Roman N. Kvyetnyy

Doctor of Technical Sciences, Professor, Corresponding Member of the National Academy of Educational Sciences of Ukraine, Professor of the Department of Automation and Intelligent Information Technologies at Vinnytsia National Technical University

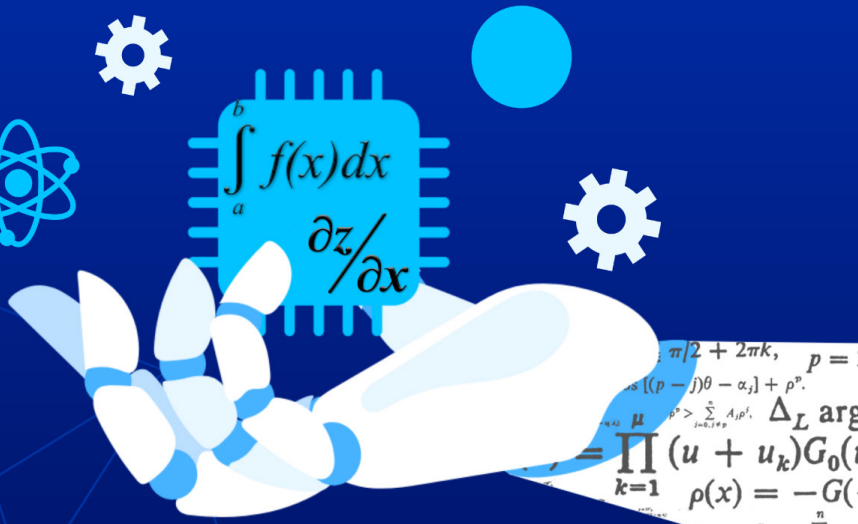
Yaroslav V. Ivanchuk

Doctor of Technical Sciences, Professor, Professor of the Department of Computer Sciences at Vinnytsia National Technical University

COMPUTATIONAL METHODS AND ALGORITHMS

R. KVYETNYY, Y. IVANCHUK

COMPUTATIONAL METHODS AND ALGORITHMS



TEXTBOOK

Ministry of Education and Science of Ukraine
Vinnytsia National Technical University

R. Kvyetnyy, Y. Ivanchuk

COMPUTATIONAL METHODS AND ALGORITHMS

Textbook

Vinnytsia
VNTU
2024

UDC 004.021+519.6
K97

Recommended by the Academic Council of the Vinnytsia National Technical University of the Ministry of Education and Science of Ukraine as a textbook for English-speaking students and postgraduates specializing in the field of information technologies (record No. 2 of 29.08.2024).

Reviewers:

P. A. Molchanov, Doctor of Technical Sciences, Professor (IPD Scientific LLC., USA)
V. V. Romaniuk, Doctor of Technical Sciences, Professor (KSTEU)
V. M. Mikhalevich, Doctor of Technical Sciences, Professor (VNTU)

Kvyetnyy, R

K97 **Computational Methods and Algorithms** : textbook [Electronic resource] / R. Kvyetnyy, Y. Ivanchuk. – Vinnytsia: VNTU, 2024. – (PDF, 281 p.)
ISBN 978-617-8163-19-8 (PDF)

The textbook examines the main applications of computational mathematics problems in engineering practice, providing examples of solutions and fragments of programs in the modern algorithmic language Python. The manual presents computational tasks that are most commonly encountered in the practice of designing computer systems and information technologies.

The textbook primarily targets specialists in the field of technical sciences. Therefore, significant attention was given to translating all the described calculation methods into specific practical algorithms. It might also be interesting for undergraduates and postgraduates learning majors related to computer science, cybernetic systems, and information technology.

UDC 004.21+519.6

ISBN 978-617-8163-19-8 (PDF)

CONTENT

INTRODUCTION 5

Chapter 1. LINEAR ALGEBRA PROBLEMS 6
1.1 Solving systems of linear algebraic equations 7
1.2 Direct calculation methods 8
1.3 Iterative methods 30
Control questions and tasks 45

Chapter 2. TASKS OF NON-LINEAR MATHEMATICS 51
2.1 Generalized formulation of the problem and the procedure for localizing the roots of the equation 53
2.2 Numerical methods of solving nonlinear algebraic equations 55
2.3 Method of determining complex roots 73
2.4 Numerical methods for solving systems of nonlinear algebraic equations 78
Control questions and tasks 85

Chapter 3. DIFFERENTIAL CALCULUS PROBLEMS 91
3.1 Generalized problem statement for ordinary differential equations 95
3.2 Numerical methods for solving ordinary differential equations for Cauchy-type problems 95
3.3 Numerical methods for solving ordinary differential equations in boundary value problems 137
3.4 Numerical methods for solving ordinary differential equations for «stiff» problems 152
Control questions and tasks 160

Chapter 4. DIFFERENTIAL EQUATIONS OF MATHEMATICAL PHYSICS 168
4.1 Classification of partial differential equations 169
4.2 Finite difference method 171
4.3 Solving various types of partial differential equations 174
Control questions and tasks 200

Chapter 5. DATA PROCESSING TASKS 205
5.1 I Interpolation 205
5.1.1 Different methods 206
5.1.2 Lagrangian interpolation 216
5.1.3 Spline interpolation 221
5.2 Data approximation 228
5.3 Statistical data processing 234
Control questions and tasks 240

Chapter 6. NUMERICAL INTEGRATION AND DIFFERENTIATION 245
6.1 Riemann sum 245
6.2 Newton-Cotes formulas 248
6.3 Chebyshev's integral 251
6.4 Gaussian quadrature 255
6.5 Algorithms for the application of numerical methods 260
6.6 Monte Carlo integration 266
6.7 Estimation of the error in numerical integration 268
6.8 Numerical differentiation 269
6.8.1 Numerical differentiation of analytically given functions 269
6.8.2 Numerical differentiation of experimental data 271
Control questions and tasks 273

Sources 279

Sources in Ukrainian 280

INTRODUCTION

The purpose of this textbook is to present the most common computational mathematics problems in engineering practice, along with solution examples and program fragments in the modern algorithmic language Python. The authors have attempted to summarize their many years of experience teaching courses in computer mathematics for students and postgraduates in specialties related to computer science, cybernetic systems, and information technologies. There are a large number of fundamental textbooks and manuals on computational mathematics, which serve as the methodological basis for this textbook.

The textbook consists of an introduction and six chapters: Introduction. Problems of linear algebra. Problems of nonlinear mathematics. Differential calculus problems. Differential equations of mathematical physics. Data processing tasks. Numerical integration and differentiation.

The authors limited themselves to computing tasks that are most often encountered in the practice of designing computer systems and information technologies. Moreover, the emphasis was placed precisely on the algorithmization of the problem-solving process. The textbook is primarily aimed at specialists in the field of technical sciences. Therefore, attention was paid to bringing all the described calculation methods to specific practical algorithms. Consequently, some methods and formulas are provided without derivation, which can be found in specialized literature. These program fragments can be easily interpreted with minimal training in the field of programming. Authors have provided illustrative examples in the popular programming languages of today, C++ and Python. The textbook is aimed at scientists and engineers in the field of computer science, information technology, robotic complexes, and computer-integrated automation and control systems. Also, the textbook will be useful for students and postgraduates of all majors in the "Information Technologies" and "Electronics and Telecommunications" fields.

At the end of the textbook, we include a list of educational literature that can be used for studying computer calculation methods. In addition to manuals and textbooks published by domestic authors and colleagues, we have also included well-known English-language sources from around the world. These works are readily available in their entirety in the information space today.

The authors show great gratitude to their colleagues who have worked with them for many years in the same team to develop a methodology for teaching disciplines related to computational methods and algorithms and their application in engineering and scientific applications.

Chapter 1. LINEAR ALGEBRA PROBLEMS

This section deals with the solution of one of the most common computational problems in linear mathematics – the solution of **linear algebraic equations systems** (LAES). Moreover, it is assumed that the user is already familiar with the main information on matrix theory.

The study of many physical systems leads to mathematical models in the form of LAES. They can appear in the process of mathematical modeling as an intermediate stage during the solution of a more complex task. There is a significant number of scientific and technical tasks in which mathematical models of complex nonlinear systems, in the form of discretization or linearization, are reduced to solving LAES.

Examples of tasks that use mathematical models in the form of LAES:

– during the simulation of economic tasks, such as management and production planning, determining the optimal placement of equipment, the optimal production plan, the optimal plan for the transportation of goods, and the distribution of personnel, a linear representation of the real world can be hypothesized. Mathematical models of such problems are described by a system of linear equations;

– during the design and operation of electrical devices, it is necessary to calculate and analyze their operation in steady-state modes. The task is reduced to the calculation of equivalent circuits, which is based on the formation and solution of LAES;

– during the construction of a mathematical model that links some parameters x_i and y_i of the object under study by functional dependence, the basis is formed by the data obtained as a result of the experiment, where $i = 1, 2, ..., n$ (setting of data approximation);

– for the study of physical processes in complex systems, mathematical models are built based on partial differential equations. As a result of approximating the original model using difference methods, under certain conditions, mathematical relations in the form of LAES are obtained.

– the essence of many physical processes is mathematically displayed using integral equations. Considering the complexity of solving many of them, it is better for the researcher to reduce the problem to solving a mathematical model in the form of LAES, using known approximation methods;

– the study of automatic control systems in a steady state often leads to static models in the form of LAES.

1.1 Solving systems of linear algebraic equations

The statement of the problem of solving LAES is to determine the unknown values $x_1, x_2, ..., x_n$ that satisfy a system of m linear algebraic equations:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1; \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2; \\ .....; \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = b_m, \end{cases} \quad (1.1)$$

where a_{11}, a_{12}, …, a_{mn} – coefficients with unknown values x_{ij}; b_1, b_2, …, b_m – free members.

A system (1.1) is called homogeneous if all of its free terms are equal to zero ($b_1=b_2=…=b_m=0$), otherwise – heterogeneous. A quadratic LAES is a system in which the number of equations coincides with the number of unknowns ($m=n$). The system of equations is indeterminate if $n>m$ and such LAES are called rectangular. If $n<m$, then the system is overdetermined.

Also, the solution of LAES is a set of n numbers c_1, c_2, …, c_n, such that when substituted into system (1.1) instead of $x_1, x_2, ..., x_n$, all of its equations become identities. System (1.1) is called compatible if it has at least one solution, and incompatible if it has no solutions. Solutions are considered different if at least one of the values of the variables does not match. A compatible system with a single solution is called determinate, and in the case of multiple solutions, it is called indeterminate.

Also, LAES (1.1) can be written in matrix form:

$$\boldsymbol{AX} = \boldsymbol{B},$$

where $\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$ – square or rectangular matrix with real or complex coefficients, $\boldsymbol{B} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$ – column of free members (given vector), $\boldsymbol{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$ – column of unknowns (searched vector).

In this section of the textbook, we will consider the methods of solving the quadratic LAES, specifically when $m = n$.

The geometric meaning of the LAES solution consists in finding the point of intersection of n-dimensional hyperplanes in an m-dimensional hyperspace. The solution is the column (vector) $\boldsymbol{X}$. If there are only two equations, then we have the case of two lines on a plane that can intersect, be parallel, or coincide. Therefore, any LAES can have: a single solution; an infinite set of solutions; or no solution at all.

A necessary and sufficient condition for the existence of a unique solution of the quadratic LAES (1.1) is the non-zero determinant A (linear independence of the equations):

$$\det \boldsymbol{A} = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{21} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} \neq 0 .$$

Methods of solving LAES can be divided into direct and iterative. Direct methods include methods that allow you to obtain an exact solution (Cramer's rule, Gaussian elimination, Tridiagonal matrix algorithm). Iterative methods include methods based on obtaining and refining successive approximations to the exact solution. Iterative methods are effective when there are many zero coefficients or a high order of the system (Gauss method is effective up to order 10^4, iterative – up to 10^6).

1.2 Direct calculation methods

The most popular direct methods include the Gaussian elimination and its variants, the Cramer's rule (determinants), the Methods of matrix inversion, and Tridiagonal matrix algorithm used in problems with diagonal matrices. However, Cramer's rule (determinants), which is discussed in detail in standard courses of higher mathematics, cannot be applied in most practical problems due to the great complexity of calculating the determinants even in the case of a small increase in the order of the system. Therefore, this section will focus on considering the Gaussian elimination, which, while inferior to iterative methods in certain practical areas, is nevertheless the most universal. Tridiagonal matrix algorithm used in problems with diagonal matrices will also be considered.

Gaussian elimination

This method is one of the most common methods of solving LAES. It is based on the idea of successively excluding unknown variables. This process transforms the original system into a triangular form, where all coefficients below the main diagonal are zero. There are various computational schemes that can be used to

implement this method. The most common schemes involve selecting the main element by row, by column, or by the entire matrix.

The classical Gaussian elimination is based on reducing the matrix $\boldsymbol{A}$ of the coefficients of system (1.1) to triangular form:

$$\begin{vmatrix} * & * & * & ... & * & * \\ 0 & * & * & ... & * & * \\ 0 & 0 & * & ... & * & * \\ \vdots & \vdots & \vdots & ... & \vdots & \vdots \\ 0 & 0 & 0 & ... & * & * \\ 0 & 0 & 0 & ... & 0 & * \end{vmatrix}$$

and consists of two staches: a direct may and an inverted substitution. The direct run step ends when one of the system's equations becomes an equation with only one unknown. Next, the inverse substitution is performed to determine all unknown x_{ij} values. In order to minimize calculation errors, the Gaussian elimination is used with the selection of the main element. By main element, we refer to the maximum element $a_{ij}^{\max}$ of matrix $\boldsymbol{A}$, chosen from a given set of rows and columns. The algorithm of this method is as follows: first, identify the main element $a_{ij}^{\max}$ of matrix $\boldsymbol{A}$ and rearrange the corresponding equations and columns to position it in the place of the element a_{11} (remember to record the rearrangement of columns to correctly match the variables with the obtained values). Then, normalize the first equation by dividing it by $a_{11}=a_{ij}^{\max}$:

$$x_1 + \frac{a_{12}}{a_{ij}^{\max}} x_2 + ... + \frac{a_{1n}}{a_{ij}^{\max}} x_n = \frac{b_1}{a_{ij}^{\max}}. \qquad (1.2)$$

By successively multiplying (1.2) by a_{21}, a_{31}, …, a_{n1}, and subtracting the respective 2nd, ..., n-th equations from system (1.1), we obtain a LAES of the form $\boldsymbol{A}^1\boldsymbol{X}=\boldsymbol{B}^1$:

$$\begin{cases} x_1 + a_{12}^1 x_2 + \ldots + a_{1n}^1 x_n = b_1^1; \\ 0 + a_{22}^1 x_2 + \ldots + a_{2n}^1 x_n = b_2^1; \\ .....; \\ 0 + a_{n2}^1 x_2 + \ldots + a_{nn}^1 x_n = b_n^1, \end{cases}$$

where $a_{ij}^1 = a_{ij} - a_{1j}^1 \cdot a_{i1}$, $b_i^1 = b_i - b_1^1 \cdot a_{i1}$ $\forall (i = \overline{2,n}$ і $j = \overline{2,n})$; $a_{1j}^1 = a_{1n} / a_{ij}^{\max}$; $b_1^1 = b_1 / a_{ij}^{\max}$.

Let the matrices $\boldsymbol{A}^{\kappa}$ and $\boldsymbol{B}^{\kappa}$ be obtained at some step (the initial matrices correspond to $\boldsymbol{A}^0$ and $\boldsymbol{B}^0$), and the diagonal element $a^k_{k+1,k+1}$ is the maximum modulus element of the matrix $\boldsymbol{A}^{\kappa}$ for rows $i > k$ and columns $j > k$ (if necessary, the corresponding equations and columns are rearranged).

We will provide formulas for calculating matrices $\boldsymbol{A}^{k+1}$ and $\boldsymbol{B}^{k+1}$:

$$a_{ij}^{k+1} = \begin{cases} a_{ij}^k & (i \le k \text{ або } j \le k); \\ m_{i,j}^k = a_{k+1,j}^k / a_{i,k+1}^k & (i = k+1,\ j > k); \\ a_{ij}^k - m_{i,j}^k \cdot a_{k+1,j}^{k+1} & (i > k+1,\ j > k), \end{cases} \tag{1.3}$$

$$b_i^{k+1} = \begin{cases} b_i^k & (i \le k); \\ b_{k+1}^k / a_{k+1,k+1}^k & (i = k+1); \\ b_i^k - b_{k+1}^{k+1} \cdot a_{i,k+1}^k & (i > k+1). \end{cases} \tag{1.4}$$

When programming the method, it is sufficient to use a two-dimensional array with n rows and $n + 1$ columns to store the information of the coefficient values of the matrices. This array will hold the extended matrix of the system ($\boldsymbol{A}|\boldsymbol{B}$).

Conditions for terminating the direct run of the Gaussian elimination

A problem may arise during the calculation process:

$$\text{element } a^k_{k+1,k+1} = 0 .$$

This situation occurs when all elements of the matrix $\boldsymbol{A}^k$ in rows $i > k$ are equal to zero. The system in this case looks like this:

$$\begin{cases} x_1 + a_{12}^k x_2 + \ldots + a_{1n}^k x_n = b_1^k; \\ 0 + a_{22}^k x_2 + \ldots + a_{2n}^k x_n = b_2^k; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\ 0 + \ldots 0 x_{k-1} + a_{kk}^k x_k + \ldots + a_{kn}^k x_n = b_k^k; \\ 0 + \ldots 0 x_{k-1} + 0 x_k + 0 x_{k+1} + \ldots + 0 x_n = b_{k+1}^k; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\ 0 + \ldots 0 x_{k-1} + 0 x_k + 0 x_{k+1} + \ldots + 0 x_n = b_n^k. \end{cases}$$

If the application of formulas (1.3) and (1.4) is impossible, then the system has an infinite set of solutions or none at all, which can be determined from the matrix $\boldsymbol{B}^k$.

If all elements $b_i^k = 0$ under the condition $i \geq k+1$, then the system has an infinite set of solutions, and the roots x_1, …, x_k are called dependent and are expressed through the values x_{k+1}, …, x_n, which are called independent. If at least one element $b_i^k \neq 0$ for $i \geq k+1$, then the system has no solutions.

In the absence of the case of the problem of dividing by zero and obtaining the triangular matrix $\boldsymbol{A}^n$, the system has a unique solution. Then the Gaussian elimination is used to find its inverse:

$$\begin{cases} x_n = b_n^n; \\ x_{n-i} = b_{n-i}^n - \sum\limits_{k=1}^{i} a_{n-k+1}^n \cdot x_{n-k+1} \quad (i = \overline{1, n-1}). \end{cases} \tag{1.5}$$

Example 1.1. Solve the LAES by the Gaussian elimination with the selection of the main element:

$$\begin{cases} 5 \cdot x_1 + 6 \cdot x_2 + 7 \cdot x_3 + 8 \cdot x_4 = 1; \\ 10 \cdot x_1 + 10 \cdot x_2 + 11 \cdot x_3 + 12 \cdot x_4 = 2; \\ 15 \cdot x_1 + 4 \cdot x_2 - 3 \cdot x_3 + 5 \cdot x_4 = 3; \\ 2 \cdot x_1 + 20 \cdot x_3 - 2 \cdot x_4 = 4. \end{cases}$$

Solution:

A matrix of coefficients $\boldsymbol{A}$ and a column of free members $\boldsymbol{B}$ are formed:

$$\boldsymbol{A} = \begin{pmatrix} 5 & 6 & 7 & 8 \\ 10 & 10 & 11 & 12 \\ 15 & 4 & -3 & 5 \\ 2 & 0 & 20 & -2 \end{pmatrix}; \quad \boldsymbol{B} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}.$$

The main element of matrix $\boldsymbol{A}$ is equal to $a_{13}^{\max} = a_{43} = 20$. After rearranging the rows, a new value of a_{11} is obtained, namely a_{11}=20, and the values of the new matrices will be obtained.

$$\boldsymbol{A} = \begin{pmatrix} 2 & 0 & 20 & -2 \\ 10 & 10 & 11 & 12 \\ 15 & 4 & -3 & 5 \\ 5 & 6 & 7 & 8 \end{pmatrix}; \quad \boldsymbol{B} = \begin{pmatrix} 4 \\ 2 \\ 3 \\ 1 \end{pmatrix}.$$

Let's calculate the elements of matrix $\boldsymbol{A}^1$ and column $\boldsymbol{B}^1$:

$$\left(\begin{array}{cccc|c} x_1 & x_2 & x_3 & x_4 & b \\ 2 & 0 & 20 & -2 & 4 \\ 10 & 10 & 11 & 12 & 2 \\ 15 & 4 & -3 & 5 & 3 \\ 5 & 6 & 7 & 8 & 1 \end{array}\right) = \left(\begin{array}{cccc|c} x_3 & x_2 & x_1 & x_4 & b \\ 20 & 0 & 2 & -2 & 4 \\ 11 & 10 & 10 & 12 & 2 \\ -3 & 4 & 15 & 5 & 3 \\ 7 & 6 & 5 & 8 & 1 \end{array}\right) =$$

$$= \left(\begin{array}{cccc|c} x_3 & x_2 & x_1 & x_4 & b \\ 20/20 & 0/20 & 2/20 & -2/20 & 4/20 \\ 11 & 10 & 10 & 12 & 2 \\ -3 & 4 & 15 & 5 & 3 \\ 7 & 6 & 5 & 8 & 1 \end{array}\right) = \left(\begin{array}{cccc|c} x_3 & x_2 & x_1 & x_4 & b \\ 1 & 0 & 0{,}1 & -0{,}1 & 0{,}2 \\ 11 & 10 & 10{,}0 & 12{,}0 & 2{,}0 \\ -3 & 4 & 15{,}0 & 5{,}0 & 3{,}0 \\ 7 & 6 & 5{,}0 & 8{,}0 & 1{,}0 \end{array}\right) =$$

$$= \left(\begin{array}{cccc|c} x_3 & x_2 & x_1 & x_4 & b \\ 1 & 0 & 0{,}1 & -0{,}1 & 0{,}2 \\ 11-(1\cdot 11) & 10-(0\cdot 11) & 10-(0{,}1\cdot 11) & 12-(-0{,}1\cdot 11) & 2-(0{,}2\cdot 11) \\ -3-(1\cdot -3) & 4-(0\cdot -3) & 15-(0{,}1\cdot -3) & 5-(-0{,}1\cdot -3) & 3-(0{,}2\cdot -3) \\ 7-(1\cdot 7) & 6-(0\cdot 7) & 5-(0{,}1\cdot 7) & 8-(-0{,}1\cdot 7) & 1-(0{,}2\cdot 7) \end{array}\right) =$$

$$= \left(\begin{array}{cccc|c} x_3 & x_2 & x_1 & x_4 & b \\ 1 & 0 & 0{,}1 & -0{,}1 & 0{,}2 \\ 0 & 10 & 8{,}9 & 13{,}1 & -0{,}2 \\ 0 & 4 & 15{,}3 & 4{,}7 & 3{,}6 \\ 0 & 6 & 4{,}3 & 8{,}7 & -0{,}4 \end{array}\right).$$

$$\boldsymbol{A}^1 = \begin{pmatrix} x_3 & x_2 & x_1 & x_4 \\ 1 & 0 & 0{,}1 & -0{,}1 \\ 0 & 10{,}0 & 8{,}9 & 13{,}1 \\ 0 & 4{,}0 & 15{,}3 & 4{,}7 \\ 0 & 6{,}0 & 4{,}3 & 8{,}7 \end{pmatrix}; \quad \boldsymbol{B}^1 = \begin{pmatrix} 0{,}2 \\ -0{,}2 \\ 3{,}6 \\ -0{,}4 \end{pmatrix}.$$

The main element of the new matrix $a_{33}^{\max} = a_{33}^1 = 15{,}3$.

Similarly, the elements of matrix $\boldsymbol{A}^2$ and column $\boldsymbol{B}^2$ are calculated:

$$\boldsymbol{A}^2 = \begin{pmatrix} x_3 & x_1 & x_2 & x_4 \\ 1 & 0{,}1 & 0{,}000 & -0{,}100 \\ 0 & 1{,}0 & 0{,}260 & 0{,}310 \\ 0 & 0{,}0 & 7{,}677 & 10{,}368 \\ 0 & 0{,}0 & 4{,}878 & 7{,}380 \end{pmatrix}; \quad \boldsymbol{B}^2 = \begin{pmatrix} 0{,}200 \\ 0{,}240 \\ -2{,}292 \\ -1{,}411 \end{pmatrix}.$$

The main element of the new matrix $a_{34}^{\max} = a_{34}^2 = 10{,}368$.

Similarly, the elements of matrix $\boldsymbol{A}^3$ and column $\boldsymbol{B}^3$ are calculated:

$$\boldsymbol{A}^3 = \begin{pmatrix} x_3 & x_1 & x_4 & x_2 \\ 1 & 0{,}1 & -0{,}10 & 0{,}000 \\ 0 & 1{,}0 & 0{,}310 & 0{,}260 \\ 0 & 0{,}0 & 1{,}000 & 0{,}740 \\ 0 & 0{,}0 & 0{,}000 & -0{,}598 \end{pmatrix}; \qquad \boldsymbol{B}^3 = \begin{pmatrix} 0{,}200 \\ 0{,}240 \\ -0{,}23 \\ 0{,}220 \end{pmatrix}.$$

The $\boldsymbol{A}^3$ matrix has a triangular shape. The system has a unique solution. To find it, the inverse of the Gaussian elimination is used:

$$x_2 = b_4^3 / a_{44}^4 = 0{,}220 / -0{,}598 = -0{,}377;$$
$$x_4 = b_3^3 - a_{34}^3 \cdot x_2 = -0{,}23 - 0{,}74 \cdot -0{,}377 = 0{,}058;$$
$$x_1 = b_2^3 - a_{24}^3 \cdot x_2 - a_{23}^3 \cdot x_4 = 0{,}24 - (0{,}26 \cdot -0{,}377) - (0{,}310 \cdot 0{,}058) = 0{,}316;$$
$$x_3 = b_1^3 - a_{14}^3 \cdot x_2 - a_{13}^3 \cdot x_4 - a_{12}^3 \cdot x_1 =$$
$$= 0{,}2 - (0 \cdot 0{,}377) - (-0{,}10 \cdot 0{,}058) - (0{,}1 \cdot 0{,}316) = 0{,}174.$$

Solution:

$$\boldsymbol{X} = \begin{pmatrix} 0{,}316 \\ -0{,}377 \\ 0{,}174 \\ 0{,}058 \end{pmatrix}.$$

The algorithm of the method is presented in Figure 1.1.

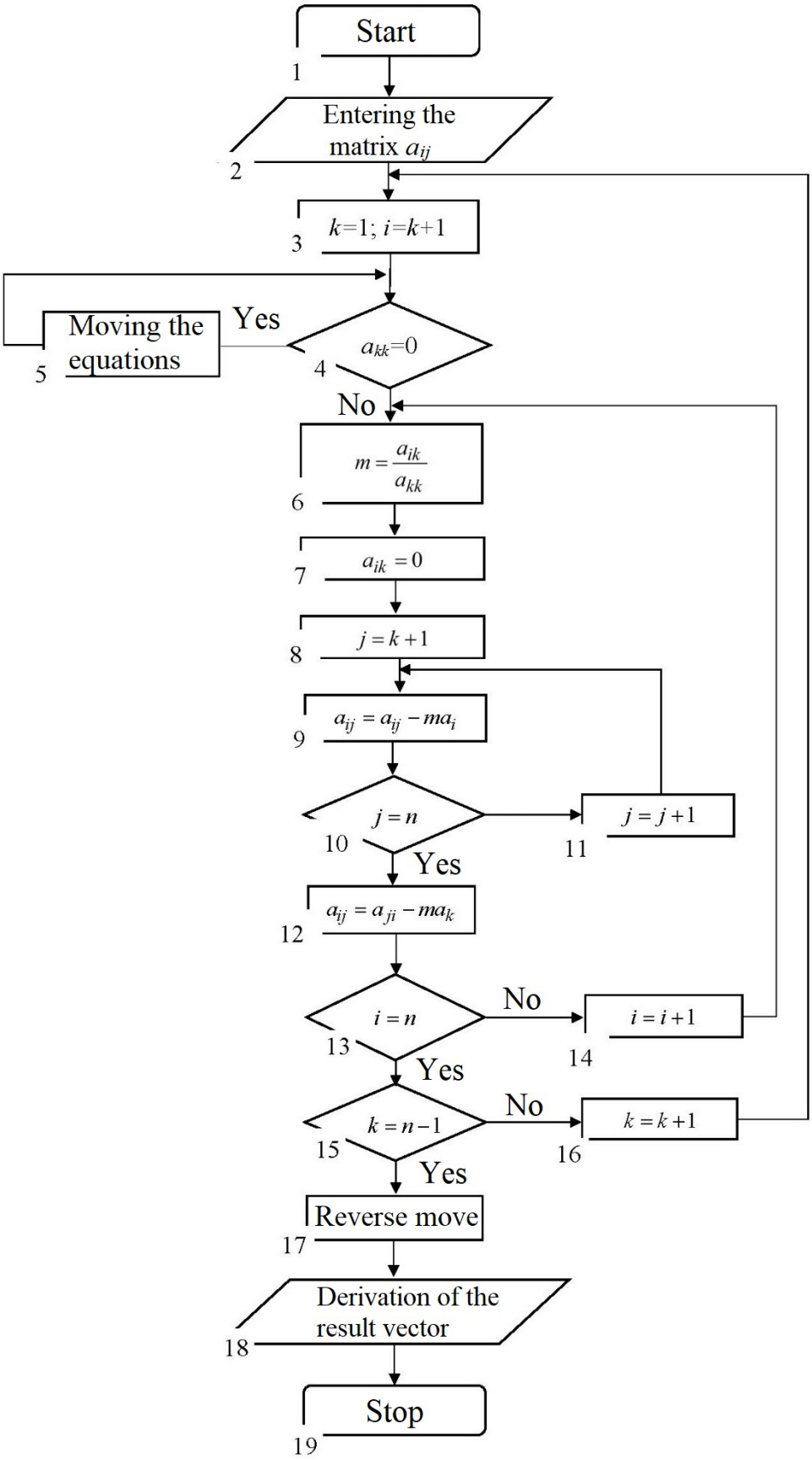


Figure 1.1 – Scheme of the Gaussian elimination algorithm

Mechanical interpretation of the Gaussian elimination

Consider an arbitrary elastic static system S fixed at the edges (for example, a string, an elastic rod, a multi-span rod, a membrane, a plate, or a discrete system), and take n points $x_1, x_2, ..., x_n$ on it.

We consider the displacement (deflections) $y_1, y_2, ..., y_n$ of points $x_1, x_2, ..., x_n$, xp of the system S under the action of forces $F_1, F_2, ..., F_n$ applied at these points. It is assumed that forces and displacements are parallel to one and the same direction and are therefore determined by their algebraic values (Fig. 1.2).

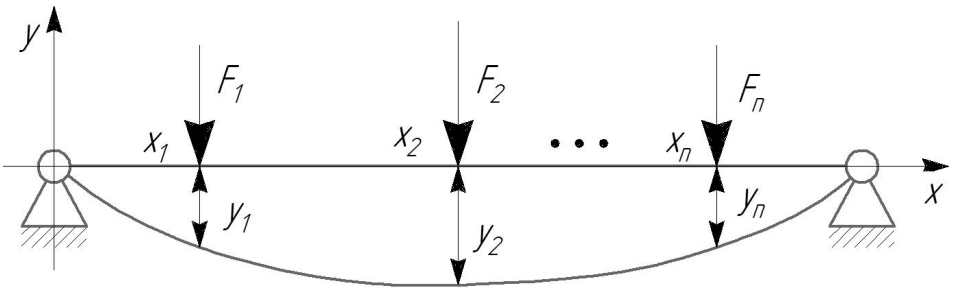


Figure 1.2 – Calculation diagram of the deflection of an elastic beam

In addition, it is necessary to accept that the principle of linear imposition of forces applies:

1) with the total superposition of two systems of forces, the corresponding deflections add up;

2) when the values of all forces are multiplied by the same real number, all deflections are multiplied by this number.

Let a_{ij} denote the coefficient of influence of point x_j on point x_i, that is, the deflection at point x_i under the action of a unit force applied at point x_j (i, j=1, 2, …, n) (Fig. 1.3). Then, with the joint action of forces $F_1, F_2, ..., F_n$ deflections $y_1, y_2, ..., y_n$ are determined by the formulas:

$$\sum_{j=1}^{n} a_{ij} F_j = y_i \quad (i=1, 2, \ldots, n). \tag{1.6}$$

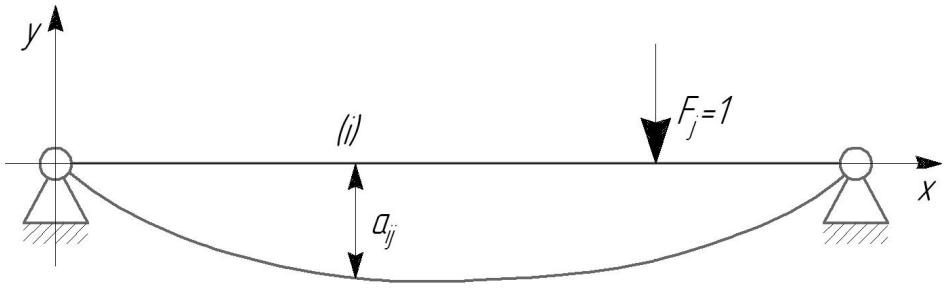


Figure 1.3 – Calculation diagram of deflection under the action of a unit force

By comparing the system of equations (1.6) with the original system of equations (1.1), we can interpret the problem of finding a solution to the system of equations as determining the deflections $y_1, y_2, ..., y_n$ to the corresponding forces $F_1, F_2, ..., F_n$.

Let us denote by S_p the static system obtained from S by introducing p fixed hinged supports at points x_1, x_2, ..., x_p ($p \le n$). The influence coefficients for the remaining moving points x_{p+1}, ..., x_n of the system S_p will be denoted by a_{ij}^p (i, j=p+1, …, n) (Fig. 1.4 for p=1).

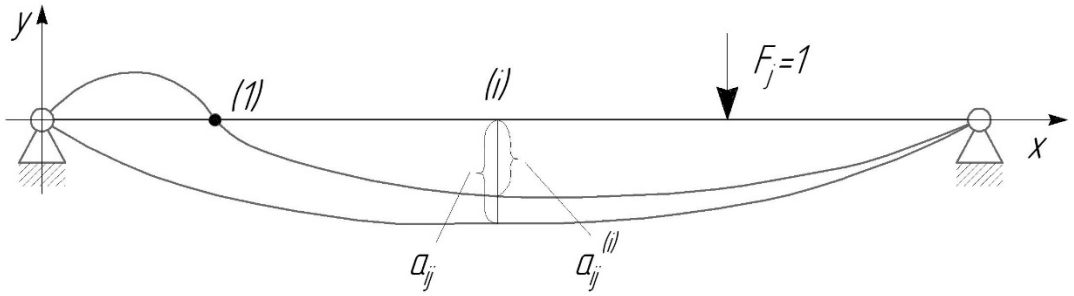


Figure 1.4 – Calculation diagram of the effect of neighboring points on the overall deflection of an elastic beam

Coefficient a_{ij}^p can be considered as the deflection at point x_i of system S under the action of a unit force at point x_k and reaction forces R_1, R_2, ..., R_n at fixed points x_1, x_2, ..., x_n. Therefore:

$$a_{ij}^p = R_1 a_{i1} + \ldots + R_p a_{ip} + a_{ij}. \tag{1.7}$$

With the same forces, the deflections of system S at points x_1, x_2, ..., x_n are equal to zero:

$$\begin{cases} R_1 a_{11} + \ldots + R_p a_{1p} + a_{1j} = 0; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ R_1 a_{p1} + \ldots + R_p a_{pp} + a_{pj} = 0. \end{cases} \tag{1.8}$$

If

$$A\begin{pmatrix} 1 & 2 & \ldots & p \\ 1 & 2 & \ldots & p \end{pmatrix} \neq 0, \tag{1.9}$$

then we can determine R_1, R_2, ..., R_p from (1.8) and substitute the obtained expressions in (1.7). This exception R_1, R_2, ..., R_p can also be done. To the system of equalities (1.8), we add equality (1.7), written in the form:

$$R_1 a_{i1} + \ldots + R_p a_{ip} + a_{ij} - a_{ij}^p = 0. \tag{1.10}$$

Considering (1.8) and (1.10) as a system p+1 of homogeneous equations with a nonzero solution R_1, R_2, ..., R_{p+1}=1, we obtain that the determinant of this system is zero:

$$\begin{vmatrix} a_{11} & \cdots & a_{1p} & a_{1j} \\ \cdots & \cdots & \cdots & \cdots \\ a_{p1} & \cdots & a_{pp} & a_{pj} \\ a_{i1} & \cdots & a_{ip} & a_{ij} - a_{ij}^{k} \end{vmatrix} = 0. \qquad (1.11)$$

Where from

$$a_{ij}^{p} = \frac{A\begin{pmatrix} 1 & 2 & \cdots & p & i \\ 1 & 2 & \cdots & p & j \end{pmatrix}}{A\begin{pmatrix} 1 & 2 & \ldots & p \\ 1 & 2 & \ldots & p \end{pmatrix}} \ (i, j=p+1, \ldots, n). \qquad (1.9)$$

According to these formulas, the influence coefficients of the «support» system S_p are expressed through the influence coefficients of the initial system S.

For any $p \le n$-1, the coefficients a_{ij}^{p} (i, j=p+1, …, n) in the Gaussian elimination are the coefficients of the influence of the support system S_p.

To confirm this basic premise, it is possible to ensure this through purely mechanical considerations. To do so, we will first consider the special case of one support: p=1 (see Fig. 1.4). In this case, the influence coefficients of system S_1 are defined as:

$$a_{ij}^{1} = \frac{A\begin{pmatrix} 1 & i \\ 1 & j \end{pmatrix}}{A\begin{pmatrix} 1 \\ 1 \end{pmatrix}} = a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j} \ (i, j=1, 2, \ldots, n). \qquad (1.10)$$

Thus, if the coefficients a_{ij} (i, j=1, 2, …, n) in the system of equations (1.1) represent the coefficients of influence of static system S, then the coefficients a_{ij}^{1} (i, j=2, …, n) in the Gaussian elimination are the coefficients of influence of system S_1. By applying the same considerations to system S_1 and introducing the second resistance at point x_2, it can be concluded that the coefficients a_{ij}^{2} (i, j=3, …, n) in the system of equations (1.2) are the coefficients of influence of the support system S_2. In general, for any $p \le n$–1, the coefficients a_{ij}^{p} (i, j=p+1, …, n) in the Gaussian elimination are the coefficients of influence of the support system S_p.

From mechanical considerations, it is evident that the sequential introduction of p supports is equivalent to the simultaneous introduction of these supports.

Consider the implementation of the Gaussian elimination in the C++ programming language:

```
int main()
{
    int i, j, n, m;
    //we create an array
    cout << "Number of equations: ";
```

```
    cin >> n;
    cout << "Number of variables: ";
    cin >> m;
    m += 1;
    float **matrix = new float *[n];
    for (i = 0; i<n; i++)
        matrix[i] = new float[m];
    for (i = 0; i<n; i++)
        for (j = 0; j<m; j++)
        {
            cout << " Element " << "[" << i + 1 << " , " << j + 1 << "]: ";
            cin >> matrix[i][j];
        }
    //we output the array
    cout << "matrix: " << endl;
    for (i = 0; i<n; i++)
    {
        for (j = 0; j<m; j++)
            cout << matrix[i][j] << " ";
        cout << endl;
    }
    cout << endl;
    //Gauss method, straight line
    float  tmp;
    int k;
    float *xx = new float[m];
    for (i = 0; i<n; i++)
    {
        tmp = matrix[i][i];
        for (j = n; j >= i; j--)
            matrix[i][j] /= tmp;
        for (j = i + 1; j<n; j++)
        {
            tmp = matrix[j][i];
            for (k = n; k >= i; k--)
                matrix[j][k] -= tmp*matrix[i][k];
        }
    }
       //Gauss method, reverse course
    xx[n - 1] = matrix[n - 1][n];
    for (i = n - 2; i >= 0; i--)
    {
        xx[i] = matrix[i][n];
        for (j = i + 1; j<n; j++) xx[i] -= matrix[i][j] * xx[j];
    }
     //We make a decision
    for (i = 0; i<n; i++)
        cout << xx[i] << " ";
    cout << endl;
    delete[] matrix;
    system("pause");
    return 0;
}.
```

Modified Gaussian elimination

In many cases, it is necessary to solve the LAES with a matrix of variable coefficients and constant values of the column of free members. Most often, the modified Gaussian elimination is used to solve such problems. In this method, the matrix of coefficients A from the matrix equation (1.1) is presented as a product of left and right triangular matrices:

$$\boldsymbol{L}\cdot\boldsymbol{R}=\boldsymbol{A}.$$

Since the diagonal elements of one of the matrices are equal to one, they cannot be memorized. This then allows both matrices to be stored in the memory of the computer system in place of the matrix of coefficients.

In a variant of Crout matrix decomposition, the following sequence is used to find elements of matrices $\boldsymbol{L}$ and $\boldsymbol{R}$:

$$\begin{cases} l_{ik}=a_{ik}-\sum\limits_{p=1}^{k-1}l_{ip}r_{pk}\ \ (k=1,\ 2,\ \ldots,n;\ i=k,k+1,\ldots,n); \\ l_{kk}=\dfrac{1}{l_{kk}}; \\ r_{kj}=l_{kk}\left(a_{kj}-\sum\limits_{p=1}^{k-1}l_{kp}r_{pj}\right)\ (j=k+1,\ \ldots,n); \\ r_{kk}=1. \end{cases}$$

The system $\boldsymbol{AX}=\boldsymbol{C}$ is reduced to the system $\boldsymbol{LRX}=\mathbf{C}$, the solution of which is replaced by the solution of two systems with triangular matrices:

$$\begin{cases} \boldsymbol{LY}=\boldsymbol{C}; \\ \boldsymbol{RX}=\boldsymbol{Y}. \end{cases}$$

The elements of the matrices $\boldsymbol{Y}$ and $\boldsymbol{X}$ are determined from the following ratios:

$$\begin{cases} y_1=l_{11}c_1; \\ y_i=l_{ii}\left(c_i-\sum\limits_{p=1}^{i-1}l_{ip}y_p\right)\ (i=2,\ldots,n); \\ x_i=y_i-\sum\limits_{p=i+1}^{n}r_{ip}x_p\ \ (i=n,n-1,\ldots,1). \end{cases}$$

The number of arithmetic operations required to solve the LAES by this method is $N=2n^2$.

Example 1.2. Three pipelines with corresponding capacities of x_1, x_2, and x_3 are connected to a measuring tank that can be filled with liquid. To determine the capacities of the connected pipelines, three experiments are carried out on filling and emptying this tank. Additionally, the pipelines operate for different periods of time. In the first case, the liquid is drained from the filled container, as indicated by the «minus» sign before the volume value. In the second case, the tank is filled, as indicated by the «plus» sign before the volume value. In the third case, the

cavity remains empty when the three pipelines work together. It will determine the values of the throughput of the connected pipelines x_1, x_2, and x_3. Data on the time and volume of filling the container are indicated in Table 1.1.

Table 1.1 – Filling tank data

Method of filling	Pipeline operation time (min) / type of operation (filling / draining)			Filling volume (m^3)
	1	2	3	
1	1 (filling)	3 (draining)	2 (filling)	–5
2	3 (draining)	1 (filling)	3 (draining)	3
3	3 (draining)	3 (draining)	1 (filling)	0

Solution:

According to the problem statement and based on the data provided in Table 1.1, we formulate a system of equations:

$$\begin{cases} x_1 - 3x_2 + 2x_3 = -5; \\ -2x_1 + x_2 - x_3 = 3; \\ -x_1 - 2x_2 + 3x_3 = 0. \end{cases}$$

Solve the system of equations using the modified Gaussian elimination with calculation of the forward and backward steps.

Straight stroke:

1) exclude x_1 from the 2nd and 3rd equations:

$-m_2^{(1)} = -a_{21}/a_{11} = 2/1 = 2;$ $\quad a_{21}^{(1)} = a_{21}+m_2^{(1)}\cdot a_{11} = -2+2\cdot 1 = 0;$

$a_{22}^{(1)} = a_{22}+m_2^{(1)}\cdot a_{12} = 1-2\cdot 3 = -5;$ $\quad a_{23}^{(1)} = a_{23}+m_2^{(1)}\cdot a_{13} = -1+2\cdot 2 = 3;$

$b_2^{(1)} = b_2+m_2^{(1)}\cdot b_1 = 3-2\cdot 5 = -7.$

$-m_3^{(1)} = -a_{31}/a_{11} = 1/1 = 1;$ $\quad a_{31}^{(1)} = a_{31}+m_3^{(1)}\cdot a_{11} = -1+1\cdot 1 = 0;$

$a_{32}^{(1)} = a_{32}+m_3^{(1)}\cdot a_{12} = -2-1\cdot 3 = -5;$ $\quad a_{33}^{(1)} = a_{33}+m_3^{(1)}\cdot a_{13} = 3+1\cdot 2 = 5;$

$b_3^{(1)} = b_3+m_3^{(1)}\cdot b_1 = 0-1\cdot 5 = -5;$

$$\begin{cases} x_1 - 3x_2 + 2x_3 = -5; \\ 0\cdot x_1 - 5x_2 + 3x_3 = -7; \\ 0\cdot x_1 - 5x_2 + 5x_3 = -5. \end{cases}$$

2) exclude x_2 from the 3rd equation:

$-m_3^{(2)} = -a_{32}^{(1)}/a_{22}^{(1)} = -5/5 = -1;$ $\quad a_{32}^{(2)} = a_{32}^{(1)}+m_3^{(2)}\cdot a_{22}^{(1)} = -5+1\cdot 5 = 0;$

$a_{33}^{(2)} = a_{33}^{(1)}+a_{33}^{(2)} = a_{33}^{(1)}+m_3^{(2)}\cdot a_{23}^{(1)} = 5-1\cdot 3 = 2;$

$b_3^{(2)} = b_3^{(1)}+m_3^{(2)}\cdot b_2^{(1)} = -5+1\cdot 7 = 2;$

$$\begin{cases} x_1 - 3x_2 + 2x_3 = -5; \\ -5x_2 + 3x_3 = -7; \\ 2x_3 = 2. \end{cases}$$

Reverse stroke:

$x_3 = b_3^{(2)}/a_{33}^{(2)} = 2/2 = 1{,}0$ m^3/min;

$x_2 = (b_2^{(2)} - a_{23}^{(2)} \cdot x_3)/ a_{22}^{(2)} = (-7-3\cdot 1)/(-5) = 2{,}0$ m^3/min;

$x_1 = (b_1^{(2)} \cdot a_{33}^{(1)} + a_{13}^{(2)} \cdot x_3 \; a_{33}^{(1)} + a_{12}^{(2)} \cdot x_2)/a_{11}^{(2)} = (-5 - (2\cdot 1) - (-3\cdot 2))/1 =$
$= -1{,}0$ m^3/min.

The «minus» sign, at the value of pipeline capacity x_1, means that the first pipeline was working all the time to drain the liquid.

Consider the implementation of the modified Gaussian elimination in the C++ programming language:

```
const int n = 3;
float A[n][n]= {2, 4, 1,
                1, 1, 2,
                4, 2, 1};
float B[n] =   {8, 6, 8};
int main(int argc, char *argv[]) {
  // Derivation of the initial matrix
  for (int i=0; i<n; i++)
  {
    for (int j=0; j<n; j++)
      cout << A[i][j] << "  ";
    cout << B[i] << endl;
  }
    cout << endl;
// Modified Gauss method
  for (int i=0; i<n; i++)
  {
    for (int j=0 ; j<n; j++)
     if (i!= j)
     {
        float d = A[j][i]/A[i][i];
        for (int k=0; k<n; k++)
               A[j][k]= A[i][k]*d - A[j][k];
           B[j] = B[i]*d - B[j];  }
// Derivation of intermediate results
  for (int i=0; i<n; i++)
  {
    for (int j=0; j<n; j++)
      cout << A[i][j] << "  ";
    cout << B[i] << endl;
  }
  cout << endl; }
// Print x
  for (int j=0; j<n; j++)
      cout << "x[" << j << "] = " << B[j]/A[j][j] << endl;
  system("PAUSE");
  return 0;
    }.
```

The algorithm of the modified Gaussian elimination is shown in Figure 1.5.

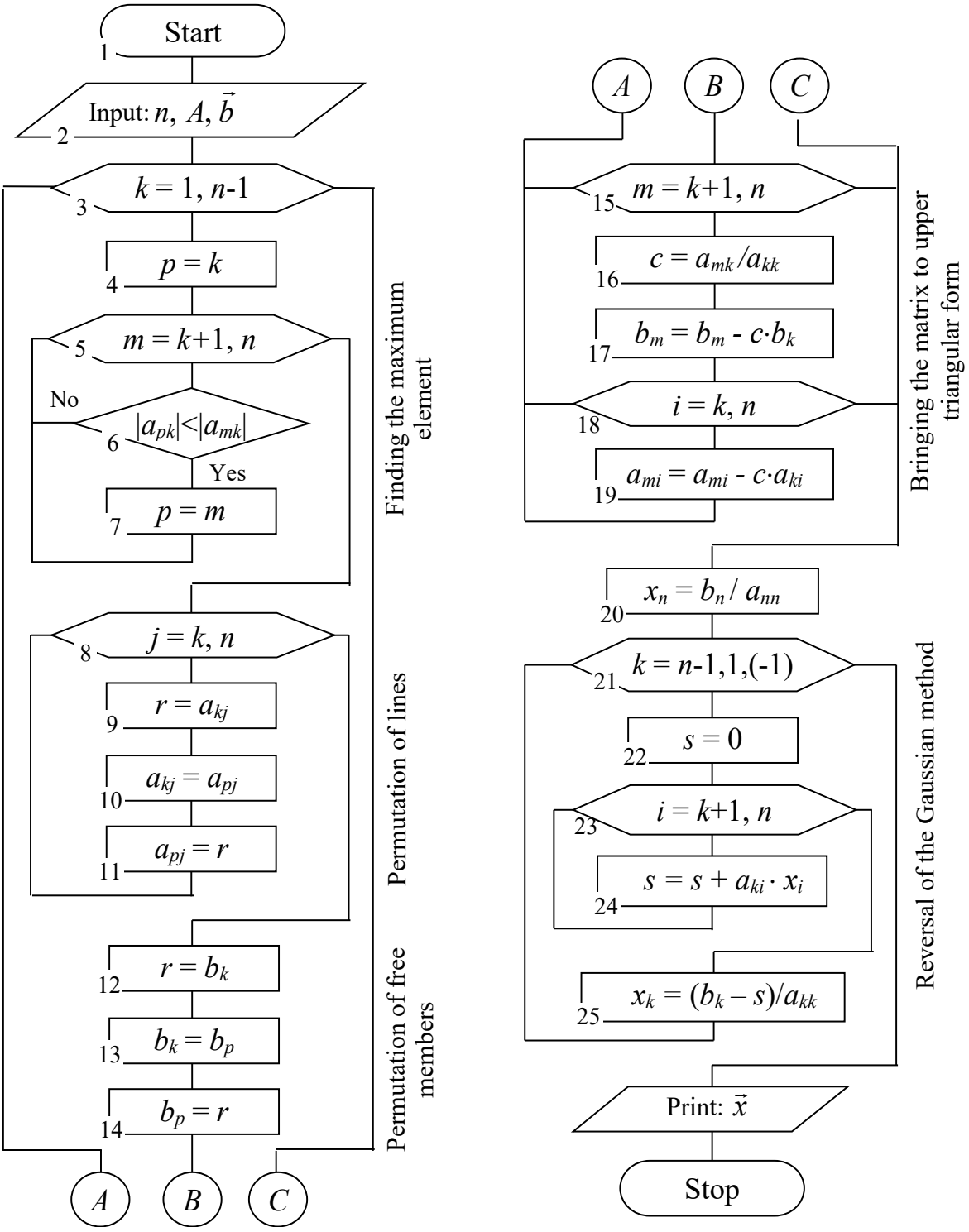


Figure 1.5 – Scheme of the modified Gaussian elimination algorithm

Application of the Gaussian straight line elimination for finding determinants

Two approaches are used to calculate the determinants of matrices:

– recursive calculation using the expansion of the matrix of coefficients by row or column;

– calculation based on the direct course of the Gaussian elimination.

The first method is based on the use of the property of determinants, where the determinant of the matrix is equal to the sum of the products of the elements of any row or column by their algebraic complement:

$$\det(\boldsymbol{A}) = \sum_{j=1}^{n} a_{ij} \cdot \boldsymbol{A}_{ij}, \forall\, i = \overline{1, n}.$$

Thus, the calculation of one determinant of the n-th order is reduced to the calculation of n determinants of the n–1 order. This method is implemented using recursion.

The recursive method is convenient for applying to rows or columns with a large number of zero elements. If there are no or very few zero elements in the matrix, then using this method is extremely inefficient. For a determinant of order n, it will be necessary to calculate $n!/2$ determinants of the second order.

The second method is based on the Gaussian straight-line elimination, which uses the property of the determinant of a triangular matrix. For such a matrix, the determinant is equal to the product of the elements on the main diagonal.

To calculate the determinant, the algorithm for constructing the sequence of matrices $\boldsymbol{A} \rightarrow \boldsymbol{A}^1 \rightarrow \boldsymbol{A}^2 \rightarrow \ldots \rightarrow \boldsymbol{A}^n$ of the Gaussian elimination is used, with the difference that the sign of the determinant changes to the opposite during the permutation of rows or columns. The value of the determinant is calculated according to the formula:

$$\det(\boldsymbol{A}) = (-1)^m \cdot a_{11} \cdot a_{22}^1 \cdot \ldots \cdot a_{nn}^{n-1},$$

where m – number of permutations.

This method allows you to calculate the determinants of matrices of high order.

Methods of matrix inversion

If the problem of solving LAES is addressed in an application package that implements the function of calculating the inverse matrix, then the formula can be used to find a solution:

$$\boldsymbol{X} = \boldsymbol{A}^{-1}\boldsymbol{B},$$

where $\boldsymbol{A}^{-1}$ – inverse matrix.

Recall the definition of the inverse matrix.

The inverse of the square matrix $\boldsymbol{A}$ is the matrix $\boldsymbol{A}^{-1}$ for which the relation holds:

$$\boldsymbol{A} \cdot \boldsymbol{A}^{-1} = \boldsymbol{A}^{-1} \cdot \boldsymbol{A} = \boldsymbol{E},$$

where $\boldsymbol{E}$ – unit matrix.

Cramer's rule

This method consists of calculating the determinant $\det(\boldsymbol{A})$ of the coefficient matrix $\boldsymbol{A}$, as well as the determinants $\det(\boldsymbol{A}_k)$ of the matrices $\boldsymbol{A}_k$ ($k = \overline{1, n}$). The matrices $\boldsymbol{A}_k$ are obtained from matrix $\boldsymbol{A}$ by replacing the k-th column of coefficients with the column $\boldsymbol{B}$ of the free members of the system of linear equations LAES.

In this case, the following variants of the determinants' values themselves are possible:

1. If det$\boldsymbol{A}\neq 0$, then the system has a unique solution $\overline{x} = (x_1, ..., x_n)$, which is determined by the formula :

$$x_k = \frac{\det(\boldsymbol{A}_k)}{\det(\boldsymbol{A})} \quad (k = \overrightarrow{1, n}).$$

2. If det($\boldsymbol{A}$)=0, as well as all det($\boldsymbol{A}_k$)=0 for $k = \overline{1, n}$, then the system has an infinite set of solutions.

3. If det($\boldsymbol{A}$)=0 and at least one det($\boldsymbol{A}_k$) $\neq$0, then the system has no solutions.

Cramer's rule (determinants) cannot be applied to most practical problems due to the complexity of directly calculating the determinants themselves, even in the case of a small increase in the system's order.

Example 1.3. The trading network consists of three different trading enterprises with profitability x_1, x_2, and x_3, respectively. Over the course of three days, the trading network accumulates a total profit based on the sales results of each enterprise. We need to determine the value of the profitability of the different trading enterprises. Table 1.2 displays the data on the turnover of the enterprises and the trading network as a whole on a daily basis.

Table 1.2 – Data on the total turnover of retail chain funds

Number of working days	Purchase of differential trade enterprises (million USD)			Total network revenue (million USD)
	1	2	3	
1	3,0	–1,0	1,0	12,0
2	5,0	1,0	2,0	3,0
3	1,0	1,0	2,0	3,0

Solution:

Based on the data in Table 1.2, we compile the LAES:

$$\begin{cases} 3x_1 - x_2 + x_3 = 12; \\ 5x_1 + x_2 + 2x_3 = 3; \\ x_1 + x_2 + 2x_3 = 3. \end{cases}$$

We solve LAES using the Gaussian elimination, the matrix method, and the Cramer's rule. Let's solve the system using the matrix method. For this, we will calculate the inverse matrix $\boldsymbol{A}^{-1} = \frac{1}{\Delta \boldsymbol{A}} \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$, where A_{ij} – algebraic addition to the elements of the matrix of coefficients $\boldsymbol{A}$ (i, j=1, 2, 3).

Matrix of coefficients: $\boldsymbol{A}=\begin{pmatrix}3 & -1 & 1\\ 5 & 1 & 2\\ 1 & 1 & 2\end{pmatrix}$.

The matrix determinant value: $\Delta\boldsymbol{A}=\begin{vmatrix}3 & -1 & 1\\ 5 & 1 & 2\\ 1 & 1 & 2\end{vmatrix}=(3\cdot1\cdot2)-(3\cdot2\cdot1)-$

$-(-1\cdot5\cdot2)+(-1\cdot2\cdot1)+(1\cdot5\cdot1)-(1\cdot1\cdot1)=12\neq0$ – the matrix is nondegenerate.

Algebraic matrix complements $\boldsymbol{A}$:

$$A_i^j=(-1)^{i+j}\det(\boldsymbol{C}_i^j),$$

where $\boldsymbol{C}_i^j$ – the matrix that is obtained from the coefficient matrix $\boldsymbol{A}$ by deleting the row numbered i and the column numbered j.

Then:

– (i=1, j=1) $A_{11}=(-1)^{1+1}\begin{vmatrix}1 & 2\\ 1 & 2\end{vmatrix}=(2-2)=0$, ($i$=1, j=2)

$A_{12}=(-1)^{1+2}\begin{vmatrix}5 & 2\\ 1 & 2\end{vmatrix}=-(10-2)=-8$, ($i$=1, j=3) $A_{13}=(-1)^{1+3}\begin{vmatrix}5 & 1\\ 1 & 1\end{vmatrix}=5-1=4$;

– (i=2, j=1) $A_{21}=(-1)^{2+1}\begin{vmatrix}-1 & 1\\ 1 & 2\end{vmatrix}=-(-2-1)=3$, (i=2, j=2)

$A_{22}=(-1)^{2+2}\begin{vmatrix}3 & 1\\ 1 & 2\end{vmatrix}=6-1=5$, (i=2, j=3) $A_{23}=(-1)^{2+3}\begin{vmatrix}3 & -1\\ 1 & 1\end{vmatrix}=-(3+1)=-4$;

– (i=3, j=1) $A_{31}=(-1)^{3+1}\begin{vmatrix}-1 & 1\\ 1 & 2\end{vmatrix}=-2-1=-3$, ($i$=3, j=2)

$A_{32}=(-1)^{3+2}\begin{vmatrix}3 & 1\\ 5 & 2\end{vmatrix}=-(6-5)=-1$, ($i$=3, j=3) $A_{33}=(-1)^{3+3}\begin{vmatrix}3 & -1\\ 5 & 1\end{vmatrix}=3+5=8$.

The value of the inverse matrix:

$$\boldsymbol{A}^{-1}=\frac{1}{\Delta\boldsymbol{A}}\begin{pmatrix}A_{11} & A_{12} & A_{13}\\ A_{21} & A_{22} & A_{23}\\ A_{31} & A_{32} & A_{33}\end{pmatrix}=\frac{1}{12}\begin{pmatrix}0 & 3 & -3\\ -8 & 5 & -1\\ 4 & -4 & 8\end{pmatrix}.$$

The value of the matrix (vector) of unknown arguments $\boldsymbol{X}$:

$$\boldsymbol{X}=\boldsymbol{A}^{-1}\cdot\boldsymbol{B}=\frac{1}{12}\begin{pmatrix}0 & 3 & -3\\ -8 & 5 & -1\\ 4 & -4 & 8\end{pmatrix}\cdot\begin{pmatrix}12\\ 3\\ 3\end{pmatrix}=\frac{1}{12}\begin{pmatrix}0\cdot12+3\cdot3+(-3)\cdot3\\ -8\cdot12+5\cdot3+(-1)\cdot3\\ 4\cdot12+(-4)\cdot3+8\cdot3\end{pmatrix}=\begin{pmatrix}0\\ -7\\ 5\end{pmatrix}.$$

The actual values of the profitability of different trading enterprises were obtained, namely: $x_1=0$ million USD; $x_2=-7{,}0$ million USD; $x_3=5$ million USD.

That is, only one enterprise is profitable, namely the enterprise with a profitability of x_3 = 5,0 million USD.

Solving LAES by Cramer's rule.

The main determinant of the coefficient matrix $\boldsymbol{A}$: $\Delta = \det \boldsymbol{A} = \begin{vmatrix} 3 & -1 & 1 \\ 5 & 1 & 2 \\ 1 & 1 & 2 \end{vmatrix}$.

The main determinant of the matrix of coefficients $\boldsymbol{A}$ can be decomposed into the elements of the first row:

$$\Delta = \det \boldsymbol{A} = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} \cdot A_{11} + a_{12} \cdot A_{12} + a_{13} \cdot A_{13},$$

$$\Delta = \begin{vmatrix} 3 & -1 & 1 \\ 5 & 1 & 2 \\ 1 & 1 & 2 \end{vmatrix} = 3 \cdot (-1)^{1+1} \cdot \begin{vmatrix} 1 & 2 \\ 1 & 2 \end{vmatrix} + (-1) \cdot (-1)^{1+2} \begin{vmatrix} 5 & 2 \\ 1 & 2 \end{vmatrix} + 1 \cdot (-1)^{1+3} \begin{vmatrix} 5 & 1 \\ 1 & 1 \end{vmatrix} = 12 \neq 0.$$

Let's write down and calculate the auxiliary determinants:

$$\Delta_1 = \begin{vmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{32} & a_{33} \end{vmatrix}; \ \Delta_2 = \begin{vmatrix} a_{11} & b_1 & a_{13} \\ a_{21} & b_2 & a_{23} \\ a_{31} & b_3 & a_{33} \end{vmatrix}; \ \Delta_2 = \begin{vmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ a_{31} & a_{32} & b_3 \end{vmatrix};$$

$$\Delta x_1 = \begin{vmatrix} 12 & -1 & 1 \\ 3 & 1 & 2 \\ 3 & 1 & 2 \end{vmatrix} = 12 \cdot (-1)^{1+1} \cdot \begin{vmatrix} 1 & 2 \\ 1 & 2 \end{vmatrix} + (-1) \cdot (-1)^{1+2} \begin{vmatrix} 3 & 2 \\ 3 & 2 \end{vmatrix} + 1 \cdot (-1)^{1+3} \begin{vmatrix} 3 & 1 \\ 3 & 1 \end{vmatrix} = 0;$$

$$\Delta x_2 = \begin{vmatrix} 3 & 12 & 1 \\ 5 & 3 & 2 \\ 1 & 3 & 2 \end{vmatrix} = 3 \cdot (-1)^{1+1} \cdot \begin{vmatrix} 3 & 2 \\ 3 & 2 \end{vmatrix} + 12 \cdot (-1)^{1+2} \begin{vmatrix} 5 & 2 \\ 1 & 2 \end{vmatrix} + 1 \cdot (-1)^{1+3} \begin{vmatrix} 5 & 3 \\ 1 & 3 \end{vmatrix} = -84;$$

$$\Delta x_3 = \begin{vmatrix} 3 & -1 & 12 \\ 5 & 1 & 3 \\ 1 & 1 & 3 \end{vmatrix} = 3 \cdot (-1)^{1+1} \cdot \begin{vmatrix} 3 & 2 \\ 3 & 2 \end{vmatrix} + (-1) \cdot (-1)^{1+2} \begin{vmatrix} 5 & 3 \\ 1 & 3 \end{vmatrix} + 12 \cdot (-1)^{1+3} \begin{vmatrix} 5 & 1 \\ 1 & 1 \end{vmatrix} = 60.$$

Then: $x_1 = \dfrac{\Delta x_1}{\Delta} = \dfrac{0}{12} = 0$, $x_2 = \dfrac{\Delta x_2}{\Delta} = -\dfrac{84}{12} = -7$, $x_3 = \dfrac{\Delta x_3}{\Delta} = \dfrac{60}{12} = 5$.

So, when solving the same LAES by all the above methods, the same answer was obtained.

Consider the implementation of Cramer's rule in the C++ programming language:

```
void Print_Arr(double** arr, double* barr, const int& rows){
    for(int i = 0; i < rows; ++i){
        for(int j = 0; j < rows;  ++j){
            cout << arr[i][j] << " ";
        }
```

```
        cout << barr[i] << endl;
    }
}
double Det(double** a, const int& rows, const int& columns){
   // the determinant search function
   // square matrix of size n*n
    double** B = new double* [rows];
    for(int i = 0; i < rows; ++i){
        B[i] = new double[columns];
        for(int j = 0; j < columns; ++j){
            B[i][j] = a[i][j];
        }
    }
    int n = rows;
    // bringing the matrix to the upper triangular form
    for(int step = 0; step < n - 1; step++)
        for(int row = step + 1; row < n; row++){
            double coeff = -1 * (B[row][step]) / B[step][step]; // Gauss
     method
            for(int col = step; col < n; col++)
                B[row][col] += B[step][col] * coeff;
        }
    // Calculation of the determinant as a product of the elements of the
     main diagonal
    double det = 1.;
    for(int i = 0; i < n; i++)
        det *= B[i][i];
    return det;
}
void MethodKramer(double** arr, double* barr, const int& n){ // function of
     Cramer's rule
    double det = Det(arr, n, n);
    int j = 0;
    while(j < n){
        for(int i = 0; i < n; ++i){
            swap(arr[i][j], barr[i]);
        }
        double detj = Det(arr, n, n);
        cout << "x" << j + 1 << " = " << detj/det << endl;
        for(int i = 0; i < n; ++i){
            swap(arr[i][j], barr[i]);
        }
        ++j; }}.
```

The Cramer's rule algorithm is shown in Figure 1.6.

Tridiagonal matrix algorithm

The Tridiagonal matrix algorithm is used to decouple LAES from a tridiagonal matrix. This system of equations is written in the form:

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i \quad (i = 1, 2, ..., n) \tag{1.11}$$

where a_1=0, c_n=0.

The tridiagonal matrix algorithm, which can be shortened to the Gaussian elimination, consists of a direct and reverse calculation method. The direct calculation path lies with the derived elements of the system matrix (1.11), which lie below the main diagonal. Each step will involve no more than two unknowns, and the formula for the reverse calculation can be written in this form:

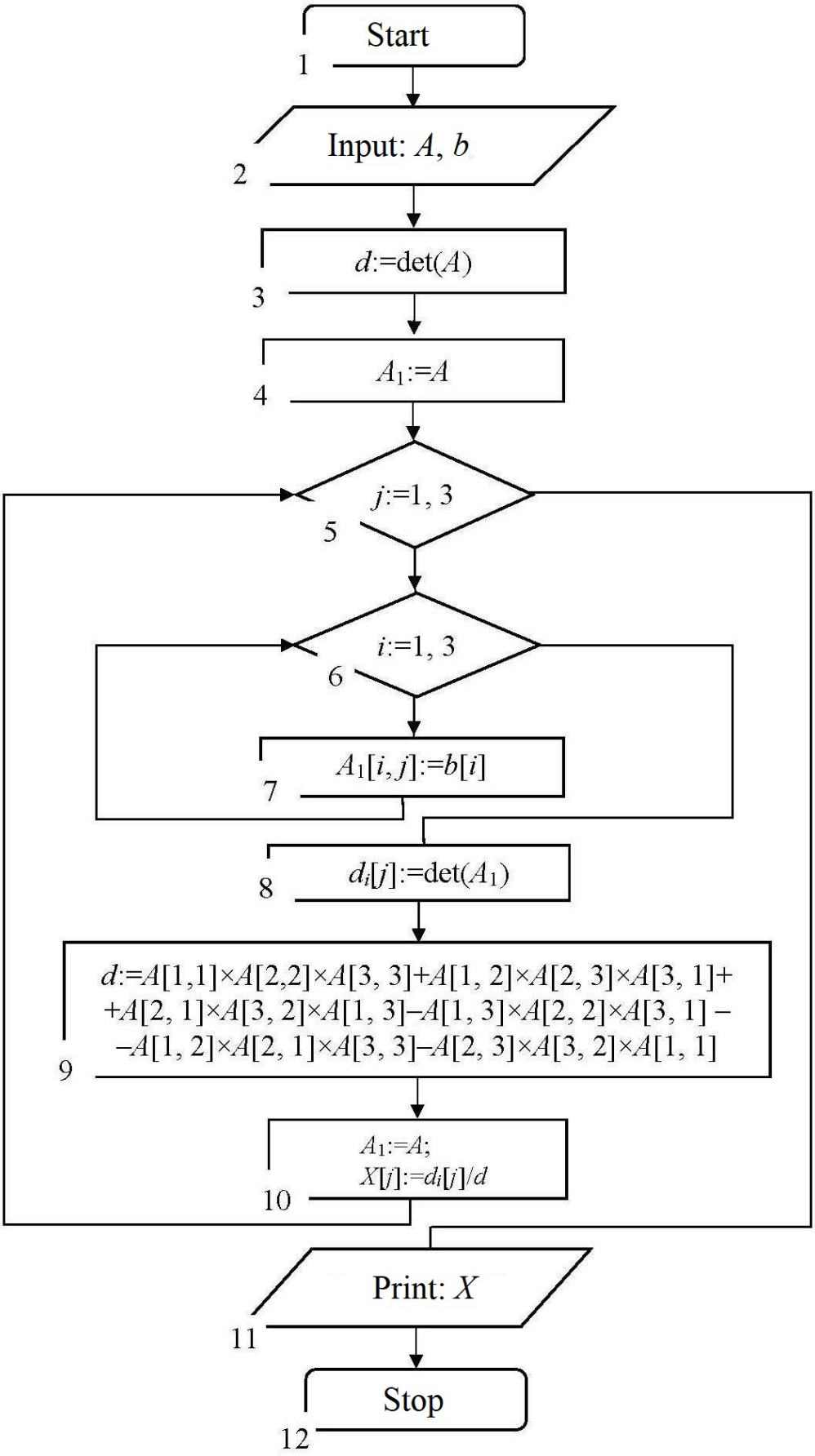


Figure 1.6 – Scheme of the Cramer's rule algorithm

$$x_i = U_i x_{i+1} + V_i \quad (i = n, n-1, \ldots, 1). \tag{1.12}$$

If we express $x_{i-1} = U_{i-1} x_i + V_{i-1}$ and substitute into the system (1.11), we will have $a_i(U_{i-1} x_i + V_{i-1}) + b_i x_i + c_i x_{i+1} = d_i$, whence:

$$x_i = -\frac{c_i}{a_i U_{i-1} + b_i} x_{i+1} + \frac{d_i - a_i V_{i-1}}{a_i U_{i-1} + b_i}. \tag{1.13}$$

Equating (1.12) and (1.13), we get:

$$U_i = -\frac{c_i}{a_i U_{i-1} + b_i}, \quad V_i = \frac{d_i - a_i V_{i-1}}{a_i U_{i-1} + b_i} \quad (i = 1, 2, ..., n). \tag{1.14}$$

As $a_1 = 0$, then:

$$U_1 = -c_1 / b_1, \quad V_1 = d_1 / b_1. \tag{1.15}$$

Now, according to formulas (1.14) and (1.15), the running coefficients U_i and V_i (i=1, 2, …, n) can be calculated during the course of running. Knowing the sweep coefficients, you can calculate all x_i (reverse stroke of sweep) using formula (1.7).

The Tridiagonal matrix algorithm can be easily algorithmized, making it frequently used in the standard mathematical support of computer systems. The implementation of the sweep method allows for a significant reduction in the number of arithmetic operations compared to the Gaussian elimination $N \approx 3n$.

Example 1.4. Solve LAES using the Tridiagonal matrix algorithm:

$$\begin{cases} 10x_1 + x_2 = 5; \\ -2x_1 + 9x_2 + x_3 = -1; \\ 0{,}1x_2 + 4x_3 - x_4 = -5; \\ -x_3 + 8x_4 = 40. \end{cases}$$

Solution:

Coefficients for unknown variables are recorded in the form of Table 1.3.

Table 1.3 – Values of LAES coefficients

i	a_i	b_i	c_i	d_i
1	0,0	10,0	1,0	5,0
2	– 2,0	9,0	1,0	– 1,0
3	0,1	4,0	– 1,0	– 5,0
4	– 1,0	8,0	0,0	40,0

Sweep forward run. Formulas (1.14) and (1.15) determine the driving coefficients U_i and V_i:

$U_1 = -c_1 / b_1 = -1/10 = -0,1;$

$V_1 = d_1 / b_1 = 5/10 = 0,5;$

$U_2 = -c_2 / (a_2 U_1 + b_2) = -1/(2 \cdot 0,1 + 9) = -0,1087;$

$V_2 = (d_2 - a_2 V_1) / (a_2 U_1 + b_2) = (-1 + 2 \cdot 0,5) / (2 \cdot 0,1 + 9) = 0;$

$U_3 = -c_3 / (a_3 U_2 + b_3) = 1/(-0,1 \cdot 0,1087 + 4) = 0,2507;$

$V_3 = (d_3 - a_3 V_2) / (a_3 U_2 + b_3) = (-5 - 0,1 \cdot 0) / (-0,1 \cdot 0,1087 + 4) = -1,2534;$

$U_4 = -c_4 / (a_4 U_3 + b_4) = 0$, because c_4=0;

$V_4 = (d_4 - a_4 V_3) / (a_4 U_3 + b_4) = (40 - 1 \cdot 1,2534) / (-1 \cdot 0,2507 + 8) = 5,0.$

The reverse course of the sweep. Formulas (1.13) calculate all unknown values x_i:

$x_4 = V_4 = 5,0$ ($U_4 = 0$);

$x_3 = U_3 x_4 + V_3 = 0,2507 \cdot 5 - 1,2534 = 0,0001 \approx 0;$

$x_2 = U_2 x_3 + V_2 = -1,1087 \cdot 0,0001 + 0 = -0,0001 \approx 0;$

$x_1 = U_1 x_2 + V_1 = 0,1 \cdot 0,0001 + 0,5 = 0,5001 \approx 0,5.$

We will present the implementation of the Tridiagonal matrix algorithm in the C++ programming language:

```
// Straight course of the Tridiagonal matrix algorithm
int N1 = N - 1;
y = matA[0][0];
   a[0] = -matA[0][1] / y;
   B[0] = matB[0] / y ;
for (int i = 1; i < N1; i++) {
   y = matA[i][i] + matA[i][i - 1] * a[i - 1];
   a[i] = -matA[i][i + 1] / y;
   B[i] = (matB[i] - matA[i][i - 1] * B[i - 1]) / y;
   }
// On the return stroke, the roots of the equation system are calculated:
matRes[N1] = (matB[N1] - matA[N1][N1 - 1] * B[N1 - 1]) / (matA[N1][N1] +
matA[N1][N1 - 1] * a[N1 - 1]);
for (int i = N1 - 1; i >= 0; i--) {
matRes[i] = a[i] * matRes[i + 1] + B[i];
}
```

The algorithm of the Tridiagonal matrix algorithm is given in Figure 1.7.

1.3 Iterative method

In the case when the LAES has a large number of unknowns and also contains a matrix of highly sparse coefficients (a large number of coefficients with zero values), the use of the Gaussian elimination, which gives an exact solution, becomes very difficult. In this case, it is convenient to use Iterative method to determine the roots of the system. For this purpose, the LAES is reduced to this form:

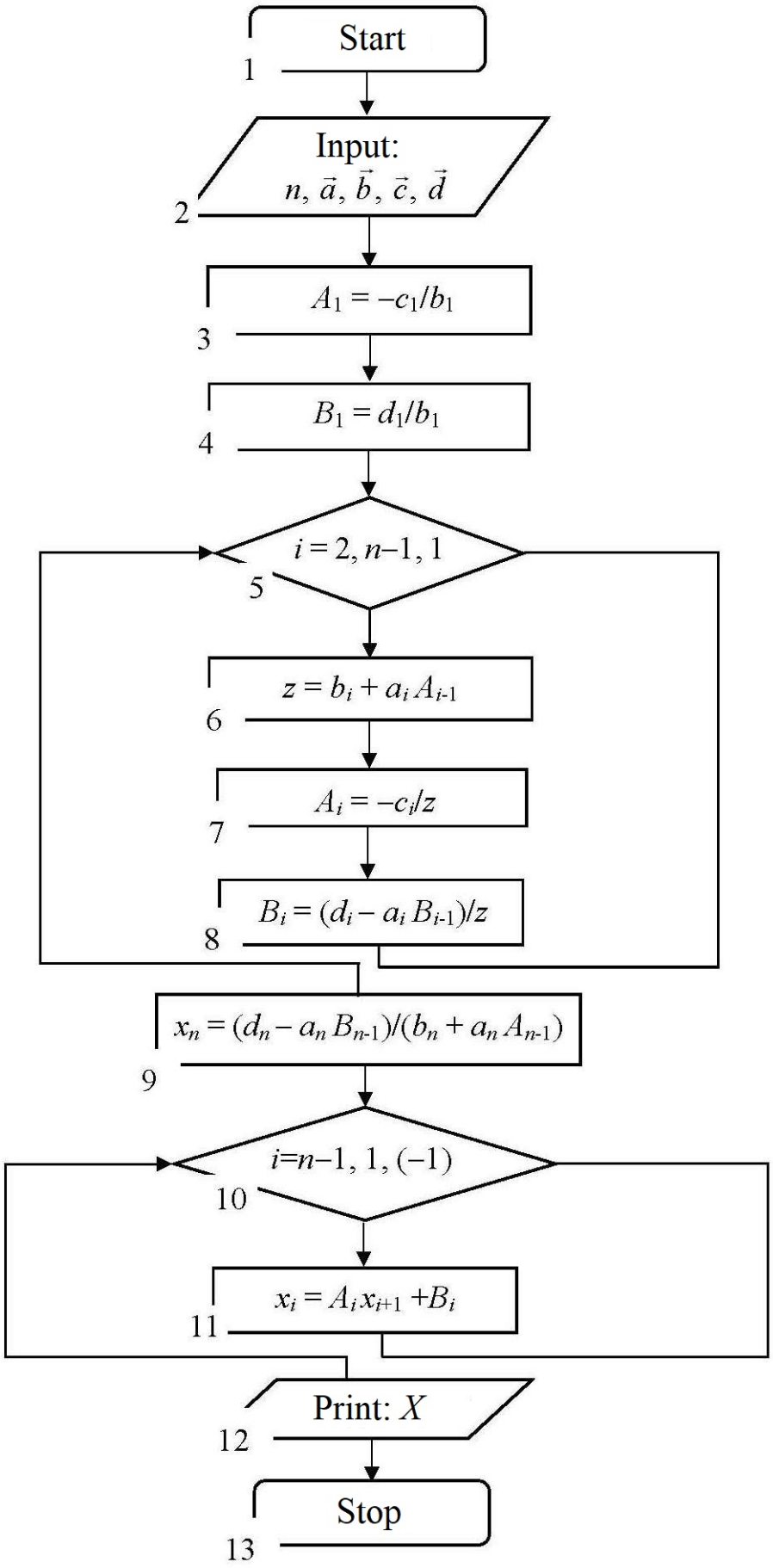


Figure 1.7 – Scheme of the algorithm for the tridiagonal matrix algorithm

$$\begin{cases} x_1 = b_{1,n}x_n + b_{1,n-1}x_{n-1} + ... + b_{1,1}x_1 + b_{1,0}; \\ x_2 = b_{2,n}x_n + b_{2,n-1}x_{n-1} + ... + b_{2,1}x_1 + b_{2,0}; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\ x_n = b_{n,n}x_n + b_{n,n-1}x_{n-1} + ... + b_{n,1}x_1 + b_{n,0}. \end{cases} \quad (1.16)$$

or in matrix form:

$$\boldsymbol{X} = \boldsymbol{B}\boldsymbol{X} + \boldsymbol{B}_0,$$

where $\boldsymbol{B} = \begin{vmatrix} b_{11} & b_{12} & \ldots & b_{1n} \\ b_{21} & b_{22} & \ldots & b_{2n} \\ \vdots & \vdots & \ldots & \vdots \\ b_{n1} & b_{n2} & \ldots & b_{nn} \end{vmatrix}$, $\boldsymbol{B}_0 = \begin{vmatrix} b_{10} \\ b_{20} \\ \vdots \\ b_{n0} \end{vmatrix}$.

There are several main types of iterative methods, which include Jacobi method (simple iteration), Gauss–Seidel method, and successive over-relaxation. These methods are based on the systematic refinement of the values of the variables specified at the beginning of the calculation.

Jacobi method (simple iteration)

In the Jacobi (simple iteration) method, the initial values of the variables are used to calculate the new values $x_1, x_2, \ldots, x_{i-1}$ using the following equations. The process stops when all the new values are close enough to the original values. Otherwise, the new values are used instead of the original values. This procedure is repeated until the convergence condition is satisfied or the process diverges. In this method, the replacement of the values of all variables is performed simultaneously (simultaneous displacement).

Let LAES be given in the following form:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1; \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2; \\ .....; \\ a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n. \end{cases} \quad (1.17)$$

To solve such a system using the Jacobi method (simple iteration), it is necessary to reduce the system (1.17) to the following form:

$$\begin{cases} x_1 = -\dfrac{a_{12}}{a_{11}}x_2 - \dfrac{a_{13}}{a_{11}}x_3 - \ldots - \dfrac{a_{1n}}{a_{11}}x_n + \dfrac{b_1}{a_{11}}; \\ x_2 = -\dfrac{a_{21}}{a_{22}}x_1 - \dfrac{a_{23}}{a_{22}}x_3 - \ldots - \dfrac{a_{2n}}{a_{22}}x_n + \dfrac{b_2}{a_{22}}; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\ x_n = -\dfrac{a_{n1}}{a_{nn}}x_1 - \dfrac{a_{n2}}{a_{nn}}x_2 - \ldots - \dfrac{a_{nn-1}}{a_{nn}}x_{n-1} + \dfrac{b_n}{a_{nn}}. \end{cases} \quad (1.18)$$

In the system (1.17), the *i*-th equation is the *i*-th equation of the system (1.18), solved with respect to the *i*-th unknown ($i = \overline{1,n}$).

The method of solving LAES (1.17) by means of reduction to system (1.18), followed by its solution using an iterative method, is called the Jacobi (simple iteration) method for system (1.17).

Thus, the formulas for the Jacobi method (simple iteration) for solving system (1.17) will have the following form:

$$\begin{cases} x_1^{k+1} = -\dfrac{a_{12}}{a_{11}}x_2^k - \dfrac{a_{13}}{a_{11}}x_3^k - \ldots - \dfrac{a_{1n}}{a_{11}}x_n^k + \dfrac{b_1}{a_{11}} = -\sum\limits_{j=2}^{n}\dfrac{a_{1j}}{a_{11}}x_j^k + \dfrac{b_1}{a_{11}}; \\ x_2^{k+1} = -\dfrac{a_{21}}{a_{22}}x_1^k - \dfrac{a_{23}}{a_{22}}x_3^k - \ldots - \dfrac{a_{2n}}{a_{22}}x_n^k + \dfrac{b_2}{a_{22}} = -\sum\limits_{j=1,j\neq 2}^{n}\dfrac{a_{2j}}{a_{22}}x_j^k + \dfrac{b_2}{a_{22}}; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\ x_n^{k+1} = -\dfrac{a_{n1}}{a_{nn}}x_1^k - \dfrac{a_{n2}}{a_{nn}}x_2^k - \ldots - \dfrac{a_{nn-1}}{a_{nn}}x_{n-1}^k + \dfrac{b_n}{a_{nn}} = -\sum\limits_{j=1}^{n-1}\dfrac{a_{nj}}{a_{nn}}x_j^k + \dfrac{b_n}{a_{nn}} \ (k = 0, 1, 2, \ldots). \end{cases} \quad (1.19)$$

Formulas (1.19) can be written in the following form:

$$x_i^{k+1} = -\sum_{j=1,j\neq i}^{n}\frac{a_{ij}}{a_{ii}}x_j^k + \frac{b_i}{a_{ii}} \ (i=1, 2, \ldots, n;\ k=0, 1, 2, \ldots).$$

Example 1.5. Solve the LAES (system with dominant diagonal coefficients) using the numerical Jacobi method (simple iteration) with the specified accuracy $\Delta = 0{,}065$:

$$\begin{cases} 4{,}300x_1 + 0{,}217x_2 = 2{,}663; \\ 0{,}100x_1 - 3{,}400x_2 - 0{,}207x_3 = 2{,}778; \\ 0{,}090x_2 + 2{,}500x_3 + 0{,}197x_4 = 2{,}533; \\ 0{,}080x_3 - 1{,}600x_4 = 1{,}928. \end{cases}$$

Solution:

Let's write down the equivalent system of equations:

$$\begin{cases} x_1 = \dfrac{2,663}{4,300} - \dfrac{0,217}{4,300} x_2 = 0,6193 - 0,050 x_2; \\ x_2 = -\dfrac{2,778}{3,4} - \dfrac{0,207}{3,4} x_4 + \dfrac{0,1}{3,4} x_1 = -0,817 - 0,061 x_4 + 0,029 x_1; \\ x_3 = \dfrac{2,533}{2,500} - \dfrac{0,197}{2,500} x_4 - \dfrac{0,09}{2,50} x_2 = 1,013 - 0,079 x_4 - 0,036 x_2; \\ x_4 = -\dfrac{1,928}{1,600} + \dfrac{0,08}{1,60} x_3 = -1,205 + 0,05 x_3. \end{cases}$$

Since the sum of the coefficients in each row on the right-hand side of the system is obviously less than one, the «weak» convergence condition will be satisfied. The following values can be considered as an initial approximation of zero: $x_1^{(0)} = 0,619$; $x_2^{(0)} = -0,8170$; $x_3^{(0)} = 1,013$; $x_4^{(0)} = -1,205$.

Based on the system, the calculation of the first iteration is performed:

$$\begin{cases} x_1^{(1)} = 0,6193 - 0,050 x_2^{(0)} = 0,6193 - 0,050 \cdot (-0,817) = 0,6604; \\ x_2^{(1)} = -0,817 - 0,061 x_4^{(0)} + 0,029 x_1^{(0)} = -0,817 - 0,061 \cdot (-1,205) + \\ +0,029 \cdot 0,619 = -0,8606; \\ x_3^{(1)} = 1,013 - 0,079 x_4^{(0)} - 0,036 x_2^{(0)} = 1,013 - 0,079 \cdot (-1,205) - \\ -0,036 \cdot (0,817) = 1,1373; \\ x_4^{(1)} = -1,205 + 0,05 x_3^{(0)} = -1,205 + 0,05 \cdot 1,013 = -1,155. \end{cases}$$

After that, the calculation for the second iteration is performed:

$$\begin{cases} x_1^{(2)} = 0,6193 - 0,050 x_2^{(1)} = 0,6193 - 0,050 \cdot (-0,8606) = 0,6623; \\ x_2^{(2)} = -0,817 - 0,061 x_4^{(1)} + 0,029 x_1^{(1)} = -0,817 - 0,061 \cdot (-1,155) + \\ +0,029 \cdot 0,6604 = -0,8678; \\ x_3^{(2)} = 1,013 - 0,079 x_4^{(1)} - 0,036 x_2^{(1)} = 1,013 - 0,079 \cdot (-1,155) - \\ -0,036 \cdot (-0,8606) = 1,072; \\ x_4^{(2)} = -1,205 + 0,05 x_3^{(1)} = -1,205 + 0,05 \cdot 1,1373 = -1,148. \end{cases}$$

The calculation of the third iteration is performed:

$$\begin{cases} x_1^{(3)} = 0,6193 - 0,050x_2^{(2)} = 0,6193 - 0,050 \cdot (-0,8678) = 0,6623; \\ x_2^{(3)} = -0,817 - 0,061x_4^{(2)} + 0,029x_1^{(2)} = -0,817 - 0,061 \cdot (-1,148) + \\ +0,029 \cdot 0,6623 = -0,862; \\ x_3^{(3)} = 1,013 - 0,079x_4^{(2)} - 0,036x_2^{(2)} = 1,013 - 0,079 \cdot (-1,148) - \\ -0,036 \cdot (-0,8678) = 1,133; \\ x_4^{(3)} = -1,205 + 0,05x_3^{(2)} = -1,205 + 0,05 \cdot 1,072 = -1,1218. \end{cases}$$

We will evaluate the calculation error:

$$\begin{cases} \left|x_1^{(2)} - x_1^{(1)}\right| = \left|0,6623 - 0,6604\right| = 0,0019; \\ \left|x_2^{(2)} - x_2^{(1)}\right| = \left|-0,8678 - (-0,8606)\right| = 0,0072; \\ \left|x_3^{(2)} - x_3^{(1)}\right| = \left|1,072 - 1,1373\right| = 0,0653; \\ \left|x_4^{(2)} - x_4^{(1)}\right| = \left|(-1,148) - (-1,155)\right| = 0,0070; \end{cases}$$

$$\Delta_1 = \max\left\{0,0019;\ 0,0072;\ \ 0,0653;\ \ 0,0070\right\} = 0,0653 > \Delta = 0,065;$$

$$\begin{cases} \Delta_1^{(2)} = \left|x_1^{(3)} - x_1^{(2)}\right| = \left|0,6623 - 0,6623\right| = 0,0000; \\ \Delta_2^{(2)} = \left|x_2^{(3)} - x_2^{(2)}\right| = \left|(-0,8620) - (-0,8678)\right| = 0,0058; \\ \Delta_3^{(2)} = \left|x_3^{(3)} - x_3^{(2)}\right| = \left|1,133 - 1,072\right| = 0,0610; \\ \Delta_4^{(2)} = \left|x_4^{(3)} - x_4^{(2)}\right| = \left|(-1,1218) - (-1,1480)\right| = 0,0262; \end{cases}$$

$$\Delta_2 = \max\left\{0,0000;\ 0,0058;\ \ 0,0610;\ \ 0,0262\right\} = 0,0610 < \Delta = 0,065.$$

The fulfillment of the condition $\Delta_2 < 0,065$ indicates the achievement of the specified accuracy of the calculation. Therefore, the solution of the system with an error of $\Delta = 0,065$:

$$\begin{cases} x_1^{(3)} = 0,6623; & x_2^{(3)} = -0,8620; \\ x_3^{(3)} = 1,1330; & x_4^{(3)} = -1,1218. \end{cases}$$

In the method of successive upper relaxation, the new values of each variable are calculated as follows:

$$x_i^{(m+1)} = x_i^{(m)} + \omega(\bar{x}_i^{(m+1)} - x_i^{(m)}),$$

where $\bar{x}_i^{(m+1)}$ – refined value $x_i^{(m)}$ using the Gauss-Seidel method, ω – relaxation parameter $(1,0 \le \omega \le 2,0)$.

When the value of the parameter $\omega = 1.0$, this method is identical to the Gauss-Seidel method. The rate of convergence depends on the value of the relaxation parameter ω. The algorithm for the numerical implementation of the

Jacobi method (simple iteration) for solving the LAES is presented in Figure 1.8.

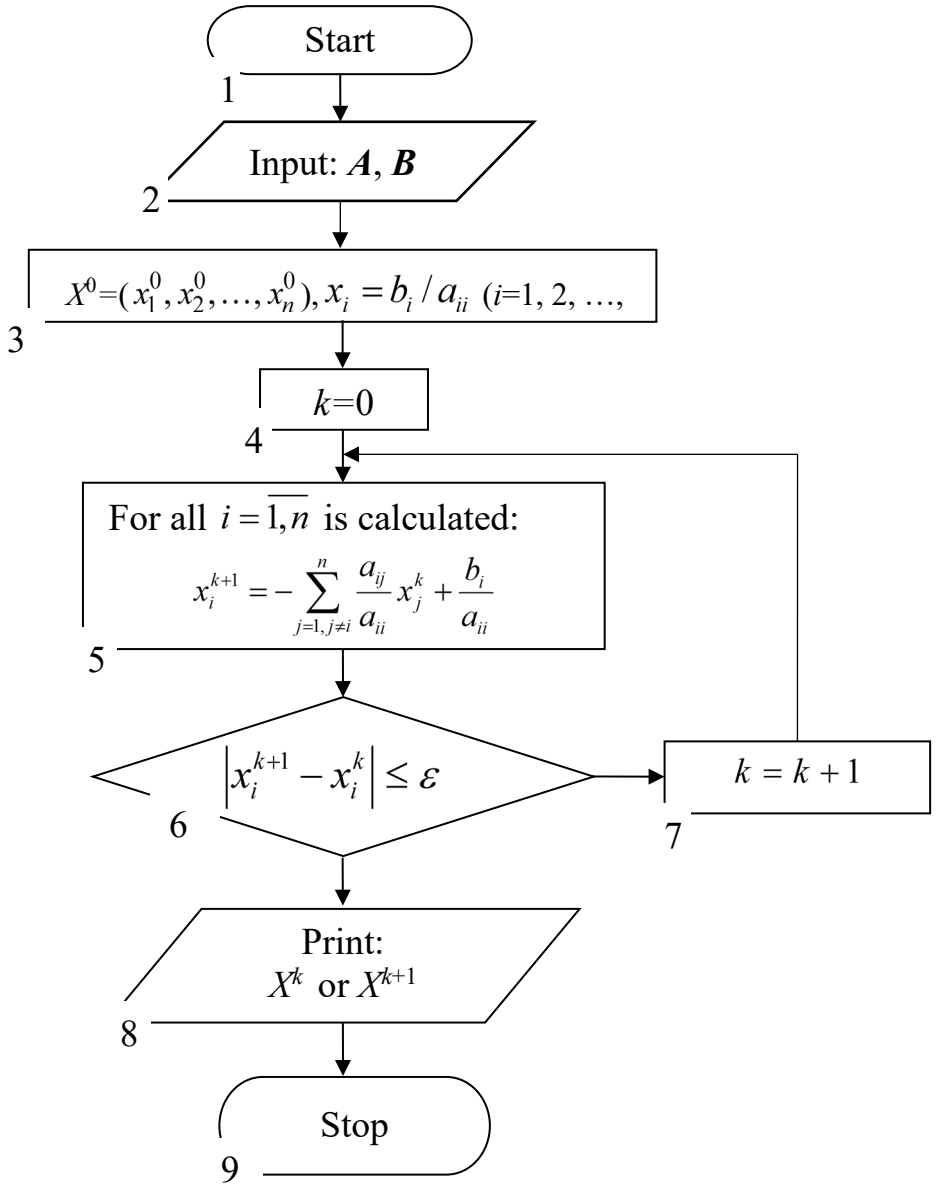


Figure 1.8 – Scheme of the algorithm of the Jacobi method (simple iteration) for LAES

Consider the implementation of the simple iteration method in the C++ programming language:

```
float G1 (float x1, float x2, float x3, float x4)
  { return -0.6 + 0.7*x1 - 0.5*x2 - 0.3*x3- 0.5*x4;}
float G2 (float x1, float x2, float x3, float x4)
  { return -0.3 - 0.3*x1 + 0.6*x2 -0.1*x3 -0.2*x4;}
float G3 (float x1, float x2, float x3, float x4)
  { return  -0.8 - 0.6*x1 -0.2*x2 +0.2*x3 - 0.2*x4;}
float G4 (float x1, float x2, float x3, float x4)
  { return  -0.8 - 0.3*x1 -0.5*x2 -0.3*x3 + 0.3*x4;}
int main(int argc, char *argv[]) {
   float delta = 0.01;
   float x1 = 1, x2 = 1, x3 = 1, x4 = 1;
```

```
   float x10, x20, x30, x40;
   do
    {
        x10 = x1;
        x20 = x2;
        x30 = x3;
        x40 = x4;
        x1 = G1(x10, x20, x30, x40);
        x2 = G2(x10, x20, x30, x40);
        x3 = G3(x10, x20, x30, x40);
        x4 = G4(x10, x20, x30, x40);
    }
 while ((fabs(x1-x10) >= delta) && (fabs(x2-x20)>= delta) && (fabs(x3-x30)
     >= delta)&&(fabs(x4-x40) >= delta));
   //
   cout << "x1 = " << x1 << endl;
   cout << " x2 = " << x2 << endl;
   cout << " x3 = " << x3 << endl;
   cout << " x4 = " << x4 << endl << endl;
   system("PAUSE");
       return 0;}
```

Gauss-Seidel method

In the case of using the Gauss-Seidel method, when calculating the (k+1)-th approximation of the unknown value x_i ($i > 1$), the previously calculated (k+1)-th approximations of the unknowns $x_1, x_2, \ldots, x_{i-1}$ are used.

Let's consider this method using the example of solving LAES:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1; \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2; \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3. \end{cases}$$

Assume that the diagonal elements a_{11}, a_{22}, and a_{33} are different from zero. Then, expressing the unknown values x_1, x_2 and x_3, respectively, from the first, second and third equations of the original system, we get:

$$\begin{cases} x_1 = \dfrac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3); \\ x_2 = \dfrac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3); \\ x_3 = \dfrac{1}{a_{33}}(b_3 - a_{31}x_1 - a_{32}x_2). \end{cases} \qquad (1.20)$$

If we specify some initial (zero) approximations of the values of the unknowns $x_1 = x_1^{(0)}$, $x_2 = x_2^{(0)}$, and x_3, and substitute them into the right-hand side of the system equations (1.20), we will obtain a new approximation (first iteration) for x_1, x_2, and x_3, respectively:

$$\begin{cases} x_1^{(1)} = \dfrac{1}{a_{11}}(b_1 - a_{12}x_2^{(0)} - a_{13}x_3^{(0)}); \\ x_2^{(1)} = \dfrac{1}{a_{22}}(b_2 - a_{21}x_1^{(1)} - a_{23}x_3^{(0)}); \\ x_3^{(1)} = \dfrac{1}{a_{33}}(b_3 - a_{31}x_1^{(1)} - a_{32}x_2^{(1)}). \end{cases}$$

For the second iteration, the new values for x_1, x_2, and x_3 will be used, namely: $x_1 = x_1^{(2)}, x_2 = x_2^{(2)}, x_3 = x_3^{(2)}$ etc.

Then the *k*-th approximation can be given in the form:

$$\begin{cases} x_1^{(k)} = \dfrac{1}{a_{11}}(b_1 - a_{12}x_2^{(k-1)} - a_{13}x_3^{(k-1)}); \\ x_2^{(k)} = \dfrac{1}{a_{22}}(b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k-1)}); \\ x_3^{(k)} = \dfrac{1}{a_{33}}(b_3 - a_{31}x_1^{(k)} - a_{32}x_2^{(k)}). \end{cases}$$

The iterative process continues until the $x_1^{(k)}$, $x_2^{(k)}$, $x_3^{(k)}$ values become close to the $x_1^{(k-1)}$, $x_2^{(k-1)}$, $x_3^{(k-1)}$ values within a given error.

Now consider a system of n linear equations with *n* unknowns ($i = 1, 2, ..., n$). Assume that all diagonal elements are nonzero. Let's write the *i*-th equation:

$$a_{i1}x_1 + ... + a_{i,i-1}x_{i-1} + a_{ii}x_i + a_{i,i+1}x_{i+1} + ... + a_{in}x_n = b_i .$$

Then, according to the Gauss-Seidel method, the *k*-th approximation of the solution can be written in the form:

$$x_i^{(k)} = \frac{1}{a_{ii}}(b_i - a_{i1}x_1^{(k)} - ... - a_{i,i-1}x_{i-1}^{(k)} - a_{i,i+1}x_{i+1}^{(k-1)} - ... - a_{in}x_n^{(k-1)}) .$$

The iterative process will continue until all values of $x_i^{(k)}$ become close to $x_i^{(k-1)}$. The proximity of these values is characterized by the maximum absolute value of their difference δ. Then, with a given permissible error $\Delta>0$, the condition for the termination of the iterative process can be written in the form:

$$\delta = \max_{1 \le i \le n} \left| x_i^{(k)} - x_i^{(k-1)} \right| < \Delta .$$

This condition is a criterion based on absolute deviation, which can be replaced by a criterion based on relative differences. That is, the termination condition of the iterative process can be written in the form (for $|x_i| >> 1$):

$$\max_{1 \le i \le n} \left| \frac{x_i^{(k)} - x_i^{(k-1)}}{x_i^k} \right| < \varepsilon .$$

If one of these conditions is fulfilled, the iterative process of solving by the Gauss-Seidel method is called convergent. In this case, the maximum differences δ between the values of the variables in the next two iterations decrease, and these values themselves will approach the solution of the system of equations.

One of the main conditions for the application of iterative methods is the fulfillment of the convergence condition. The analysis of the convergence of iterative methods for solving the LAES is connected with the use of the concept of matrix norm. The norm of a matrix $\boldsymbol{B}$ is a characteristic number $\|\boldsymbol{B}\|$, that has the following properties: the norm is always greater than zero ($\|\boldsymbol{B}\|>0$) and is equal to zero only for the zero matrix; if the matrix $\boldsymbol{B}$ is multiplied by a real coefficient λ, then the norm must be multiplied by the modulus of this coefficient – $|\lambda|\cdot\|\boldsymbol{B}\|$; the norm of the sum of two matrices $\boldsymbol{A}$, $\boldsymbol{B}$ is not greater than the sum of the norms (the condition of additivity of the norm – $\|\boldsymbol{A}+\boldsymbol{B}\|\le\|\boldsymbol{A}\|+\|\boldsymbol{B}\|$), and the norm of the product of two matrices $\boldsymbol{A}$, $\boldsymbol{B}$ is not greater than the product of the norms (the condition of multiplicativity of the norm – $\|\boldsymbol{AB}\|\le\|\boldsymbol{A}\|\cdot\|\boldsymbol{B}\|$).

The following matrix norms have become the most widespread: $\|\boldsymbol{B}\|_1=\max\limits_{1\le j\le n}\sum\limits_{i=1}^{n}|b_{ij}|$ – the first norm; $\|\boldsymbol{B}\|_2=\max\limits_{1\le j\le n}\sum\limits_{j=1}^{n}|b_{ij}|$ – the second norm; $\|\boldsymbol{B}\|_E=\sqrt{\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n}b_{ij}^2}$ – E-norm (Euclidean norm).

There are several approaches to determining convergence using norm estimation. In the general case, it is enough that at least one of the norms of the matrix $\boldsymbol{B}$ is less than one, namely: $||\boldsymbol{B}||<1$.

In mathematics, such a condition is called «ordinary» or «strong». In many cases, convergence is ensured by the fulfillment of the so-called «weak» feature. For example, a «weak» sign of convergence based on row sums is as follows: for all row coefficient sums (i = 1, 2, ..., n), the relation – $\sum\limits_{j=i}^{n}|b_{ij}|\le 1$ holds, but there is one row p for which $\sum\limits_{j=i}^{n}|b_{pj}|<1$.

Similarly, the «weak» sign of the sums for the columns of the matrix is determined.

The «weak» sign is used in those cases when the coefficient matrix $\boldsymbol{A}$ of the LAES is not decomposable. In other words, it is a square matrix $\boldsymbol{A}$ that cannot be reduced to the form of a decomposable matrix:

$$\begin{pmatrix}\boldsymbol{A}_1 & \boldsymbol{A}_2\\ 0 & \boldsymbol{A}_3\end{pmatrix},$$

where $\boldsymbol{A}_1$, $\boldsymbol{A}_2$, $\boldsymbol{A}_3$ – square matrices.

For decomposable matrices, LAES decomposes into two systems of equations that are solved sequentially. The primary sources listed at the end of this textbook

contain many more detailed proofs and analyses of properties and features for evaluating convergence. However, for a general engineering approach to solving many practical problems, the above data is sufficient.

In practice, for the convergence of the iterative process, it is sufficient that the magnitudes of the diagonal coefficients of each equation in the system are not less than the sum of the magnitudes of all other coefficients:

$$\left|a_{ij}\right| \geq \sum_{i \neq j} \left|a_{ij}\right|,$$

This condition is sufficient for the convergence of the method, but it is not necessary because for some systems, the iterations converge even if these conditions are violated.

Example 1.6. A manufacturing enterprise produces four types of products using four types of technological equipment. Each type of equipment has its own coefficient of useful action (CUE), namely: x_1, x_2, x_3, and x_4. Table 1.4 presents the energy consumption required for each type of equipment to produce the different products. To calculate the CUE of the technological equipment, refer to Table 1.4.

Table 1.4 – Energy consumption data for manufacturing products

Product type	Energy consumption by technological equipment, MJ				Total energy consumption, MJ
	1	2	3	4	
I	0,401	0,301	0,000	0,000	0,122
II	0,029	0,500	0,018	0,000	0,253
III	0,000	0,050	1,400	0,039	0,988
IV	0,000	0,000	0,007	2,300	2,082

Solution:

Based on the data in Table 1.4, we will compile the characteristic equation for energy consumption by technological equipment in the manufacturing of various types of products.

$$\begin{cases} 0,401x_1 + 0,301x_2 = 0,122; \\ 0,029x_1 + 0,5x_2 + 0,018x_3 = 0,253; \\ 0,050x_2 + 1,4x_3 + 0,039x_4 = 0,988; \\ 0,007x_3 + 2,3x_4 = 2,082. \end{cases}$$

The obtained LAES must be solved using the Gauss-Seidel method with the given accuracy Δ=0,001.

Let's reduce the system to an equivalent form:

$$\begin{cases} x_1 = \dfrac{1}{0,401}(0,122 - 0,301x_2) = 0,304 - 0,751x_2; \\ x_2 = \dfrac{1}{0,5}(0,253 - 0,018x_3 - 0,029x_1) = 0,506 - 0,036x_3 - 0,058x_1; \\ x_3 = \dfrac{1}{1,4}(0,988 - 0,050x_2 - 0,039x_4) = 0,706 - 0,036x_2 - 0,028x_4; \\ x_4 = \dfrac{1}{2,3}(2,082 - 0,007x_3) = 0,905 - 0,003x_3. \end{cases}$$

Since the sum of the coefficients in the rows on the right side of the system is obviously less than unity, the «weak» convergence condition will be fulfilled.

All x_i values equal to zero in the right part of the system are taken as initial approximations, namely: $x_1^{(0)}=x_2^{(0)}=x_3^{(0)}=x_4^{(0)}=0$.

Then the values of the unknown arguments at the first iteration:

$$\begin{cases} x_1^{(1)} = 0,304 - 0,751x_2^{(0)} = 0,304 - 0,751 \cdot 0 = 0,304; \\ x_2^{(1)} = 0,506 - 0,036x_3^{(0)} - 0,058x_1^{(1)} = 0,506 - 0,036 \cdot 0 - 0,058 \cdot 0,304 = 0,488; \\ x_3^{(1)} = 0,706 - 0,036x_2^{(1)} - 0,028x_4^{(0)} = 0,706 - 0,036 \cdot 0,488 - 0,028 \cdot 0 = 0,688; \\ x_4^{(1)} = 0,905 - 0,003x_3^{(1)} = 0,905 - 0,003 \cdot 0,688 = 0,903. \end{cases}$$

Based on the results of the first iteration, the second calculation iteration is performed:

$$\begin{cases} x_1^{(2)} = 0,304 - 0,751x_2^{(1)} = 0,304 - 0,751 \cdot 0,488 = -0,062; \\ x_2^{(2)} = 0,506 - 0,036x_3^{(1)} - 0,058x_1^{(2)} = 0,506 - 0,036 \cdot 0,688 - \\ -0,058 \cdot (-0,062) = 0,485; \\ x_3^{(2)} = 0,706 - 0,036x_2^{(2)} - 0,028x_4^{(1)} = 0,706 - 0,036 \cdot 0,485 - 0,028 \cdot 0,903 = 0,663; \\ x_4^{(2)} = 0,905 - 0,003x_3^{(2)} = 0,905 - 0,003 \cdot 0,663 = 0,903. \end{cases}$$

Based on the results of two iterations, the accuracy of the current calculation is checked:

$$\begin{cases} \left|x_1^{(2)} - x_1^{(1)}\right| = \left|-0,062-0,304\right| = 0,366; \\ \left|x_2^{(2)} - x_2^{(1)}\right| = \left|0,488 - 0,485\right| = 0,003; \\ \left|x_3^{(2)} - x_3^{(1)}\right| = \left|0,663-0,688\right| = 0,025; \\ \left|x_4^{(2)} - x_4^{(1)}\right| = \left|0,903-0,903\right| = 0; \end{cases}$$

$$\Delta_1 = \max\{0,366;\ 0,003;\ 0,025;\ 0\} = 0,366 > \Delta = 0,001.$$

The value $\Delta_1 > 0{,}001$ means that the specified calculation accuracy has not yet been achieved, indicating the need to perform the third calculation iteration:

$$\begin{cases} x_1^{(3)} = 0{,}304 - 0{,}751 x_2^{(2)} = 0{,}304 - 0{,}751 \cdot 0{,}485 = -0{,}060; \\ x_2^{(3)} = 0{,}506 - 0{,}036 x_3^{(2)} - 0{,}058 x_1^{(3)} = 0{,}506 - 0{,}036 \cdot 0{,}663 - 0{,}058 \cdot (-0{,}060) = 0{,}486; \\ x_3^{(3)} = 0{,}706 - 0{,}036 x_2^{(3)} - 0{,}028 x_4^{(2)} = 0{,}706 - 0{,}036 \cdot 0{,}486 - 0{,}028 \cdot (0{,}903) = 0{,}663; \\ x_4^{(3)} = 0{,}905 - 0{,}003 x_3^{(3)} = 0{,}905 - 0{,}003 \cdot 0{,}663 = 0{,}903. \end{cases}$$

Checking the accuracy of the current calculation is performed:

$$\begin{cases} \left| x_1^{(3)} - x_1^{(2)} \right| = \left| -0{,}060 - 0{,}062 \right| = 0{,}002; \\ \left| x_2^{(3)} - x_2^{(2)} \right| = \left| 0{,}486 - 0{,}485 \right| = 0{,}001; \\ \left| x_3^{(3)} - x_3^{(2)} \right| = \left| 0{,}663 - 0{,}663 \right| = 0; \\ \left| x_4^{(3)} - x_4^{(2)} \right| = \left| 0{,}903 - 0{,}903 \right| = 0; \end{cases}$$

$$\Delta_2 = \max\{0{,}002;\ 0{,}001;\ \ 0;\ \ 0\} = 0{,}002 > \Delta = 0{,}001.$$

Since $\Delta_2 > 0{,}001$, the specified accuracy of the calculation was also not achieved, which indicates the need to perform the fourth iteration:

$$\begin{cases} x_1^{(4)} = 0{,}304 - 0{,}751 x_2^{(3)} = 0{,}304 - 0{,}751 \cdot 0{,}486 = -0{,}060; \\ x_2^{(4)} = 0{,}506 - 0{,}036 x_3^{(3)} - 0{,}058 x_1^{(4)} = 0{,}506 - 0{,}036 \cdot 0{,}663 - \\ -0{,}058 \cdot (-0{,}060) = 0{,}486; \\ x_3^{(4)} = 0{,}706 - 0{,}036 x_2^{(4)} - 0{,}028 x_4^{(3)} = 0{,}706 - 0{,}036 \cdot 0{,}486 - \\ -0{,}028 \cdot (0{,}903) = 0{,}663; \\ x_4^{(4)} = 0{,}905 - 0{,}003 x_3^{(4)} = 0{,}905 - 0{,}003 \cdot 0{,}663 = 0{,}903. \end{cases}$$

Checking the accuracy of the current calculation is performed:

$$\begin{cases} \left| x_1^{(4)} - x_1^{(3)} \right| = \left| -0{,}060 - 0{,}060 \right| = 0; \\ \left| x_2^{(4)} - x_2^{(3)} \right| = \left| 0{,}486 - 0{,}486 \right| = 0; \\ \left| x_3^{(4)} - x_3^{(3)} \right| = \left| 0{,}663 - 0{,}663 \right| = 0; \\ \left| x_4^{(4)} - x_4^{(3)} \right| = \left| 0{,}903 - 0{,}903 \right| = 0; \end{cases}$$

$$\Delta_3 = \max\{0;\ 0;\ \ 0;\ \ 0\} = 0 < \Delta = 0{,}001.$$

Now $\Delta_3 < 0{,}001$, which indicates that the specified calculation accuracy has been achieved. Therefore, the solution of the system with an error of Δ=0,001:

$$\begin{cases} x_1^{(4)} = -0,060; \\ x_2^{(4)} = 0,486; \\ x_3^{(4)} = 0,663; \\ x_4^{(4)} = 0,903. \end{cases}$$

The «minus» sign in the obtained CUE value shows that an exothermic chemical reaction with heat release is taking place on the first technological equipment. This type of technological equipment with CUE x_1 refers to chemical devices.

The algorithm for the numerical implementation of the Gauss-Seidel method for solving the LAES is presented in Figure 1.9.

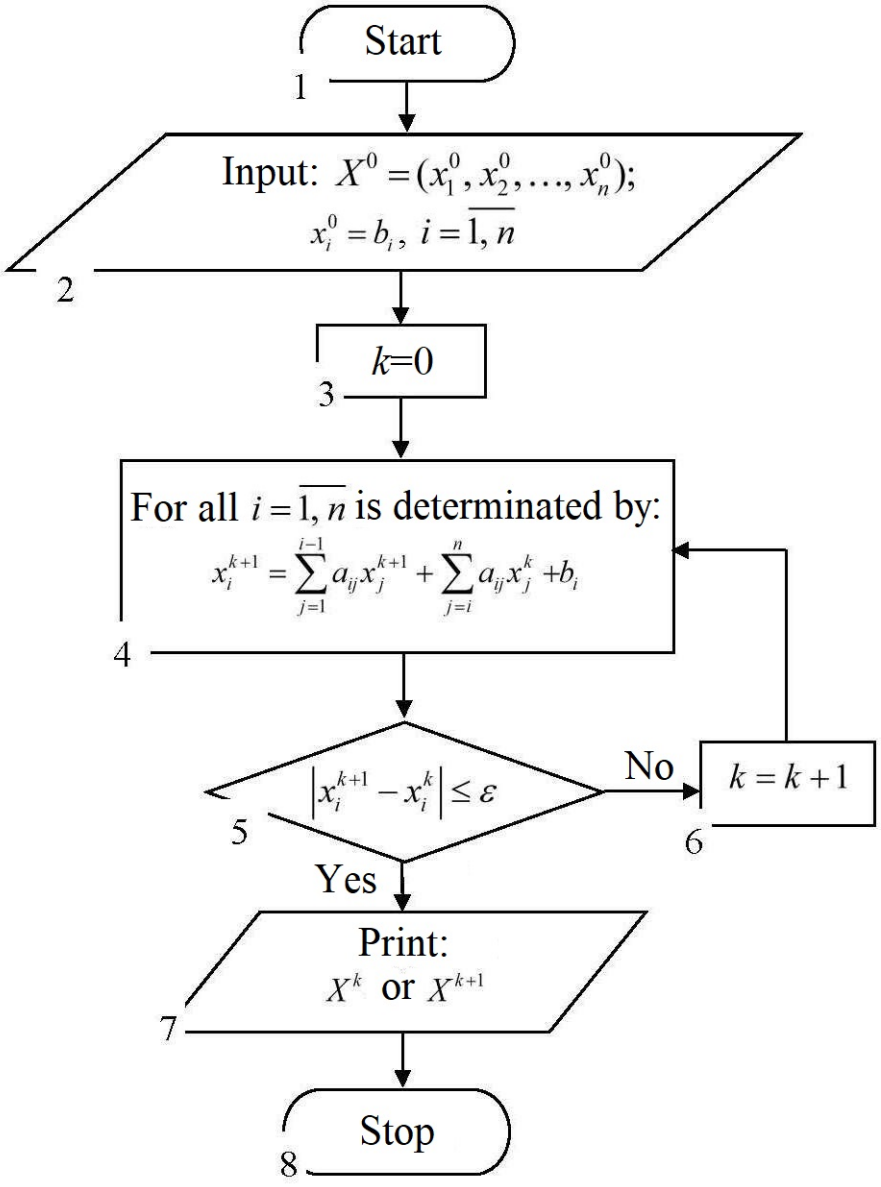


Figure 1.9 – Scheme of the Gauss-Seidel method algorithm

Consider the implementation of the Gauss-Seidel method in the C++ programming language:

```
public class GauseZeydel {
public static void GauseZeydelMethod()
{
Scanner scanner = new Scanner(System.in);
PrintWriter printWriter = new PrintWriter(System.out);
 int size;
System.out.println("Enter the number of unknowns"); size = scanner.nextInt();
double[][] matrix = new double[size][size + 1]; System.out.println("Enter the
     matrix ");
for (int i = 0; i < size; i++)
{
for (int j = 0; j < size + 1; j++)
{
matrix[i][j] = scanner.nextDouble();
}
}
double eps;
System.out.println("Input the precision"); eps = scanner.nextDouble();
double[] previousVariableValues = new double[size]; for (int i = 0; i < size;
     i++)
{
previousVariableValues[i] = 0.0;
}
while (true)
{
double[] currentVariableValues = new double[size]; for (int i = 0; i < size;
     i++)
{
currentVariableValues[i] = matrix[i][size]; for (int j = 0; j < size; j++)
{
if (j < i)
{
currentVariableValues[i] -= matrix[i][j] * currentVariableValues[j];
}
if (j > i)
{
currentVariableValues[i] -= matrix[i][j] * previousVariableValues[j];
}
}
currentVariableValues[i] /= matrix[i][i];
}
double error = 0;
for (int i = 0; i < size; i++)
{
error += Math.abs(currentVariableValues[i] - previousVariableValues[i]);
}
if (error < eps)
{
break;
}
previousVariableValues = currentVariableValues;
}
for (int i = 0, j=1; i < size; i++, j++)
{
printWriter.print("X" + j + "="  + previousVariableValues[i] + " ");
}
scanner.close(); printWriter.close();
}
}.
```

Conclusions regarding the application of LAES solution methods

In cases of small orders of the system (up to 6–7), you can use Cramer's rule, which allows you to obtain an exact solution and which, for such orders, does not require too many computational operations. Such problems often arise during calculations of small electrical networks or subsystems of automatic control. However, with the increase in the number of equations in the system, the number of operations to calculate determinants of matrices begins to increase very quickly, which leads to a decrease in the efficiency of the method.

Starting with the 9th-10th order of LAES, the use of Gaussian methods and their modifications has advantages and is the most reliable method. However, if the coefficient matrices are very sparse, meaning they contain many zeros, implementing the Gaussian elimination starts to require a large computational cost to change the order of the rows of the matrices in order to ensure a nonzero value of the main element at a certain step.

Sparse matrix problems often arise in data processing and mathematical physics problems, in which a sufficiently dense discretization with a small step is performed to ensure accuracy. In general, for problems of solving LAES of large order (tens, hundreds, thousands, etc.), iterative methods have no competitors, provided that convergence is ensured. It should be noted that in many practical problems, the condition of convergence is ensured by the very formulation of the problem (for example, in the process of solving the Laplace's equation in problems of mathematical physics).

Control questions and tasks

1. What is the difference between direct and indirect methods of solving LAES? Provide a comparative assessment.
2. To reveal the essence of the Cramer's rule, Gaussian elimination (and its variants), and Tridiagonal matrix algorithm.
3. In what form is the LAES presented and which iterative methods are used for its solution?
4. Solve the following system of equations using Cramer's rule. Write an algorithm and a solution program.

$$\begin{cases} x_1 + x_2 + x_3 + x_4 + x_5 = 15; \\ x_1 - x_3 + 7x_4 = 26; \\ x_1 - 2x_2 + 3x_3 - 4x_4 = -10; \\ x_1 - x_2 + 2x_3 + 3x_5 = 20; \\ x_1 + x_3 - x_4 + 10x_5 = 50. \end{cases}$$

5. Solve the LAES from example 4 using the Gaussian elimination. Compose a computational algorithm and program.
6. Solve the LAES from example 4 using the modified Gaussian elimination. Compose a computational algorithm and program.

7. Solve the following LAES using the Jacobi method (simple iterations). Compose a computational algorithm and program. Check the convergence condition.

$$\begin{cases} x_1 = \frac{x_2}{2} + \frac{x_3}{3} - \frac{x_4}{4}; \\ x_2 = \frac{x_1}{10} + \frac{x_2}{10} - \frac{x_3}{10} + \frac{x_4}{2}; \\ x_3 = \frac{x_1}{5} + \frac{x_2}{5} + \frac{x_3}{3} + \frac{x_4}{10}; \\ x_4 = \frac{x_3}{3} + \frac{x_4}{4} + 2. \end{cases}$$

8. Solve the LAES from example 4 using the Gauss-Seidel method. Develop a computational algorithm and program. Verify the convergence condition.

9. How to check the convergence condition of iterative algorithms for calculating LAES?

10. What is the difference between the Jacobi and Gauss-Seidel iterative algorithms?

11. Solve LAES using Cramer's rule:

$$\begin{cases} 2x_1 - x_2 + x_3 + 3x_4 = -1; \\ x_1 + x_2 - x_3 - 4x_4 = 6; \\ 3x_1 - x_2 + x_3 + x_4 = 4; \\ x_1 - 3x_2 + 3x_4 = -5. \end{cases}$$

12. Solve LAES using the Gaussian elimination:

a) $\begin{cases} 0,542x + 2,43y - 3,75z - 0,208 = 0; \\ 2,31x - 3,68y - 4,51z + 4,08 = 0; \\ 13,4x + 2,36y - 9,75z - 36,4 = 0. \end{cases}$

b) $\begin{cases} 0,542x + 2,43y - 3,75z - 0,208 = 0; \\ 2,31x - 3,68y - 4,51z + 4,08 = 0; \\ 13,4x + 2,36y - 9,75z - 36,4 = 0. \end{cases}$

c) $\begin{cases} x_1 - 4x_2 - x_4 = 6; \\ x_1 + x_2 + 2x_3 + 3x_4 = -1; \\ 3x_1 - 5x_2 + 4x_3 = 0; \\ x_1 + 17x_2 + 4x_3 = 0. \end{cases}$

13. Solve the system of equations using the method of iterations:

a) $\begin{cases} x = 0,12x - 0,06y + 0,09z + 0,32; \\ y = 0,17x + 0,11y - 0,07z - 0,21; \\ z = 0,14x - 0,15y - 0,08z + 0,18. \end{cases}$

b) $\begin{cases} x = 0,12x - 0,06y + 0,09z + 0,32; \\ y = 0,17x + 0,11y - 0,07z - 0,21; \\ z = 0,14x - 0,15y - 0,08z + 0,18. \end{cases}$

c) $\begin{cases} 60x - 2y + 6z = 30; \\ 10x - 80y - 4z = 20; \\ 12x + 6y - 90z = 45. \end{cases}$

14. Solve LAES by Gauss-Seidel iterative method with calculation error Δ=0,0001:

a) $\begin{cases} 0,63x_1 + 1,00x_2 + 0,71x_3 + 0,34x_4 = 2,08; \\ 1,17x_1 + 0,18x_2 - 0,65x_3 + 0,71x_4 = 0,17; \\ 2,71x_1 - 0,75x_2 + 1,17x_3 - 2,35x_4 = 1,28; \\ 3,58x_1 + 0,28x_2 - 3,45x_3 - 1,18x_4 = 0,05. \end{cases}$

b) $\begin{cases} 7,6x_1 + 0,5x_2 + 2,4x_3 = 1,9; \\ 2,2x_1 + 9,1x_2 + 4,4x_3 = 9,7; \\ -1,3x_1 + 0,2x_2 + 5,8x_3 = -1,4. \end{cases}$

15. Solve LAES using the method of iterations with a calculation error of Δ=0,001:

a) $\begin{cases} 8x_1 + x_2 + x_3 = 26; \\ x_1 + 5x_2 - x_3 = 7; \\ x_1 - x_2 + 5x_3 = 7. \end{cases}$

b) $\begin{cases} 7,6x_1 + 0,5x_2 + 2,4x_3 = 1,9; \\ 2,2x_1 + 9,1x_2 + 4,4x_3 = 9,7; \\ -1,3x_1 + 0,2x_2 + 5,8x_3 = -1,4. \end{cases}$

16. Solve LAES using the Gaussian elimination with a calculation error of Δ=0,001:

a) $\begin{cases} 1{,}14x_1 - 2{,}15x_2 - 5{,}11x_3 = 2{,}05; \\ 0{,}42x_1 - 1{,}13x_2 + 7{,}05x_3 = 0{,}80; \\ -0{,}71x_1 + 0{,}83x_2 - 0{,}02x_3 = -1{,}07. \end{cases}$

b) $\begin{cases} 1{,}14x_1 - 2{,}15x_2 - 5{,}11x_3 = 2{,}05; \\ 0{,}42x_1 - 1{,}13x_2 + 7{,}05x_3 = 0{,}80; \\ -0{,}71x_1 + 0{,}83x_2 - 0{,}02x_3 = -1{,}07. \end{cases}$

17. Solve the LAES given in the matrix form $\boldsymbol{X}=\boldsymbol{C}\boldsymbol{X}+\boldsymbol{d}$ (Table 1.5) using the method of simple iteration with calculation error Δ=0,001.

Table 1.5 – Output data for the task

№	$\boldsymbol{C}$	$\boldsymbol{d}$	№	$\boldsymbol{C}$	$\boldsymbol{d}$
1	2	3	1	2	3
1	$\begin{pmatrix} 0 & 0{,}3 & -0{,}1 & 0{,}2 \\ 0{,}2 & 0 & -0{,}21 & 0{,}2 \\ 0{,}3 & -0{,}1 & 0 & 0{,}3 \\ 0{,}3 & -0{,}1 & -0{,}2 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 \\ -4 \\ 2 \\ 0{,}1 \end{pmatrix}$	2	$\begin{pmatrix} 0 & 0{,}13 & -0{,}4 & 0{,}2 \\ 0{,}25 & 0 & -0{,}14 & 0{,}2 \\ 0{,}3 & -0{,}1 & 0 & 0{,}3 \\ 0{,}3 & -0{,}4 & -0{,}2 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 \\ -4 \\ 2 \\ 0{,}1 \end{pmatrix}$
3	$\begin{pmatrix} 0 & 0{,}27 & -0{,}1 & 0{,}2 \\ 0{,}2 & 0 & -0{,}26 & 0{,}2 \\ 0{,}3 & -0{,}1 & 0 & 0{,}5 \\ 0{,}2 & -0{,}1 & -0{,}2 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 \\ -4 \\ 2 \\ 0{,}1 \end{pmatrix}$	4	$\begin{pmatrix} 0 & 0{,}23 & -0{,}2 & 0{,}2 \\ 0{,}1 & 0 & -0{,}24 & -0{,}1 \\ 0{,}2 & -0{,}1 & 0 & 0{,}2 \\ 0{,}23 & -0{,}4 & -0{,}2 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 \\ -2 \\ 2 \\ 0{,}1 \end{pmatrix}$
5	$\begin{pmatrix} 0 & 0{,}3 & -0{,}1 & 0{,}2 \\ 0{,}2 & 0 & 0{,}1 & -0{,}2 \\ 0{,}3 & -0{,}1 & 0 & 0{,}3 \\ 0{,}3 & 0{,}1 & -0{,}2 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 \\ -4 \\ 2 \\ 0{,}1 \end{pmatrix}$	6	$\begin{pmatrix} 0 & 0{,}3 & 0{,}4 & 0{,}2 \\ 0{,}1 & 0 & -0{,}14 & 0{,}14 \\ 0{,}1 & 0{,}1 & 0 & 0{,}3 \\ 0{,}3 & -0{,}4 & -0{,}2 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 \\ -1 \\ 2 \\ 0{,}1 \end{pmatrix}$
7	$\begin{pmatrix} 0 & 0{,}3 & -0{,}4 & 0{,}2 \\ 0{,}1 & 0 & -0{,}14 & -0{,}1 \\ 0{,}1 & -0{,}1 & 0 & 0{,}3 \\ 0{,}3 & -0{,}4 & -0{,}2 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 \\ -1 \\ 2 \\ 0{,}1 \end{pmatrix}$	8	$\begin{pmatrix} 0 & 0{,}1 & -0{,}1 & 0{,}2 \\ 0{,}2 & 0 & -0{,}2 & 0{,}1 \\ 0{,}13 & -0{,}2 & 0 & 0{,}3 \\ 0{,}1 & -0{,}1 & -0{,}2 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 \\ -1 \\ 2 \\ 0{,}1 \end{pmatrix}$
9	$\begin{pmatrix} 0 & 0{,}1 & -0{,}4 & 0{,}2 \\ 0{,}15 & 0 & 0{,}1 & 0{,}2 \\ 0{,}3 & -0{,}1 & 0 & 0{,}3 \\ 0{,}1 & -0{,}14 & -0{,}2 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 \\ -2 \\ 2 \\ 0{,}1 \end{pmatrix}$	10	$\begin{pmatrix} 0 & 0{,}3 & -0{,}1 & 0{,}2 \\ 0{,}2 & 0 & 0{,}1 & -0{,}2 \\ 0{,}1 & -0{,}2 & 0 & 0{,}1 \\ 0{,}1 & 0{,}2 & -0{,}2 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 \\ -0{,}5 \\ 2 \\ 0{,}1 \end{pmatrix}$
11	$\begin{pmatrix} 0 & 0{,}3 & 0{,}1 & -0{,}2 \\ 0{,}2 & 0 & -0{,}1 & -0{,}2 \\ -0{,}1 & 0{,}2 & 0 & -0{,}1 \\ 0{,}1 & 0{,}2 & -0{,}2 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 \\ 0{,}5 \\ -2 \\ 0{,}1 \end{pmatrix}$	12	$\begin{pmatrix} 0 & 0{,}3 & 0{,}14 & 0{,}2 \\ 0{,}11 & 0 & -0{,}41 & -0{,}1 \\ 0{,}1 & 0{,}1 & 0 & 0{,}13 \\ 0{,}13 & -0{,}4 & -0{,}2 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 \\ -1 \\ 2 \\ 0{,}1 \end{pmatrix}$

18. Solve the LAES ***AX***=***B*** (Table 1.6) using the Gauss-Seidel method with a calculation error of Δ=0,001:

Table 1.6 – Output data for the task

№	***A***	***B***	№	***A***	***B***
1	2	3	1	2	3
1	$\begin{pmatrix} 0,9 & 0,3 & -0,1 & 0,2 \\ 0,2 & -2 & 0 & -0,2 \\ 0,1 & -0,2 & 1 & -0,1 \\ 0,1 & 0,2 & -0,2 & -1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 5 \\ 2 \\ 0,1 \end{pmatrix}$	2	$\begin{pmatrix} 9 & 3 & -1 & 2 \\ 2 & -7 & 1 & -1 \\ 1 & -2 & 6 & -2 \\ 1 & 1 & -2 & -9 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 5 \\ 2 \\ 1 \end{pmatrix}$
3	$\begin{pmatrix} 1 & 0,1 & 0 & 0,2 \\ 0,2 & -1 & 0 & -0,2 \\ 0,1 & -0,2 & 1 & -0,1 \\ 0,1 & 0,2 & -0,2 & -1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 5 \\ 2 \\ 0,1 \end{pmatrix}$	4	$\begin{pmatrix} 7 & 3 & 0 & 2 \\ 1 & -4 & 0 & 1 \\ 1 & -1 & 6 & -2 \\ 1 & 2 & -2 & -9 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 6 \\ 2 \\ 11 \end{pmatrix}$
5	$\begin{pmatrix} 1 & 0,23 & -0,1 & 0,2 \\ 0,2 & -1 & 0 & 0,2 \\ 0,1 & -0,2 & 1 & 0 \\ 0,14 & 0,2 & -0,2 & -1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 2 \\ 2 \\ 1 \end{pmatrix}$	6	$\begin{pmatrix} 12 & 3 & -2 & 2 \\ 1 & -7 & 1 & -1 \\ 1 & -2 & 11 & -2 \\ 11 & 2 & -2 & -19 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 6 \\ 2 \\ 11 \end{pmatrix}$
7	$\begin{pmatrix} 2 & 0,3 & -0,1 & 0,2 \\ 0,2 & -3 & 0,1 & -0,2 \\ 0,1 & -0,52 & 2 & -0,1 \\ 0,1 & 0,2 & -0,2 & -2 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 5 \\ 2 \\ 0,1 \end{pmatrix}$	8	$\begin{pmatrix} 7 & 3 & -1 & 2 \\ 1 & -7 & 1 & 0 \\ 1 & 0 & 6 & -2 \\ 1 & 2 & -2 & -9 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 6 \\ 2 \\ 11 \end{pmatrix}$
9	$\begin{pmatrix} 4 & 0,2 & -0,1 & 0,3 \\ -0,1 & 3 & 0,2 & -0,3 \\ 0,1 & -0,5 & 2 & -0,1 \\ 0,3 & 0,2 & -0,2 & -2 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 2 \\ 5 \\ 0,1 \end{pmatrix}$	10	$\begin{pmatrix} 6 & 1 & -1 & 2 \\ 3 & 5 & 1 & 0 \\ 2 & 1 & -6 & 1 \\ 1 & 4 & 3 & -8 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 3 \\ 2 \\ 5 \end{pmatrix}$
11	$\begin{pmatrix} 1 & 0,13 & -0,15 & 0,24 \\ 0,2 & -1 & 0,11 & -0,42 \\ 0,11 & -0,2 & 1 & -0,11 \\ 0,1 & 0,12 & -0,42 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 5 \\ 2 \\ 0,1 \end{pmatrix}$	12	$\begin{pmatrix} 7 & 3 & -1 & 0 \\ 1 & -7 & 0 & -1 \\ 1 & -1 & 6 & -2 \\ 1 & 2 & 0 & -9 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 6 \\ -2 \\ 11 \end{pmatrix}$

19. The movement of the crankshaft mechanism (Fig. 10) is described by the equation:

$$K_1 s_i \cos(\varphi_i) + K_2 \sin(\varphi_i) - K_3 = s_i^2 \ \ (i=1, 2, 3),$$

where $a_1 = \frac{K_1}{2}$, $a_2 = \sqrt{a_1^2 + a_3^2 - K_2}$, $a_3 = \frac{K_3}{2a_1}$.

The crankshaft mechanism must satisfy the following conditions presented in Table 1.7.

Table 1.7 – Mechanism design data

i	S_i	φ_i
1	1,0	20°
2	1,2	45°
3	2,0	60°

Design a device that meets all three conditions above. To do this, you need to write down an equation describing the operation of the mechanism and find the values of K_i. What values of a_1, a_2, a_3 correspond to the desired solution?

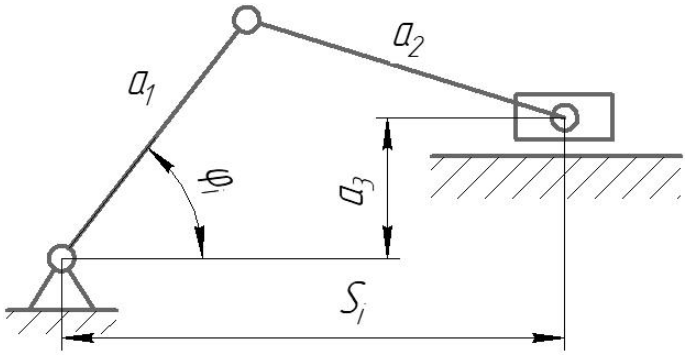


Figure 1.10 – The principal diagram of the crank mechanism

20. Two direct current sources with parameters E_1=11,5 V, R_1=2,5 Ohm, E_2=16,5 V, R_2=6,0 Ohm are connected in parallel to each other, along with a load resistor with resistance R_H=30,0 Ohm (Fig. 1.11). Determine the value and direction of the currents through the direct current sources and the load resistor. To create a system of equations, it is necessary to use Kirchhoff's law and the conditions for equality of currents and voltages for parallel and series connections of electrical elements.

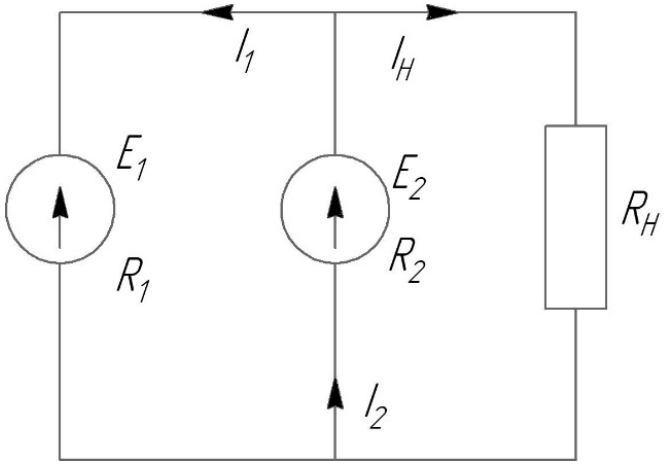


Figure 1.11 – Circuit diagram

Chapter. 2 TASKS OF NON-LINEAR MATHEMATICS

At the current stage, the development of many fundamental and applied sciences is closely related to the use of methods and means of mathematical and computer modeling. The ideology of mathematical modeling involves the formalization of the original object using a mathematical and/or algorithmic description (model), and the study of the object's behavior based on software-implemented computing experiments. This has made it possible to move from simple calculations and evaluations of various structures or processes to a new stage of work – detailed mathematical modeling (computational experiment), which significantly reduces the need for natural experiments and, in some cases, can replace them altogether. The computational experiment is based on the solution of a mathematical model by numerical methods. Often, a mathematical model is reduced to nonlinear equations, algebraic or transcendental, and the presence of any non-linear function other than a power polynomial turns the equations into transcendental ones. In practice, the task is reduced to finding one root on a given interval or all existing roots. Such a problem arises, for example, when assessing the stability of automatic control systems. There are many examples of such problems arising in the process of designing and researching objects of various nature.

In particular, in the field of science known as ballistics, the task of determining the flight parameters of an artillery projectile is important. The mathematical model of projectile flight is defined by the equations of body motion, assuming that the force of air resistance acting on the projectile is proportional to its speed ($R = – kmv$):

– in parametric form (with parameter t representing projectile flight time) in the projection on the vertical axis of the ordinate y:

$$y = \frac{1}{k^2}\left(g + k\upsilon_0 \sin\alpha\right)\left(1 - e^{-kt}\right) - \frac{g}{k}t; \tag{2.1}$$

– in projections on the horizontal and vertical axes xy:

$$y = \frac{1}{k}\left(\frac{g + k\upsilon_0 \sin\alpha}{\upsilon_0 \cos\alpha}\right)x + \frac{g}{k^2}\ln\left(1 - \frac{k}{\upsilon_0 \cos\alpha}x\right), \tag{2.2}$$

where m is the mass of the projectile, α is the angle to the horizon at which the projectile flew out of the artillery barrel with the initial speed υ_0, k is the proportionality factor, t is the projectile flight time, g=9,82 m/sec^2 is the acceleration of free fall.

Using equation (2.1), you can determine the total flight time of the projectile, t, by equating the left side to zero and solving the corresponding transcendental equation. From equation (2.2), by also equating the left side to zero and solving the corresponding transcendental equation, one can find either the initial velocity, υ_0, of the projectile or the range of the projectile flight $x = S$.

The problem of modeling in the field of economics, particularly in finance, is especially relevant today. A typical task involves determining the interest rate and yield of bonds. The mathematical model of profitability is represented by the following transcendental equation:

$$(I+P)\cdot(1+i)^{-n}+P\cdot(1+i)^{-n+1}+A_n\cdot(1+i)-(A_n+I)=0, \qquad (2.3)$$

where A_n is the current value of the bond (in monetary units), P is the nominal value (in monetary units); N is the term that has passed since the bond was issued (in years); T is the maturity (in years); $n=T–N$ is the term remaining until the bond's maturity (in years); k is the coupon interest rate (in fractions of a unit); $I = P\cdot k$ is the amount of coupon payments (the product of the nominal value P by the coupon interest rate k) (in monetary units).

A well-known problem in the construction industry is to determine the critical force (loss of vertical balance of the rod) applied along a rod. One end of the rod is fixed while the other end can move in the vertical direction. This problem can be defined by the equation:

$$tg\left(\sqrt{\frac{PL}{EI}}\right)-\sqrt{\frac{PL}{EI}}=0, \qquad (2.4)$$

where P is the critical force (N), EI is the bending stiffness of the rod (N·m^2), L is the length of the rod (m).

Metrologists often solve the problem of finding the zero point of a nonlinear measuring or grading characteristic, which boils down to solving the same set of nonlinear equations.

The presented and mentioned mathematical models (2.1) – (2.4) cannot be investigated by the methods of exact solution of equations, since the variable arguments are under the sign of a transcendental function or a nonlinear function of high order (and it is known that exact analytical formulas for calculating the roots exist only for algebraic equations not higher than the third order). However, the exact solution of the equation in technical problems is not absolutely necessary. The task of finding the roots of the equation can be considered practically solved if it is possible to determine the roots with guaranteed accuracy and indicate the limits of possible error. If there are exact analytical formulas for power functions, then for transcendental equations and any systems of equations, such methods do not exist at all, and only approximate iterative methods and algorithms need to be used, the most common of which will be discussed below.

This section deals with solving computational problems in nonlinear mathematics: solving transcendental equations and systems of nonlinear equations, as well as finding complex roots of algebraic equations (polynomial equations).

2.1 Generalized formulation of the problem and the procedure for localizing the roots of the equation

The study of mathematical models (2.1) – (2.4) involves the task of calculating the real roots of equations of the type $f(x) = 0$ on a given interval $[a; b]$, where $f: R_1 \mapsto R_2$ is an algebraic or transcendental function. It is assumed that the function $f(x)$ is a piecewise continuous function of a real argument, which is continuous on the interval $[a; b]$ and has a piecewise continuous derivative.

The number $x = \xi$ is called the root of the function $f(x)$ if $f(\xi) \equiv 0$.

The number ξ is called the root of the k-th multiplicity if, for $x = \xi$, all its derivatives up to and including the order k-1, together with the function, are equal to zero:

$$f(\xi) = f'(\xi) = \cdots = f^{k-1}(\xi) = 0, \text{ but } f^{(k)}(\xi) \neq 0. \tag{2.5}$$

When determining the approximate values of the roots of the equation $f(x) = 0$, two problems must be solved:

1) separation of roots, i.e. determination of sufficiently small intervals, in each of which there is one and only one root of the equation (simple or multiple):

2) refinement of roots with a predetermined number of correct signs.

In the case of a graphical separation of the roots of the equation $f(x) = 0$, it is necessary to convert this equation into the form:

$$\varphi_1(x) = \varphi_2(x), \tag{2.6}$$

and plot graphs of functions: $y_1 = \varphi_1(x)$, $y_2 = \varphi_2(x)$.

Indeed, the roots of the equation $f(x) = \varphi_1(x) - \varphi_2(x) = 0$ are the abscissas of the points of intersection of these graphs.

Of all the ways in which the equation $f(x) = 0$ can be transformed into the form (2.6), the one that provides the simplest construction of graphs $y_1 = \varphi_1(x)$ and $y_2 = \varphi_2(x)$ is chosen. In particular, we can take $\varphi_2(x) = 0$, and then we will plot the graph of the function $y = f(x)$, where it intersects (or is tangent to) the line $y_2 = \varphi_2(x)$, which represents the x-axis. These points of intersection or tangency are the roots of the equation $f(x) = 0$ that we are seeking.

In general, when plotting graphs:

$$y_1 = \varphi_1(x); \quad y_2 = \varphi_2(x), \tag{2.7}$$

first of all, it is necessary to determine the behavior of each of the functions $\varphi_1(x)$ and $\varphi_2(x)$ under the conditions $x \to -\infty$ and $x \to +\infty$. Find the values of x for which $\varphi_j(x) = \infty$ (j=1, 2). Determine the points of intersection of these functions with the x and y axes and calculate a number of intermediate, most characteristic values, starting with the values of $\varphi_j(x)$ for $x = \pm 1$, for which it is usually easier to calculate the values of any function.

Example 2.1. Separate the real roots of the equation:

$$\frac{x+1}{x-1} - \sin x = 0. \tag{2.8}$$

Solution:

Having written equation (2.8) in the form $\sin x = \frac{x+1}{x-1}$, we construct graphs $y_1 = \sin(x)$, $y_2 = \frac{x+1}{x-1}$. The abscissas of the intersection points determine all the real roots of the original equation (Fig. 2.1). In this case, all roots are negative, since the right-hand side of the hyperbola $y_2 = \frac{x+1}{x-1}$ does not intersect the sinusoid $y_1 = \sin(x)$ anywhere. Although the number of roots is infinite, they are all isolated because only one root occurs in each of the intervals $(0; -\pi/2)$, $(-\pi/2; -3\pi/2)$, …, $((1-2k)\pi/2; (-1-2k)\pi/2)$, where $k = 1, 2, 3, \ldots$.

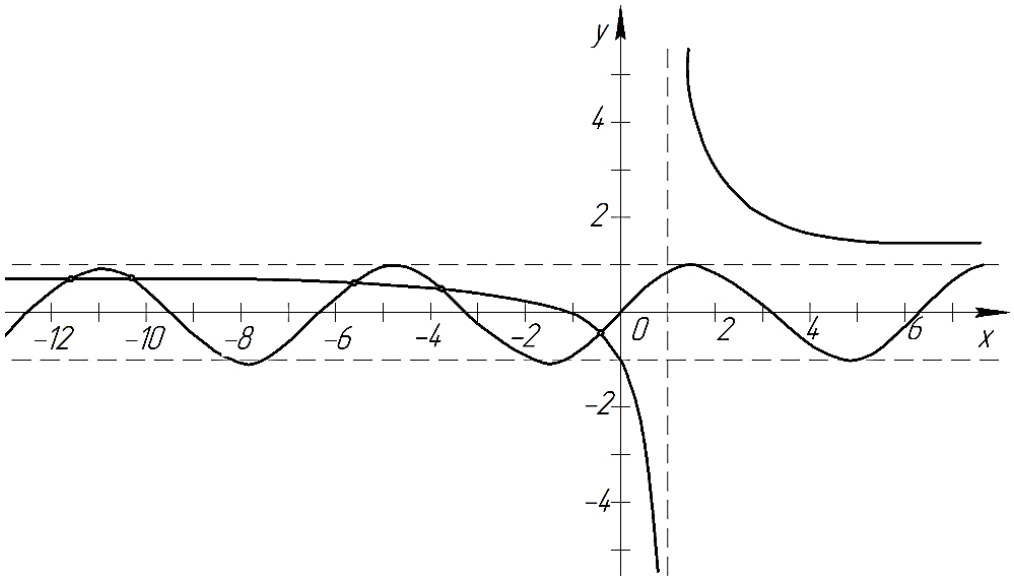


Figure 2.1 – Diagram illustrating the graphical determination of the equation's roots

Among the analytical methods of separating roots, the two most common general methods are used, which are equally suitable for both algebraic and transcendental equations. One of them is to find a simpler equation that has roots that are approximately equal to the unknown roots of this equation. This can often be achieved by neglecting the small terms in the given equation. The second method uses theorems that directly follow from known properties of continuous functions.

Theorem 2.1 (theorem about the root of a continuous function). If the function $f(x)$ is continuous on the segment $[a, b]$ and its values at the endpoints of the segment, $f(a)$ and $f(b)$, have opposite signs, then there is at least one real root of the equation $f(x) = 0$ between the points a and b.

Theorem 2.2. If the function $f(x)$ is strictly monotonic on the segment $[a; b]$, that is, it strictly increases or decreases on $[a; b]$, then on this segment the equation $f(x) = 0$ cannot have more than one root.

Example 2.2. For the function $f(x) = x^3-4x+2$ it is necessary to find the intervals. Solving the inequality $f'(x) = 3x^2-4 > 0$, we get: $x \in \left(-\infty; -\frac{2}{\sqrt{3}}\right) \cup \left(\frac{2}{\sqrt{3}}; +\infty\right)$. The function increases on these two intervals. It is clear that the function decreases on the interval $x \in \left(-\frac{2}{\sqrt{3}}; \frac{2}{\sqrt{3}}\right)$.

The value of the function at the extremum points:

$$f\left(-\frac{2}{\sqrt{3}}\right) = 2 + \frac{16}{3\sqrt{3}} > 0, \quad f\left(\frac{2}{\sqrt{3}}\right) = 2 - \frac{16}{3\sqrt{3}} = \frac{8(\sqrt{3}-2)}{3\sqrt{3}} < 0$$

means that the root ξ^{**} is separated on the descent segment $\left[-\frac{2}{\sqrt{3}}; \frac{2}{\sqrt{3}}\right]$. Since it is obvious that $f(x) \to -\infty$ for $x \to -\infty$ and $f(x) \to +\infty$ for $x \to +\infty$, there are two more roots: $\xi^* \in \left(-\infty; -\frac{2}{\sqrt{3}}\right)$ and $\xi^{***} \in \left(\frac{2}{\sqrt{3}}; +\infty\right)$.

To separate these roots, the values at points -3 and 3 are additionally calculated. For example, we have : $f(-3)=-13<0$ and $f(3) = 17 > 0$. Consequently, the following segments are obtained on which the roots are separated:

$$\xi^* \in [-3; -\frac{2}{\sqrt{3}}]; \quad \xi^{***} \in [\frac{2}{\sqrt{3}}; 3].$$

The function $f(x)$ changes sign when passing through the root ξ^*, i.e. if $f'(\xi^*) \neq 0$.

In the general case, if the specified procedures are complicated for analysis, the entire definition area or a certain part of it, which for some reasons is of interest, is divided into segments by points xi located at a conditionally insignificant distance h. Having determined the values at all these points, the fulfillment of the condition $f(x_{i-1}) \cdot f(x_i) \leq 0$ is checked. If the number of roots in the studied area is previously known, then by narrowing down the step h of the search, one can either localize all of them or assert that the presence of pairs of roots not determined with accuracy $h = \varepsilon$ is possible.

2.2 Numerical methods of solving nonlinear algebraic equations

After the study of the equation $f(x) = 0$ is finished and for each real root ξ the interval in which this root is located is established, we proceed to the solution of the second problem – refinement of the found roots.

Separation of the root ξ, i.e. the establishment of the double inequality $a < \xi < b$, by itself makes it possible to obtain its rough approximate value. For example, you can take the center of the interval $[a; b]$. In this case, the absolute error will be less than $\varepsilon < \frac{|a-b|}{2}$.

After substituting the value of ξ_1 into the equation $f(x)=0$ and ensuring that it does not yield the desired accuracy, either $f(\xi_1) = a_1$ or $f(\xi_1) = b_1$ is chosen. As a result, a more precise inequality is satisfied: $a_1<\xi<b$ or $a<\xi<b_1$.

Method of dividing in half (dichotomy)

Suppose that the root ξ of the equation $f(x)=0$ is determined in the interval $[a; b]$, and in this case, the signs of the functions $f(a)$ and $f(b)$ are different (the function $f(x)$ changes its sign when crossing the value of the root ξ).

Let's assume that $a_0 = a$ and $b_0 = b$, and calculate the value of the function at the left end of the segment $f(a_0)$, as well as at its middle $f(c_0)$, where $c_0 = \frac{a_0 + b_0}{2}$. We compare the signs of the values of $f(a_0)$ and $f(c_0)$. If these signs are different, then the value of the root ξ lies in the interval $[a_0; c_0]$. If they are the same, then the signs of the values of f $f(c_0)$ and $f(b_0)$ are different, and the root lies in the interval $[c_0; b_0]$. There is a possible case when $f(c_0) = 0$, then the value of the root $\xi = c_0$ will already be considered found. In both cases of sign change, the value of the root will be in the segment $[a_0; c_0]$ or $[c_0; b_0]$, the length of which will be two times smaller than the length of the original segment $a_0; b_0] = [a; b]$. This segment is denoted as half the length by $[a_1; b_1]$ (that is, $a_1 = a_0$, $b_1 = c_0$ is assumed in the case when the values of $f(a_0)$ and $f(c_0)$ have different signs; $a_1 = c_0$, $b_1 = b_0$ in the case when $f(a_0)$ and $f(c_0)$ have the same sign).

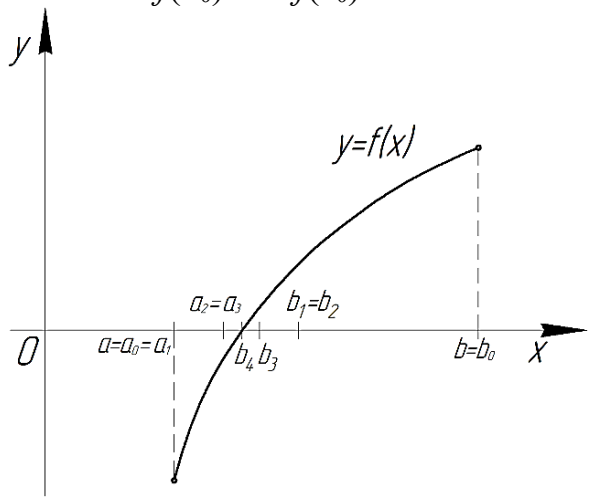


Figure 2.2 – Diagram of halving the segment and approximation to the value of the root ξ

Afterwards, the standard process is repeated for the segment $[a_1; b_1]$. Specifically, the middle of c_1 is searched for, as well as the value of the function $f(c_1)$, while simultaneously comparing the sign of this number with the sign of the value of the function $f(a_1)$. If these signs differ, then the value of the root lies within the segment $[a_2; b_2] = [a_1; b_1]$; if they are the same, then it lies within $[a_2; b_2] = [c_1; b_1]$ (if $f(c_1) = 0$, then the value of the root is considered found). Additionally, the length of the segment on which the root is located is reduced by another two times.

Repeating the typical process, we find that after k divisions, the length of the segment on which the root is located is reduced by 2^k times, and is equal to $\delta_k = \frac{b-a}{2^k}$. If the value of the root ξ was not precisely determined at some previous stage, that is, if it does not coincide with the value of c_i for some i.

The algorithm of dividing in half (dichotomy) is shown in figure 2.3.

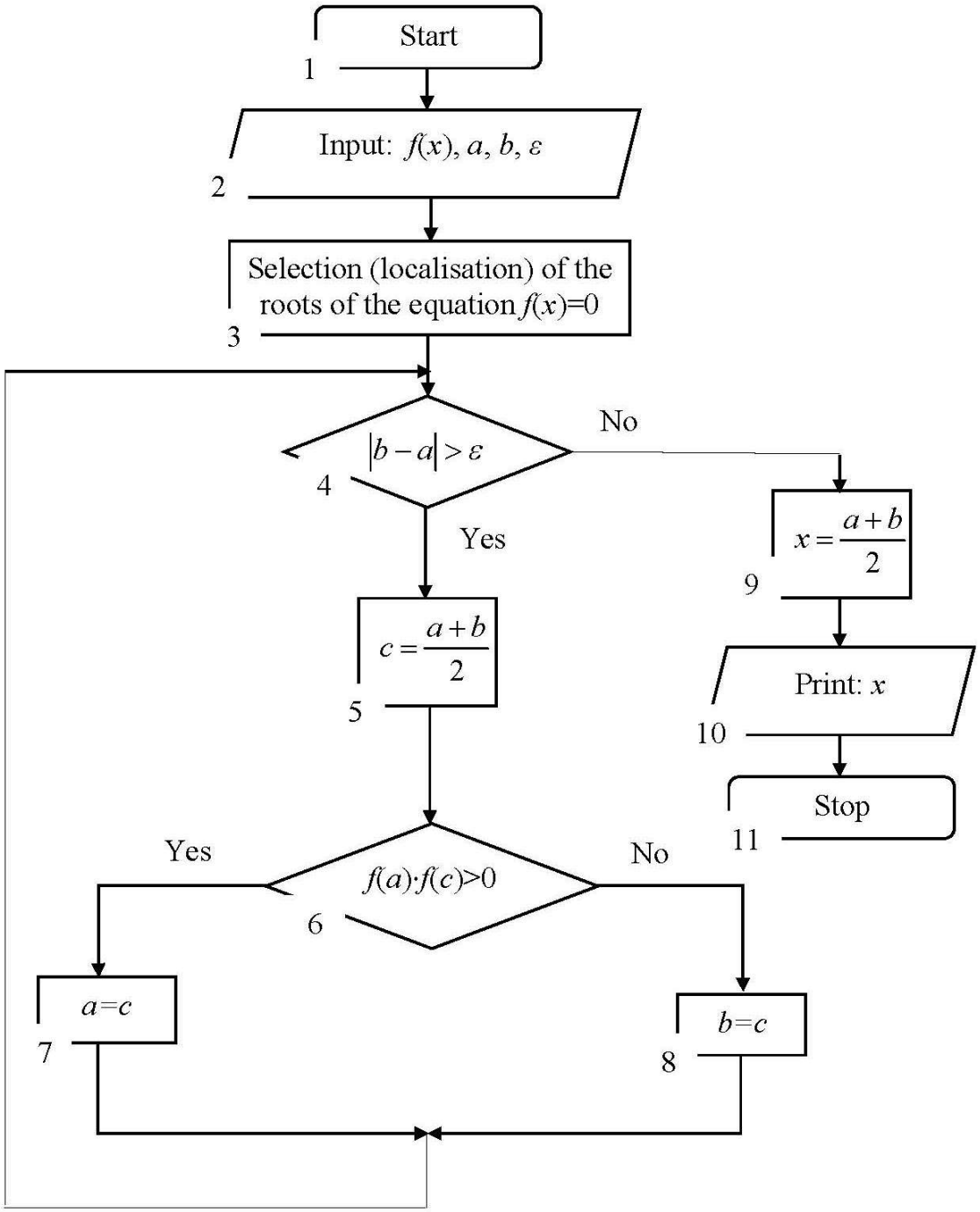


Figure 2.3 – Scheme of the algorithm for the method of dividing in half (dichotomy)

Example 2.3. Solve equations by the numerical method of dividing in half (dichotomy) with an accuracy of $\varepsilon = 0{,}001$:

$$x^3+2x^2+3x+5 = 0.$$

Solution:

The solution begins with the use of the method of halving on the segment $[-2; -1]$, on which the root ξ is separated by the method of graphically constructing the graph of the function $\varphi(x) = x^3+2x^2+3x+5$ (Fig. 2.4).

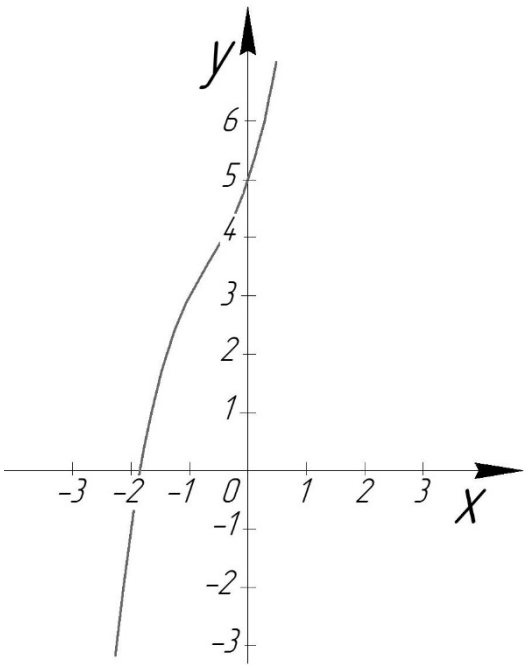


Figure 2.4 – Graphical determination of the interval of the roots' ownership of the equation

The values of the function in the middle of the obtained segments are determined sequentially: $f(-1,5) = 1,625$; $f(-1,75) = 0,515625$; $f(-1,875) = -0,185547$; …; $f(-1.841797) = 0.011269$, after which the calculation stops at the ninth step because the next segment has a length $\frac{1}{2^9} = \frac{1}{512} < 2\varepsilon = \frac{1}{500}$. In this case, the middle of the last segment is a point –1,842773. It was found that the approximate value of the $\tilde{x}$ root ξ with an accuracy of 0.001 is $\tilde{x} \approx -1,843$.

Please note that the method of dividing a segment in half, as well as the method of simple sorting, does not require the smoothness of the function (i.e., the existence of its derivative). It is sufficient for the function to be continuous.

Consider the implementation of the method of half division (dichotomy) in the PYTHON programming language:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import *
from sympy import *
x = np.arange(-5, 3, 0.1)
# we define a function
func_x=(x**3)+(2*x**2)+(3*x)+5
# we construct a graph and localize the roots of the equation
plt.figure(figsize=(7, 7))
plt.xlabel('x',fontsize=15, color='blue')
plt.ylabel('y',fontsize=15, color='blue')
plt.plot(x, func_x)
plt.legend(['f(x)=X^3+2x^2+3x+5'], loc=1)
plt.grid(True)
plt.xlim([-4, 1])
plt.ylim([-5, 5])
plt.show()
# we set the initial limits of localization of the root of the equation
a=-2; b=-1; e=0.001
# we set the function itself
def f(x):
    return (x**3)+(2*x**2)+(3*x)+5
from timeit import default_timer as timer
# we determine the root of the equation
i=0
start= timer()
while abs(b-a)>=e:
    i=i+1
    z = (a+b)/2
```

```
    if f(a)*f(z)<=0:
        b=z
    else:
        a=z
end = timer()
print("x =", z, "Time taken:", end-start, 'Number of iterations:', i).
```

Newton's method (tangents)

Newton's method, or the method of tangents, is one of the most widely used methods for refining roots, equally suitable for both algebraic and transcendental equations.

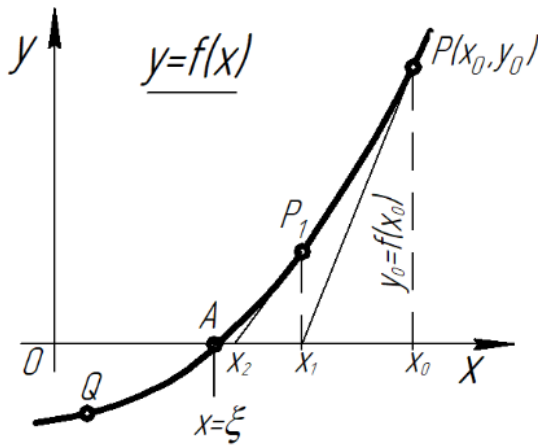


Figure 2.5 – Scheme for determining the roots of the equation using Newton's method

Let the arc PAQ be the arc of the curve $y = f(x)$ (Fig. 2.5), which intersects the axis Ox at point A, such that the abscissa $x = \xi$ of point A is the root of the equation $f(x) = 0$.

Suppose that the arc AP is inverted by convexity to the axis Ox. Let's draw a tangent to the curve $y = f(x)$ through the point P with coordinates $(x_0, y_0 = f(x_0))$. The slope of the tangent is equal to the value of the derivative of the function $f(x)$. The slope of the tangent is equal to the value of the derivative of the function $f(x)$ at the point of contact; $k = f'(x_0)$, respectively, the equation of the tangent passing through the point $P(x_0, y_0)$:

$$y - y_0 = f'(x_0)(x - x_0)\,. \tag{2.9}$$

Putting $y = 0$ and $y_0 = f(x_0)$ into equation (2.9), we can determine the point of intersection with the x-axis ($y = 0$), which we denote as x_1.

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}\,. \tag{2.10}$$

Through the point $P_1\,(x_1, y_1 = f(x_1))$, it is necessary to draw a tangent line. By continuing this process, we arrive at Newton's formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (n = 0, 1, 2, \ldots)\,, \tag{2.11}$$

which makes it possible to calculate increasingly accurate root values step by step. In other words, the values x_0, x_1, x_2, … calculated by formula (2.11) form a sequence that approaches the value of the root $f(x) = 0$.

If we start the process from the point Q, where the curve is concave to the Ox-axis, then the first step will lead to the other side of the Ox-axis, where the curve is convex to it. In the future, we will approach the value of the root in the same way as earlier.

In those cases when the calculation of the second derivative for the function $f(x)$ is inverted by convexity on the Ox axis at those points for which the relation holds:

$$f(x) \cdot f''(x) > 0, \tag{2.12}$$

then this condition must be satisfied by the selected initial value x_0.

In the process of calculating the root according to Newton's method, the error of each new approximation decreases in proportion to the square of the error of the previous approximation.

The algorithm of Newton's (tangent) method is shown in Figure 2.6.

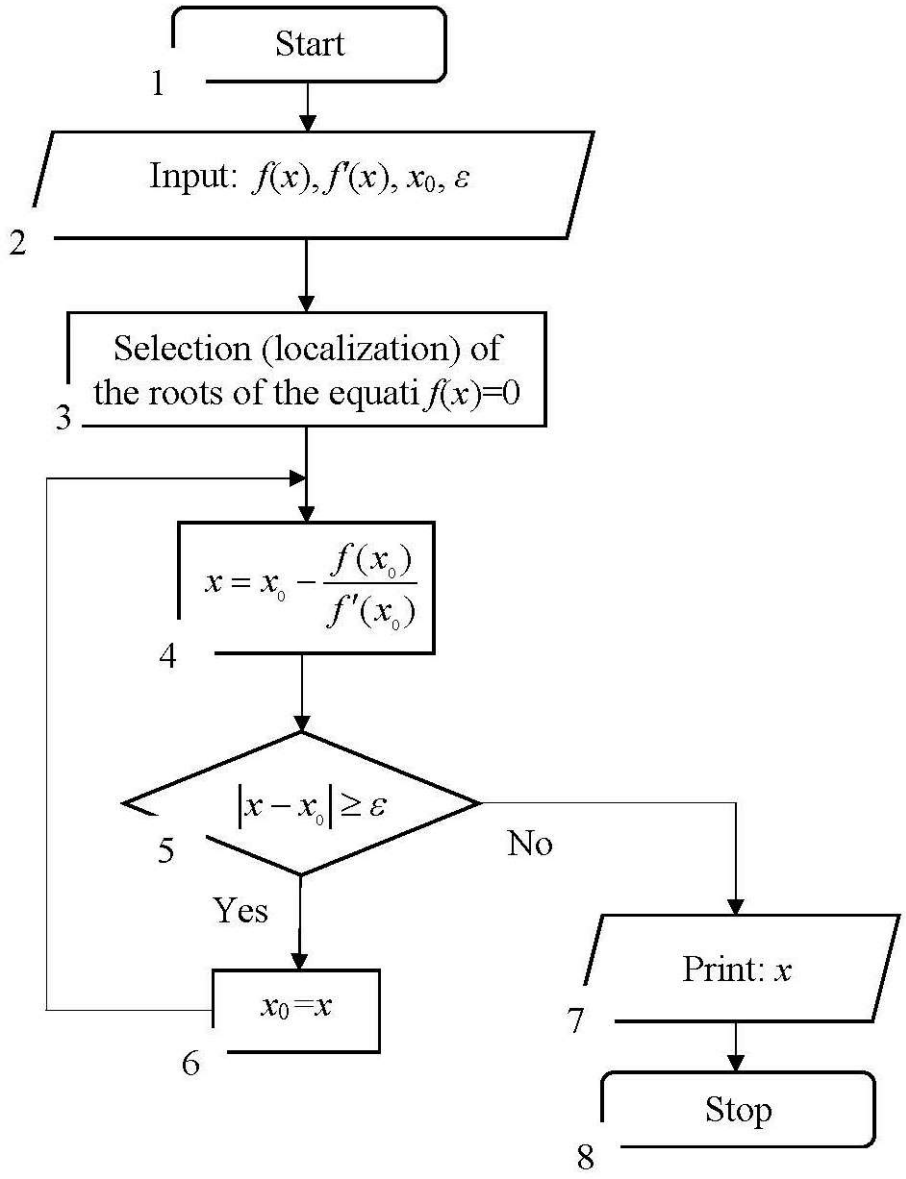


Figure 2.6 – Scheme of the Newton's method algorithm (tangents)

Example 2.4. Calculate the smallest positive root of the equation:

$$e^x - 3x = 0 \qquad (2.13)$$

with precision ε=0,0001.

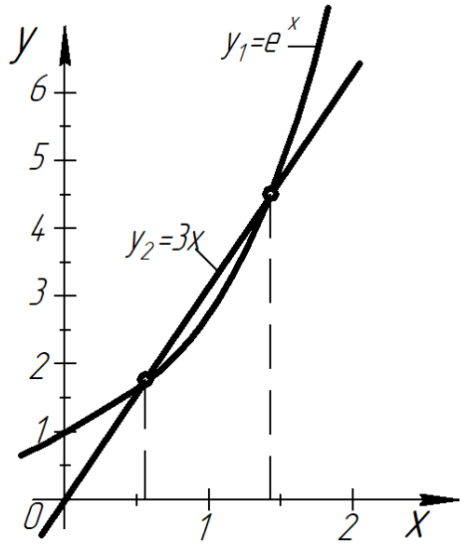


Figure 2.7 – Graphical depiction of the range of values for which the roots of the equation belong

Solution:

In order to separate the roots, equation (2.13) is reduced to the form $e^x = 3x$, and graphs of the functions are constructed: $y_1 = e^x$ and $y_2 = 3x$ (Fig. 2.7). The scales along the *Ox* and *Oy* axes may be different.

According to Figure 2.7, the smallest positive root lies in the interval [0; 1]. Moving on to clarification in this example, it is necessary to define:

$$f(x) = e^x - 3x; \quad f'(x) = e^x - 3; \quad f''(x) = e^x,$$

then formula (2.11) takes the form:

$$x_{n+1} = x_n - \alpha_n, \qquad (2.14)$$

where $\alpha_n = \dfrac{f(x_n)}{f'(x_n)}$.

because according to the criterion (2.12):

$$f(x) \cdot f''(x)\big|_{x=0} = (e^x - 3x)e^x\big|_{x=0} = 1 > 0,$$

whereas

$$f(x) \cdot f''(x)\big|_{x=1} = (e^x - 3x)e^x\big|_{x=1} = (e-3)e < 0.$$

All further approximations calculated by formula (2.14) are listed in Table 2.1. Thus, the unknown root with the required precision $\varepsilon = 0{,}0001$ is $x_3 = 0{,}6191$.

Table 2.1 – Results of solving the equation

n	$x=x_n$	e^x	$3x$	$f(x_n)$	$f'(x_n)$	α_n
0	0,10000	1,0000	0,0000	1,0000	– 2,00	– 0,500000
1	0,50000	1,6500	1,5000	0,1500	– 1,35	– 0,110000
2	0,61000	1,8404	1,8300	0,0104	– 1,16	– 0,009000
3	0,61900	1,8571	1,8570	0,0001	– 1,14	– 0,000088
4	0,61909	1,8573	1,8573	0,0000	–	–

After performing one more calculation step, we get a more accurate value of this root: $x_4 = 0{,}61906129$.

When calculating the roots of the equation $f(x) = 0$ using Newton's method, the process of successive approximations always coincides if the initial approximation x_0 is taken so close to the root $x = \xi$ that on the interval $[\xi; x_0]$:

1) the slope of the curve $y = f(x)$ is not equal to zero, i.e. $f'(x) \neq 0$;

2) the curve $y = f(x)$ has no inflection points, i.e. $f''(x) \neq 0$.

Practically, this means that according to formula (2.11), the root of the equation $f(x) = 0$ can be calculated with any arbitrarily high degree of accuracy, if only the root $x = \xi$ is not a multiple ($f'(x) \neq 0$) and if the zero approximation x0 is taken sufficiently close to the sought root ξ.

Multiple roots can also be defined by Newton's formula as corresponding to the roots of the equation $f'(x) = 0$, or, more generally, $f^{(k)}(x) = 0$ ($k = 1, 2, 3, ...$).

Consider the implementation of Newton's (tangent) method in the PYTHON programming language:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import *
from sympy import *
x = np.arange(-5, 5, 0.1)
# we define a function
value_funcm=[]
for i in x:
    value_funcm.append(math.exp(i)-(3*i))
M = np.array(value_funcm)
# we construct a graph and localize the roots of the equation
plt.figure(figsize=(7, 7))
plt.xlabel('x',fontsize=15, color='blue')
plt.ylabel('y',fontsize=15, color='blue')
plt.plot(x, M)
plt.legend(['f(x)=exp(x)-3*x'], loc=1)
plt.grid(True)
plt.xlim([0, 2])
plt.ylim([-2, 2])
plt.show()
# we set the function itself
def f(x):
    return math.exp(x)-(3*x)
from sympy import *
# we define the first derivative of the function
x_arg = Symbol('x_arg')
func_m=exp(x_arg)-(3*x_arg)
m_first=func_m.diff(x_arg)
func_m = lambdify(x_arg, m_first)
# we define the second derivative of the function
m_second=m_first.diff(x_arg)
func_mm = lambdify(x_arg, m_second)
# we define the first derivative of the function
print("The first derivative of a function f'(x)=",m_first,". The second
    derivative of a function f''(x)=",m_second)
```

```
# we set the initial iteration point and the accuracy for determining the
#root of the equation.
x_n=0; e=0.001
if f(x_n)*func_mm(x_n)<0:
    print("Invalid starting point value. Choose another point value!")
from timeit import default_timer as timer
# definition of Newton's iterative formula
def x_nn(x):
    return x_n-(f(x_n)/func_m(x_n))
k=0
# we determine the root of the equation
start= timer()
while abs(x_nn(x_n)-x_n)>=e:
    k=k+1
    x_n=x_nn(x_n)

end = timer()
print("x =", x_n, "Time taken:", end-start, "Number of iterations:", k).
```

Secant method (linear interpolation)

The idea of the method is that, from two points $M_{i-1}(x_{i-1}, f(x_{i-1}))$ and $M_i(x_i, f(x_i))$, it is necessary to construct a straight line $M_{i-1}M_i$ (a chord connecting two points on the function $y=f(x)$ on the Cartesian coordinate axis) and take as the next approximation x_{i+1} the abscissa of the point of intersection of this line with the Ox axis. That is, at this step, the function $f(x)$ is replaced by its linear interpolation determined by two values of x: x_{i-1} and x_i. It will be considered that the linear interpolation of the function $f(x)$ is such a linear function $l(x)$, the values of which coincide with the values of $f(x)$ at two fixed points, in this case – at the points x_{i-1} and x_i.

Depending on whether the points x_{i-1} and x_i lie on different sides of the root ξ or on the same side, the following principle schemes for determining the roots of the equation in the Cartesian coordinate system, which are shown in Figure 2.8, will be obtained.

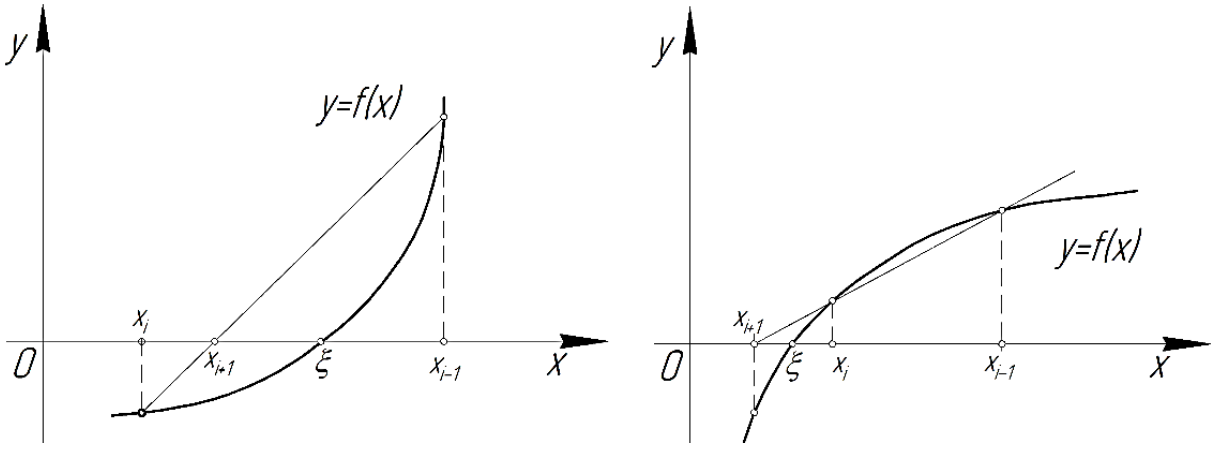


Figure 2.8 – Schemes for determining the roots of the equation using the secant method

The next successive approximation will depend on the previous two values, $x_{i+1} = \varphi(x_{i-1}, x_i)$. The interpolated linear function $l(x)$ will be defined as a function with a slope equal to the difference ratio:

$$k_i = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}, \tag{2.15}$$

constructed for the segment between x_{i-1} and x_i, the graph passes through the point M_i:

$$l(x) = f(x_i) + \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}(x_i - x_{i-1}). \tag{2.16}$$

Solving equation (2.16) under the condition that $l(x) = 0$, we determine:

$$x_{i+1} = \frac{x_{i-1}f(x_i) - x_i f(x_{i-1})}{f(x_i) - f(x_{i-1})} = x_i - \frac{f(x_i)}{\dfrac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}} \tag{2.17}$$

or

$$x_{i+1} = x_i - \frac{f(x_i)}{k_i}. \tag{2.18}$$

The value k_i can be considered as a difference approximation for the derivative $f'(x)$ at the point x_i. That is, the obtained formula (2.17) is a difference analogue of Newton's method's iterative formula.

The calculation according to formula (2.18) is much more acceptable than the calculation according to formula (2.17), even though both are mathematically identical. This is because using formula (2.18) in calculations with rounding results in less loss of significant figures.

There are two ways to apply formula (2.18). The first method involves carrying out the calculation directly according to formula (2.18) for $i = 1, 2, 3, ...$, starting from two approximations x_0 and x_1 that are as close as possible to the root ξ. It is important to note that it is not assumed that ξ lies between x_0 and x_1 (the values of the function $f(x)$ at the points x_0 and x_1 may have different signs), and there is also no guarantee that the root will fall between x_{i-1} and x_i at any subsequent step (although it is not impossible). In such a case, it is difficult to give an estimate of the error with which x_{i+1} approximates the true value of the root ξ, and therefore we settle for the following empirical rule: calculations stop when the inequality $|x_{i+1} + x_i| < \varepsilon$ is fulfilled, where ε is the established accuracy of finding the root of the equation. In this case, the approximate value of the root is chosen, which is equal to $\xi = x_{i+1}$.

Note that the sufficient conditions that ensure the calculation of the root of the equation $f(x) = 0$ by the chord method with an arbitrary given degree of accuracy will be the same as those for Newton's method. However, Newton's

method generally offers better convergence than the chord method in most cases.

In cases where the calculation of $f(x)$ and its derivative $f'(x)$ is time-consuming, the secant method provides greater cost savings. In this case, the main factor affecting the acceleration of process convergence is the size of the segment $[x_1; \xi]$ under consideration, where ξ is the true value of the root. Therefore, all other conditions being equal, it is necessary to give preference to the segment $[x_{n-1}; x_n]$, which is smaller in absolute value.

The secant method (linear interpolation) algorithm is shown in Figure 2.9.

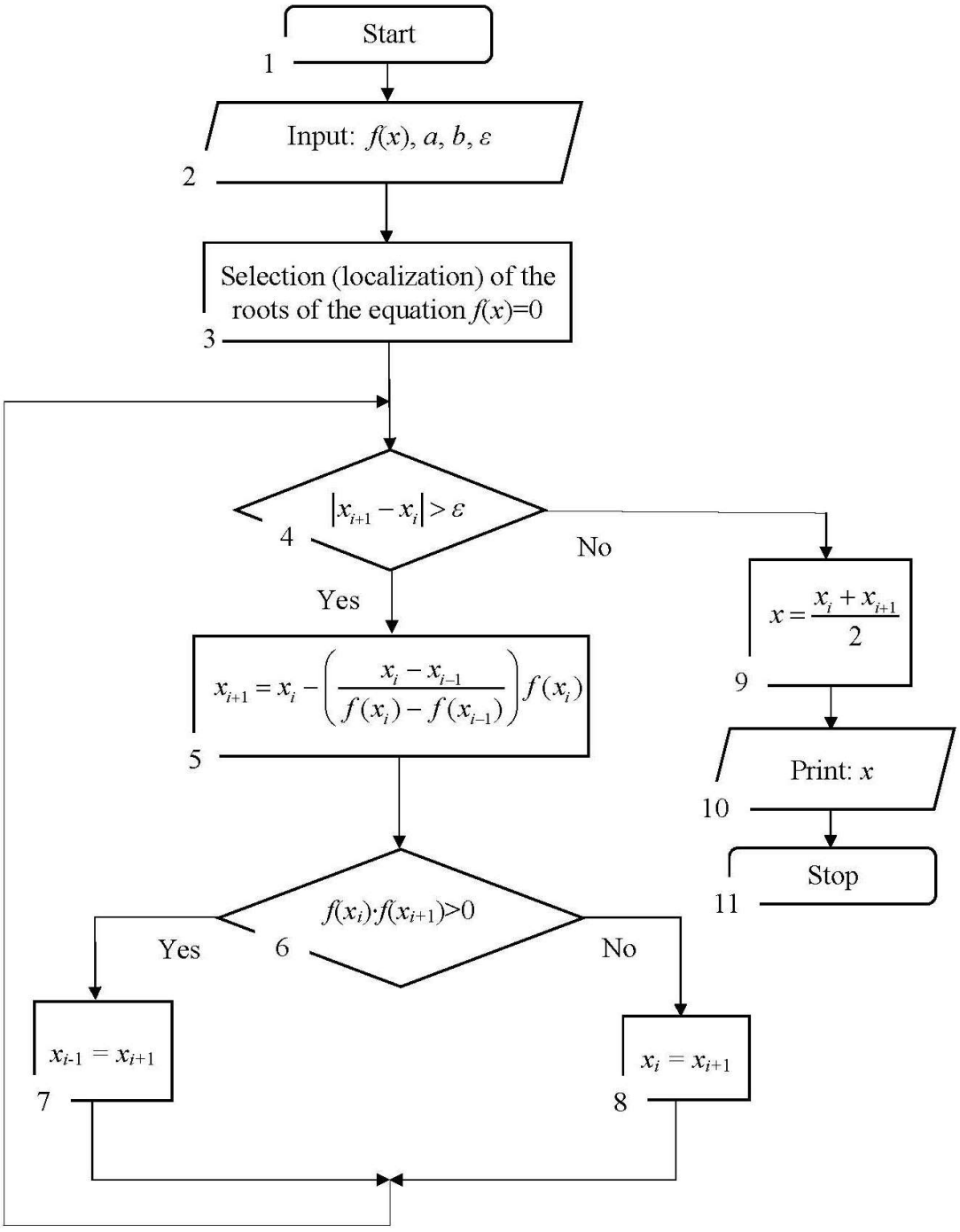


Figure 2.9 – Scheme of the secant method algorithm (linear interpolation)

Example 2.5. Determine the value of the real root of the equation:

$$x^3 - 2x - 5 = 0 \qquad (2.19)$$

with precision ε=0,00001.

Solution:

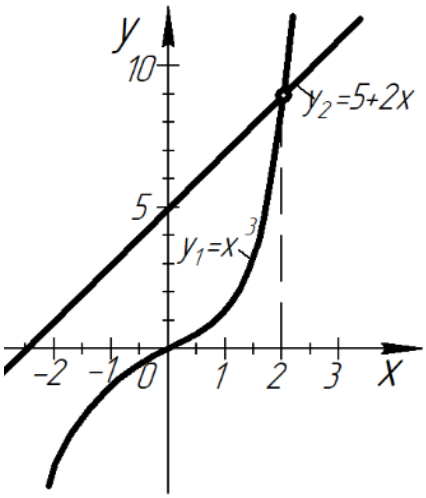


Figure 2.10 – Diagram illustrating the graphic determination of the interval of ownership of the roots of the equation

To begin with, it is advisable to present equation (2.19) in the form $y_1=x^3$, $y_2=5+2x$. Then the graph in Figure 2.10 allows you to choose two initial values necessary for the chord method: x_1=2,2 and x_2=2,0. Using the formula (2.17), we determine the root step by step with the required accuracy. At the same time, x_1=2.200000 has a fixed value, and x_2 is refined in each subsequent step. The numbers x_1 and $f(x_1)$=1.248, which are used in all subsequent steps, can be considered exact and selected with as many decimal places as necessary in this step. All the necessary calculations are given in table 2.2, according to which the root of the equation $x = 2{,}09455$ is determined with the specified accuracy.

Table 2.2 – Results of solving the equation

n	$x=x_n$	x_n^3	$f(x_n)$	$f(x_i) - f(x_{i-1})$
0	2,200000	10,648000	1,248000	–
1	2,000000	8,000000	– 1,000000	2,248000
2	2,090000	9,129000	– 0,051000	1,299000
3	2,094300	9,185790	– 0,002810	1,250810
4	2,094530	9,188820	– 0,000240	1,248240
5	2,094550	9,189084	– 0,000016	–

Consider the implementation of secant method (linear interpolation) in the PYTHON programming language:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import *
from sympy import *
```

```
x = np.arange(-5, 4, 0.1)
# we define a function
func_x=(x**3)-(2*x)-5
# we construct a graph and localize the roots of the equation
plt.figure(figsize=(7, 7))
plt.xlabel('x',fontsize=15, color='blue')
plt.ylabel('y',fontsize=15, color='blue')
plt.plot(x, func_x)
plt.legend(['f(x)=X^3-2x-5'], loc=1)
plt.grid(True)
plt.xlim([-2, 4])
plt.ylim([-7, 4])
plt.show()
# we set the function itself
def f(x):
    return (x**3)-(2*x)-5
from timeit import default_timer as timer
# we set the initial limits of localization of the root of the equation
x_n=1; x_nm=2; e=0.000001
# definition of the iterative formula of the chord method
def x_np(x_n, x_nm):
    k=(f(x_n)-f(x_nm))/(x_n-x_nm)
    return x_n-(f(x_n)/k)
k=0
# we determine the root of the equation
start= timer()
while abs(x_np(x_n, x_nm)-x_n)>=e:
    k=k+1
    x_n=x_np(x_n, x_nm)
    if f(x_n)*f(x_np(x_n, x_nm))>0:
        x_nm=x_np(x_n, x_nm)
    else:
        x_n=x_np(x_n, x_nm)
end = timer()
print("x =", (x_n+x_np(x_n, x_nm))/2,"Time taken:", end-start, "Number of
     iterations:", k).
```

Fixed-point iteration

The algebraic or transcendental equation $f(x)=0$ can be reduced to this form:

$$x = \varphi(x),$$

which can be done in different ways and obtain different expressions for the function $\varphi(x)$.

For the approximate value of the root x_0, a more accurate result is determined using the formula $x_1 = \varphi(x_0)$ or in a more general form:

$$x_{n+1} = \varphi(x_n) \quad (n = 0, 1, 2, \ldots). \tag{2.20}$$

Repeating this process, that is, integrating several times, it is possible to obtain the value of the root with any degree of accuracy, if a sufficient condition is met:

$$\left|\varphi'(x)\right| < 1 \text{ on the segment } [\xi; x_0], \tag{2.21}$$

where $x = \xi$ – the exact value of the root.

If condition (2.21) is not fulfilled, then the equation $f(x) = 0$ can always be

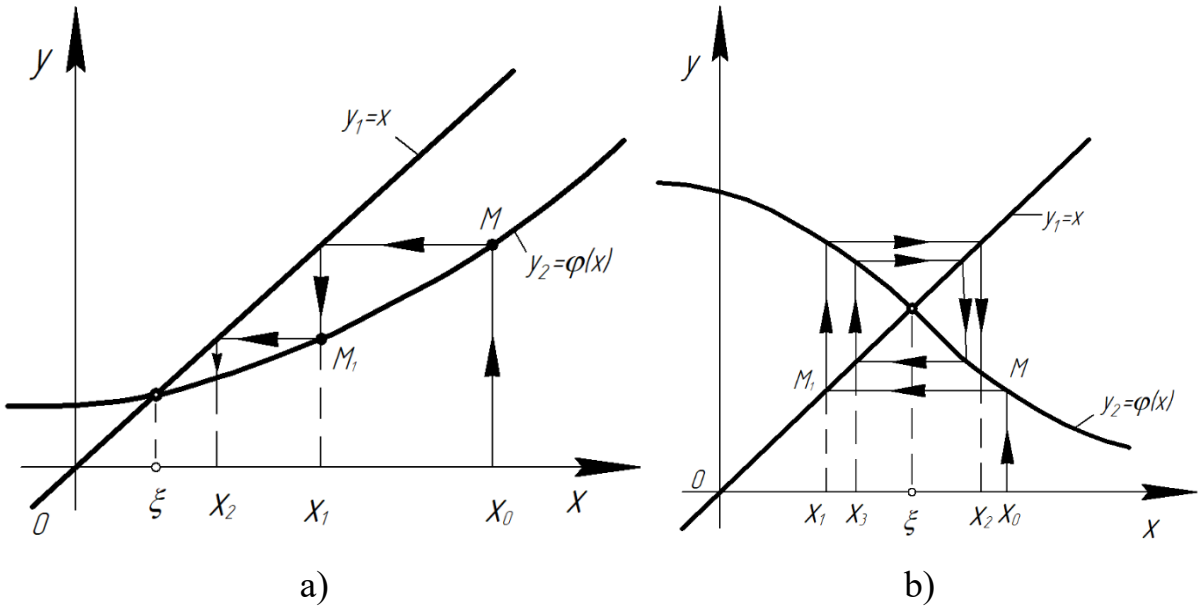
written as $x = x–c\cdot f(x)$, and the constant c can be chosen in such a way that condition (2.21) holds for the function $\varphi(x) = x–c\cdot f(x)$. Then, using formula (2.20), we obtain:

$$x_{n+1} = x_n - cf(x). \qquad (2.22)$$

The fixed-point iteration makes it possible to «guess» new values of x_n during any step n, which is equivalent to starting iterations with a new, more successful value of x_0. Accordingly, in cases where the process converges slowly, appropriate adjustments can be made, taking into account the results of the previous steps.

The mentioned properties of the method of iterations, its simplicity, and unlimited possibilities laid down in formula (2.22), made it possible to use it effectively when solving differential, integral, and integro-differential equations.

The geometric content of the solution to the equation $x = \varphi(x)$ using the fixed-point iteration is shown in Figure 2.11.



a) b)

Figure 2.11 – Scheme for determining the roots of the equation using the method of simple iterations:
a) $0 < \varphi'(x) < 1$; b) $-1 < \varphi'(x) < 0$

Figure 2.11, a) plots the functions $y = \varphi(x)$ and $y = x$. The root of the equation is the x-coordinate of the point of intersection of the curve $y = \varphi(x)$ with the bisector of the coordinate angle. If x_0 is an initial approximation of the root, then $x_1 = \varphi(x_0)$ is equal to the y-coordinate of the corresponding point M on the curve or the x-coordinate of the point M_1. The following approximations are determined in a similar manner (see Fig. 2.11, a). The role of the $|\varphi'(x)| < 1$ condition can also be established.

Figure 2.11, a) depicts the case when $0<\varphi'(x)<1$, so that the curve intersects the bisector from left to right and the right side lies under the bisector. In this case, the iterative process converges, and the approximations decrease monotonically if $x_0 > \overline{x}$ or increase monotonically for $x_0 < \overline{x}$. Figure 2.11, b) shows the case when the derivative $\varphi'(x)$ is negative ($-1 < \varphi'(x) < 0$). If at the same time $|\varphi'(x)| < 1$, then the iterative process converges, but the approximations fluctuate around the true value of the root.

To determine the loss of accuracy in the fixed-point iteration, it is necessary to analyze the effect of rounding errors on the final results of intermediate calculations.

In particular, when solving the equation using the fixed-point iteration for two successive approximations, the following relationship can be established:

$$x_{n+1} - \xi \approx \varphi'(\xi)(x_n - \xi), \tag{2.23}$$

where ξ – the exact value of the sought root.

Thus, the accuracy of the result in the process of determining the root by the fixed-point iteration increases approximately like a geometric progression with the denominator $\varphi'(\xi)$.

The algorithm for the fixed-point iteration is shown in Figure 2.12.

Example 2.6. Determine the value of the roots of the equation:

$$4x - 5\ln x = 5 \tag{2.24}$$

with precision ε=0,00001.

Solution:

Having written the equation in the form $\ln x = \frac{4x-5}{5}$, the zero approximation is determined graphically (Fig. 2.13), finding the intersection of the logarithmic curve y_2 with the straight line $y_1 = \frac{4}{5}x - 1$. From Figure 2.13, two approximate root values are determined: $x_0 = 2{,}28$ and $x_0 = 0.57$, which will be taken as the initial approximation.

For a more accurate search for the correct root, equation (2.24) is written in the form $x = 1{,}25(1+\ln x)$, where $\varphi(x) = 1{,}25(1+\ln x)$. The iterative process is convergent, as the value of the derivative function $\varphi'(x) = \frac{1{,}25}{x}$ in the vicinity of the correct root is positive and less than unity under condition (2.21). The calculation is presented in Table 2.3.

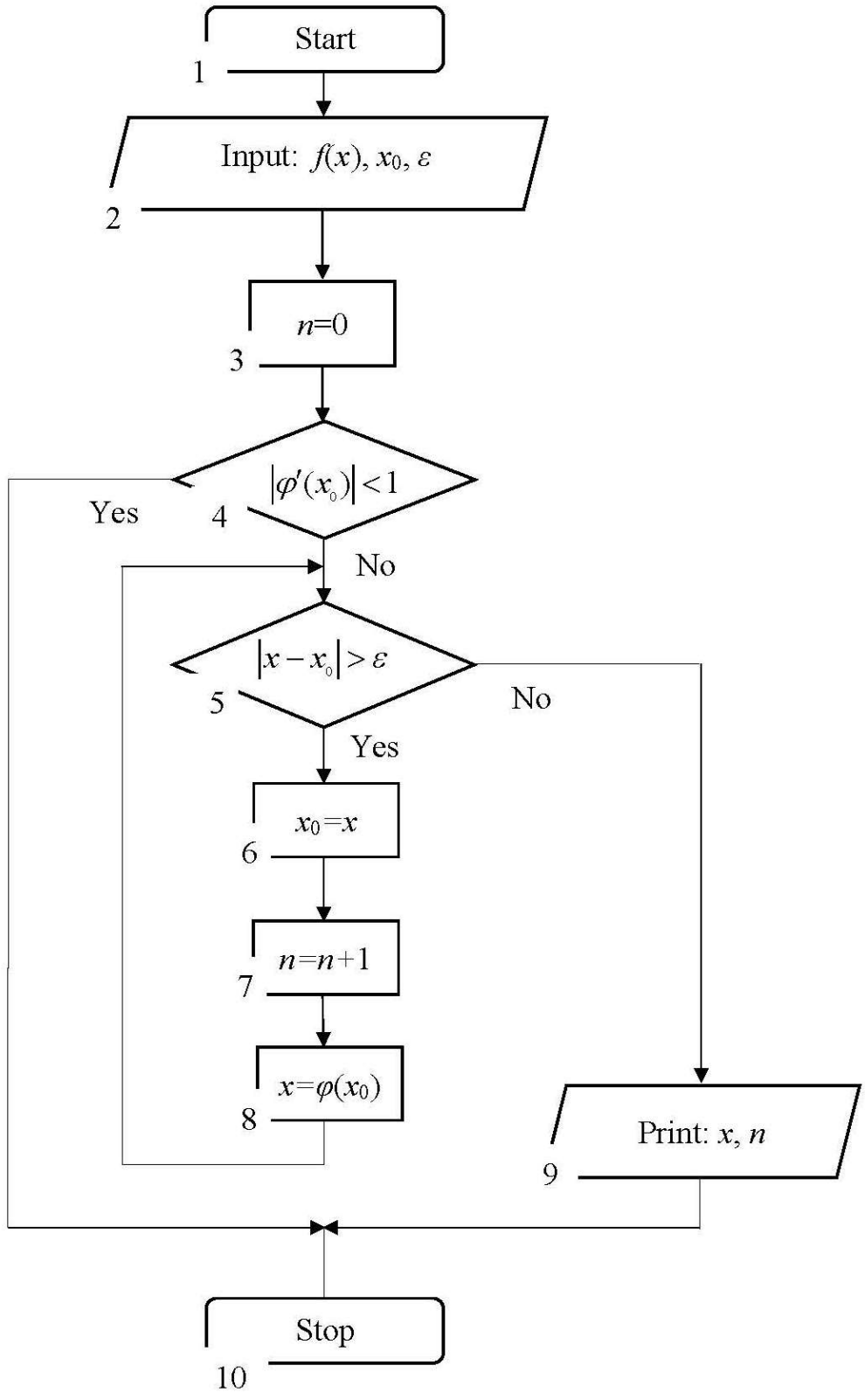


Figure 2.12 – Scheme of the algorithm for the Fixed-point iteration

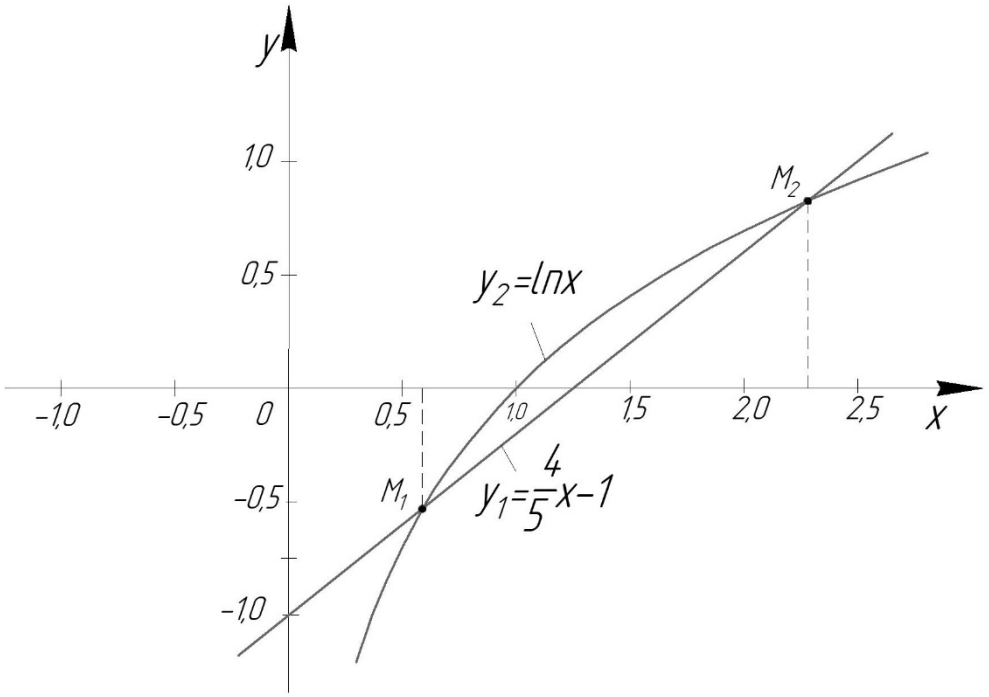


Figure 2.13 – Diagram illustrating the graphical determination of the interval properties of the roots of the equation

Table 2.3 – Results of solving the equation

n	x_n	$\ln(x_n)+1$	$1{,}25(\ln(x_n)+1)$
0	2,28000	1,82418	2,28022
1	2,28022	1,82427	2,28034
2	2,28034	1,82432	2,28040
3	2,28040	1,82435	2,28044
4	2,28044	1,82437	2,28046
5	2,28046	1,82438	2,28048
6	2,28048	1,82439	2,28049
7	2,28049	1,82439	2,28049

Eight steps were taken to find the root of the equation to five digits. The fast convergence is due to the small value of the derivative around the root of 0.55, as well as the successful choice of the initial approximation.

In the process of finding the left root, the iterative process is divergent because $\varphi'(x) = 1{,}25/x$ in the region $x = 0{,}57$ has a value of about 2.2. Therefore, the initial equation (2.24) must be rewritten in the form:

$$x = e^{0,8x-1}, \tag{2.25}$$

then $\varphi(x) = e^{0,8x-1}$ and $\varphi'(x) = 0{,}8e^{0,8x-1}$.

For $x_0 = 0.57$, the value of the function $\varphi'(x) \approx 0{,}46 < 1$ and the iterative process converges. As shown in Table 2.4, it takes eleven steps to find the root

with an accuracy of five decimal places. In this case, the process converges more slowly compared to the previous case, despite the smaller value of the derivative. The reason for this slowdown is a less accurate choice of the initial approximation compared to the previous case. In the first case, the initial approximation differed from the true root by a value on the order of 10^{-4}, while in the second case, it differed by a value on the order of 10^{-2}.

Table 2.4 – Results of solving the equation

n	x_n	$0{,}8x_n$	$0{,}8x_n$-1	$e^{0,8x_n-1}$
0	0,57000	0,45600	– 0,55500	0,58042
1	0,58042	0,46434	– 0,53566	0,58528
2	0,58528	0,46822	– 0,53178	0,58756
3	0,58756	0,47005	– 0,52995	0,58863
4	0,58863	0,47090	– 0,52910	0,58913
5	0,58913	0,47130	– 0,52870	0,58937
6	0,58937	0,47150	– 0,52850	0,58949
7	0,58949	0,47159	– 0,52841	0,58954
8	0,58954	0,47163	– 0,52837	0,58957
9	0,58957	0,47166	– 0,52834	0,58958
10	0,58958	0,47166	– 0,52834	0,58958

Consider the implementation of the fixed-point iteration in the PYTHON programming language:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import *
from sympy import *
x = np.arange(-5, 4, 0.1)
from numpy import log as ln
# we define a function
func_x=(4*x)-(5*ln(x))-5
# we construct a graph and localize the roots of the equation
plt.figure(figsize=(7, 7))
plt.xlabel('x',fontsize=15, color='blue')
plt.ylabel('y',fontsize=15, color='blue')
plt.plot(x, func_x)
plt.legend(['f(x)=(4*x)-(5*ln(x))-5'], loc=1)
plt.grid(True)
plt.xlim([-2, 4])
plt.ylim([-2, 4])
plt.show()
# the first form of expressing a function in terms of x
def f_first(x):
    return 1.25*(1+ln(x))
# the second form of expressing the function in terms of x
def f_second(x):
    return math.exp((0.8*x)-1)
from sympy import *
# we determine the derivatives of the functions through which the desired #
argument x can be expressed
```

```
x_arg = Symbol('x_arg')
func_first=1.25*(1+ln(x_arg))
func_second=exp((0.8*x_arg)-1)
d_first=func_first.diff(x_arg)
d_second=func_second.diff(x_arg)
print("Derivative of the 1st function f1'(x)=",d_first,". Derivative of the
     2nd function f2'(x)=",d_second)
from timeit import default_timer as timer
#we set the point of zero approximation to determine the root of the equation
x[0]=2.2; x[1]=0.55; e=0.00001; k_1=0; k_2=0
func_first = lambdify(x_arg, d_first)
func_second = lambdify(x_arg, d_second)
for i in range(0,2):
    if abs(func_first(x[i]))<1:
        # we determine the root of the equation
        start= timer()
        x_first=x[i]
        while abs(f_first(x_first)-x_first)>e:
            k_1=k_1+1
            x_first=f_first(x_first)
        end = timer()
        time_1=end-start
    if abs(func_second(x[i]))<1:
        x_second=x[i]
        # we determine the root of the equation
        start= timer()
        while abs(f_second(x_second)-x_second)>e:
            k_2=k_2+1
            x_second=f_second(x_second)
        end = timer()
        time_2=end-start
print("x1 =", x_first, "Time taken for x1:",time_1, "Number of iterations
     for x1:",k_1)
print("x2 =", x_second, "Time taken for x2=:",time_2, "Number of iterations
for x2:",k_2).
```

2.3 Method of determining complex roots

To determine complex roots, the same methods as for real roots can be used, but the arithmetic of complex numbers is used. Convergence and error control are carried out by the modulus of the complex number, which may not always be convenient for the user.

There are a number of special methods that allow you to evaluate complex roots by performing calculations with real numbers. One of the well-known approaches is Lin's method, which is based on the transformation of the initial algebraic equation $P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 = 0$ into a product of quadratic coefficients of the type $x^2 + px + q$, namely:

$$P_n(x) = (x^2 + px + q)Q_{n-2}(x) + R(x), \qquad (2.26)$$

where $Q_{n-2}(x) = b_{n-2}x^{n-2} + b_{n-3}x^{n-3} + \ldots + b_{n-2-j}x^{n-2-j} + \ldots + b_1 x + b_0$ ($j = 0,\ 1,\ 2,\ \ldots,\ n–2$), and the residual term $R(x)$ is zero.

If the polynomial $P_n(x)$ contains complex roots, then they can be expressed in terms of coefficients p and q, namely:

$$x_{1,2} = \alpha \pm i\beta \ (\alpha = -0{,}5p; \quad \beta = \sqrt{q - 0{,}25p^2}\,)).$$

To separate the real and complex roots of a polynomial equation:

$$P_n(x) \equiv x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 = 0, \qquad (2.27)$$

the presence and number of roots are determined in advance using such theorems.

Theorem 2.3 (on the number of roots of an algebraic equation). The algebraic equation (2.27) of the n-th degree has n roots (real or complex), provided that each root is counted as many times as its multiplicity.

Theorem 2.4 (on the even conjugation property of complex roots of equation (2.43)). If $x_{*i}=\alpha+i\beta$ is a root of the algebraic equation (2.27) with multiplicity k, then the number $\overline{x_{*_i}} = \alpha - i\beta$ is also a root of the same multiplicity. A consequence of this theorem is that an algebraic equation of odd degree has at least one real root.

Theorem 2.5 (Descartes' rule of signs on the number of real roots of algebraic equations).

The number S_1, which represents the number of positive roots (accounting for multiplicity) of the algebraic equation $P_n(x)=0$, is equal to the number of sign changes in the sequence of coefficients a_n, a_{n-1}, …, a_0 (excluding coefficients that are equal to zero) of the polynomial $P_n(x)=0$. Alternatively, it can be less than this number by an even number. The number S_2, on the other hand, denotes the number of negative roots (taking into account their multiplicity) of the algebraic equation $P_n(x)=0$. It is also determined by the number of sign changes in the sequence of coefficients a_n, a_{n-1}, …, a_0 (excluding coefficients that are equal to zero) of the polynomial $P_n(-x)=0$. Similarly, it could be less than this number by an even number.

Theorem 2.6 (De Gua's theorem on the necessary condition for the validity of all roots of an algebraic equation). If the algebraic equation (2.27) has all real roots, then the square of each nonextreme coefficient is greater than the product of its two adjacent coefficients. A consequence of this theorem is that if the inequality $a_k^2 \le a_{k-1}a_{k+1}$ holds for any k, then equation (2.27) has at least one pair of complex roots.

Opening the brackets in equation (2.26) and equating the coefficients with the same powers of x, we obtain:

$$\begin{cases} b_{n-2} = a_n, \\ b_{n-3} = a_{n-1} - pb_{n-2}, \\ b_{j-2} = a_j - pb_{j-1} - qb_j, \\ qb_1 + b_0 p = a_1, \\ qb_0 = a_0, \\ j = n-2, n-3, \ldots, 2. \end{cases} \qquad (2.28)$$

Solving the system of linear equations (2.28), determine the values of the quadratic equation p and q, as well as the coefficients of the polynomial b, which now has an order two lower than the original ($b_1 = b_0 = 0$). The search for the roots of the quadratic equation $x^2+px+q = 0$ is also carried out according to the classical scheme.

Lin's method uses the running method or simple iteration method to find a solution to the system of equations (2.28).

Given the initial values of p^0 and q^0, the $b_{n-1}^0, b_{n-2}^0, \ldots, b_2^0$ values, as well as the values of p^1 and q^1, are determined sequentially. Choosing the values of p and q as the next approximation, repeat the procedure of finding b, p, and q until convergence. Figure 2.14 shows the scheme of the algorithm for finding complex roots of an algebraic equation of the n-th order using Lin's method.

It should also be noted that the method of simple iterations is conditionally convergent, but there are no practically acceptable criteria for the convergence of this method. During the practical implementation of Lin's method, it is necessary to monitor the number of iterations. If it goes beyond a reasonable limit, it should be considered that the method is divergent for the selected algebraic equation.

Example 2.7. Determine the value of the complex roots of the algebraic equation:

$$x^4 - 2x^3 + 18x^2 + 3x - 5 = 0, \tag{2.29}$$

with precision $\varepsilon = 0{,}0001$.

Solution:

To begin with, it is necessary to determine the presence of real and complex roots of the algebraic equation (2.29). According to Theorem 2.3, the algebraic equation (2.29) has four roots, since the degree of the polynomial is n=4. Given that $a_k^2 \le a_{k-1}a_{k+1}$, where $k = 2$, $a_1 = 1$, $a_2 = -2$ and $a_3 = 18$, according to Theorem 2.6, the algebraic equation (2.29) has one pair of complex roots.

We write the algebraic equation (2.29) in the form (2.26):

$$x^4 - 2x^3 + 18x^2 + 3x - 5 = \left(x^2 + px + q\right)\left(x^2 + b_1x^2 + b_0\right) = 0. \tag{2.30}$$

Opening the brackets in equation (2.30) and equating the coefficients with the same powers of x, we obtain:

$$\begin{cases} b_1 = a_3 - p; \\ b_0 = a_2 - pb_1 - q; \\ qb_1 + b_0p = a_1; \\ qb_0 = a_0, \end{cases} \tag{2.31}$$

where $a_0 = -5$, a_1=3, a_2=18 and $a_3 = -2$.

For the initial approximation values $p = 1$ and $q = 1$, the iterative solution of the system of equations (2.31) is performed. The results of solving the system of equations (2.31) by iterations are presented in Table 2.5.

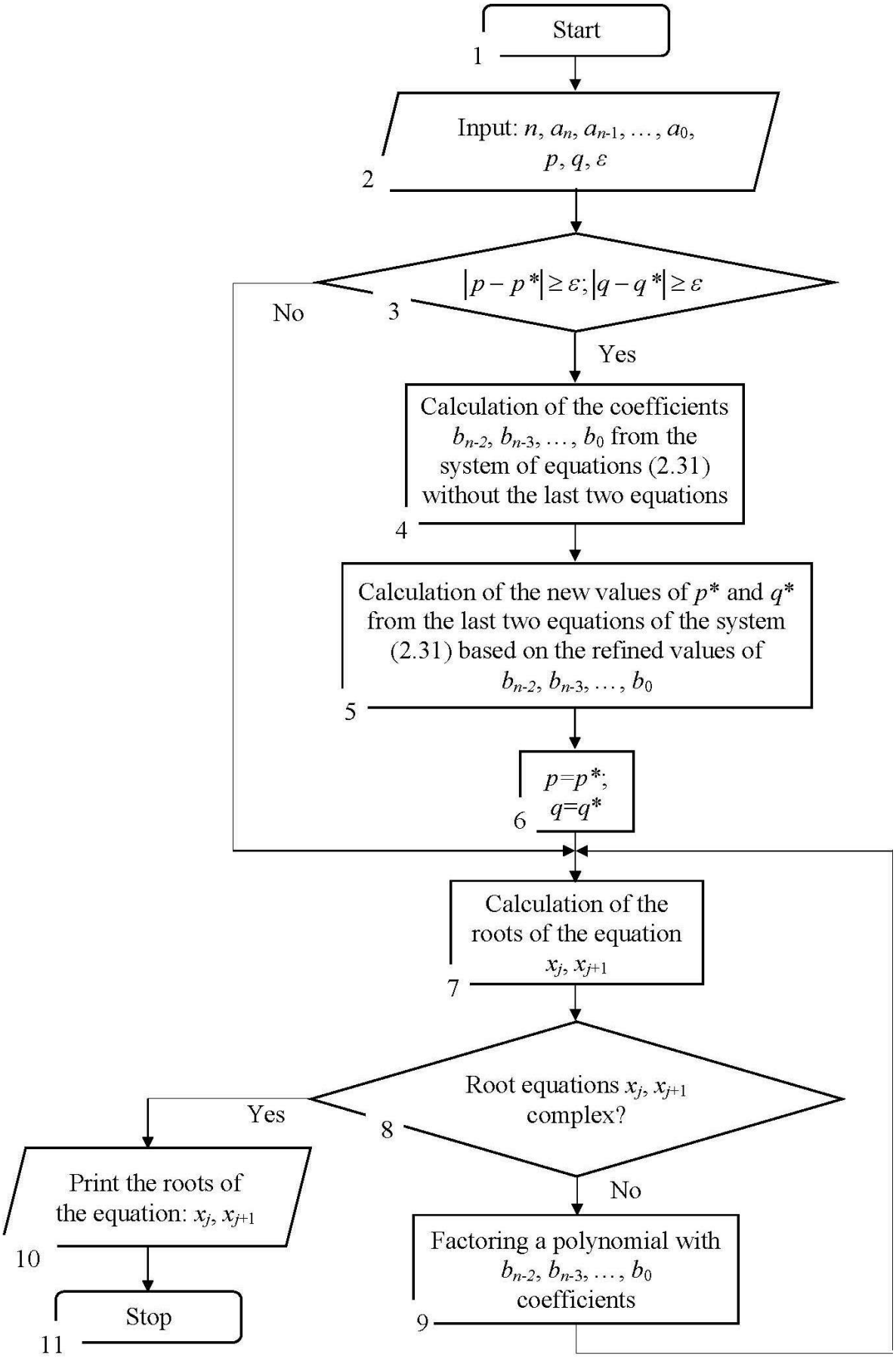


Figure 2.14 – Scheme of the Lin's method algorithm

Table 2.5 – Results of the iterative solution of the system of equations

n	p	q	b_0	b_1	b_2
0	1,00000	1,00000	24,0000	– 4,00000	1,0
1	0,45833	– 0,20833	19,33507	– 2,45833	1,0
2	0,12867	– 0,25860	18,52349	– 2,12867	1,0
3	0,13217	– 0,26980	18,55161	– 2,13070	1,0
4	0,13070	– 0,26957	18,54801	– 2,13070	1,0
5	0,13078	– 0,26957	18,54800	– 2,13070	1,0
6	0,13077	– 0,26956	18,54824	– 2,13078	1,0

Based on the results of the iterative solution of the system of equations (2.31), the decomposition of the polynomial (2.29) into the following quadratic factors was obtained:

$$\begin{aligned} x^4 - 2x^3 + 18x^2 + 3x - 5 &= \left(x^2 + px + q\right)\left(x^2 + b_1x^2 + b_0\right) = \\ &= \left(x^2 + 0{,}13077x - 0{,}26956\right)\left(x^2 - 2{,}13078x^2 + 18{,}54824\right) = 0. \end{aligned} \quad (2.32)$$

The result of solving the quadratic equation (2.32) is the real and complex roots of equation (2.29):

$$x_1 = 0{,}45791; \quad x_2 = -0{,}58868; \quad x_3 = 1{,}06539 \pm i \cdot 4{,}17291.$$

Consider the implementation of Lin's method in the PYTHON programming language:

```
n=4 # the value of the power of the polynomial
a=[-5, 3, 18, -2, 1] # the value of the coefficients of the polynomial
# general form of the polynomial
print(' It is necessary to determine the complex roots of an algebraic
     equation of the form:')
print('({0}*x^{1})'.format(a[n],n),end='')
for j in range(n-1,0,-1):
    print('+({0}*x^{1})'.format(a[j],j),end='')
print('+({0})=0'.format(a[0]))
# we determine the conditions for the presence of complex roots of the
#equation
for k in range(1,n):
    if (a[k]^2)<=a[k-1]*a[k+1]:
        print('An algebraic equation has at least one pair of complex
      roots ')
        break
p=1; q=1 # Initial approximation values for numbers p and q
e=0.00001 # The value of the degree of accuracy of the calculation of the #
complex roots of the equation
b=[0]*(n+1)
p_new=p+1; q_new=q+1
reg=0
while (abs(p_new-p)>=e) and (abs(q_new-q)>=e):
    p=p_new; q=q_new
    for i in range(n-2,-1,-1):
        b[i]=a[i+2]-(p*b[i+1])-(q*b[i+2])
    reg=reg+1
```

```
    print(' The value of the coefficients of the polynomial factor (x^2+px+q)
     on', reg, 'iterations-', b)
    q_new=a[0]/b[0]
    p_new=(a[1]/b[0])-((b[1]/b[0])*q)
    print(' The value of the coefficients of the multiplier p_new=', p_new,
     'q_new=', q_new)
    print('-------------------------------------------------------------------
     ----------------------')
# Checking for complex roots of the quadratic factor of the polynomial #
equation (x^2+px+q)
import math
discr=(p_new*p_new)-(4*q_new)
if discr<0:
    alfa=-0.5*p_new
    bet=math.sqrt(abs(discr))
    print(' The value of the complex roots of the equation:')
    print('x1={0}+i*{1}'.format(alfa,bet))
    print('x2=({0})-i*({1})'.format(alfa,bet))
else:
    print(' The value of the real roots of the equation:')
    print('x1=',(-0.5*p_new)+(0.5*math.sqrt(discr)))
    print('x2=',(-0.5*p_new)-(0.5*math.sqrt(discr)))
# determination of complex roots of the equation from the residual factor of
# the polynomial (x^2+px+q)
c=[]
for cell in b:
    if cell!=0: c.append(cell)
print (c)
discr_2=(c[1]*c[1])-(4*c[0]*c[2])
if discr_2<0:
    alfa=(-0.5*c[1])/c[2]
    bet=0.5*math.sqrt(abs(discr_2))/c[2]
    print(' The value of the complex roots of the equation:')
    print('x3={0}+i*{1}'.format(alfa,bet))
    print('x4=({0})-i*({1})'.format(alfa,bet))
else:
    print(' The value of the real roots of the equation:')
    print('x3=',(alfa)+(0.5*math.sqrt(discr_2)/c[2]))
    print('x4=',(alfa)-(0.5*math.sqrt(discr_2)/c[2])).
```

2.4 Numerical methods for solving systems of nonlinear algebraic equations

In the general case, a system of n nonlinear equations with n unknowns is given in the form:

$$\begin{cases} f_1(x_1, x_2, ..., x_n) = 0; \\ f_2(x_1, x_2, ..., x_n) = 0; \\ \vdots \\ f_n(x_1, x_2, ..., x_n) = 0. \end{cases} \tag{2.33}$$

Since the nonlinear functions included in the system (2.33) cannot be described by any specific general form, no analytical direct method can be proposed for solving such a system. Of the approximate iterative methods, the simplest is the simple iteration method, which is based on reducing the system (2.33) to a system of nonlinear equations in the form:

$$\begin{cases} x_1 = g_1(x_1, x_2, ..., x_n), \\ x_2 = g_2(x_1, x_2, ..., x_n), \\ ..... \\ x_n = g_n(x_1, x_2, ..., x_n). \end{cases} \tag{2.34}$$

Or in matrix form:

$$\boldsymbol{X}=\boldsymbol{G}(\boldsymbol{X}), \tag{2.35}$$

where $\boldsymbol{G}(\boldsymbol{X}) = \begin{bmatrix} g_1(x_1, x_2, ..., x_n), \\ g_2(x_1, x_2, ..., x_n), \\ ..... \\ g_n(x_1, x_2, ..., x_n) \end{bmatrix}$.

Next, an algorithm similar to the Gauss-Seidel method for systems of linear equations can be applied. It is based on iterative equations connecting (m+1) and m iterations:

$$\begin{cases} x_1^{(m+1)} = g_1\left(x_1^{(m)}, x_2^{(m)}, ..., x_n^{(m)}\right), \\ x_2^{(m+1)} = g_2\left(x_1^{(m+1)}, x_2^{(m)}, ..., x_n^{(m)}\right), \\ \vdots \\ x_n^{(m+1)} = g_n\left(x_1^{(m+1)}, x_2^{(m+1)}, ..., x_n^{(m)}\right). \end{cases} \tag{2.36}$$

For this method, it is very difficult to ensure convergence, and the convergence interval can be so narrow that the selection of initial approximations becomes much more challenging.

In the general case, this method will match if $\|\boldsymbol{G}'(\boldsymbol{X})\| < 1$, where $\|\boldsymbol{G}'(\boldsymbol{X})\|$ is the norm of the matrix of partial derivative functions with respect to the variables $x_1, x_2, \ldots, x_n$:

$$\boldsymbol{G}'(\boldsymbol{X}) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial g_n}{\partial x_1} & \frac{\partial g_n}{\partial x_2} & \cdots & \frac{\partial g_n}{\partial x_n} \end{bmatrix}. \tag{2.37}$$

A more stable method – Newton's method – has become widely used for solving systems of nonlinear equations. It is an analogue of Newton's method for one equation and is based on the expansion of all equations into a Taylor series:

$$\begin{cases} f_1\left(x_1+\Delta x_1,...,x_n+\Delta x_n\right)=f_1\left(x_1,...,x_n\right)+\Delta x_1\dfrac{\partial f_1}{\partial x_1}+...+\Delta x_n\dfrac{\partial f_1}{\partial x_n}+R_n; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\ f_n\left(x_1+\Delta x_1,...,x_n+\Delta x_n\right)=f_n\left(x_1,...,x_n\right)+\Delta x_1\dfrac{\partial f_n}{\partial x_1}+...+\Delta x_n\dfrac{\partial f_n}{\partial x_n}+R_n, \end{cases}$$

where R_n – members of the second and higher orders, who are subsequently rejected.

The problem boils down to solving a system of linear equations:

$$\begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \ldots & \dfrac{\partial f_1}{\partial x_n} \\ \ldots & \ldots & \ldots \\ \dfrac{\partial f_n}{\partial x_1} & \ldots & \dfrac{\partial f_n}{\partial x_n} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_n \end{bmatrix} = \begin{bmatrix} -f_1 \\ -f_2 \\ \vdots \\ -f_n \end{bmatrix}. \tag{2.38}$$

In this system, the matrix of partial derivatives is called the Jacobian matrix and denoted by $\boldsymbol{W}(\boldsymbol{X})$.

The values of Δx_i found for a certain $(m+1)$ iteration step are used as corrections to the previous approximations:

$$\begin{cases} x_1^{(m+1)}=x_1^{(m)}+\Delta x_1; \\ \ldots\ldots\ldots\ldots\ldots; \\ x_n^{(m+1)}=x_n^{(m)}+\Delta x_n. \end{cases} \tag{2.39}$$

The general iterative formula in matrix representation looks like this:

$$\boldsymbol{X}^{(m+1)}=\boldsymbol{X}^{(m)}-\boldsymbol{W}^{-1}\left[\boldsymbol{X}^{(m)}\right]\boldsymbol{F}\left[\boldsymbol{X}^{(m)}\right], \tag{2.40}$$

where $\boldsymbol{F}\left[\boldsymbol{X}^{(m)}\right]$ – column vector of function values $f_1, f_2, \ldots, f_n$ for approximations $\boldsymbol{X}^{(m)}$, $\boldsymbol{W}^{-1}\left[\boldsymbol{X}^{(m)}\right]$ – inverse Jacobi matrix.

Certain difficulties arise in the process of implementing Newton's method algorithm, particularly during the inversion of the Jacobi matrix. To address this, matrix inversion methods known from linear algebra are employed.

There are also many options for applying Newton's method. For example, a modified Newton method:

$$\boldsymbol{X}^{(m+1)}=\boldsymbol{X}^{(m)}-\boldsymbol{W}^{-1}\left[\boldsymbol{X}^{(0)}\right]\boldsymbol{F}\left[\boldsymbol{X}^{(m)}\right]. \tag{2.41}$$

In this method, it is not necessary to calculate the inverse Jacobi matrix at each step of the calculation, which simplifies the algorithm but slows down the convergence and makes the method more sensitive to the choice of initial approximation.

There is also Newton's method with the parameter τ:

$$\boldsymbol{X}^{(m+1)} = \boldsymbol{X}^{(m)} - \tau \boldsymbol{W}^{-1}\left[\boldsymbol{X}^{(m)}\right]\boldsymbol{F}\left[\boldsymbol{X}^{(m)}\right]. \tag{2.42}$$

This method is somewhat similar to the method of successive over-relaxation for systems of linear equations. A variety of hybrid methods are also used, in which Newton's method is combined with the method of simple iteration.

The convergence of Newton's method is evaluated by calculating the exponent:

$$q = \frac{M^2 LP}{2} < 1, \tag{2.43}$$

where $M \geq \left\|\boldsymbol{W}^{-1}(\boldsymbol{X})\right\|$, $L \geq \left\|\boldsymbol{W}(\boldsymbol{X})\right\|$, $P \geq \left\|\boldsymbol{F}(\boldsymbol{X})\right\|$, moreover $\lim\limits_{l\to\infty} MP \sum\limits_{m=0}^{l} q^{2^m-1} \to 0$

.

The error at the mth iteration is determined by the inequality:

$$\Delta \leq MP\frac{q^{2^m-1}}{1-q^{2^m}}. \tag{2.44}$$

The algorithm of the method is shown in the Figure 2.15.

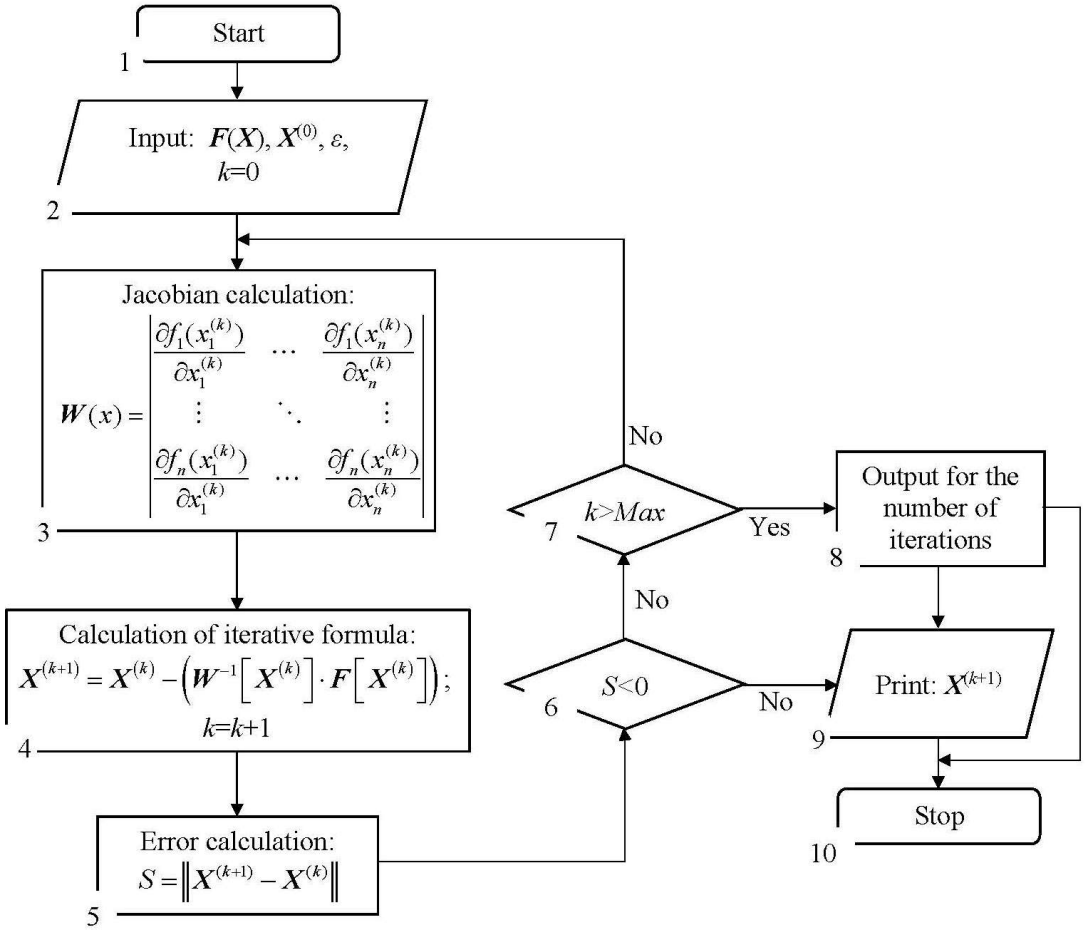


Figure 2.15 – Scheme of the Newton's solution method algorithm for systems of nonlinear equations

Example 2.8. Determine the values of the roots of the system of nonlinear equations:

$$\begin{cases} x^2 + y^2 + z^2 = 1; \\ 2x^2 + y^2 - 4z^2 = 0; \\ 3x^2 - 4y + z^2 = 0. \end{cases} \tag{2.45}$$

with precision $\varepsilon = 0{,}001$.

Solution:

Enter the notation:

$$\boldsymbol{X} = \begin{vmatrix} x \\ y \\ z \end{vmatrix}; \quad \boldsymbol{F}(\boldsymbol{X}) = \begin{vmatrix} f_1 \\ f_2 \\ f_3 \end{vmatrix} = \begin{vmatrix} x^2 + y^2 + z^2 - 1 \\ 2x^2 + y^2 - 4z^2 \\ 3x^2 - 4y + z^2 \end{vmatrix};$$

$$\boldsymbol{W}(\boldsymbol{X}) = \begin{vmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} \\ \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial z} \end{vmatrix} = \begin{vmatrix} 2x & 2y & 2z \\ 4x & 2y & -4 \\ 6x & -4 & 2z \end{vmatrix}.$$

An initial approximation is selected:

$$\boldsymbol{X}^0 = \begin{vmatrix} x_0 \\ y_0 \\ z_0 \end{vmatrix} = \begin{vmatrix} 0{,}5 \\ 0{,}5 \\ 0{,}5 \end{vmatrix}; \quad \boldsymbol{F}(\boldsymbol{X}^{(0)}) = \begin{vmatrix} f_1^{(0)} \\ f_2^{(0)} \\ f_3^{(0)} \end{vmatrix} = \begin{vmatrix} x_0^2 + y_0^2 + z^2 - 1 \\ 2x_0^2 + y_0^2 - 4z_0^2 \\ 3x_0^2 - 4y_0 + z_0^2 \end{vmatrix} = \begin{vmatrix} -0{,}25 \\ -0{,}25 \\ -1{,}00 \end{vmatrix};$$

$$\boldsymbol{W}(\boldsymbol{X}_0) = \begin{vmatrix} 2x_0 & 2y_0 & 2z_0 \\ 4x_0 & 2y_0 & -4 \\ 6x_0 & -4 & 2z_0 \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 \\ 2 & 1 & -4 \\ 3 & -4 & 1 \end{vmatrix}.$$

The value of the inverse Jacobian matrix of the function $\boldsymbol{F}(\boldsymbol{X})$ is determined:

$$\boldsymbol{W}^{-1}(\boldsymbol{X}_0) = \begin{vmatrix} 0{,}375 & 0{,}125 & 0{,}125 \\ 0{,}350 & 0{,}050 & -0{,}150 \\ 0{,}275 & -0{,}175 & 0{,}025 \end{vmatrix}.$$

Using Newton's modified method, we perform iterative calculations based on the iterative formula (2.41):

$$\boldsymbol{X}^{(1)} = \boldsymbol{X}^{(0)} - \boldsymbol{W}^{-1}\left[\boldsymbol{X}^{(0)}\right]\boldsymbol{F}\left[\boldsymbol{X}^{(0)}\right] =$$

$$= \begin{vmatrix} 0,5 \\ 0,5 \\ 0,5 \end{vmatrix} - \begin{vmatrix} 0,375 & 0,125 & 0,125 \\ 0,350 & 0,050 & -0,150 \\ 0,275 & -0,175 & 0,025 \end{vmatrix} \begin{vmatrix} -0,25 \\ -0,25 \\ -1,00 \end{vmatrix} = \begin{vmatrix} 0,750 \\ 0,450 \\ 0,550 \end{vmatrix},$$

$$\boldsymbol{F}(\boldsymbol{X}^{(1)}) = \begin{vmatrix} f_1^{(1)} \\ f_2^{(1)} \\ f_3^{(1)} \end{vmatrix} = \begin{vmatrix} 0,0675 \\ 0,1175 \\ 0,1900 \end{vmatrix};$$

$$\boldsymbol{X}^{(2)} = \boldsymbol{X}^{(1)} - \boldsymbol{W}^{-1}\left[\boldsymbol{X}^{(0)}\right]\boldsymbol{F}\left[\boldsymbol{X}^{(1)}\right] =$$

$$= \begin{vmatrix} 0,750 \\ 0,450 \\ 0,550 \end{vmatrix} - \begin{vmatrix} 0,375 & 0,125 & 0,125 \\ 0,350 & 0,050 & -0,150 \\ 0,275 & -0,175 & 0,025 \end{vmatrix} \begin{vmatrix} 0,0675 \\ 0,1175 \\ 0,1900 \end{vmatrix} = \begin{vmatrix} 0,68625 \\ 0,44900 \\ 0,54725 \end{vmatrix},$$

$$\boldsymbol{F}(\boldsymbol{X}^{(2)}) = \begin{vmatrix} f_1^{(2)} \\ f_2^{(2)} \\ f_3^{(2)} \end{vmatrix} = \begin{vmatrix} -0,0279 \\ -0,0545 \\ -0,0837 \end{vmatrix}; \ldots; \ \boldsymbol{X}^{(7)} = \begin{vmatrix} 0,70641 \\ 0,44896 \\ 0,54748 \end{vmatrix}; \boldsymbol{F}(\boldsymbol{X}^{(7)}) = \begin{vmatrix} f_1^{(3)} \\ f_2^{(3)} \\ f_3^{(3)} \end{vmatrix} = \begin{vmatrix} -0,000786 \\ -0,001571 \\ -0,002357 \end{vmatrix}.$$

At this step of the calculations, the measurement error was obtained:

$$E_x = \left\| \boldsymbol{X}^{(7)} - \boldsymbol{X}^{(6)} \right\| = \left\| \begin{vmatrix} 0,70641 \\ 0,44896 \\ 0,54748 \end{vmatrix} - \begin{vmatrix} 0,70563 \\ 0,44896 \\ 0,54748 \end{vmatrix} \right\| = \max\left(\left|0,00078; 0; 0\right|\right) = 0,00078 \le \varepsilon .$$

Consider the implementation of Newton's method for solving a system of nonlinear equations in the PYTHON programming language:

```
import numpy as np
from scipy import *
from sympy import *
# we determine the column vector of the values of the functions of the system
of nonlinear equations
def F_X(x_num,y_num,z_num):
    F_x=np.array([(x_num**2)+(y_num**2)+(z_num**2)-1,
                  (2*(x_num**2))+(y_num**2)-(4*(z_num**2)),
                  (3*(x_num**2))-(4*y_num)+(z_num**2)])
    return F_x
def Yak_Xrev(x_num,y_num,z_num):
    x,y,z=symbols('x y z')
    f_1=(x**2)+(y**2)+(z**2)-1
    f_2=(2*(x**2))+(y**2)-(4*(z**2))
    f_3=(3*(x**2))-(4*y)+(z**2)
    g_1=[diff(f_1,var) for var in [x,y,z]]
    g_2=[diff(f_2,var) for var in [x,y,z]]
```

```
    g_3=[diff(f_3,var) for var in [x,y,z]]
    g_1num = lambdify([x,y,z], g_1)
    g_2num = lambdify([x,y,z], g_2)
    g_3num = lambdify([x,y,z], g_3)
    Yak = np.array([g_1num(x_num,y_num,z_num),
                    g_2num(x_num,y_num,z_num),g_3num(x_num,y_num,z_num)])
    return np.linalg.inv(Yak)
# solution of the iterative equation
X=np.array([0.4,0.4,0.4])
X_1=np.array([0.5,0.5,0.5])
e=0.001
k=0
while np.linalg.norm(X-X_1,np.inf)>=e:
    X=X_1
    X_1=X-(Yak_Xrev(0.5,0.5,0.5).dot(F_X(X[0],X[1],X[2])))
    k=k+1
    print('k=',k)
    print('X',X)
    print('F_X(X[0],X[1],X[2])=',F_X(X[0],X[1],X[2]))
    print('X_1=',X_1)
    print('Current error value',np.linalg.norm(X-X_1,np.inf))
    print('----------------------------').
```

Conclusions on the application of methods for solving transcendental equations, systems of nonlinear equations, as well as finding complex roots of polynomial equations

To solve finite nonlinear equations, it is advisable to use the following methods: dividing in half (dichotomy), Newton (tangents), secant (linear interpolation), fixed-point iteration. Before applying these methods, the roots are separated using analytical methods such as finding a simpler equation that has roots approximately equal to the unknown roots of this equation (neglecting the small terms of the equation) and using theorems based on known properties of continuous functions. It should be noted that the method of dividing the segment in half does not require any smoothness conditions for the function; it is only necessary that the function is continuous. This makes it very popular in simple problems involving finding a single root within a given interval (for example, in the field of metrology). When calculating the roots of an equation using Newton's method, the process of successive approximations always converges if the initial approximation is taken to be very close to the root on a certain interval. Therefore, it is particularly useful for problems where the root is only roughly known. The calculation of the equation's root using the secant method with any desired level of accuracy will yield the same result as Newton's method. However, Newton's method generally exhibits better convergence than the secant method in most cases. In situations where computing the function and its derivative is time-consuming, the secant method offers greater efficiency. Along with the above-mentioned methods, the fixed-point iteration makes it possible to «guess» new values during the implementation of each step. The properties mentioned of the iteration method, its simplicity and limitless possibilities expressed in the formula, allow for its effective use in solving differential, integral, and integro-differential equations. Such tasks often arise in the process of ballistic calculations, space travel, or determining credit conditions in the financial sector.

Problems with finding complex roots of polynomial equations are relevant in the field of automatic control of complex objects. When determining complex roots, the control of convergence and error is carried out by the modulus of the complex number. In these types of problems, it is effective to use Lin's method, which allows you to estimate complex roots by calculating with real numbers. Before applying this method, the real and complex roots of the polynomial equation are also separated using the De Gua's theorem and Descartes' rule. Lin's method uses the running method or the simple iteration method to find the solution to the system of iterative equations. It should be noted that the method of simple iterations is conditionally convergent, but there are no practically acceptable criteria for the convergence of this method. In the process of practical implementation of Lin's method, it is necessary to monitor the number of iterations. If it exceeds a reasonable limit, it is necessary to consider that the method is divergent for the selected algebraic polynomial equation.

It is also common to solve problems in the dynamics of complex systems and solid state physics using a mathematical model in the form of a system of nonlinear equations. The most effective and stable method for solving systems of nonlinear equations is Newton's method, which is based on expanding all component equations into a Taylor series. Generally, the problem of solving a system of nonlinear equations is reduced to solving a system of linear equations, which is accomplished by using the inverse Jacobi matrix. To simplify the overall algorithm, a modified Newton method can be employed, which eliminates the need to calculate the value of the inverse Jacobi matrix at each step. However, a disadvantage of this approach is that it slows down convergence and becomes more sensitive to the choice of the initial approximation. Various hybrid methods, combining Newton's method with a simple iteration method, are also used.

Control questions and tasks

1. What number is called the root of a function?
2. What are the known methods for separating the roots of a function?
3. What is the condition for the existence of the roots of the equation in a certain domain of the function's definition?
4. To reveal the essence of numerical methods dividing in half (dichotomy), Newton (tangents), secant (linear interpolation), as well as simple iterations for solving nonlinear algebraic equations?
5. How to check the convergence of iterative algorithms for solving nonlinear algebraic equations?
6. What are the differences between algorithms dividing in half (dichotomy), Newton (tangents), secant (linear interpolation), fixed-point iteration?
7. Using the half-division method, determine the flight time of the projectile with an accuracy of ε=0.00001, provided that the force of air resistance to the movement of the projectile is proportional to its speed ($R= -kmv$). The equation

of motion of the projectile is given by a function in a parametric form (parameter t is the flight time of the projectile) in the projection on the vertical axis of the ordinate y:

$$y = \frac{1}{k^2}\left(g + k\upsilon_0 \sin\alpha\right)\left(1 - e^{-kt}\right) - \frac{g}{k}t,$$

where g=9,82 m/sec^2 – acceleration of gravity. The initial data for the options are presented in Table 2.6.

Table 2.6 – Output data for the task

Version	Air resistance coefficient (k, sec^{-1})	Initial velocity (υ_0, m/sec)	The angle between the initial velocity vector and the horizontal surface (α, grad)
1	0,25	1200,0	30°
2	0,33	1300,0	45°
3	0,15	1450,0	50°
4	0,42	1560,0	60°

8. Using Newton's method (tangents) determine the flight range of the projectile with an accuracy of ε=0.00001, provided that the force of air resistance to the movement of the projectile is proportional to its speed (R= –kmv). The equation of motion of the projectile is given by a function in projections on the horizontal and vertical xy axes

$$y = \frac{1}{k}\left(\frac{g + k\upsilon_0 \sin\alpha}{\upsilon_0 \cos\alpha}\right)x + \frac{g}{k^2}\ln\left(1 - \frac{k}{\upsilon_0 \cos\alpha}x\right),$$

where g=9,82 m/sec^2 – acceleration of gravity. The initial data for the options is presented in Table 2.7.

Table 2.7 – Output data for the task

Version	Air resistance coefficient (k, sec^{-1})	Initial velocity (υ_0, m/sec)	The angle between the initial velocity vector and the horizontal surface (α, grad)
5	0,25	1200,0	30°
6	0,33	1300,0	45°
7	0,15	1450,0	50°
8	0,42	1560,0	60°

9. Using the method of chords (linear interpolation) determine the interest rate and yield of bonds with an accuracy of ε = 0,0001. The yield function is given by the following transcendental equation:

$$(I+P)\cdot(1+i)^{-n}+P\cdot(1+i)^{-n+1}+A_n\cdot(1+i)-(A_n+I)=0,$$

where $n = T–N$ – the term remaining until the bond's maturity (in years); $I = P·k$ – amount of coupon payments (in monetary units). The initial data for the options is presented in Table 2.8.

Table 2.8 – Output data for the task

Version	Current value of the bond (A_n, in monetary units)	Nominal value (P, in monetary units)	Amount of years (N, amount of years)	Repayment term (T, amount of years)	Coupon interest rate (k, fraction from one)
9	1150,0	1000,0	1,0	10,0	0,12
10	1300,0	1200,0	2,0	15,0	0,14
11	1420,0	1300,0	3,0	16,0	0,16
12	1500,0	1500,0	4,0	20,0	0,18

10. Using Newton's method (tangent) determine the interest rate i with an accuracy of ε=0.0001, based on the compound interest formula:

$$S=\frac{Q}{(i/12)}\left[\left(1+\frac{i}{12}\right)^{N}-1\right].$$

The initial data for the options is presented in Table 2.9.

Table 2.9 – Output data for the task

Version	Amount of security deposit (S, in monetary units)	The period during which equal payments will be made (N, number of months)	The amount of the monthly contribution is equal (Q, in monetary units)
13	$1{,}0\cdot10^6$	9,0	$1{,}0\cdot10^5$
14	$2{,}0\cdot10^6$	10,0	$1{,}5\cdot10^5$
15	$2{,}5\cdot10^6$	11,0	$2{,}0\cdot10^5$
16	$3{,}0\cdot10^6$	12,0	$3{,}2\cdot10^5$

11. In the Landau-Ginsburg-Devonshire equation determine the value of the polarization due to the action of the external field of crystals with first-order phase transitions using the method of simple iterations, with an accuracy of ε=0,0001:

$$\alpha P+\beta P^3-\gamma P^5+E_m=0.$$

The initial data for the options are presented in Table 2.10.

Table 2.10 – Output data for the task

Version	Normalized thermodynamic parameters			The intensity of the electromagnetic field (E_m, V/m)
	α	β	γ	
17	1,0	1,00	0,01	0,0
18	1,0	0,01	0,00	1,0
19	1,0	1,50	0,10	1,2
20	1,2	1,10	0,08	1,4

12. Using the secant method (linear interpolation) with an accuracy of ε=0.00001 determine the critical force (loss of vertical balance of the rod) applied along the rod, one end of which is rigidly fixed and the other is hinged and can only move in the vertical direction. This critical force is determined from the equation:

$$tg\left(\sqrt{\frac{PL}{EI}}\right)-\sqrt{\frac{PL}{EI}}=0.$$

The initial data for the options are presented in Table 2.11.

Table 2.11 – Output data for the task

Version	Bending stiffness of the rod (EI, N·m^2)	The length of the rod (L, m)
21	450,0	3,0
22	500,0	5,0
23	560,0	7,0
24	630,0	9,0

13. Using the example of a well-known problem in cosmonautics and the fixed-point iteration determine the optimal ratio of the elements of the rocket's mass so that the maximum value CUE of the equation is reached:

$$\frac{2z}{1+z}-\ln(1+z)=0,$$

where z – the ratio of the mass of fuel for a jet engine to the mass of the payload.

14. Determining the roots of the characteristic equation allows us to find out how an increase in the gain affects the relative stability of the control system. Since the positive real parts of the roots of the system's characteristic equation correspond to the exponential development of transient processes, they should be avoided at all costs. For the regulation topic shown in Figure 2.16, it looks like:

$$12D^3+7D^2+D+K=0.$$

Find and plot the roots D_i in the plane of a complex variable depending on increasing values of K. At what value of the gain K does the system lose stability?

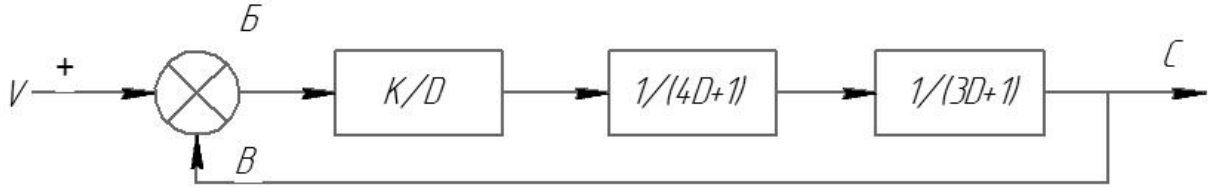


Figure 2.16 – Scheme of the system regulation system

15. What is the difference between real and complex roots of an equation?

16. Formulate theorems that determine the presence of complex roots in an equation?

17. Reveal the essence of Lin's method for determining the complex roots of a polynomial equation.

18. Describe the convergence criteria used when applying Lin's method for factoring a polynomial?

19. Calculate the real roots of a polynomial equation to ε=0,0001 precision:

$$x^5 + 5x^4 - 3x^3 + 16x^2 + 5x + 9 = 0.$$

20. In the chemical reaction:

$$CO + \frac{1}{2}O_2 \rightleftarrows CO_2$$

the percentage of X dissociated mole CO_2 is determined by the equation:

$$\left(\frac{P}{K^2 - 1}\right)X^3 + 3X - 2 = 0,$$

where P – pressure expressed in atmospheres, and K – equilibrium constant depends on temperature. Find X at K=1,648 (which corresponds to 2800,0 K) and P=1,0 atm.

21. Reveal the essence of Newton's method for solving systems of nonlinear equations.

22. Describe the principle of expanding a function into a Taylor series.

23. How the Jacobian matrix is determined?

24. How to check the convergence of the iterative algorithm for solving systems of nonlinear algebraic equations using Newton's method?

25. What are the variants of Newton's method for solving systems of nonlinear algebraic equations?

26. How is the error determined during iterative calculation using Newton's method for solving systems of nonlinear algebraic equations?

27. Define the concept of «norm» and characterize all its types in matrix calculation.

28. Calculate the roots of a system of nonlinear equations with accuracy ε=0,001:

$$\begin{cases} 2x^2 - y^2 + 9z^2 = 1; \\ x^2 + 6y^2 - z^2 = 0; \\ x^2 + 2y - 3z^2 = 4. \end{cases}$$

29. Calculate the roots of the system of nonlinear equations with an accuracy of ε=0,001 and an initial approximation:

$$\begin{cases} x_1 + 3\ln x_1 - x^2 = 0; \\ 2x_1^2 - x_1 x_2 - 5x_1 = 1, \end{cases} \quad x^{(0)} = \begin{vmatrix} 3,4 \\ 2,2 \end{vmatrix}.$$

30. Perform five iterations using Newton's method to find the root values for a system of nonlinear equations, given an initial approximation vector:

$$\begin{cases} x_2 = 2x_1^4 - 1; \\ x_2 = -5x_1^2 + 3x_1, \end{cases} \quad \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Chapter 3. DIFFERENTIAL CALCULUS PROBLEMS

Modern computer systems are widely used in all spheres of human activity. One of the main areas of application for computer systems is solving problems that arise in science and technology, specifically the construction of solutions for mathematical models that describe various physical phenomena. This field of application is known as scientific programming. Nowadays, it is almost impossible to find any field of science and technology where high-performance computing systems are not used, such as calculating the trajectories of Earth's satellites, modeling the airflow around aircraft to design more effective designs, analyzing the strength characteristics of building structures, and forecasting climatic conditions. Mathematical models for all the aforementioned problems, as well as most other scientific and engineering problems, involve systems of differential equations. Depending on the number of independent variables and, accordingly, the type of derivatives included in them, the equations are divided into ordinary differential equations, which contain one independent variable and its derivatives, and partial differential equations, which have several independent variables and their (partial) derivatives.

Ordinary differential equations (ODE) can be used to describe problems related to the motion of a system of interacting material points, kinetics, electric circuits, resistance of materials, etc. A number of important problems for partial differential equations are also reduced to problems involving ODE. For example, if a multidimensional problem allows for the separation of variables (e.g., problems involving finding the natural oscillations of elastic beams and membranes in a simpler form or determining the spectrum of eigenvalues of the energy of a particle in a spherically symmetric field), or if its solution depends only on some combination of variables (automodal decision). Thus, the solution of ODE holds a significant place among the applied problems in physics, chemistry, and engineering. Therefore, this section reveals the main methods of numerical research for ODE.

An ordinary differential equation is an equation containing unknown functions, an independent variable, and derivatives of the unknown functions (or their differentials).

General view of the ODE:

$$F\left(x, y, y', y'', \ldots, y^{(n)}\right) = 0 \text{ або } F\left(x, y, \frac{dy}{dx}, \frac{d^2y}{dx^2}, \ldots, \frac{d^ny}{dx^n}\right) = 0, \qquad (3.1)$$

where $y=y(x)$ – the function that is defined; $y^{(n)} = \frac{d^ny}{dx^n}$ – derivative (differential) of the n-th order of the function $y(x)$ with respect to the variable x; F – a valid function from its own arguments, which are also considered valid.

Differential equations can be linear or non-linear. A linear equation is an equation in which the unknown function and its derivatives appear to the first

degree. The order of a differential equation is determined by the highest derivative (or differential) present in equation (3.1). In general, the solution to (3.1) is found through n successive integrations, resulting in the general solution of *n*-th order that contains n arbitrary constants:

$$y = y\left(x, C_1, C_2, \ldots, C_n\right). \tag{3.2}$$

If the general solution of equation (3.1) is obtained implicitly:

$$\Phi\left(x, y, C_1, C_2, \ldots, C_n\right) = 0, \tag{3.3}$$

then it is called the general integral of this equation. In other words, in those cases when equality (3.3) can be solved with respect to the desired function *y*, it is called a general solution of the differential equation (3.1), and if it remains unsolved with respect to *y*, it is called a general integral. By assigning specific numerical values to arbitrary constants C_1, C_2, …, C_n, a particural integral will be obtained from (3.3).

In order to extract a partial solution from the general solution, it is necessary to add some additional conditions to the differential equation. Typically, initial conditions are used. These conditions serve as a mathematical record of the initial state of the process, especially when studying a time-evolving process.

For a system of *n* ordinary differential equations of the first order:

$$\frac{dU(x)}{dx} = F(x, U), \tag{3.4}$$

where $U = (y_1, y_2, \ldots, y_n)^T$, $F(x, U) = \left(f_1(x, U), f_2(x, U), \ldots, f_n(x, U)\right)^T$ – In vectors, the general solution contains n derivative constants $U = \Phi\left(x, C_1, \ldots, C_n\right)$, and in order to select its partial solutions, it is also necessary to set *n* additional conditions, that is, as many equations as the system contains.

ODE are increasingly being used in mathematical models developed to simulate processes and phenomena that occur in various fields of technology, science, and production.

In the field of microbiology, the problem of determining the dependence of the growth of the number of bacteria over time is well-known. This is provided that there is a certain number of N_0 bacteria at the beginning, under favorable conditions for reproduction. It is also experimentally known that the rate of reproduction of bacteria is proportional to their number.

To solve this problem, it is advisable to use $N(t)$ to denote the number of multiplying bacteria at time *t*, where $N(0) = N_0$. Bearing in mind that the number can only be measured in whole numbers, we will assume that $N(t)$ changes continuously and differently over time. Therefore, the rate of reproduction is the derivative of the function $N(t)$. Thus, the biological experimental law specified in the problem allows us to formulate the differential equation of bacterial reproduction:

$$\frac{dN(t)}{dt} = kN(t),\ k > 0. \tag{3.5}$$

The experimental coefficient k depends on the type of bacteria and the conditions in which they are found. The problem was reduced to a purely mathematical problem of finding the solution $N = N(t)$ of equation (3.5), for which $N(0) = N_0$. Since $N(t) > 0$, dividing both parts of equation (3.5) by $N(t)$, we obtain $\frac{d}{dt}(\ln N(t)) = k$, from which:

$$\ln N(t) = kt + C_1, \tag{3.6}$$

where C_1 – arbitrary constant.

Denoting $C_1 = \ln C$ ($C > 0$), we find from equation (3.6).

$$N(t) = Ce^{kt}. \tag{3.7}$$

To select the function (3.7) that describes the process of bacterial reproduction, you can use the condition $N(0) = N_0$. Then, from equation (3.7), we have $N_0 = C$. Finally:

$$N(t) = N_0 e^{kt}, \tag{3.8}$$

shows that the number of bacteria grows exponentially.

In ballistics, it is important to determine the relationship between the velocity of the vertical fall of a body with mass m and time. It is given that the initial height of the fall is h, and the force of viscous friction acting on the body is proportional to the velocity: $F_{fr} = -\alpha\upsilon$, where $\alpha > 0$ is the coefficient of friction.

When solving this problem, we consider that $\upsilon(t)$ is the velocity of the body at time t. Two oppositely directed forces act on the body: gravity $F_g = mg$ and viscous friction $F_{fr} = -\alpha\upsilon$.

In this case, the differential equation describes Newton's second law.:

$$ma = F = F_g + F_{fr} \quad \Rightarrow \quad m\frac{d\upsilon}{dt} = mg - \alpha\upsilon. \tag{3.9}$$

Dividing both sides of equation (3.9) by m, we obtain a differential equation.

$$\frac{d\upsilon}{dt} = -\frac{\alpha}{m}\upsilon + g. \tag{3.10}$$

The solution to the differential equation (3.10) will be the following expression:

$$\upsilon(t) = \frac{mg}{\alpha} + Ce^{-\frac{\alpha}{m}t}. \tag{3.11}$$

If the body starts moving with zero speed ($\upsilon(0) = 0$), then $C = -\frac{mg}{\alpha}$:

$$\upsilon(t) = \frac{mg}{\alpha}\left(1 - e^{-\frac{\alpha}{m}t}\right). \tag{3.12}$$

During free fall without friction $\left(\frac{d\upsilon}{dt} = g\right)$, the velocity increases linearly: $\upsilon(t) = gt$. In the presence of viscous friction, the increasing speed tends to a constant value $\upsilon = \frac{mg}{\alpha}$.

Many real processes are modeled by differential equations that contain the second derivative of an unknown function. These differential equations are referred to as second-order equations. An example of a problem from the field of ballistics is the task of determining the law of motion for a point mass m falling vertically downwards, while disregarding air resistance.

To solve such a problem, a reference point O is chosen on the vertical axis along which the point falls, and the positive direction is determined – from point O downwards. The position of the point is determined by the coordinate $y(t)$, which changes with time t. The point falls under the force of gravity $F_g = mg$, therefore, according to Newton's second law, $ma = F$. Then $m\frac{d^2y}{dt^2} = mg$ or:

$$\frac{d^2y}{dt^2} = g\,. \tag{3.13}$$

Integrating the ratio (3.13) twice, we can determine:

$$\frac{dy}{dt} = gt + C_1, \quad y(t) = \frac{gt^2}{2} + C_1t + C_2\,. \tag{3.14}$$

Formula (3.14) defines the law of motion of a material point, but it contains constants of integration, in this case – two. Knowing the initial position of the falling point relative to point O – $y(0) = y_0$ and its initial speed $\upsilon(t) = \upsilon_0$, one chooses from the set of functions (3.14) that describes the motion of the point.

Since the speed of movement of the point $\upsilon(t) = \frac{dy}{dt}$, then under the indicated initial conditions $C_1 = \upsilon_0$ and $C_2 = y_0$, the function that describes the law of movement of the point is sought:

$$y = \frac{gt^2}{2} + \upsilon_0 t + y_0\,. \tag{3.15}$$

Thus, the well-known formula for the path traveled by a point during uniform accelerated motion was obtained.

This section deals with solving computational problems of differential calculus, specifically the solution of ordinary differential equations and their systems. Moreover, it is assumed that the user is already familiar with the main information presented in the sections of mathematical analysis.

3.1 Generalized problem statement for ordinary differential equations

There are three main types of tasks for ODE: Cauchy problems, boundary value problems and eigenvalue problems.

The statement of the Cauchy problem for the first-order system of n ODE is formulated as follows in the general case. Find the solution of the differential equation:

$$\frac{dU(x)}{dx} = F(x, U) \text{ at } x>x_0,\ U(x_0)=U^0, \tag{3.16}$$

where x_0 – initial value x; U^0 – the initial value of the vector U ($U = (y_1, y_2, \ldots, y_n)^T$); $F(x,U) = \left(f_1(x,U), f_2(x,U), \ldots, f_n(x,U)\right)^T$, or in expanded form:

$$\frac{dy_i(x)}{dx} = f_i\left(x, y_1, y_2, \ldots, y_n\right);\ x > x_0;\ y_i(x_0) = y_i^{(0)}\ (i = 1, 2, \ldots, n).$$

For equations of the first to n-th orders, the Cauchy problem is formulated as follows find a solution to differential equations:

a) $\frac{dy}{dx} = f(x, y)$ for $x > x_0$ by $y(x_0) = C_0$;

b) $\frac{d^2y}{dx^2} = f(x, y, y')$ for $x > x_0$ by $y(x_0) = C_0,\ y'(x_0) = C_1$;

c) $\frac{d^ny}{dx^n} = f(x, y, y', \ldots, y^{(n)})$ for $x > x_0$ by $y(x_0) = C_0,\ y'(x_0) = C_1,\ \ldots,$ $y^{(n-1)}(x_0) = C_{n-1}$,

where $C_1, C_2, \ldots, C_n$ – some constants; $y', y'', \ldots, y^{(n)}$ – derived functions y.

3.2 Numerical methods for solving ordinary differential equations for Cauchy-type problems

In general, the methods of solving ODE are conditionally divided into exact, approximate, and numerical methods. Analytical methods, which allow you to express the solution of a differential equation in terms of elementary functions or present it using quadratures of elementary functions, belong to the exact methods. Finding an exact, and moreover, general solution to problem (3.16) facilitates qualitative research of this solution and further actions with it. However, the classes of equations for which methods of obtaining exact solutions have been developed are relatively narrow and cover only a small part of the problems that arise in practice.

Methods in which the solution is obtained as the limit of $y(x)$ from some sequence $y_n(x)$ are called approximate methods, and $y_n(x)$ is expressed through elementary functions or using quadratures. By restricting to a finite number n, an approximate expression for $y(x)$ can be obtained. Approximate methods include the expansion of the solution into a generalized power series, the Chaplygin method, the Picard, Kantorovich method, and others. However, these methods are convenient in cases where most of the intermediate operations can be performed accurately (for example, finding an explicit expression for the coefficients of a series). This can only be done for relatively simple problems (linear), which greatly narrows the scope of application of approximate methods.

Numerical methods are algorithms for calculating approximate (and sometimes exact) values of the desired solution $y(x)$ on a chosen grid of values of the argument x_n. The solution will be obtained in the form of a table. Numerical methods do not allow finding the general solution of system (3.16); they can only provide a partial solution, such as the solution of the Cauchy problem (3.16). However, these methods can be applied to a wide class of equations and all types of problems associated with them.

Numerical methods can only be applied to correctly posed (or well-posed) problems. However, it should be noted that the formal fulfillment of correctness conditions may not be sufficient for the successful application of numerical methods. It is necessary for the problem to be well-conditioned, meaning that small changes in the initial conditions would result in sufficiently small changes in the integral curves. If this condition is not met, meaning the problem is ill-conditioned (weakly stable), then small changes in the initial conditions or small errors in the numerical method can introduce significant distortions.

Available numerical methods for solving Cauchy problems are classified into two types:

1) one-step methods, in which information about only one previous step is needed to find the next point on the curve. Well-known one-step methods are the Euler and Runge-Kutta methods.

2) methods of «prediction and correction» (multi-step methods), in which information about more than one of the previous data points is needed to determine the next point on the curve, are used. Iterations are often employed to obtain a reasonably accurate numerical value. Examples of such methods include the Adams, Milne, and Moulton methods.

Euler's method

Euler's method is the simplest method for solving the Cauchy problem (3.16) for $x \in [a; b]$ and $i = 1$. In order to obtain calculation formulas, the interval of continuous change of the variable x can be replaced by a set of points $x_j = a+jh$, which will be called grid nodes:

$$h = \frac{b-a}{n};\ x_0 = a;\ x_n = b,$$

where h – grid step.

The numerical solution of problem (3.16) is a table of values:

$$x_j; y_j\ (j = 0, 1, \ldots, n),$$

where y_j – difference or numerical value of the solution at the node x_j.

Equation (3.16) for $x = x_j$ and $n = 1$, by definition of the derivative:

$$\left.\frac{dy}{dx}\right|_{x=x_j} = \lim_{h\to 0}\frac{y(x_j+h)-y(x_j)}{h}. \qquad (3.17)$$

Discarding the limit in (3.17), the derivative $\left.\frac{dy}{dx}\right|_{x=x_j}$ is replaced with a finite-difference relation:

$$\left.\frac{dy}{dx}\right|_{x=x_j} \approx \frac{y(x_j+h)-y(x_j)}{h}. \qquad (3.18)$$

If we substitute equation (3.18) into equation (3.16), we get:

$$\frac{y(x_j+h)-y(x_j)}{h} \approx f\left(x_j, y(x_j)\right) \quad (j = 0, 1, \ldots, N-1), \qquad (3.19)$$

where $y(x_j)$ – the value of the solution $y(x)$ to the Cauchy problem at node x_j. Denoting by y_j the numerical solution that satisfies the difference equation, we can write:

$$\frac{y_{j+1}-y_j}{h} = f\left(x_j, y_j\right) \quad (j = \overline{0, n-1}) \qquad (3.20)$$

or

$$y_{j+1} = y_j + hf\left(x_j, y_j\right) \quad (j = \overline{0, n-1}). \qquad (3.21)$$

Given the initial condition (3.16) and using the difference equation (3.21), it is possible to consistently determine a series of iterative equations:

$$y_1 = y_0 + hf\left(x_0, y_0\right) \quad (j = 1);$$

$$y_2 = y_1 + hf\left(x_1, y_1\right) \quad (j = 2);$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots;$$

$$y_n = y_{n-1} + hf\left(x_{n-1}, y_{n-1}\right) \quad (j = n-1).$$

The graphical interpretation of the approximate solution obtained by Euler's method is a broken curve (Fig. 3.1, a) that connects points M_0, M_1, …, M_n in a series and is called Euler broken curve.

For the geometric interpretation of the error that occurs when applying the Euler's method, it is necessary to consider the point x_n at which the numerical solution of the problem y_n is obtained. Putting $y_n=y(x_n)$, a hypothetical integral curve $y(x)$ is drawn through this point (y_n, x_n) and $f(x_n, y(x_n))$ is tangent to it (Fig. 3.1, b). Under the given assumptions, it is equal to $f(x_n, y_n)$.

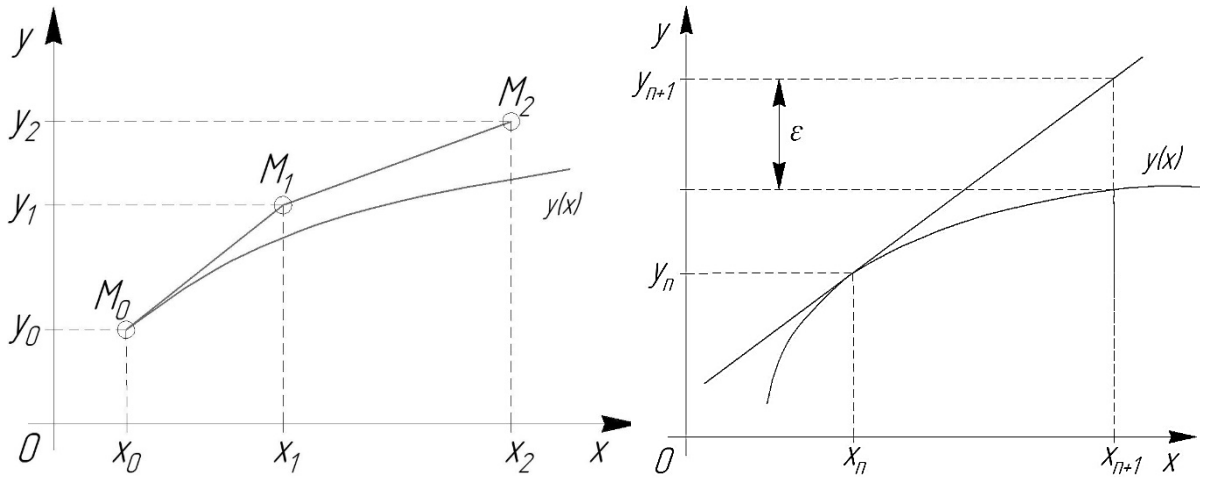


Figure 3.1 – Scheme of the geometric interpretation of Euler's method numerical solution of the Cauchy problem:
a) – approximation scheme; b) – error determination scheme

The point of intersection of this tangent with the perpendicular to the x-axis, which passes through the point x_{n+1}, gives the approximate value of the function y_{n+1} at the point x_{n+1}. At the same time, the solution error is equal to $\varepsilon=y_{n+1}-y(x_{n+1})$. Accordingly, Euler method is a linear extrapolation of the function y_n to the point x_{n+1} using the values of the function and its derivative at the point x_n: $y_{n+1} = y_n + f\left(x_n, y_n\right)(x_{n+1} - x_n)$.

Based on the expansion in the Taylor series of equation (3.21), it is determined that the Euler's method has a large error, namely, an error of the first order $O(h)$ and often turns out to be unstable, since a small error in the initial data or due to rounding in the calculation process increases with the growth of x.

In order to improve the accuracy of the solution to the Cauchy problem, a refined Euler's method is employed for the ODE. This method is based on calculating the function $y(x_{n+1})$ at the next point x_{n+1} using the average value of the slopes of the tangent lines to the integral curve $y(x)$ at points x_n and x_{n+1}. In this scenario, the solution to the problem y_n at point x_n is assumed to be already known.

The method consists of two steps:

1) according to the Euler's method (3.21), the approximate value of $\overline{y}_{n+1}$ at the point $x=x_n+h$ is preliminarily determined by the formula

$$\overline{y}_{n+1} = y_n + hf(x_n, y_n), \tag{3.22}$$

and the function $f(x_{n+1}, \overline{y}_{n+1})$ is calculated in it;

2) the tangents of the tangent angles at the points (x_n, y_n) and $\left(x_{n+1}, \overline{y}_{n+1}\right)$ are added and the average arithmetic value $\Phi(x_n, y_n, h)$ is determined:

$$\Phi(x_n, y_n, h) = \frac{1}{2}\left[f\left(x_n, y_n\right) + f\left(x_{n+1}, \overline{y}_{n+1}\right)\right], \tag{3.23}$$

where $f\left(x_{n+1}, \overline{y}_{n+1}\right) = f\left(x_n + h, y_n + hf\left(x_n, y_n\right)\right)$.

After that, the final (refined) value of the function y_{n+1} at the point x_{n+1} is determined by the formula $y_{n+1}=y_n+h\Phi(x_n, y_n, h)$ or in expanded form:

$$y_{n+1} = y_n + \frac{h}{2}\left[f\left(x_n, y_n\right) + f\left(x_n + h, y_n + hf\left(x_n, y_n\right)\right)\right]. \tag{3.24}$$

Thus, when calculating y_{n+1}, the function $f(x, y)$ has to be calculated twice at the points (x_n, y_n) and $(x_n+h, y_n+h\,y'_n)$.

With the help of the refined Euler's method, it is possible to perform accuracy control by comparing the value of $\overline{y}_{n+1}$ from formula (3.22) and y_{n+1} from formula (3.24), which will allow choosing the appropriate value of the step h in each calculation node based on this. That is, if the value of $|\overline{y}_{n+1} - y_{n+1}|$ is comparable to the calculation errors, then the step must be increased; otherwise, if this difference is large enough (for example, $|\overline{y}_{n+1} - y_{n+1}| > 0{,}01|\overline{y}_{n+1}|$), the value of h must be reduced. Using these estimates, it is possible to build an algorithm of the Euler method with automatic step selection (Fig. 3.2).

For the purpose of geometric interpretation of the refined Euler's method, at the point (x_n, y_n) (Fig. 3.3), the tangent line L_1 to the hypothetical integral curve $y(x)$ is constructed, and the initial value of $\overline{y}_{n+1}$ is determined at the point of intersection of this tangent with the perpendicular to the x-axis passing through point x_{n+1}. After that, at the point $(x_n+h, y_n+h\,y'_n)$, the tangent line L_2 to the curve $y(x)$ is constructed again.

Next, the arithmetic mean of the tangents of the angles of inclination of these tangents (curve L_3) is determined, and the line L_0, parallel to line L_3, is drawn through this point (x_n, y_n). The point of intersection of this line with the ordinate passing through the point x_{n+1} gives the refined value of the function y_{n+1} at point x_{n+1}.

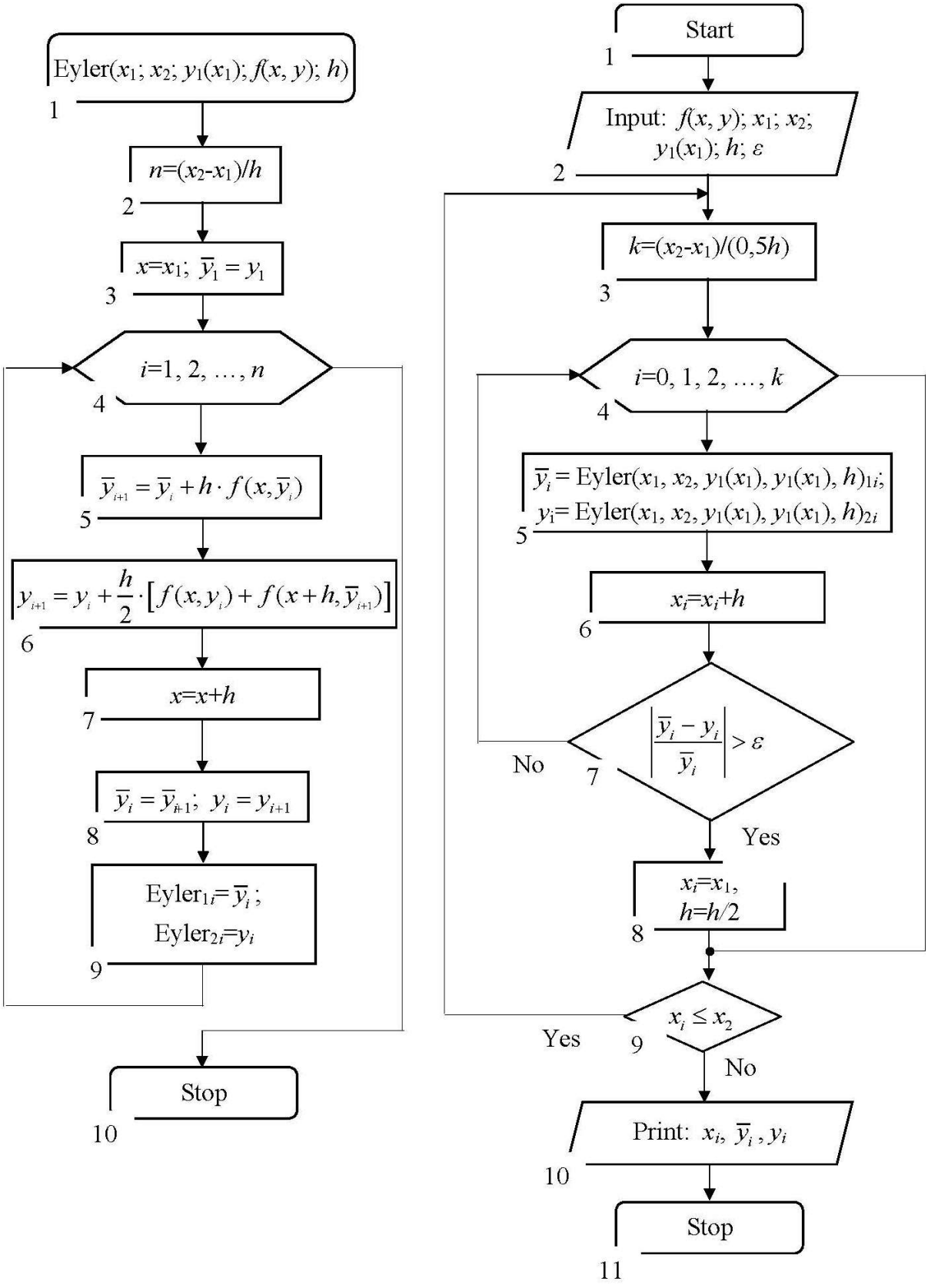


Figure 3.2 – Scheme of the Euler's method algorithm

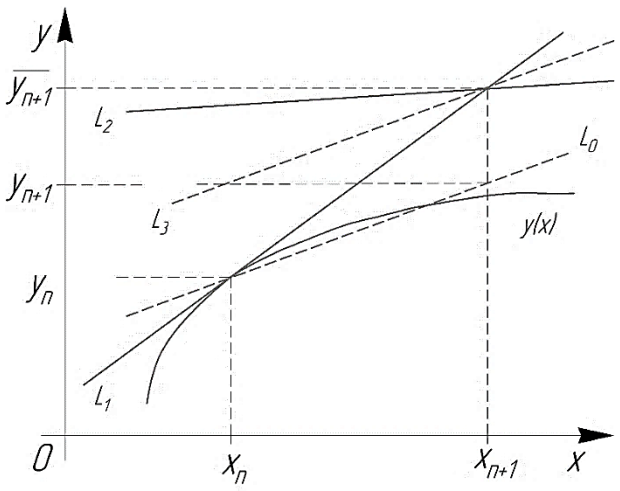


Figure 3.3 – Scheme of the geometric interpretation of the refined Euler's method

Estimating the approximation error of formula (3.24) based on the Taylor series expansion of the functions $y(x_n)$ and $f\left(x_n + h, y_n + hf\left(x_n, y_n\right)\right)$ shows that this method has a second order of approximation $O(h^2)$.

Euler method is successfully applied to systems of ODE. For a system of first-order n ODE, the Euler's method takes the following form:

$$\begin{cases} \dfrac{\overline{y_{j+1}} - \overline{y_j}}{h} = \overline{f}\left(x_j, \overline{y_j}\right) (j = 0, 1, ..., n-1); \\ \overline{y}(0) = \overline{y}_0; \;\; x \in [0; 1], \end{cases} \tag{3.25}$$

where $\overline{y}_j = \left(y_{1j}, y_{2j}, \ldots, y_{n,j}\right)^T$.

Then Euler's method is rewritten for each component of the vector $\overline{y}_j$:

$$\begin{cases} \dfrac{y_{1,j+1} - y_{1,j}}{h} = f_{1,j}; \\ \dfrac{y_{2,j+1} - y_{2,j}}{h} = f_{2,j}; \\ \ldots\ldots\ldots\ldots\ldots\ldots; \\ \dfrac{y_{n,j+1} - y_{n,j}}{h} = f_{n,j} \;\; (j = \overline{0, N-1}); \end{cases}$$

$$y_1(0) = y_{10}, y_2(0) = y_2, \ldots, y_n(0) = y_{n0}.$$

To estimate the total error of the numerical solution to the Cauchy problem for ODE systems, Runge's method (other name Richardson's extrapolation) can be used. This method relies on comparing the results of calculating the values of $\overline{y}_n$ and $\tilde{y}_n$ at the point x_n with different steps $\overline{h}$ та $\tilde{h}$, respectively, in order to determine the error of the approximation:

$$R_n^h = \left(\tilde{y}_n - \overline{y}_n\right) / \left(1 - 2^{-p}\right), \tag{3.26}$$

where p – the order of approximation of the difference scheme used, particularly p=1 – the Euler method.

Algorithms for Euler's method in ODE systems with automatic step selection are shown in Figure 3.4.

Example 3.1. To solve the Cauchy problem for the first-order ODE using Euler numerical methods:

$$\frac{dy}{dx} = y + x\,;\; x \in [0;\, 1{,}0],$$

which satisfies the initial condition for $x_0 = 0$, $y_0 = 1.0$, and the relative error of the calculation ε = 10,0 %.

Solution:

To solve this problem, the segment [0; 1,0] can be divided into ten parts by the points x_0 = 0; 0.1; 0.2; ..., 1.0. Accordingly, h = 0,1. The values of y_1, y_2, …, y_n will be determined by the Euler's method according to formula (3.21), then:

$$y_1 = y_0 + hf\left(x_0, y_0\right) = 1 + 0{,}1 \cdot (0 + 1) = 1{,}10;$$

$$y_2 = y_1 + hf\left(x_1, y_1\right) = 1{,}1 + 0{,}1 \cdot (1{,}1 + 0{,}1) = 1{,}21;$$

…… .

Also, the values of y_1, y_2, …, y_n can be determined using the refined Euler's method according to formula (3.24). Then:

$$\overline{y}_1 = y_0 + \frac{h}{2}\left[f\left(x_0, y_0\right) + f\left(x_0 + h, y_0 + hf\left(x_0, y_0\right)\right)\right] =$$

$$= 1{,}0 + \frac{0{,}1}{2}\left[(0 + 1{,}0) + (0 + 0{,}1 + 1{,}0 + 0{,}1 \cdot (0 + 1{,}0))\right] = 1{,}11000;$$

$$\overline{y}_2 = \overline{y}_1 + \frac{h}{2}\left[f\left(x_1, \overline{y}_1\right) + f\left(x_1 + h, \overline{y}_1 + hf\left(x_1, \overline{y}_1\right)\right)\right] =$$

$$= 1{,}11 + \frac{0{,}1}{2}\left[(0{,}1 + 1{,}11) + (0{,}1 + 0{,}1 + 1{,}11 + 0{,}1 \cdot (0{,}1 + 1{,}11))\right] = 1{,}24205;$$

…… .

Also, for a comparative analysis of the accuracy of the solution using the usual and refined Euler's method, an analytical solution of the ODE was obtained, namely:

$$\tilde{y} = 2e^x - x - 1. \tag{3.27}$$

In the process of solving, Table 3.1 of the calculation results was compiled:

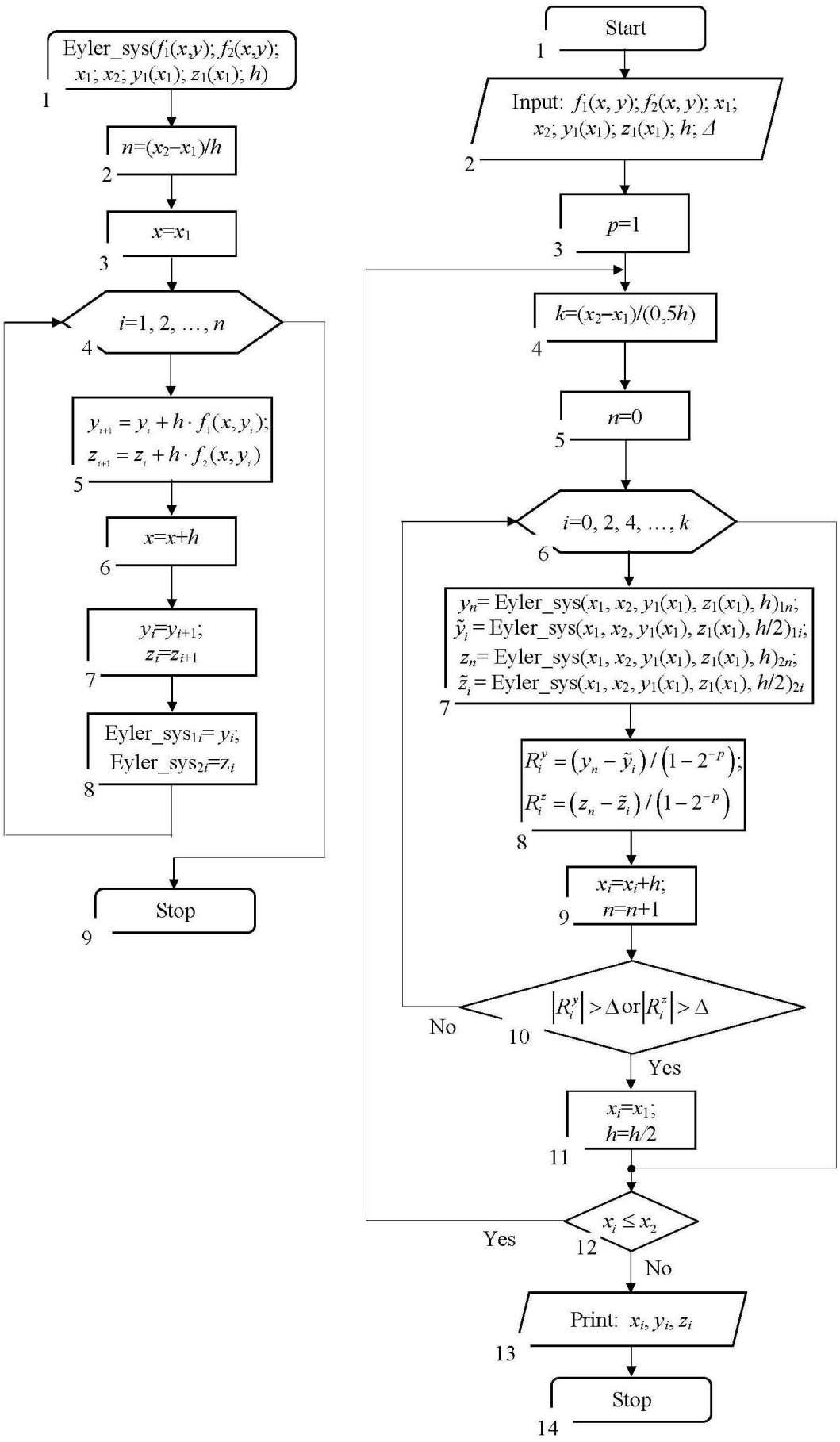


Figure 3.4 – Scheme of the Euler's method algorithm for ODE systems

Table 3.1 – Calculation results for the solution of the differential equation

n	x_n	$y_n(x_n)$	$\overline{y}_n(x_n)$	$\tilde{y}_n(x_n)$
0	0,0	1,0000	1,00000	1,00000
1	0,1	1,1000	1,11000	1,11034
2	0,2	1,2200	1,24205	1,24281
3	0,3	1,3620	1,39847	1,39972
4	0,4	1,5240	1,58180	1,58365
5	0,5	1,7164	1,79489	1,79744
6	0,6	1,9380	2,04086	2,04424
7	0,7	2,1918	2,32315	2,32751
8	0,8	2,4730	2,64558	2,65108
9	0,9	2,8003	3,01236	3,01921
10	1,0	3,1703	3,42816	3,43656

From the obtained results, shown in Table 3.1, it can be seen that the errors made during the determination of the solution grow towards the end of the table. In particular, there is a relative error at the points $x_5 = 0,5$ and $x_{10} = 1,0$:

– by comparing the values of the analytical solution and the usual Euler's method

$$\varepsilon_{x_5} = \left|\frac{\tilde{y}_{x_5} - y_{x_5}}{\tilde{y}_{x_5}}\right| \cdot 100\% = \left|\frac{1,79744 - 1,71640}{1,79744}\right| \cdot 100\% = 4,51\%,$$

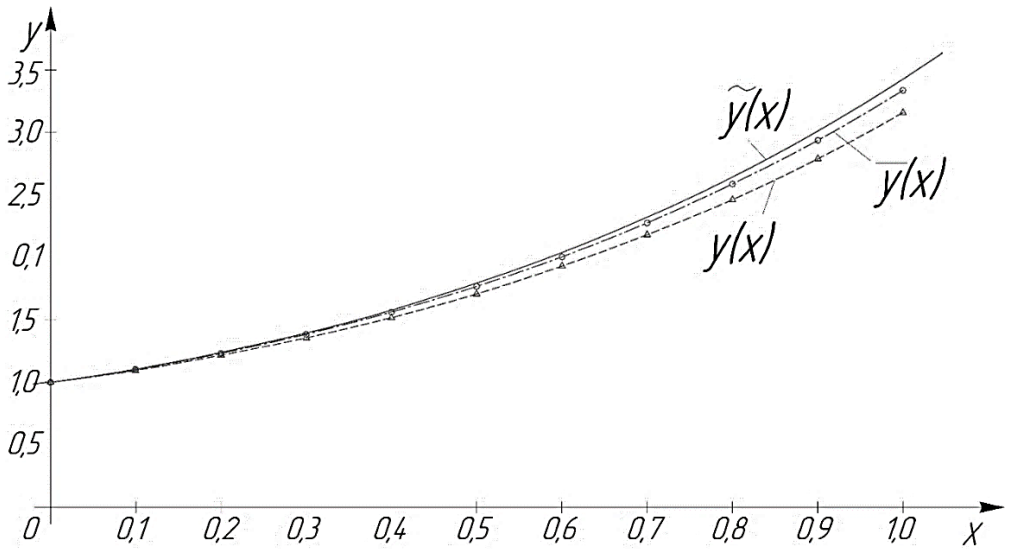
$$\varepsilon_{x_{10}} = \left|\frac{\tilde{y}_{x_{10}} - y_{x_{10}}}{\tilde{y}_{x_{10}}}\right| \cdot 100\% = \left|\frac{3,43656 - 3,17030}{3,43656}\right| \cdot 100\% = 7,75\%;$$

– by comparing the values of the analytical solution and the refined Euler's method

$$\overline{\varepsilon}_{x_5} = \left|\frac{\tilde{y}_{x_5} - \overline{y}_{x_5}}{\tilde{y}_{x_5}}\right| \cdot 100\% = \left|\frac{1,79744 - 1,79489}{1,79744}\right| \cdot 100\% = 0,14\%,$$

$$\overline{\varepsilon}_{x_{10}} = \left|\frac{\tilde{y}_{x_{10}} - \overline{y}_{x_{10}}}{\tilde{y}_{x_{10}}}\right| \cdot 100\% = \left|\frac{3,43656 - 3,42816}{3,43656}\right| \cdot 100\% = 0,24\%.$$

The numerical results of the calculation of the solution of the differential equation, which are summarized in Table 3.1, are conveniently presented on a graph (Fig. 3.5).



$y(x)$ – Euler's method;
$\bar{y}(x)$ – refined Euler's method;
$\tilde{y}(x)$ – is the exact solution

Figure 3.5 – Diagram of numerical solution results problems of Cauchy ODE

It is also possible to note the rather high accuracy of the refined Euler's method in the process of solving the Cauchy ODE problem, which, compared to the usual Euler's method, is approximately ten times higher.

Consider the implementation of Euler's method in the PYTHON programming language:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import *
from sympy import *
# introducing an integration step
h_0=float(input())
# entering the initial value of the integration argument (total integration
     #interval)
a=int(input()); x_mtr=np.array(a)
# entering the initial value of the diff function argument. equation
y_0=int(input()); y_mtr=np.array(y_0)
# entering the final value of the total integration interval
b=int(input())
# entering the value of the relative error of the calculation
e=float(input())/100
# setting the function of the differential equation
def f(x,y):
    return x+y
# calculation of iterative formulas
def eyler(a,b,y_0,h):
    x=a; y=y_0; y_plus=y_0
    x_mas=[x]; y_mas=[y]; y_masplus=[y]
    x_mtr=np.array(a)
    y_mtr=np.array(y_0)
    while x<=b:
        # iterative formula of the Euler's method
```

```
        y=y+(h*f(x,y))
        x=x+h
        # iterative formula of the refined Euler's method
        y_plus=y_plus+((h/2)*(f(x-h,y_plus)+f(x,y_plus+(h*f(x,y_plus)))))
        x_mas.append(x)
        y_mas.append(y)
        x_mtr=np.append(x_mtr, x)
        y_masplus.append(y_plus)
    return x_mas, y_mas, y_masplus, x_mtr
# control of calculation accuracy and adjustment of the calculation step in
     #the node
h=h_0; x=a
while x<=b:
    for k in range(0,len(eyler(a,b,y_0,h)[0])):
        x=x+h
        if abs(eyler(a,b,y_0,h)[1][k]-
     eyler(a,b,y_0,h)[2][k])>0.01*abs(eyler(a,b,y_0,h)[1][k]):
            h=h/2; x=a; break
# we construct graphs of the solutions of the differential equation
plt.figure(figsize=(7, 7))
plt.xlabel('x',fontsize=15, color='blue')
plt.ylabel('y',fontsize=15, color='blue')
plt.plot(eyler(a,b,y_0,h)[0], eyler(a,b,y_0,h)[1])
plt.plot(eyler(a,b,y_0,h)[3], (2*(np.exp(eyler(a,b,y_0,h)[3])))-
     eyler(a,b,y_0,h)[3]-1)
plt.plot(eyler(a,b,y_0,h)[0], eyler(a,b,y_0,h)[2])
plt.legend(['Euler's method','f(x)=2*exp(x)-x-1','Refined Euler's method'],
     loc=1)
plt.grid(True)
plt.xlim([0, 1])
plt.ylim([1, 4])
plt.show().
```

Example 3.2. It is known that when finding solutions to Laplace and Helmholtz differential equations in cylindrical and spherical coordinates, as well as when solving problems of wave propagation, static potentials, etc., Bessel differential equation arises:

$$y'' + \frac{1}{x}y' + y = 0\,;\ x \in [0;\ 1{,}0], \tag{3.28}$$

find a solution with initial conditions $x_0 = 1{,}0$; $y_0 = 0{,}765$; $y'_0 = -0{,}440$ and error of cut $R^h = 0{,}005$.

Solution:

To solve this problem, the segment [1,0; 2.0] can be divided into ten parts by the points x_0=1,0; 1,1; 1,2; …, 2,0. Accordingly, h=0,1.

To apply Euler numerical methods, it is necessary to reduce the second-order differential equation (3.28) to a system of two first-order equations with two unknown functions. This can be achieved by substituting $y' = z$. Since $y'' = z'$, equation (3.28) can be written in the form of a system:

$$\begin{cases} y' = z; \\ z' = -\dfrac{z}{x} - y; \\ x_0 = 1,0;\ y_0 = 0,765;\ z_0 = -0,440. \end{cases} \tag{3.29}$$

The values of $y_1, y_2, \ldots, y_n$ and $z_1, z_2, \ldots, z_n$ will be determined by the Euler's method according to formula (3.22), namely:

$$\begin{cases} \dfrac{y_1 - y_0}{h} = z_0; \\ \dfrac{z_1 - z_0}{h} = -\dfrac{z_0}{x_0} - y_0; \end{cases} \Rightarrow \begin{cases} y_1 = y_0 + hz_0; \\ z_1 = z_0 + h\left(-\dfrac{z_0}{x_0} - y_0\right); \end{cases}$$

$$\begin{cases} \dfrac{y_2 - y_1}{h} = z_1; \\ \dfrac{z_2 - z_1}{h} = -\dfrac{z_1}{x_1} - y_1; \end{cases} \Rightarrow \begin{cases} y_2 = y_1 + hz_1; \\ z_2 = z_1 + h\left(-\dfrac{z_1}{x_1} - y_1\right); \end{cases}$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \tag{3.30}$$

$$\begin{cases} \dfrac{y_{n+1} - y_n}{h} = z_n; \\ \dfrac{z_{n+1} - z_n}{h} = -\dfrac{z_n}{x_n} - y_n; \end{cases} \Rightarrow \begin{cases} y_{n+1} = y_n + hz_n; \\ z_{n+1} = z_n + h\left(-\dfrac{z_n}{x_n} - y_n\right); \end{cases}$$

$$x_0 = 1,0;\ y_0 = 0,765;\ z_0 = -0,440.$$

Also, for a comparative analysis of the accuracy of the solution by the usual and refined Euler's methods, an analytical solution of the Bessel differential equation (3.28) was obtained in the form of a Taylor series expansion. Specifically, it was expressed as a Bessel function of the first kind with zero order:

$$\tilde{y}(x) = J_0(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{k!(k+1)!}\left(\frac{x}{2}\right)^{2k}, \tag{3.31}$$

where k – integer.

Also, the derivative of the Bessel function of the first order zero is equal to the negative Bessel function of the first order:

$$\tilde{y}'(x) = J_0'(x) = -J_1'(x)\ . \tag{3.32}$$

The values of the Bessel functions (3.31) and (3.32) are determined using ready tabular data, which are given in Table 3.2 for $x \in [1,0;\ 2.0]$. To determine the calculation error using the Runge method (3.26), the values of $\overline{y}_n(x_n)$ were determined with a step of $h/2$=0,05. In the process of solving, Table 3.2 of the calculation results was compiled

Table 3.2 – Calculation results of the solution to the differential equation

n	x_n	$y_n(x)$	$\overline{y}_n(x_n)$	$y'_n(x_n)$	$\tilde{y}_n$	$\tilde{y}'_n(x_n)$
0	1,0	0,76500	0,76500	-0,44000	0,7652	-0,4400
1	1,1	0,72100	0,72019	-0,47250	0,7196	-0,4709
2	1,2	0,67375	0,67229	-0,50165	0,6711	-0,4983
3	1,3	0,62359	0,62166	-0,52722	0,6201	-0,5220
4	1,4	0,57086	0,56866	-0,54902	0,5669	-0,5419
5	1,5	0,51596	0,51367	-0,56689	0,5118	-0,5579
6	1,6	0,45927	0,45709	-0,58069	0,4554	-0,5699
7	1,7	0,40120	0,39933	-0,59033	0,3980	-0,5778
8	1,8	0,34217	0,34080	-0,59572	0,3400	-0,5815
9	1,9	0,28260	0,28192	-0,59684	0,2818	-0,5812
10	2,0	0,22291	0,22311	-0,59369	0,2239	-0,5767

From the obtained results, given in Table 3.2, it can be seen that the errors made during the determination of the solution grow towards the end of the table for the values of the first derivative, namely $y'_n(x_n)$ and $\tilde{y}'_n(x_n)$. In particular, at the points $x_5 = 1{,}5$ and $x_{10} = 2{,}0$:

– relative error can be determined by comparing the values of the analytical solution and Euler method

$$\varepsilon_{x_5} = \left|\frac{\tilde{y}_{x_5} - y_{x_5}}{\tilde{y}_{x_5}}\right| \cdot 100\% = \left|\frac{0{,}51596 - 0{,}51180}{0{,}51596}\right| \cdot 100\% = 0{,}81\%,$$

$$\varepsilon_{x_{10}} = \left|\frac{\tilde{y}_{x_{10}} - y_{x_{10}}}{\tilde{y}_{x_{10}}}\right| \cdot 100\% = \left|\frac{0{,}22390 - 0{,}22291}{0{,}22390}\right| \cdot 100\% = 0{,}44\%;$$

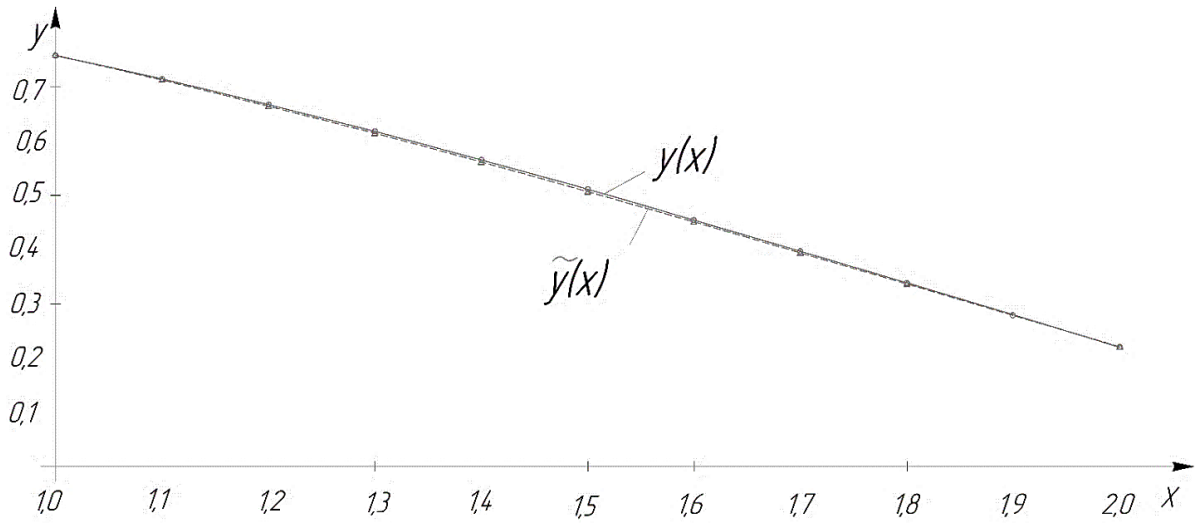
– error of cut by the Runge method (see 3.26)

$$R^h_{x_5} = \left(y_{x_5} - \overline{y}_{x_5}\right) / \left(1 - 2^{-p}\right) = \left(0{,}51596 - 0{,}51367\right) / \left(1 - 2^{-1}\right) = 0{,}0046;$$

$$R^h_{x_{10}} = \left(y_{x_{10}} - \overline{y}_{x_{10}}\right) / \left(1 - 2^{-p}\right) = \left(0{,}22291 - 0{,}22311\right) / \left(1 - 2^{-1}\right) = -0{,}0004.$$

The numerical results of the calculation of the solution of the differential equation (3.28), presented in Table 3.2, are conveniently displayed in a graph (Fig. 3.6).

From the obtained results, presented in Table 3.2 and Figure 3.6, it is clear that the absolute and relative errors allowed during the determination of the solution increase in the middle of the considered interval $x \in [1{,}0;\ 2.0]$.



$y(x)$ – Euler's method;
$\tilde{y}(x)$ – is the exact solution

Figure 3.6 – Diagram of the results of the numerical solution of the higher-order Cauchy ODE problem

Consider the implementation of Euler's method in the PYTHON programming language:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import *
from sympy import *
# introducing an integration step
h_0=float(input())
# entering the initial value of the integration argument (total integration
    interval)x0
a=int(input()); x_mtr=np.array(a)
# entering the initial value of the argument of the differential equation
    function y0
y_0=float(input()); y_mtr=np.array(y_0)
# entering the initial value of the argument of the differential equation
    function z0
z_0=float(input()); z_mtr=np.array(z_0)
# entering the final value of the total integration interval
b=int(input())
# entering the permissible calculation error
e_0=float(input())
# setting the functions of the differential equation system
def f1(z):
    return z
def f2(x,y,z):
    return (-z/x)-y
# calculation of iterative formulas
def eyler(a,b,y_0,z_0,h):
    x=a; y=y_0; z=z_0
    x_mas=[x]; y_mas=[y]; z_mas=[z]
    x_mtr=np.array(a)
    y_mtr=np.array(y_0)
    z_mtr=np.array(z_0)
    while x<b:
        # iterative formula of the Euler method
        y_buf=y
```

```
        z_buf=z
        y=y+(h*f1(z_buf))
        z=z+(h*f2(x,y_buf,z_buf))
        x=x+h
        x_mas.append(x)
        y_mas.append(y)
        z_mas.append(z)
    return x_mas, y_mas, z_mas
y_Bessel=[0.7652, 0.7196, 0.6711, 0.6201, 0.5669,
                  0.5118, 0.4554, 0.3980, 0.34, 0.2818, 0.2239]
# determination of the calculation error by the Runge method and adjustment
     of the integration step
p=1; x=a; h=h_0
while x<=b:
    n=0
    for k in range(0,len(eyler(a,b,y_0,z_0,h/2)[1]),2):
        Rh_y=(eyler(a,b,y_0,z_0,h)[1][n]-eyler(a,b,y_0,z_0,h/2)[1][k])/(1-
     (1/2**p))
        Rh_z=(eyler(a,b,y_0,z_0,h)[2][n]-eyler(a,b,y_0,z_0,h/2)[2][k])/(1-
     (1/2**p))
        #print(abs(Rh_y))
        #print(abs(Rh_z))
        n=n+1
        x=x+h
        if abs(Rh_y)>e_0 or abs(Rh_z)>e_0:
            h=h/2; x=a
            break
# we construct graphs of the solutions of the differential equation
plt.figure(figsize=(7, 7))
plt.xlabel('x',fontsize=15, color='blue')
plt.ylabel('y',fontsize=15, color='blue')
plt.plot(eyler(a,b,y_0,z_0,h)[0], eyler(a,b,y_0,z_0,h)[1])
plt.plot(eyler(a,b,y_0,z_0,h_0)[0], y_Bessel)
plt.legend(['Euler's method',' Exact solution based on calculation
     tables'], loc=1)
plt.grid(True)
plt.xlim([1, 2])
plt.ylim([0.2, 0.8])
    plt.show().
```

Runge-Kutta method

To construct a difference integration scheme for the numerical solution of the Cauchy problem of the ODE, it is necessary to use the expansion of equation (3.16) for i=1 in the Taylor series:

$$y(x_{k+1}) = y(x_k) + y'(x_k)h + y''(x_k)\frac{h^2}{2!} + \ldots + y^{(n)}(x_k)\frac{h^n}{n!}. \qquad (3.33)$$

For the expression (3.33), K. Runge proposed (later V. Kutta developed this idea of the method) for the difference $\Delta y(h) = y(x_{k+1}) - y(x_k)$ to search for linear approximation combinations of the following form:

$$\Delta y(h) = y(x_{k+1}) - y(x_k) = p_{r1}K_1(h) + p_{r2}K_2(h) + \ldots + p_{rm}K_r(h), \qquad (3.34)$$

where p_{rm} (m=1, 2, …, r) – some constant coefficients; r – the order of accuracy of the numerical solution of the differential equation, and $K_r(h)$ are functions that are calculated according to the formulas:

$$\begin{cases} K_1(h) = f(x_k, y_k); \\ K_2(h) = f(x_k + \dfrac{h}{2}, y_k + \dfrac{h}{2}K_1); \\ K_3(h) = f(x_k + \dfrac{h}{2}, y_k + \dfrac{h}{2}K_2); \\ K_4(h) = f(x_k + h, y_k + hK_3). \end{cases} \tag{3.35}$$

The equation (3.34) based on functions (3.35) is then rewritten in the following form:

$$y_{k+1} = y_k + \frac{h}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right). \tag{3.36}$$

The iterative equation (3.36) is the Runge-Kutta formula for the numerical solution of the ODE with a calculation error of the fourth order $O(h_4)$, which has also become the most widely used in practical calculations.

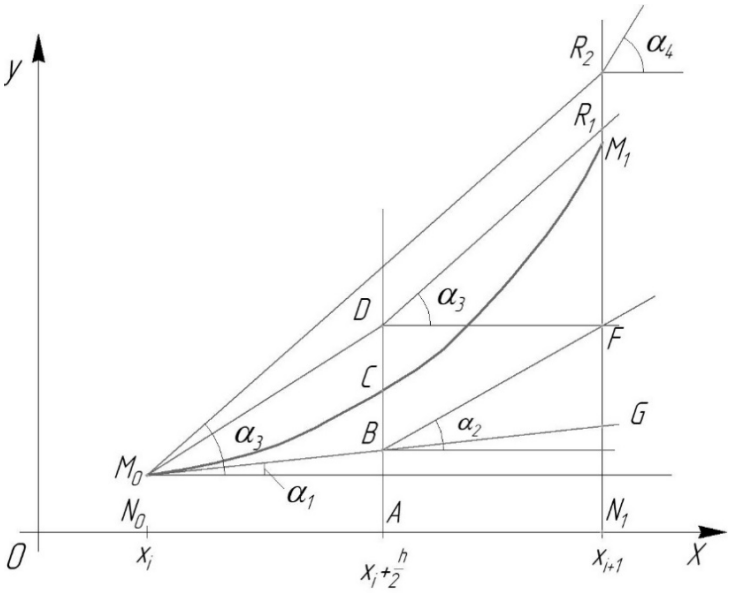


Figure 3.7 – Scheme depicting the geometric interpretation of the Runge-Kutta method

Increasing the order of accuracy of the numerical one-step Runge-Kutta methods leads to a rapid increase in the complexity of calculations, since in one step it is necessary to calculate the value of the function $f(x, y(x))$ for different values of the arguments. Therefore, in practice, the calculation scheme of the fourth-order Runge-Kutta method (3.36) is mainly used.

Components K_1, K_2, K_3, K_4 of the calculation scheme (3.36) of the Runge-Kutta method of the fourth order have a simple geometric interpretation (Fig. 3.7). Let the curve M_0CM_1 represent the solution of the Cauchy problem (3.16) ODE. Point C of this curve lies on a line perpendicular to the Ox axis and bisects segment N_0N_1, B and G are points of intersection of the tangent drawn to the curve at point M_0 with ordinates AC and N_1M_1. Then the number K_1 with accuracy up to the factor h is the angular coefficient (α_1) of the tangent at the point M_0 to the integral curve M_0CM_1, i.e. $K_1 = f(x_k, y_k)$.

Point B has coordinates $x = x_i+h/2$ and $y = y_i+K_1/2$, respectively. The number K_2, with accuracy up to the factor h, represents the angle coefficient (α_2) of the tangent drawn to the integral curve at point B (BF – tangent segment).

A line parallel to line BF is drawn through point M_0. Point D has coordinates $x = x_i+h/2$, $y = y_i+K_1/2$, and the value K_3, accurate up to the factor h, represents the angle coefficient (α_3) of the tangent drawn to the integral curve at point D (DR_1 is the segment of this tangent). Then, a straight line DR_1 is drawn through point M_0, intersecting the extension of M_1N_1 at point $R_2(x_i+h, y_i+K_3)$. As a result, the angular coefficient (α_4) of the tangent drawn to the integral curve at point R_2 is represented by the value K_4, accurate up to the factor h.

The Runge-Kutta algorithm with automatic step selection is shown in Figure 3.8.

To estimate the total error of the numerical solution of the Cauchy problem for the ODE by the Runge-Kutta method, and to choose the integration step at each iterative step of the calculation, the Collatz method was proposed. Namely, if in the process of calculations the value of $R = \left|\frac{K_2 - K_3}{K_1 - K_2}\right|$ exceeds a few hundredths, then the step must be reduced (see Fig. 3.8).

Example 3.3. To solve the fourth-order Cauchy problem for the first-order ODE using the fourth-order Runge-Kutta numerical method:

$$\frac{dy}{dx} = y + x\,;\ \ x \in [0;\ 1{,}0], \tag{3.37}$$

which satisfies the initial conditions for $x_0 = 0$ and $y_0 = 1{,}0$.

Solution:

To solve this problem, the segment [0; 1,0] can be divided into ten parts by the points $x_0 = 0$; 0,1; 0,2; …, 1,0. Accordingly, $h = 0{,}1$. The values of y_1, y_2, …, y_n will be determined by the fourth-order Runge-Kutta method using formula (3.36). Then:

$$\begin{cases} K_1^{(0)} = y_0 + x_0; \\ K_2^{(0)} = y_0 + \dfrac{h}{2}K_1^{(0)} + x_0 + \dfrac{h}{2}; \\ K_3^{(0)} = x_0 + \dfrac{h}{2} + y_0 + \dfrac{h}{2}K_2^{(0)}; \\ K_4^{(0)} = x_0 + h + y_0 + hK_3^{(0)}; \\ y_1 = y_0 + \dfrac{h}{6}\left(K_1^{(0)} + 2K_2^{(0)} + 2K_3^{(0)} + K_4^{(0)}\right); \end{cases} \Rightarrow \begin{cases} K_1^{(0)} = 1 + 0 = 1{,}0; \\ K_2^{(0)} = 1{,}0 + \dfrac{0{,}1}{2}1{,}0 + 0 + \dfrac{0{,}1}{2} = 1{,}10000; \\ K_3^{(0)} = 0 + \dfrac{0{,}1}{2} + 1{,}0 + \dfrac{0{,}1}{2}1{,}1 = 1{,}10500; \\ K_4^{(0)} = 0 + 0{,}1 + 1{,}0 + 0{,}1 \cdot 1{,}105 = 1{,}21050; \\ y_1 = 1 + \dfrac{0{,}1}{6}\left(\begin{array}{l} 1 + 2 \cdot 1{,}1 + \\ +2 \cdot 1{,}105 + 1{,}2105 \end{array}\right) = 1{,}11034; \end{cases}$$

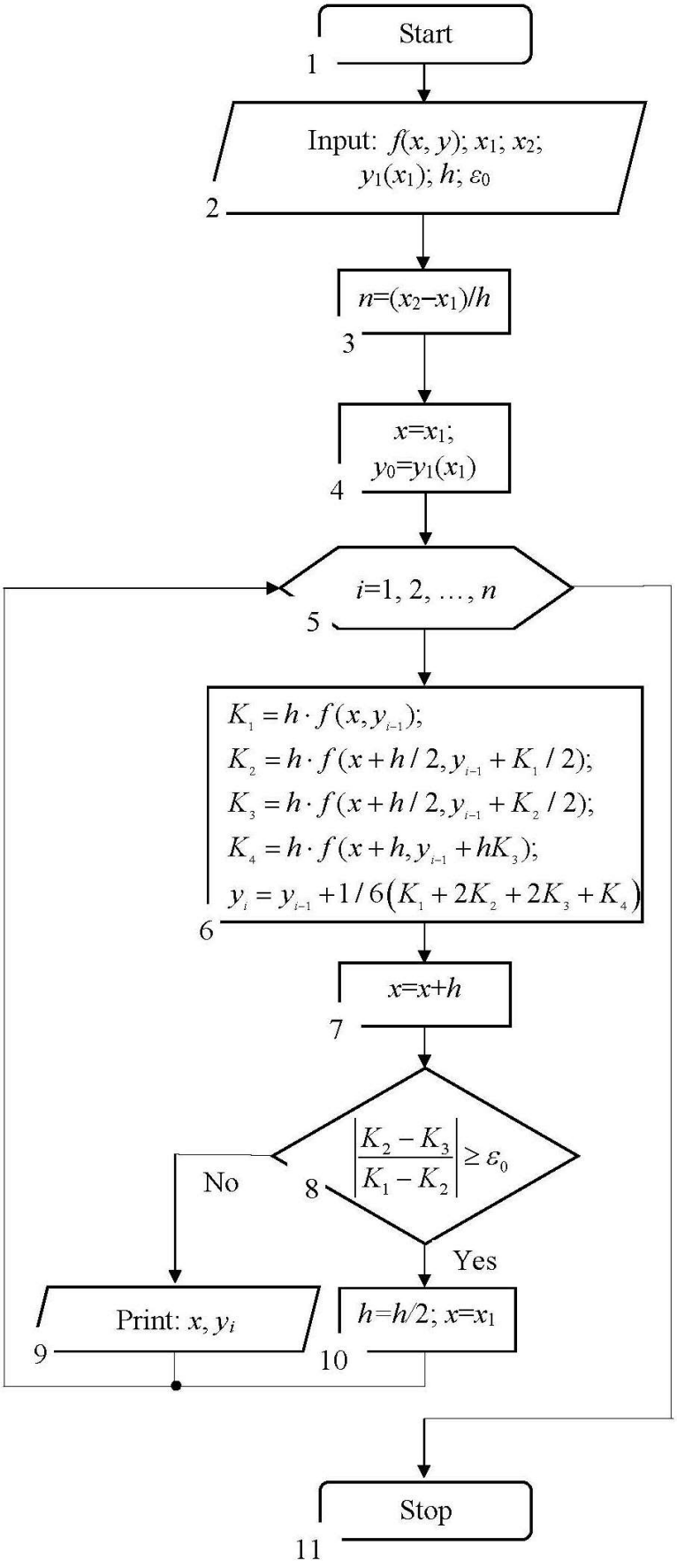


Figure 3.8 – Scheme of the algorithm for the fourth-order Runge-Kutta method

$$\begin{cases} K_1^{(1)} = y_1 + x_1; \\ K_2^{(1)} = y_1 + \dfrac{h}{2}K_1^{(1)} + x_1 + \dfrac{h}{2}; \\ K_3^{(1)} = x_1 + \dfrac{h}{2} + y_1 + \dfrac{h}{2}K_2^{(1)}; \\ K_4^{(1)} = x_1 + h + y_1 + hK_3^{(1)}; \\ y_2 = y_1 + \dfrac{h}{6}\left(K_1^{(1)} + 2K_2^{(1)} + 2K_3^{(1)} + K_4^{(1)}\right); \end{cases}$$

$$\begin{cases} K_1^{(1)} = 1{,}11034 + 0{,}1 = 1{,}2100; \\ K_2^{(1)} = 1{,}11034 + \dfrac{0{,}1}{2}1{,}210 + 0{,}1 + \dfrac{0{,}1}{2} = 1{,}3210; \\ K_3^{(1)} = 0{,}1 + \dfrac{0{,}1}{2} + 1{,}11034 + \dfrac{0{,}1}{2}1{,}321 = 1{,}3270; \\ K_4^{(1)} = 0{,}1 + 0{,}1 + 1{,}11034 + 0{,}1 \cdot 1{,}327 = 1{,}4430; \\ y_2 = 1{,}11034 + \dfrac{0{,}1}{6}\begin{pmatrix} 1{,}210 + 2 \cdot 1{,}321 + \\ +2 \cdot 1{,}327 + 1{,}443 \end{pmatrix} = 1{,}2428; \end{cases}$$

…… .

Also, for a comparative analysis of the accuracy of the fourth-order Runge-Kutta solution, an analytical solution of the ODE was obtained, namely:

$$\tilde{y} = 2e^x - x - 1. \tag{3.38}$$

In the process of solving, Table 3.3 of the calculation results was compiled.

From the obtained results, given in Table 3.3, a high degree of accuracy in determining the solution can be seen, particularly at the points $x_5 = 0{,}5$ and $x_{10} = 1{,}0$:

– relative error by comparing the values of the analytical solution and the Runge-Kutta method of the fourth order

$$\Delta_{x_5} = \left|\frac{\tilde{y}_{x_5} - y_{x_5}}{\tilde{y}_{x_5}}\right| \cdot 100\% = \left|\frac{1{,}79744 - 1{,}79743}{1{,}79744}\right| \cdot 100\% = 5{,}56 \cdot 10^{-4}\%,$$

$$\Delta_{x_{10}} = \left|\frac{\tilde{y}_{x_{10}} - y_{x_{10}}}{\tilde{y}_{x_{10}}}\right| \cdot 100\% = \left|\frac{3{,}43656 - 3{,}43655}{3{,}43656}\right| \cdot 100\% = 2{,}90 \cdot 10^{-4}\%;$$

– the value of the error indicator according to the Collatz's method does not exceed a few hundredths

$$R_{x_5} = \left| \frac{K_2^{x_5} - K_3^{x_5}}{K_1^{x_5} - K_2^{x_5}} \right| = \left| \frac{2{,}1328240 - 2{,}140283}{1{,}9836422 - 2{,}132824} \right| = 0{,}005;$$

$$R_{x_{10}} = \left| \frac{K_2^{x_{10}} - K_3^{x_{10}}}{K_1^{x_{10}} - K_2^{x_{10}}} \right| = \left| \frac{4{,}1651520 - 4{,}177450}{3{,}9191925 - 4{,}165152} \right| = 0{,}005.$$

It is also possible to note the relatively high accuracy of the fourth-order Runge-Kutta method compared to the conventional and refined Euler's methods when solving the Cauchy ODE problem.

Table 3.3 – Calculation results for the solution of the differential equation

n	x_n	$K_1^{(n)}$	$K_2^{(n)}$	$K_3^{(n)}$	$K_4^{(n)}$	$y_n(x_n)$	$\tilde{y}_n(x_n)$
0	0,0	0,0000000	0,000000	0,000000	0,000000	1,00000	1,00000
1	0,1	1,0000000	1,100000	1,105000	1,210500	1,11034	1,11034
2	0,2	1,2100000	1,321000	1,327000	1,443000	1,24280	1,24281
3	0,3	1,4428000	1,564940	1,571047	1,699905	1,39971	1,39972
4	0,4	1,6997113	1,834697	1,841446	1,983856	1,58364	1,58365
5	0,5	1,9836422	2,132824	2,140283	2,297671	1,79743	1,79744
6	0,6	2,2974343	2,462306	2,470550	2,644489	2,04423	2,04424
7	0,7	2,6442283	2,826440	2,835550	3,027783	2,32750	2,32751
8	0,8	3,0274948	3,228870	3,238938	3,451389	2,65107	2,65108
9	0,9	3,4510698	3,673623	3,684751	3,919545	3,01919	3,01921
10	1,0	3,9191925	4,165152	4,177450	4,436937	3,43655	3,43656

Consider the implementation of the Runge-Kutta method in the PYTHON programming language:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import *
from sympy import *
# entering the integration step
h=float(input())
# entering the initial value of the integration argument (total integration
    interval)
a=int(input()); x_mtr=np.array(a)
# inputting the initial value of the argument of the function of the
    differential equation
y_0=int(input()); y_mtr=np.array(y_0)
# entering the final value of the total integration interval
b=int(input())
```

```
# Introduction to the permissible error of calculation according to the Kolatz
    method
e_0=float(input())
# setting the function of the differential equation
def f(x,y):
    return x+y
# calculation of iterative formulas
x_mas=[a]; y_mas=[y_0]
x_mtr=np.array(a)
x=a; y=y_0
while x<b:
    # iterative formula of the Runge-Kutta method
    K1=f(x,y); K2=f(x+(h/2),y+(h/2*K1))
    K3=f(x+(h/2),y+(h/2*K2))
    K4=f(x+(h/2),y+(h*K3))
    y=y+(h/6*(K1+(2*K2)+(2*K3)+K4))
    x=x+h
    x_mas.append(x)
    y_mas.append(y)
    x_mtr=np.append(x_mtr, x)
# determination of the calculation error by the Collatz method and adjustment
    of the integration step
    if abs(K2-K3)/abs(K1-K2)>=e_0:
        h=h/2; x=a; del x_mas; del x_mtr;  del y_mas
        x_mas=[a]; x_mtr=np.array(a); y_mas=[y_0]; y=y_0
# we construct graphs of the solutions of the differential equation
plt.figure(figsize=(7, 7))
plt.xlabel('x',fontsize=15, color='blue')
plt.ylabel('y',fontsize=15, color='blue')
plt.plot(x_mas, y_mas)
plt.plot(x_mtr, (2*(e**x_mtr))-x_mtr-1)
plt.legend(['Runge-Kutta method','f(x)=2*exp(x)-x-1'],loc=1)
plt.grid(True)
plt.xlim([0, 1])
plt.ylim([1, 4])
plt.show().
```

All Runge-Kutta methods are generalized to ODE systems. Let a system of differential equations be given:

$$\frac{d\overline{y}}{dx} = \overline{f}(x, \overline{y});\ x \in [a; b], \tag{3.39}$$

with initial conditions – $\overline{y}(a) = \overline{y}_0$, де $\overline{y} = \left(y_1,\ y_2,\ \ldots, y_n\right)^T$; $\overline{f} = \left(f_1, f_2,\ \ldots,\ f_n\right)^T$; $\overline{y}_0 = \left(y_{10},\ y_{20},\ \ldots, y_{n0}\right)^T$.

The value $h > 0$ is chosen, and a uniform grid is constructed:

$$\overline{\omega_n} = \left\{ x_n \middle| x_n = a + nh; n = \overline{0, N}; N = \frac{b-a}{h} \right\}.$$

The task is to determine the value of the approximate solution $\overline{y}_n = \overline{y}(x_n)$ ($n = \overline{1, N}$) according to formulas $\overline{y}_{n+1} = \overline{y}_n + \Delta\overline{y}_n$ ($n = \overline{1, N-1}$), where $\Delta\overline{y}_n$ is calculated, for example, according to the following formula:

$$\begin{cases} \overline{y}_{n+1} = \overline{y}_n + \dfrac{h}{6}\left(\overline{K}_1^{(n)} + 2\overline{K}_2^{(n)} + 2\overline{K}_3^{(n)} + \overline{K}_4^{(n)}\right); \\ \overline{K}_1^{(n)} = \overline{f}\left(x_n, \overline{y_n}\right); \\ \overline{K}_2^{(n)} = \overline{f}\left(x_n + \dfrac{h}{2}, \overline{y_n} + \dfrac{h}{2}\overline{K}_1^{(n)}\right); \\ \overline{K}_3^{(n)} = \overline{f}\left(x_m + \dfrac{h}{2}, \overline{y_m} + \dfrac{h}{2}\overline{K}_2^{(m)}\right); \\ \overline{K}_4^{(n)} = \overline{f}\left(x_n + h, \overline{y_n} + h\overline{K}_3^{(n)}\right); \\ n = \overline{0, N-1}. \end{cases} \tag{3.40}$$

So, knowing $\overline{y}_0$, $\overline{y}_1$ is calculated using formulas (3.40). Taking $\left(x_1, \overline{y}_1\right)$ as the initial data and repeating the same process, $\overline{y}_2$ is determined, etc. Similarly, any computational scheme of the Runge-Kutta method for one equation is transferred to a system of equations of the form (3.39). Example 3.4 examines the application of numerical Runge-Kutta methods for higher-order ODE, which also additionally allows determining the features of the numerical solution of ODE systems.

To estimate the total error of the numerical solution of the Cauchy problem for ODE systems, the Runge method can be used based on equation (3.26). This equation compares the results of calculating the values $\overline{y}_n$ and $\tilde{y}_n$ at the point x_n with different steps $\overline{h}_n$ and $\tilde{h}_n$, respectively:

$$R_n^h = \left(\tilde{y}_n - \overline{y}_n\right) / \left(1 - 2^{-p}\right), \tag{3.41}$$

where p=4 – the order of approximation of the difference scheme used by the fourth-order Runge-Kutta method.

The algorithm of the Runge-Kutta method for solving the ODE systems with calculation error determination and automatic step selection is presented in Figure 3.9.

Example 3.4. Use the Runge-Kutta method to obtain a numerical solution to the equation of oscillations of a pendulum in a medium that creates resistance to movement:

$$\frac{d^2\theta}{dt^2} + 0,2\frac{d\theta}{dt} + 10\sin\theta = 0; \quad t \in [0;\ 1,0] \tag{3.42}$$

under the initial conditions $\theta(0) = 0,3$; $\dfrac{d\theta}{dt}(0) = 0$ and cutoff errors $R^h = 0,001$.

In equation (3.42), $\theta(t)$ is a function of the pendulum deflection angle, which depends on time t.

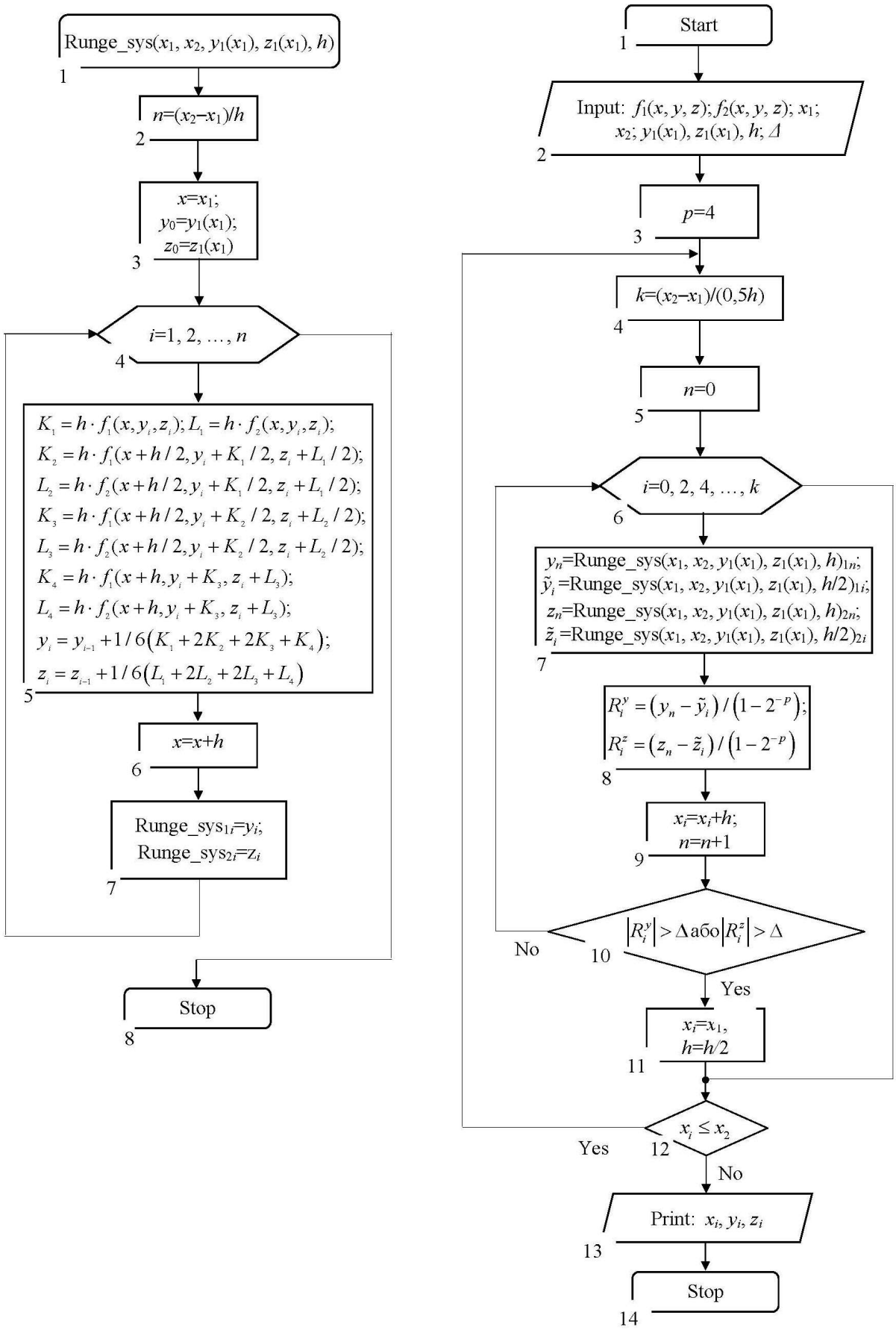


Figure 3.9 – Scheme of the fourth-order Runge-Kutta algorithm for ODE systems

Solution:

To solve this problem, the segment [0; 1,0] can be divided into ten parts by the points $x_0 = 0$; 0,1; 0,2; …, 1,0. Therefore, h=0.1. To solve the problem, substitution $z = \frac{d\theta}{dt}$ is introduced. Then equation (3.42) with the initial conditions can be presented in the form of the ODE system:

$$\begin{cases} \dfrac{d\theta}{dt} = z; \\ \dfrac{dz}{dt} = -0,2z - 10\sin\theta; \\ \theta(0) = 0,3; \quad z(0) = 0. \end{cases} \tag{3.43}$$

Also, for a comparative analysis of the accuracy of the solution obtained by using the fourth-order Runge-Kutta method, an analytical solution of the ODE type (3.42) was obtained under the assumption that $\theta \to 0$, namely:

$$\tilde{\theta}(t) = 0,3e^{-0,1t}\left[\cos(3,1607 \cdot t) + 0,03164 \cdot \sin(3,1607 \cdot t)\right]. \tag{3.44}$$

The values of θ_1, θ_2, …, θ_n and z_1, z_2, …, z_n will be determined by the Runge-Kutta method of the fourth order according to formula (3.40), namely:

$$\begin{cases} K_{1z}^{(0)} = f(\theta_0, z_0, t_0) = -0,2z_0 - 10\sin\theta_0; \\ K_{2z}^{(0)} = f(\theta_0 + \dfrac{h}{2}K_{1\theta}^{(0)}, z_0 + \dfrac{h}{2}K_{1z}^{(0)}, t_0 + \dfrac{h}{2}) = -0,2\left(z_0 + \dfrac{h}{2}K_{1z}^{(0)}\right) - 10\sin\left(\theta_0 + \dfrac{h}{2}K_{1\theta}^{(0)}\right); \\ K_{3z}^{(0)} = f(\theta_0 + \dfrac{h}{2}K_{2\theta}^{(0)}, z_0 + \dfrac{h}{2}K_{2z}^{(0)}, t_0 + \dfrac{h}{2}) = -0,2\left(z_0 + \dfrac{h}{2}K_{2z}^{(0)}\right) - 10\sin\left(\theta_0 + \dfrac{h}{2}K_{2\theta}^{(0)}\right); \\ K_{4z}^{(0)} = f(\theta_0 + hK_{3\theta}^{(0)}, z_0 + hK_{3z}^{(0)}, t_0 + h) = -0,2\left(z_0 + hK_{3z}^{(0)}\right) - 10\sin\left(\theta_0 + hK_{3\theta}^{(0)}\right); \\ z_1 = z_0 + \dfrac{h}{6}\left(K_{1z}^{(0)} + 2K_{2z}^{(0)} + 2K_{3z}^{(0)} + K_{4z}^{(0)}\right); \end{cases}$$

$$\begin{cases} K_{1z}^{(0)} = -0,2 \cdot 0 - 10 \cdot \sin 0,3 = -2,95520; \\ K_{2z}^{(0)} = -0,2 \cdot \left[0 + \dfrac{0,1}{2}(-2,9552) \right] - 10 \cdot \sin \left[0,3 + \dfrac{0,1}{2} 0 \right] = -2,92565; \\ K_{3z}^{(0)} = -0,2 \cdot \left[0 + \dfrac{0,1}{2}(-2,92565) \right] - 10 \cdot \sin \left[0,3 + \dfrac{0,1}{2}(-0,14776) \right] = -2,85529; \\ K_{4z}^{(0)} = -0,2 \cdot \left[0 + 0,1(-2,85529) \right] - 10 \cdot \sin \left[0,3 + 0,1(-0,28553) \right] = -2,75804; \\ z_1 = 0 - \dfrac{0,1}{6}\left(2,95520 + 2 \cdot 2,92565 + 2 \cdot 2,85529 + 2,75804 \right) = -0,28792; \end{cases}$$

$$\begin{cases} K_{1\theta}^{(0)} = \psi(\theta_0, z_0, t_0) = z_0; \\ K_{2\theta}^{(0)} = \psi \left[\theta_0 + \dfrac{h}{2} K_{1\theta}^{(0)}, z_0 + \dfrac{h}{2} K_{1z}^{(0)}, t_0 + \dfrac{h}{2} \right] = z_0 + \dfrac{h}{2} K_{1z}^{(0)}; \\ K_{3\theta}^{(0)} = \psi \left[\theta_0 + \dfrac{h}{2} K_{2\theta}^{(0)}, z_0 + \dfrac{h}{2} K_{2z}^{(0)}, t_0 + \dfrac{h}{2} \right] = z_0 + \dfrac{h}{2} K_{2z}^{(0)}; \\ K_{4\theta}^{(0)} = \psi \left[\theta_0 + h K_{2\theta}^{(0)}, z_0 + h K_{2z}^{(0)}, t_0 + h \right] = z_0 + h K_{3z}^{(0)}; \\ \theta_1 = \theta_0 + \dfrac{h}{6}\left(K_{1\theta}^{(0)} + 2K_{2\theta}^{(0)} + 2K_{3\theta}^{(0)} + K_{4\theta}^{(0)} \right); \end{cases}$$

$$\begin{cases} K_{1\theta}^{(0)} = z_0 = 0,00000; \\ K_{2\theta}^{(0)} = 0 + \dfrac{0,1}{2}(-2,95520) = -0,14776; \\ K_{3\theta}^{(0)} = 0 + \dfrac{0,1}{2}(-2,92565) = -0,14628; \\ K_{4\theta}^{(0)} = 0 + 0,1 \cdot (-2,85529) = -0,28553; \\ \theta_1 = 0,3 - \dfrac{0,1}{6}\left(0 + 2 \cdot 0,14776 + 2 \cdot 0,14628 + 0,28553 \right) = 0,28554; \end{cases}$$

……. .

In the process of solving, Table 3.4 was compiled with the results of the calculation using the fourth-order Runge-Kutta numerical method and the analytical solution (3.43), which are graphically interpreted in Figure 3.10. Additionally, to determine the calculation error using the Runge method (3.41), the value $\overline{\theta}_n(x_n)$ was determined with a step size of $h/2 = 0,05$.

Table 3.4 shows that the value of the angle of deviation $\theta_n(t_n)$ of the pendulum in a resisting medium decreases over time, which corresponds to the physics of the analyzed process.

Table 3.4 – Calculation results for the solution of the differential equation

n	t_n, sec	$\theta_n(t_n)$, grad	$\bar{\theta}_n(x_n)$, grad	$\frac{d\theta}{dt}(t_n), \frac{\text{grad}}{\text{sec}}$	$\tilde{\theta}(t_n)$, grad
0	0,0	0,30000	0,30000	0,00000	0,30000
1	0,1	0,28544	0,28544	– 0,28792	0,28522
2	0,2	0,24352	0,24351	– 0,54295	0,24274
3	0,3	0,17876	0,17874	– 0,74113	0,17726
4	0,4	0,09783	0,09780	– 0,86376	0,09567
5	0,5	0,00892	0,00889	– 0,89960	0,00630
6	0,6	– 0,07910	– 0,07914	– 0,84648	– 0,08191
7	0,7	– 0,15763	– 0,15767	– 0,71161	– 0,16034
8	0,8	– 0,21919	– 0,21923	– 0,51039	– 0,22147
9	0,9	– 0,25819	– 0,25821	– 0,26420	– 0,25964
10	1,0	– 0,27135	– 0,27136	0,00225	– 0,27157

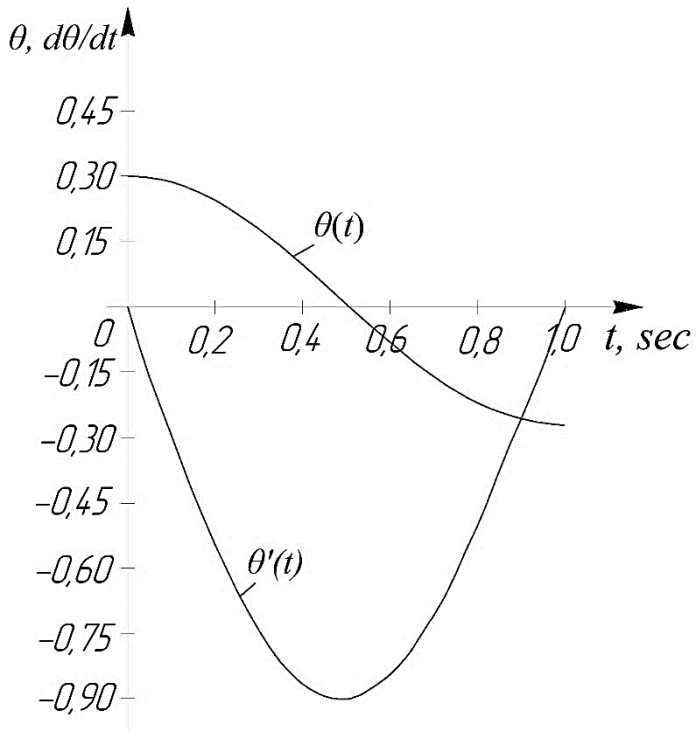


Figure 3.10 – Diagram of numerical solution results for higher-order Cauchy problems of ODE using the fourth-order Runge-Kutta method

The obtained results, given in Table 3.4, show a high degree of accuracy in determining the solution, particularly at the points $t_3 = 0,3$ sec i $t_{10} = 1,0$ sec:

– relative error by comparing the values of the analytical solution and the Runge-Kutta method of the fourth order

$$\varepsilon_{t_3} = \left| \frac{\tilde{\theta}_{t_3} - \theta_{t_3}}{\tilde{\theta}_{t_3}} \right| \cdot 100\% = \left| \frac{0{,}17876 - 0{,}17726}{0{,}17726} \right| \cdot 100\% = 0{,}850\,\%,$$

$$\varepsilon_{t_{10}} = \left| \frac{\tilde{\theta}_{t_{10}} - \theta_{t_{10}}}{\tilde{\theta}_{t_{10}}} \right| \cdot 100\% = \left| \frac{-0{,}27157 - (-0{,}27135)}{-0{,}27157} \right| \cdot 100\% = 0{,}081\,\%;$$

– error in the Runge method's approximation (see 3.41)

$$R^h_{x_3} = \left(\theta_{x_3} - \overline{\theta}_{x_3}\right) / \left(1 - 2^{-p}\right) = \left(0{,}17876 - 0{,}17874\right) / \left(1 - 2^{-4}\right) = 2{,}13 \cdot 10^{-5},$$

$$R^h_{x_{10}} = \left(\theta_{x_{10}} - \overline{\theta}_{x_{10}}\right) / \left(1 - 2^{-p}\right) = \left(-0{,}27135 - (-0{,}27136)\right) / \left(1 - 2^{-4}\right) = 1{,}07 \cdot 10^{-5}.$$

Consider the implementation of the fourth-order Runge-Kutta method for systems of differential equations in the PYTHON programming language.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import *
from sympy import *
# entering the integration step
h_0=float(input())
# entering the initial value of the integration argument (total integration
    interval) t0
a=int(input()); t_mtr=np.array(a)
#entering the initial value of the argument of the function of the
    differential equation q0
q_0=float(input()); q_mtr=np.array(q_0)
# entering the initial value of the argument of the differential equation
    function z0
z_0=float(input()); z_mtr=np.array(z_0)
#entering the final value of the total integration interval
b=int(input())
# entering the permissible calculation error
e_0=float(input())
# setting the functions of the differential equation system
def f1(z):
    return z
def f2(q,z):
    return (-0.2*z)-(10*math.sin(q))
# calculation of iterative formulas
def runge(a,b,q_0,z_0,h_0):
    t=a; q=q_0; z=z_0
    t_mas=[t]; q_mas=[q]; z_mas=[z]
    t_mtr=np.array(t)
    q_mtr=np.array(q_0)
    z_mtr=np.array(z_0)
    h=h_0
    while t<=b:
        # iterative formulas of the Runge-Kutta method
        K1_Z=f2(q,z); K1_Q=f1(z)
        K2_Z=f2(q+(h/2*K1_Q),z+(h/2*K1_Z)); K2_Q=f1(z+(h/2*K1_Z))
        K3_Z=f2(q+(h/2*K2_Q),z+(h/2*K2_Z)); K3_Q=f1(z+(h/2*K2_Z))
        K4_Z=f2(q+(h*K3_Q),z+(h*K3_Z)); K4_Q=f1(z+(h*K3_Z))
        z=z+(h/6*(K1_Z+(2*K2_Z)+(2*K3_Z)+K4_Z))
        q=q+(h/6*(K1_Q+(2*K2_Q)+(2*K3_Q)+K4_Q))
        t=t+h
        q_mas.append(q)
```

```
        z_mas.append(z)
        t_mas.append(t)
        t_mtr=np.append(t_mtr, t)
    return t_mas, q_mas, z_mas, t_mtr
# determination of the calculation error by the Runge method and adjustment
     of the integration step
p=4; x=a
while x<=b:
    n=0
    for k in range(0,len(runge(a,b,q_0,z_0,h_0/2)[1]),2):
        Rh_q=(runge(a,b,q_0,z_0,h_0)[1][n]-
     runge(a,b,q_0,z_0,h_0/2)[1][k])/(1-(1/2**p))
        Rh_z=(runge(a,b,q_0,z_0,h_0)[2][n]-
     runge(a,b,q_0,z_0,h_0/2)[2][k])/(1-(1/2**p))
        #print(Rh_q)
        #print(Rh_z)
        n=n+1
        x=x+h_0
        if abs(Rh_q>e_0) and abs(Rh_z>e_0):
            h_0=h_0/2; x=a
            break
# we construct graphs of the solutions of the differential equation
plt.figure(figsize=(7, 7))
plt.xlabel('t',fontsize=15, color='blue')
plt.ylabel('psi, dPsi(t)/dt',fontsize=15, color='blue')
plt.plot(runge(a,b,q_0,z_0,h_0)[0], runge(a,b,q_0,z_0,h_0)[1])
plt.plot(runge(a,b,q_0,z_0,h_0)[0], runge(a,b,q_0,z_0,h_0)[2])
plt.plot(runge(a,b,q_0,z_0,h_0)[3], 0.3*(e**(-
     0.1*runge(a,b,q_0,z_0,h_0)[3]))*
          (np.cos(3.1607*runge(a,b,q_0,z_0,h_0)[3])+
           (0.03164*np.sin(3.1607*runge(a,b,q_0,z_0,h_0)[3]))))
plt.legend(['The Runge-Kutta method for Psi(t)',' The Runge-Kutta method
     for dPsi(t)/dt',' The exact solution], loc=1)
plt.grid(True)
plt.xlim([0, 1])
plt.ylim([-0.95, 0.45])
plt.show().
```

It should be noted in Table 3.4 that at other time intervals t_n, there is a significant discrepancy between the results of numerical and analytical calculations. This discrepancy is caused by obtaining the analytical calculation (3.43) based on the assumption that $\theta \to 0$.

The above-mentioned numerical one-step methods for solving the Cauchy problem for ODE can be summarized by the following characteristics.

1) to obtain information at a new point, data from only one previous point is required;

2) all one-step methods are based on the decomposition of the function into a Taylor series, in which the terms containing the step h to the power of n inclusive are stored. An integer is called the order of the method, and the step error has the order of $m+1$;

3) one-step methods do not require the calculation of derivatives because only the function is calculated, but its value at several intermediate points may be required;

4) there is a possibility of changing the size of the calculation step.

Adams' method

In one-step methods for solving the Cauchy problem for a single ODE:

$$\frac{du}{dx} = f(x,u); \quad x > x_0; \quad u(x_0) = u_0, \tag{3.45}$$

which is considered correctly set, the value of u_{n+1} depends only on the information about the solution at the previous grid point x_n ($n = 0, 1, 2, \ldots$).

To increase accuracy, information about the solution at several previous grid points x_n, x_{n-1}, x_{n-2}, … can be used. Moreover, it is expedient to use the information with a forward run beyond the point x_{n+1}.

In multi-step, as well as single-step, methods, it is advisable to use a constant step size in the calculation grid:

$$\overline{\omega_h} = \left\{ x_m \middle| x_m = a + mh; m = \overline{0, N}; N = \frac{b-a}{h} \right\}. \tag{3.46}$$

The grid functions $y_n = y(x_n)$, $f_n = f(x_n, y_n)$, $u_n = u(x_n)$ defined on grid $\overline{\omega_h}$ (3.46) are entered. A linear n-step difference method is called a system of difference equations:

$$\frac{a_0 y_n + a_1 y_{n-1} + a_2 y_{n-2} + \ldots + a_m y_{n-m}}{h} = b_0 f_n + b_1 f_{n-1} + \ldots + b_m f_{n-m}, \tag{3.47}$$

or in the compact form $\sum_{k=0}^{m} \frac{a_k y_{n-k}}{h} = \sum_{k=0}^{m} b_k f_{n-k}$, which is defined for $n = m$, (m+1), …, where a_k, b_k are numerical coefficients that do not depend on n ($k = 0, 1, \ldots, m$), and $a_0 \neq 0$. Equation (3.47) is a recurrence relation for the determination of the new value $y_n = y(x_n)$ using the previously found values $y_{n-1}, y_{n-2}, \ldots, y_{n-m}$.

A partial case of multi-step methods (3.47) is the presence of the condition when the derivative $u'(x)$ is approximated only by two points x_n and x_n–1, that is, the coefficients ak acquire the following values: $a_0 = -a_1 = 1$; $a_k = 0$ ($k = 2, 3, \ldots, m$). In this case, the Adams' method is used, which generally has the form:

$$\frac{y_n - y_{n-1}}{h} = \sum_{k=0}^{m} b_k f_{n-k}. \tag{3.48}$$

for $b_0 = 0$, the methods are called explicit, and the order of approximation is equal to m. For $b_0 \neq 0$, the methods are called implicit, and the order of approximation is equal to m+1.

For explicit m-step Adams' methods, the coefficients of the highest-order method of approximation of equations (3.48) are determined for each value of m.

In particular, for m=4, an explicit method of the fourth order of approximation will be obtained:

$$\frac{y_n - y_{n-1}}{h} = \frac{1}{24}\left(55f_{n-1} - 59f_{n-2} + 37f_{n-3} - 9f_{n-4}\right) \tag{3.49}$$

with a local error $\rho = \frac{251}{720}h^5u^{(5)}(x)$.

Formula (3.49) is known as the fourth-order Adams-Bashforth method. The coefficients of the highest order approximation of equation (3.48) in this method can be obtained by using information from an even larger number of previous points, which also allows for obtaining the high-order Adams-Bashforth method. However, the accuracy of calculations increases non-linearly with increasing order (the greater the distance between the previous point and the current one, the weaker its effect on the accuracy).

Multi-step methods give rise to the same problem as single-step methods. Since multi-step methods use information about previously obtained points, unlike single-step methods, they do not have the property of being «self-starting». Therefore, before applying the multi-step method, it is necessary to calculate the raw data using single-step methods, such as the Euler or Runge-Kutta methods.

In implicit m-step Adams' methods, for each value of m, the coefficients of the method of the highest order of approximation of equations (3.48) are determined, which are equal to (m+1). In this case, the next class of implicit computing methods arises, which are called Adams-Moulton methods of the second, third, and fourth orders, respectively. In particular, for m=3, the fourth-order approximation method $O(h^4)$ will be obtained.

$$\frac{y_n - y_{n-1}}{h} = \frac{1}{24}\left(9f_n + 19f_{n-1} - 5f_{n-2} + f_{n-3}\right) \tag{3.50}$$

with a local error $\rho = -\frac{19}{720}h^5u^{(5)}(x)$.

In the formula (3.50), the value of f_n is unknown, since for the calculation $f(x_n, y_n) = f_n$, three values of y_n that are unknown must be used. Accordingly, the Adams-Moulton methods determine the value of y_n implicitly. On the other hand, the Adams-Bashforth methods are called explicit because the process of determining the value of y_n does not require solving any equations. Therefore, in practice, a common explicit and implicit formula is used when solving the ODE, which leads to the application of the «prediction and correction» method (a combination of fourth-order Adams' methods):

$$\begin{cases} y^*_n = y_{n-1} + \dfrac{h}{24}\left(55f_{n-1} - 59f_{n-2} + 37f_{n-3} - 9f_{n-4}\right); \\ f^*_n = f\left(x_n, y^*_n\right); \\ y_n = y_{n-1} + \dfrac{h}{24}\left(9f^*_n + 19f_{n-1} - 5f_{n-2} + f_{n-3}\right). \end{cases} \quad (3.51)$$

In general, the method of «prediction and correction» is clear. First, the Adams-Bashforth formula (3.49) calculates the value of y^*_n in (3.51), which is a «prediction» for y_n. The value of y^*_n is then used to calculate the approximate value of f^*_n in (3.51), which is also used in the Adams-Moulton formula (3.50). Thus, the Adams-Moulton formula «corrects» the approximation defined by the Adams-Bashforth formula (3.49). Since the series intercept errors for the Adams-Bashforth prediction formulas y^*_n (3.49) are $\rho = \frac{251}{720}h^5u^{(5)}(x)$, and for the Adams-Moulton adjustment y_n (3.50) are $\rho = -\frac{19}{720}h^5u^{(5)}(x)$, this allows further reduction of the calculation error by 3%.

The correction equations are more accurate than the forecasting formulas, allowing you to generally increase the accuracy of the calculation of the ODE, despite the occurrence of additional calculations. Additionally, to achieve the highest calculation accuracy, the correction process in the «prediction and correction» methods can be repeated several times at the same iteration step to obtain a value with a certain specified accuracy using the absolute error $\left|y^*_n - y^{(i)}_n\right| \le \Delta$, where $y^{(i)}_n$ represents the current value of the solution by the Adams-Moulton method (3.50) at a certain iterative step of the general method of «prediction and correction» methods.

The algorithm of the Adams method is presented in Figure 3.11.

Adams' method («prediction and correction») has been successfully generalized to the Cauchy problem of ODE systems. Let a system of differential equations be given:

$$\frac{d\overline{y}}{dx} = \overline{f}(x, \overline{y}); \; x \in [a; b], \quad (3.52)$$

with initial conditions $-\overline{y}(a) = \overline{y}_0$, де $\overline{y} = \left(y_1, \; y_2, \; \ldots, y_n\right)^T$, $\overline{f} = \left(f_1, f_2, \; \ldots, \; f_n\right)^T$, $\overline{y}_0 = \left(y_{10}, \; y_{20}, \; \ldots, y_{n0}\right)^T$.

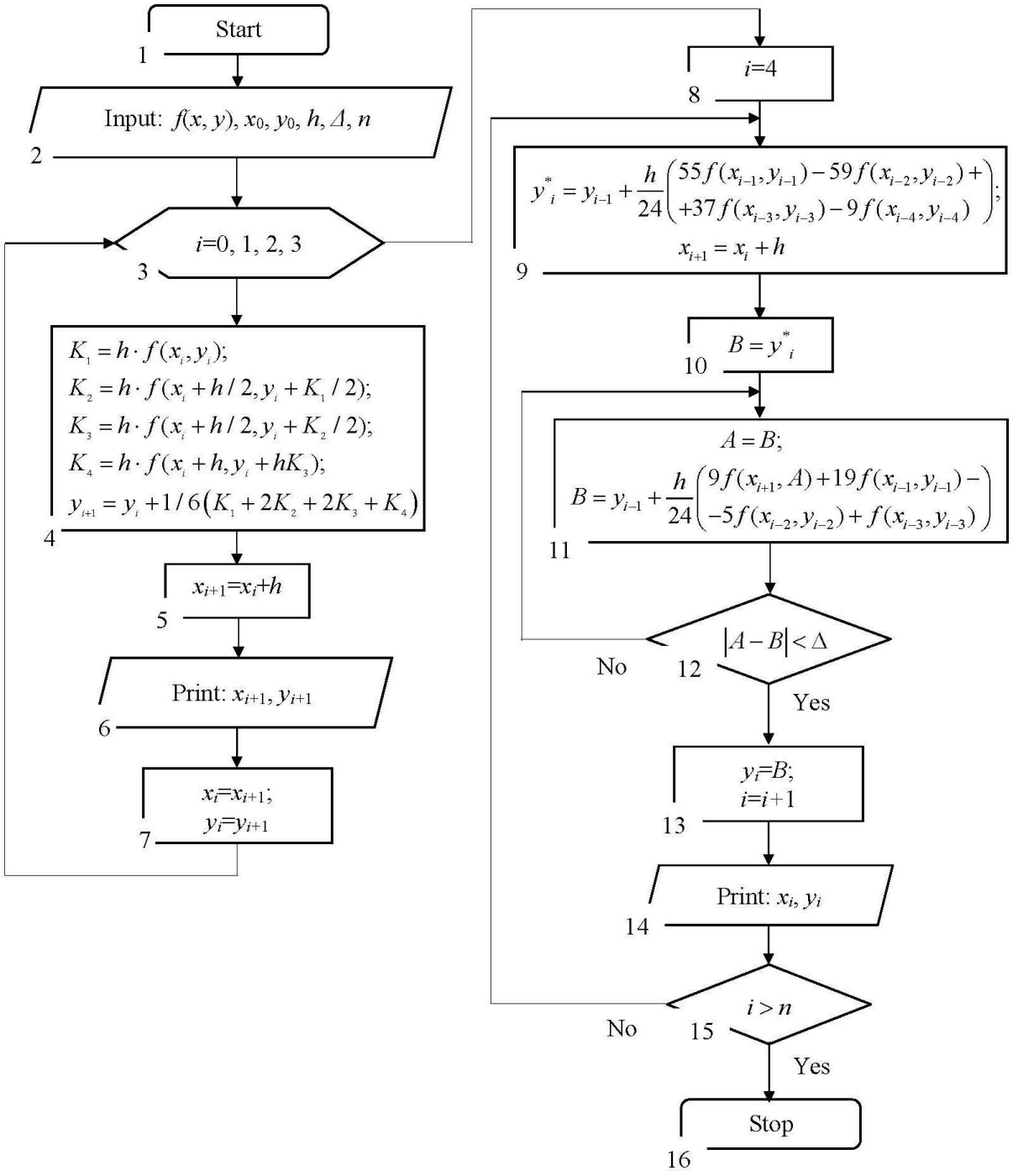


Figure 3.11 – Scheme of the Adams' method algorithm

The value $h > 0$ is chosen and a uniform grid is constructed:

$$\overline{\omega_n} = \left\{ x_n \middle| x_n = a + nh;\ n = \overline{0, N};\ N = \frac{b-a}{h} \right\}.$$

The problem is to determine the value of the approximate solution $\overline{y}_n = \overline{y}(x_n)$ ($n = \overline{1, N}$) using the following formulas (generalization of formula (3.51) for the case of differential equations):

$$\begin{cases} \overline{y^*}_m = \overline{y_{m-1}} + \dfrac{h}{24}\left(55\overline{f_{m-1}} - 59\overline{f_{m-2}} + 37\overline{f_{m-3}} - 9\overline{f_{m-4}}\right); \\ \overline{f_m^*} = \overline{f}\left(x_m, \overline{y_m^*}\right); \\ \overline{y_m} = \overline{y_{m-1}} + \dfrac{h}{24}\left(9\overline{f_m^*} + 19\overline{f_{m-1}} - 5\overline{f_{m-2}} + \overline{f_{m-3}}\right) \quad (m = \overline{1, N-1}). \end{cases} \tag{3.53}$$

So, having obtained values $\overline{y}_0$, $\overline{y}_1$, $\overline{y}_2$ $\overline{y}_3$ and $\overline{f}_0$, $\overline{f}_1$, $\overline{f}_2$, $\overline{f}_3$ according to formulas (3.40), values $\overline{y}_4$ and $\overline{f}_4$ are calculated according to formulas (3.53). Taking $\left(x_4, \overline{y}_4\right)$ and $\overline{f}\left(x_4, \overline{y}_4\right)$ as inputs and repeating the same process, $\overline{y}_5$ is determined, and so on.

The algorithm of the Adams' method for solving the ODE systems is presented in Figure 3.12.

Example 3.5. To solve the Cauchy problem for the first-order ODE using the numerical method of «prediction and correction» based on Adams' iterative formulas:

$$\frac{dy}{dx} = y + x\,; \; x \in [0;\, 1{,}0], \tag{3.54}$$

which satisfies the initial condition for x_0=0, y_0=1,0 and the absolute error at the adjustment stage Δ=10^{-4}.

Solution:

To solve this problem, the segment [0; 1,0] can be divided into ten parts by points x_0=0; 0,1; 0,2; …, 1,0. Accordingly, h=0,1. To begin with, the value of $\overline{y}_0$, $\overline{y}_1$, $\overline{y}_2$, $\overline{y}_3$ is determined by the Runge-Kutta method of the fourth order in example 3.3 (see Table 3.3). Also, for comparison, the value of $\tilde{y}_1$, $\tilde{y}_2$, $\tilde{y}_3$, $\tilde{y}_4$ of the analytical solution of the ODE (3.54) was obtained and summarized in Table 3.5. The application of the numerical method of «prediction and correction» begins with obtaining the solutions of the ODE by the explicit Adams-Bashforth method (3.49) of the difference scheme (3.51). To begin with, the «initial segment» from Table 3.5 is selected, namely: x_0=0,0; x_1=0,1; x_2=0,2; x_3=0,3; $\overline{y}_0$=1,0; $\overline{y}_1$=1,11034; $\overline{y}_2$=1,24280; $\overline{y}_3$=1,39971.

Then the «prediction» calculation is performed (using the explicit Adams-Bashforth method (3.49)):

$$\begin{cases} \overline{\overline{y^*}}_4(x_4) = \overline{y}_3(x_3) + \dfrac{h}{24}\left(55f(x_3, \overline{y_3}) - 59f(x_2, \overline{y_2}) + 37f(x_1, \overline{y_1}) - 9f(x_0, \overline{y_0})\right); \\ \overline{\overline{f}}_4^* = f\left(x_4, \overline{\overline{y}}_4^*\right) = x_4 + \overline{\overline{y^*}}_4(x_4); \end{cases}$$

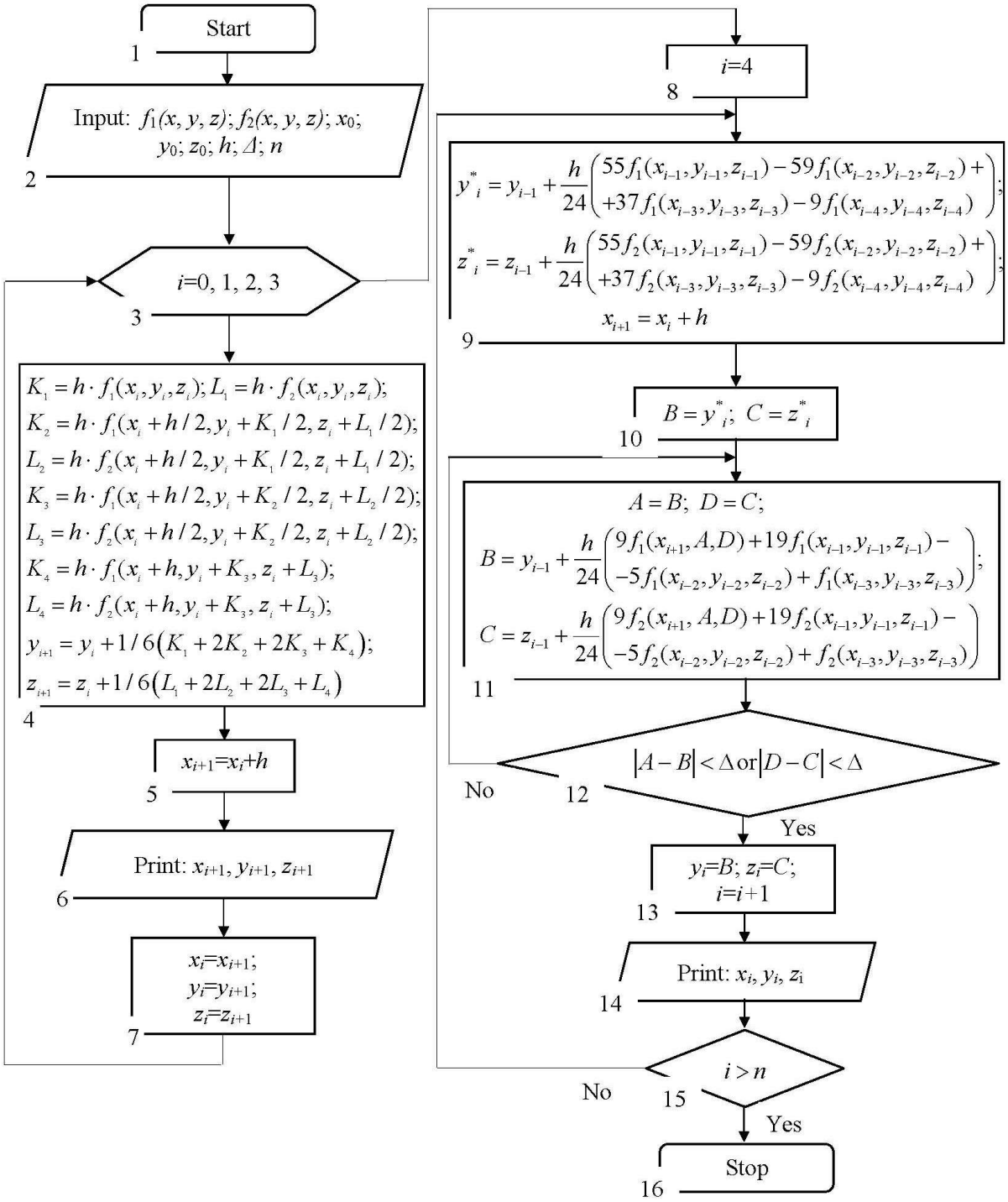


Figure 3.12 – Scheme of the Adams' method algorithm for ODE systems

$$\begin{cases} \overline{\overline{y^*}}_4(x_4) = 1{,}39971 + \frac{0{,}1}{24}\begin{pmatrix} 55(0{,}3 + 1{,}39971) - 59(0{,}2 + 1{,}24280) + \\ +37(0{,}1 + 1{,}11034) - 9(0 + 1{,}0) \end{pmatrix} = 1{,}58365; \\ \overline{\overline{f^*_4}} = f\left(x_4, \overline{\overline{y^*_4}}\right) = 0{,}4 + 1{,}58365 = 1{,}98363; \end{cases}$$

$$\begin{cases} \overline{\overline{y^*}}_5(x_5) = \overline{\overline{y}}_4(x_4) + \dfrac{h}{24}\left(55f(x_4, \overline{\overline{y_4^*}}) - 59f(x_3, \overline{y_3}) + 37f(x_2, \overline{y_2}) - 9f(x_1, \overline{y_1})\right); \\ \overline{\overline{f_5^*}} = f\left(x_5, \overline{\overline{y_5^*}}\right) = x_5 + \overline{\overline{y^*}}_5(x_5); \end{cases}$$

$$\begin{cases} \overline{\overline{y^*}}_5(x_5) = 1,58364 + \dfrac{0,1}{24}\begin{pmatrix} 55(0,4+1,58365) - 59(0,3+1,39972) + \\ +37(0,2+1,24280) - 9(0,1+1,11034) \end{pmatrix} = 1,79741; \\ \overline{\overline{f_5^*}} = f\left(x_5, \overline{\overline{y_5^*}}\right) = 0,5 + 1,79741 = 2,29741; \end{cases}$$

……. .

To correct the results obtained by the explicit Adams-Bashforth method (3.49), the implicit Adams-Moulton method (3.50) of the difference scheme (3.51) is used. To begin with, the «initial segment» from table 3.5 is chosen, namely: x_0=0,0; x_1=0,1; x_2=0,2; x_3=0,3; $\overline{y}_0$=1,0; $\overline{y}_1$=1,11034; $\overline{y}_2$=1,24280; $\overline{y}_3$=1,39971. Then the «correction» calculation is performed (implicit Adams-Moulton method (3.50)):

$$\overline{\overline{y}}_4(x_4) = \overline{y_3}(x_3) + \frac{h}{24}\left(9\overline{\overline{f^*}}(x_4, \overline{\overline{y_4^*}}) + 19f(x_3, \overline{y_3}) - 5f(x_2, \overline{y_2}) + f(x_2, \overline{y_2})\right) =$$

$$= 1,39971 + \frac{0,1}{24}\left(9 \cdot 1,98363 + 19 \cdot 1,39972 - 5 \cdot 1,24280 + 1,11034\right) = 1,58364;$$

$$\overline{\overline{y}}_5(x_5) = \overline{\overline{y}}_4(x_4) + \frac{h}{24}\left(9\overline{\overline{f^*}}(x_5, \overline{\overline{y_5^*}}) + 19\overline{\overline{f^*}}(x_4, \overline{\overline{y_4}}) - 5f(x_3, \overline{y_3}) + f(x_2, \overline{y_2})\right) =$$

$$= 1,39971 + \frac{0,1}{24}\left(9 \cdot 2,29741 + 19 \cdot 1,98363 - 5 \cdot 1,39972 + 1,24280\right) = 1,79743;$$

……. .

Also, to achieve the highest calculation accuracy, the correction process (implicit Adams–Moulton method (3.50)) can be repeated several times at the same iteration step to obtain a value with a certain specified accuracy Δ (absolute error), which was implemented in the PYTHON programming language (see Fig. 3.12):

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import *
from sympy import *
# entering the integration step
h=float(input())
# entering the initial value of the integration argument (total integration
    interval)
a=int(input()); x_mtr=np.array(a)
# entering the initial value of the argument of the differential equation
    function
```

```
y_0=int(input()); y_mtr=np.array(y_0)
# entering the final value of the total integration interval
b=int(input())
# entering the value of the calculation accuracy by the correction formula
     of the Adams method
e=float(input())
# setting the function of the differential equation
def f(x,y):
    return x+y
# determination of the first three values of the solution of the differential
     equation
x=a; y=y_0
x_mas=[x]; y_mas=[y]
x_mtr=np.array(a)
y_mtr=np.array(y_0)
while x<3*h:
    # iterative formula of the Runge-Kutta method
    K1=f(x,y); K2=f(x+(h/2),y+(h/2*K1))
    K3=f(x+(h/2),y+(h/2*K2))
    K4=f(x+(h/2),y+(h*K3))
    y=y+(h/6*(K1+(2*K2)+(2*K3)+K4))
    x=x+h
    x_mas.append(x)
    y_mas.append(y)
    x_mtr=np.append(x_mtr, x)
i=4
while x<=b-h:
    # iterative prediction formula of the Adams method
    y=y_mas[i-1]+(h/24*(55*f(x_mas[i-1],y_mas[i-1])-59*f(x_mas[i-
     2],y_mas[i-2])+
       37*f(x_mas[i-3],y_mas[i-3])-9*f(x_mas[i-4],y_mas[i-4])))
    x=x+h
    f_tran=y
    s_tran=y+1
    # iterative correction formula of the Adams method
    while abs(s_tran-f_tran)<=e:
        s_tran=f_tran
        f_tran=y_mas[i-1]+(h/24*(9*f(x,s_tran)+19*f(x_mas[i-1],y_mas[i-1])-
                                 5*f(x_mas[i-2],y_mas[i-2])+f(x_mas[i-
     3],y_mas[i-3])))
    i=i+1
    x_mas.append(x)
    y_mas.append(f_tran)
    x_mtr=np.append(x_mtr, x)
    x_mtr=np.append(x_mtr, x)
# we construct graphs of the solutions of the differential equation
plt.figure(figsize=(7, 7))
plt.xlabel('x',fontsize=15, color='blue')
plt.ylabel('y',fontsize=15, color='blue')
plt.plot(x_mas, y_mas)
plt.plot(x_mtr, (2*(e**x_mtr))-x_mtr-1)
plt.legend([' Adams method','f(x)=2*exp(x)-x-1'],loc=1)
plt.grid(True)
plt.xlim([0, 1])
plt.ylim([1, 4])
plt.show().
```

From the obtained results, given in Table 3.5, it is clear that a high degree of absolute and relative error is allowed in the process of determining the solution, particularly in points x_5=0,5 i x_{10}=1,0:

– the relative error is obtained by comparing the values of the analytical solution and the Adams' method

$$\varepsilon_{x_5} = \left| \frac{\tilde{y}_{x_5} - \overline{=} y_{x_5}}{\tilde{y}_{x_5}} \right| \cdot 100\% = \left| \frac{1,79744 - 1,79743}{1,79744} \right| \cdot 100\% = 5,56 \cdot 10^{-4} \ \%,$$

$$\varepsilon_{x_{10}} = \left| \frac{\tilde{y}_{x_{10}} - \overline{\overline{y}}_{x_{10}}}{\tilde{y}_{x_{10}}} \right| \cdot 100\% = \left| \frac{3,43656 - 3,43652}{3,43656} \right| \cdot 100\% = 1,16 \cdot 10^{-3}\ \%;$$

– absolute error by comparing the values of «prediction» $\overline{\overline{y}}^{*}_{n}(x_n)$ (Adams-Bashforth) and «adjustment» $\overline{\overline{y}}_{n}(x_n)$ (Adams-Moulton)

$$\Delta_{x_5} = \left| \overline{\overline{y}}_{x_5} - \overline{\overline{y}}^{*}_{x_5} \right| = \left| 1,79743 - 1,79741 \right| = 2,0 \cdot 10^{-5},$$

$$\Delta_{x_{10}} = \left| \overline{\overline{y}}_{x_{10}} - \overline{\overline{y}}^{*}_{x_{10}} \right| = \left| 3,43652 - 3,43644 \right| = 8,0 \cdot 10^{-5}.$$

Table 3.5 – Calculation results for the solution of the differential equation

n	x_n	$\overline{y}_n(x_n)$	$\tilde{y}_n(x_n)$	$\overline{\overline{y}}^{*}_{n}(x_n)$	$\overline{\overline{y}}_{n}(x_n)$
0	0,0	1,00000	1,00000	1,00000	1,00000
1	0,1	1,11034	1,11034	1,11034	1,11034
2	0,2	1,24280	1,24281	1,24280	1,24280
3	0,3	1,39971	1,39972	1,39971	1,39971
4	0,4	1,58364	1,58365	1,58363	1,58364
5	0,5	1,79743	1,79744	1,79741	1,79743
6	0,6	2,04423	2,04424	2,04410	2,04423
7	0,7	2,32750	2,32751	2,32745	2,32749
8	0,8	2,65107	2,65108	2,65100	2,65106
9	0,9	3,01919	3,01921	3,01911	3,01918
10	1,0	3,43655	3,43656	3,43644	3,43652

Also, in the «prediction and correction» numerical method itself, it is possible to observe an increase in the accuracy of the calculation using the «correction» process (Adams-Moulton methods) based on the previously obtained results of the numerical solution of the ODE based on the «prediction» iterative equation (Adams-Bashforth method) problems of Cauchy ODE.

Since the solution of the Cauchy problem using the «prediction and correction» method requires the initial values of the solution of the ODE system, the data from the solution obtained by applying the fourth-order Runge-Kutta method in Example 3.4 are necessary for the example.

Example 3.6. To solve the problem of oscillations of a pendulum in an environment that creates resistance to movement, we will use the numerical method of «prediction and correction» based on Adams' iterative formulas:

$$\frac{d^2\theta}{dt^2} + 0,2\frac{d\theta}{dt} + 10\sin\theta = 0;\ t \in [0;\ 1,0], \qquad (3.55)$$

under the initial conditions $\theta(0) = 0{,}3$; $\frac{d\theta}{dt}(0) = 0$ and with an absolute error at the adjustment stage $\Delta = 10^{-3}$, equation (2.60) represents $\theta(t)$ as a function of the pendulum deflection angle (grad) that varies with time t (sec).

Solution:

To solve this problem, the segment [0; 1,0] can be divided into ten parts by points x_0=0; 0,1; 0,2; …, 1,0. Accordingly, $h = 0{,}1$. To solve the problem, a substitution $z = \frac{d\theta}{dt}$ is introduced. Then equation (3.55) with the initial conditions can be written in the form of the ODE system:

$$\begin{cases} \frac{d\theta}{dt} = z; \\ \frac{dz}{dt} = -0{,}2z - 10\sin\theta; \\ \theta(0) = 0{,}3; \quad z(0) = 0. \end{cases} \tag{3.56}$$

To begin with, the values $\overline{\theta_0}$, $\overline{\theta_1}$, $\overline{\theta_2}$, $\overline{\theta_3}$ and $\overline{z_0}$, $\overline{z_1}$, $\overline{z_2}$, $\overline{z_3}$ are determined by the Runge-Kutta method of the fourth order, as shown in example 3.4 and listed in Table 3.4. Additionally, for comparison, the value of $\tilde{\theta}_0$, $\tilde{\theta}_1$, $\tilde{\theta}_2$, $\tilde{\theta}_3$ from the analytical solution of ODE (3.43) was obtained and is given in Table 3.4. The application of the numerical method of «prediction and correction» starts with obtaining solutions of the ODE using explicit Adams-Bashforth methods of the difference scheme (3.53). Initially, the «initial segment» from Table 3.6 is selected, namely: t_0=0; t_1=0,1; t_2=0,2; t_3=0,3; $\overline{\theta_0}$ =0,3; $\overline{\theta_1}$ =0,28544; $\overline{\theta_2}$ =0,24352; $\overline{\theta_3}$ =0,17876; $\overline{z_0}$ =0; $\overline{z_1}$ =-0,28792; $\overline{z_2}$ =-0,54295; $\overline{z_3}$ =-0,74113. After that, the «prediction» calculation is performed using the explicit Adams-Bashforth method of the difference scheme (3.53):

$$\begin{cases} \overline{\overline{\theta_4^*}}(t_4) = \overline{\theta}_3(t_3) + \frac{h}{24}\left(55 f(t_3, \overline{z_3}, \overline{\theta_3}) - 59 f(t_2, \overline{z_2}, \overline{\theta_2}) + 37 f(t_1, \overline{z_1}, \overline{\theta_1}) - 9 f(t_0, \overline{z_0}, \overline{\theta_0})\right); \\ \overline{\overline{z_4^*}}(t_4) = \overline{z}_3(t_3) + \frac{h}{24}\left(55\xi(t_3, \overline{z_3}, \overline{\theta_3}) - 59\xi(t_2, \overline{z_2}, \overline{\theta_2}) + 37\xi(t_1, \overline{z_1}, \overline{\theta_1}) - 9\xi(t_0, \overline{z_0}, \overline{\theta_0})\right); \\ \overline{\overline{f_4^*}} = f\left(t_4, \overline{\overline{z_4^*}}, \overline{\overline{\theta_4^*}}\right) = \overline{\overline{z_4^*}}(t_4); \\ \overline{\overline{\xi_4^*}} = \xi\left(t_4, \overline{\overline{z_4^*}}, \overline{\overline{\theta_4^*}}\right) = -0{,}2\overline{\overline{z_4^*}} - 10\sin(\overline{\overline{\theta_4^*}}); \end{cases}$$

$$\begin{cases} \overline{\overline{\theta_4^*}}(t_4) = 0{,}17876 + \dfrac{0{,}1}{24}\begin{pmatrix} 55\,(-0{,}74113) - 59(-0{,}54295) + \\ +37(-0{,}28792) - 9\cdot 0 \end{pmatrix} = 0{,}09801; \\ \overline{\overline{z_4^*}}(t_4) = -0{,}74113 + \dfrac{0{,}1}{24}\begin{pmatrix} 55\cdot(-0{,}2\cdot(-0{,}74113) - 10\sin(0{,}17876)) - \\ -59\cdot(-0{,}2\cdot(-0{,}54295) - 10\sin(0{,}24352)) + \\ +37\cdot(-0{,}2\cdot(-0{,}28792) - 10\sin(0{,}28544)) - \\ -9\cdot(-0{,}2\cdot 0 - 10\sin(0{,}3)) \end{pmatrix} = -0{,}86299; \\ \overline{\overline{f_4^*}} = \overline{\overline{z^*}}_4(t_4) = -0{,}86299; \\ \overline{\overline{\xi_4^*}} = -0{,}2\overline{\overline{z_4^*}} - 10\sin(\overline{\overline{\theta^*}}_4) = -0{,}2(-0{,}86299) - 10\sin(0{,}09801) = -0{,}80589; \end{cases}$$

…… .

To correct the results obtained by the explicit Adams-Bashfort method, the (implicit) Adams-Moulton method of the difference scheme (3.53) is used. To begin with, the «initial segment» from Table 3.6 is chosen, namely: t_0=0; t_1=0,1; t_2=0,2; t_3=0,3; $\overline{\theta_0}$=0,3; $\overline{\theta_1}$=0,28544; $\overline{\theta_2}$=0,24352; $\overline{\theta_3}$=0,17876; $\overline{z_0}$=0; $\overline{z_1}$=-0,28792; $\overline{z_2}$=-0,54295; $\overline{z_3}$=-0,74113. Then the «correction» (implicit Adams–Moulton method) of the difference scheme (3.53) is calculated:

$$\begin{cases} \overline{\overline{\theta_4}}(t_4) = \overline{\theta_3}(t_3) + \dfrac{h}{24}\left(9\overline{\overline{f^*}}(t_4, \overline{\overline{z_4}}, \overline{\overline{\theta_4}}) + 19 f\,(t_3, \overline{z_3}, \overline{\theta_3}) - 5 f\,(t_2, \overline{z_2}, \overline{\theta_2}) + f\,(t_1, \overline{z_1}, \overline{\theta_1})\right); \\ \overline{\overline{z_4}}(t_4) = \overline{z_3}(t_3) + \dfrac{h}{24}\left(9\overline{\overline{\xi^*}}(t_4, \overline{\overline{z_4}}, \overline{\overline{\theta_4}}) + 19\xi\,(t_3, \overline{z_3}, \overline{\theta_3}) - 5\xi\,(t_2, \overline{z_2}, \overline{\theta_2}) + \xi\,(t_1, \overline{z_1}, \overline{\theta_1})\right); \end{cases}$$

$$\begin{cases} \overline{\overline{\theta_4}}(t_4) = 0{,}17876 + \dfrac{0{,}1}{24}\begin{pmatrix} 9\cdot(0{,}86299) + 19\cdot(-0{,}74113) - \\ -5\cdot(-0{,}54295) + (-0{,}28792) \end{pmatrix} = 0{,}097840; \\ \overline{\overline{z_4^*}}(t_4) = -0{,}74113 + \dfrac{0{,}1}{24}\begin{pmatrix} 9\cdot(-0{,}80589) + \\ +19\cdot(-0{,}2\cdot(-0{,}74113) - 10\sin(0{,}17876)) - \\ -5\cdot(-0{,}2\cdot(-0{,}54295) - 10\sin(0{,}24352)) + \\ +(-0{,}2\cdot(-0{,}28792) - 10\sin(0{,}28544)) \end{pmatrix} = -0{,}86390; \end{cases}$$

…… .

Also, to achieve the highest calculation accuracy, the correction process (implicit Adams–Moulton method) of the difference scheme (3.53) can be repeated several times at the same iteration step to obtain a value with a certain specified accuracy Δ (absolute error), which was implemented in the programming language PYTHON (see Fig. 3.12):

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import *
from sympy import *
# entering the integration step
h=float(input())
```

```
# entering the initial value of the integration argument (total integration
    interval) t0
a=int(input()); t_mtr=np.array(a)
# entering the initial value of the argument of the differential equation
    function q0
q_0=float(input()); q_mtr=np.array(q_0)
# entering the initial value of the argument of the differential equation
    function z0
z_0=int(input()); z_mtr=np.array(z_0)
# entering the final value of the total integration interval
b=int(input())
# entering the value of the calculation accuracy by the correction formula
    of the Adams method
e=float(input())
# setting the functions of the differential equation system
def f1(z):
    return z
def f2(q,z):
    return (-0.2*z)-(10*math.sin(q))
# determination of the first three values of the solution of the differential
    equation
t=a; q=q_0; z=z_0
t_mas=[t]; q_mas=[q]; z_mas=[z]
t_mtr=np.array(t)
q_mtr=np.array(q_0)
z_mtr=np.array(z_0)
while t<3*h:
    # iterative formulas of the Runge-Kutta method
    K1_Z=f2(q,z); K1_Q=f1(z)
    K2_Z=f2(q+(h/2*K1_Q),z+(h/2*K1_Z)); K2_Q=f1(z+(h/2*K1_Z))
    K3_Z=f2(q+(h/2*K2_Q),z+(h/2*K2_Z)); K3_Q=f1(z+(h/2*K2_Z))
    K4_Z=f2(q+(h*K3_Q),z+(h*K3_Z)); K4_Q=f1(z+(h*K3_Z))
    z=z+(h/6*(K1_Z+(2*K2_Z)+(2*K3_Z)+K4_Z))
    q=q+(h/6*(K1_Q+(2*K2_Q)+(2*K3_Q)+K4_Q))
    t=t+h
    t_mas.append(t)
    q_mas.append(q)
    z_mas.append(z)
    t_mtr=np.append(t_mtr, t)
i=4
while t<=b-h:
    # iterative prediction formula of the Adams method
    q=q_mas[i-1]+(h/24*(55*f1(z_mas[i-1])-59*f1(z_mas[i-2])+
                        37*f1(z_mas[i-3])-9*f1(z_mas[i-4])))
    z=z_mas[i-1]+(h/24*(55*f2(q_mas[i-1],z_mas[i-1])-59*f2(q_mas[i-
     2],z_mas[i-2])+
                        37*f2(q_mas[i-3],z_mas[i-3])-9*f2(q_mas[i-
     4],z_mas[i-4])))
    t=t+h
    b_tran=q; c_tran=z
    a_tran=q+1; d_tran=z+1
    # iterative correction formula of the Adams method
    while abs(b_tran-a_tran)<=e or abs(c_tran-d_tran)<=e:
        a_tran=b_tran; d_tran=c_tran
        b_tran=q_mas[i-1]+(h/24*(9*f1(d_tran)+19*f1(z_mas[i-1])-
     5*f1(z_mas[i-2])+
                                 f1(z_mas[i-3])))
        c_tran=z_mas[i-1]+(h/24*(9*f2(a_tran,d_tran)+19*f2(q_mas[i-
     1],z_mas[i-1])-
                                 5*f2(q_mas[i-2],z_mas[i-2])+f2(q_mas[i-
     3],z_mas[i-3])))
    i=i+1
    t_mas.append(t)
    q_mas.append(b_tran)
    z_mas.append(c_tran)
    t_mtr=np.append(t_mtr, t)
# we construct graphs of the solutions of the differential equation
plt.figure(figsize=(7, 7))
plt.xlabel('t',fontsize=15, color='blue')
plt.ylabel('psi, dPsi(t)/dt',fontsize=15, color='blue')
plt.plot(t_mas, q_mas)
plt.plot(t_mas, z_mas)
```

```
plt.plot(t_mtr,0.3*(np.exp(-
    0.1*t_mtr))*(np.cos(3.1607*t_mtr)+(0.03164*np.sin(3.1607*t_mtr))))
plt.legend(['The Runge-Kutta method for Psi(t)','The Runge-Kutta method for
    dPsi(t)/dt','Exact solution'], loc=1)
plt.grid(True)
plt.xlim([0, 1])
plt.ylim([-0.95, 0.45])
plt.show().
```

From the obtained results, given in Table 3.6, it is clear that the calculations are highly accurate compared to the results of analytical calculation ($\tilde{\theta}(t_n)$) (see Table 3.4), in particular, at the points $t_4 = 0{,}4$ sec i $t_{10} = 1{,}0$ sec:

– relative error is calculated by comparing the values of the analytical solution and the Adams' method

$$\varepsilon_{t_4} = \left|\frac{\tilde{\theta}_{t_4} - \overline{\overline{\theta}}_{t_4}}{\tilde{\theta}_{t_4}}\right| \cdot 100\% = \left|\frac{0{,}09567 - 0{,}09783}{0{,}09567}\right| \cdot 100\% = 2{,}26\%,$$

$$\varepsilon_{t_{10}} = \left|\frac{\tilde{\theta}_{t_{10}} - \overline{\overline{\theta}}_{t_{10}}}{\tilde{\theta}_{t_{10}}}\right| \cdot 100\% = \left|\frac{(-0{,}27157) - (-0{,}27163)}{-0{,}27157}\right| \cdot 100\% = 0{,}02\%;$$

– absolute error by comparing the values of «prediction» $\overline{\overline{\theta}}^{*}(t_n)$ (Adams-Bashforth) and «correction» $\overline{\overline{\theta}}(x_n)$ (Adams-Moulton)

$$\Delta_{t_4} = \left|\overline{\overline{\theta}}_{t_4} - \overline{\overline{\theta}}^{*}_{t_4}\right| = \left|0{,}09783 - 0{,}09800\right| = 1{,}7 \cdot 10^{-4},$$

$$\Delta_{t_{10}} = \left|\overline{\overline{\theta}}_{t_{10}} - \overline{\overline{\theta}}^{*}_{t_{10}}\right| = \left|(-0{,}27157) - (-0{,}27163)\right| = 6{,}0 \cdot 10^{-5}.$$

Table 3.6 – Calculation results for the solution of the differential equation

n	t_n, sec	$\overline{\overline{\theta}}^{*}(t_n)$, grad	$\frac{d\overline{\theta}^{*}}{dt}(t_n)$, grad/sec	$\overline{\overline{\theta}}(t_n)$, grad	$\frac{d\overline{\overline{\theta}}}{dt}(t_n)$, grad/sec	$\tilde{\theta}(t_n)$, grad
0	0,0	0,30000	0,00000	0,300000	0,000000	0,30000
1	0,1	0,28544	– 0,28792	0,285440	– 0,287920	0,28522
2	0,2	0,24352	– 0,54295	0,243520	– 0,542950	0,24274
3	0,3	0,17876	– 0,74113	0,178760	– 0,741130	0,17726
4	0,4	0,09800	– 0,86298	0,097830	– 0,86390	0,09567
5	0,5	0,00914	– 0,89904	0,008901	– 0,89987	0,00630
6	0,6	– 0,07884	– 0,84631	– 0,07916	– 0,84684	– 0,08191
7	0,7	– 0,15740	– 0,71199	– 0,15775	– 0,71194	– 0,16034
8	0,8	– 0,21906	– 0,51117	– 0,21939	– 0,51060	– 0,22147
9	0,9	– 0,25819	– 0,26504	– 0,25845	– 0,26418	– 0,25964
10	1,0	– 0,27147	0,00166	– 0,27163	0,00256	– 0,27157

It should be noted that in Table 3.6, at other time intervals t_n, there is a significant discrepancy between the results of numerical and analytical calculations (see Table 3.4). This discrepancy is also caused by obtaining the analytical calculation (3.43) under the assumption that $\theta \to 0$. However, the results of the numerical calculation using the «prediction and correction» method show close values to the results of the fourth-order Runge-Kutta calculation in Table 3.4.

Compared to one-step methods, «prediction and correction» methods have a number of special features:

1. To implement «prediction and correction» methods, it is necessary to have information about several previous points. Therefore, they do not belong to the «self-starting» methods, and a certain one-step method must be used to start the solution. Therefore, in the process of solving differential equations, the integration step cannot be changed.

2. One-step methods and «prediction and correction» methods provide approximately the same accuracy of results, but the latter, unlike the former, allows for easy estimation of the step error.

3. Using the fourth-order Runge-Kutta method, four function values must be calculated at each step, while two function values are sufficient to ensure convergence in the «prediction and correction» method of the same order of accuracy. Therefore, «prediction and correction» methods require almost half as much machine time as Runge-Kutta methods of comparable accuracy.

3.3 Numerical methods for solving ordinary differential equations in boundary value problems

During the formulation of the boundary value problem, it is necessary to find the solution of the n-th order differential equation $y^{(n)} = f\left(x, y, y', y'', \ldots, y^{(n-1)}\right)$ on segment $a \le x \le b$, subject to the specified boundary conditions:

$$
\begin{aligned}
& y(a) = A_0, \quad y(b) = B_0; \\
& y'(a) = A_1, \quad y'(b) = B_1; \\
& \ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\
& y^{(k)}(a) = A_k, \; y^{(m)}(b) = B_m; \\
& \ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\
& y^{(i)}(a) = A_i, \quad y^{(j)}(b) = B_j,
\end{aligned}
$$

where A_k, B_m are some constants (k=0, 1, 2, …, i; m=0, 1, 2, …, j). The total number of additional conditions at the ends of the segment [a; b] must be equal to the order of the differential equation $i+j+2=n$.

In the case of a system of differential equations, the same rule must be followed. Moreover, additional conditions cannot be concentrated at one end of the segment. For example, for differential equations of the second and third orders, the problem is formulated as follows.

Solve the boundary value problem for the equations:

– of the second order $y'' = f\left(x, y, y'\right)$ on segment $a \le x \le b$ under the boundary conditions $y(a) = A$, $y(b) = B$;

– of the third order $y''' = f\left(x, y, y', y''\right)$ on segment $a \le x \le b$ under the boundary conditions $y(a) = A$, $y'(a) = C$, $y(b) = B$ or under the boundary conditions $y(a) = A$, $y(b) = B$, $y'(b) = D$.

Constants A, B, C and D correspond to the values of the specified functions at points a and b.

Specific boundary conditions are selected from the physical formulation of the problem.

The Cauchy problem differs from boundary-value problems in that the region in which the solution must be determined is not specified in advance. However, the Cauchy problem can be considered as one of the boundary value problems. The Cauchy problem usually arises during the analysis of processes determined by the differential law of evolution and the initial state (the mathematical expression of which is the equation and the initial condition). Moreover, there are methods that allow the search for the solution of the boundary value problem to lead to the search for the solutions of a series of Cauchy problems for the corresponding differential equation. One such well-known numerical method for solving the ODE for boundary value problems is the «shooting» method.

«Shooting» method

The «shooting» method reduces the solution of the boundary value problem for the ODE to the solution of an iterative sequence of Cauchy problems. Suppose it is necessary to solve the following boundary value problem of the type:

$$y''(x) = f(x, y, y'), x \in [a; b]; \tag{3.57}$$

$$y(a) = A; \tag{3.58}$$

$$y(b) = B. \tag{3.59}$$

Instead of the boundary value problem (3.57) – (3.59), the following Cauchy problem is considered:

$$y''(x) = f(x, y, y'), x \in [a; b]; \tag{3.60}$$

$$y(a) = A; \tag{3.61}$$

$$y'(a) = tg\alpha, \alpha: y(b, \alpha) = B, \tag{3.62}$$

in which the integral curve $y(x, \alpha)$ depends not only on the variable x but also on the parameter α, which is called the shooting angle (the angle between the tangent to the curve of the ODE solution and the abscissa axis). It is chosen based on the condition that the value of the integral curve on the right boundary, $y(b, \alpha)$, is equal to the value of B with a predetermined accuracy ε (Fig. 3.13):

$$\left|y(b,\alpha) - B\right| \le \varepsilon . \tag{3.63}$$

The shooting angle that satisfies the inequality (3.63) will be denoted by α^*. The integral curve obtained from the solution of the Cauchy problem (3.60)–(3.62) with an angle close to this value, according to inequality (3.63), will be the solution of problem (3.57)–(3.59) with an accuracy of ε.

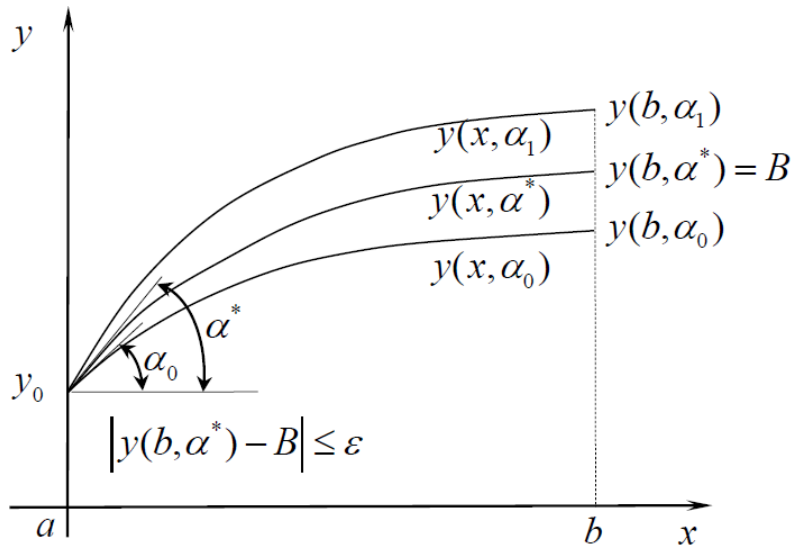


Figure 3.13 – Diagram of the geometric interpretation of the «shooting» method

Thus, to implement the «shooting» method, the initial value of the angle α_0 from condition $tg\alpha_0 = \dfrac{B-A}{b-a}$ is initially selected. With this value of α_0, one of the known numerical methods discussed above solves the Cauchy problem (3.60)–(3.62) to obtain $y(x, \alpha_0)$ and $y(b, \alpha_0)$. If condition (3.63) is fulfilled at the same time, then the boundary value problem (3.57)–(3.59) is solved with accuracy ε.

Otherwise, there may be two options:

a) if $y(b, \alpha_0) > B$, then the aiming angle is reduced in some way and the Cauchy problem (3.60)–(3.62) is solved using the same numerical method until the condition $y(b, \alpha_0) < B$ is met;

b) if $y(b, \alpha_0) < B$, then the angle of attack is increased in any way and the Cauchy problem is solved until the condition $y(b, \alpha_0) > B$ is fulfilled.

Thus, the aiming angle is in the middle of the interval $\alpha\in[\alpha_0; \alpha_1]$. After this, the true value α^* of the aiming angle is determined by the method of halving, namely: the next value of the angle is determined using the iterative formula $\alpha_{k+1} = (\alpha_{k-1}+\alpha_k)/2$. The value of the ordinates $y(x, \alpha_{k+1})$ and $y(b, \alpha_{k+1})$ is determined at the corresponding points. After that, the inequality $\left|y(b,\alpha) - B\right| \le \varepsilon$ is analyzed. If it is fulfilled, then $\alpha^* = (\alpha_{k-1}+\alpha_k)/2$ and $y(x^*, \alpha_{k+1})$ is a true integral curve. If the inequality does not hold, then the iterative process is repeated from the beginning.

The halving method converges very slowly, and it is necessary to solve a significant number of Cauchy problems for different «shooting» angles α_k. Newton's iterative formula is used to accelerate the convergence of the iterative process:

$$\alpha_{k+1} = \alpha_k + \frac{B - y(b,\alpha_k)}{y(b,\alpha_k) - y(b,\alpha_{k-1})}(\alpha_k - \alpha_{k-1}). \tag{3.64}$$

To determine the preliminary value of the aiming angle α_{k-1} in equation (3.64), the intermediate value determined at the first iteration step for k=1 is used, namely:

$$\alpha_1 = \alpha_0 + \frac{B - y(b,\alpha_0)}{y(b,\alpha_0 + \delta) - y(b,\alpha_0)}\delta, \tag{3.65}$$

where δ – a small value of the angle increment (δ=10°… 20°).

The algorithm for the «shooting» method to solve ODE boundary value problems is presented in Figure 3.14.

Example 3.7. To solve the boundary value problem for the second-order ODE with the accuracy of ε = 0,01 using the numerical "shooting" method:

$$y'' = \frac{1}{x}y' + x^2, \tag{3.66}$$

which satisfies the initial conditions $y(1, 0) = 0$ i $y(2, 0) = 1$.

Solution:

The boundary value problem (3.66) reduces to the Cauchy problem:

$$\begin{cases} y'' = \dfrac{1}{x}y' + x^2; \\ y(1) = 0; \\ y'(1) = tg\alpha^*; \\ y(2,\alpha^*) = 1; \quad (\alpha = \alpha^*). \end{cases} \tag{3.67}$$

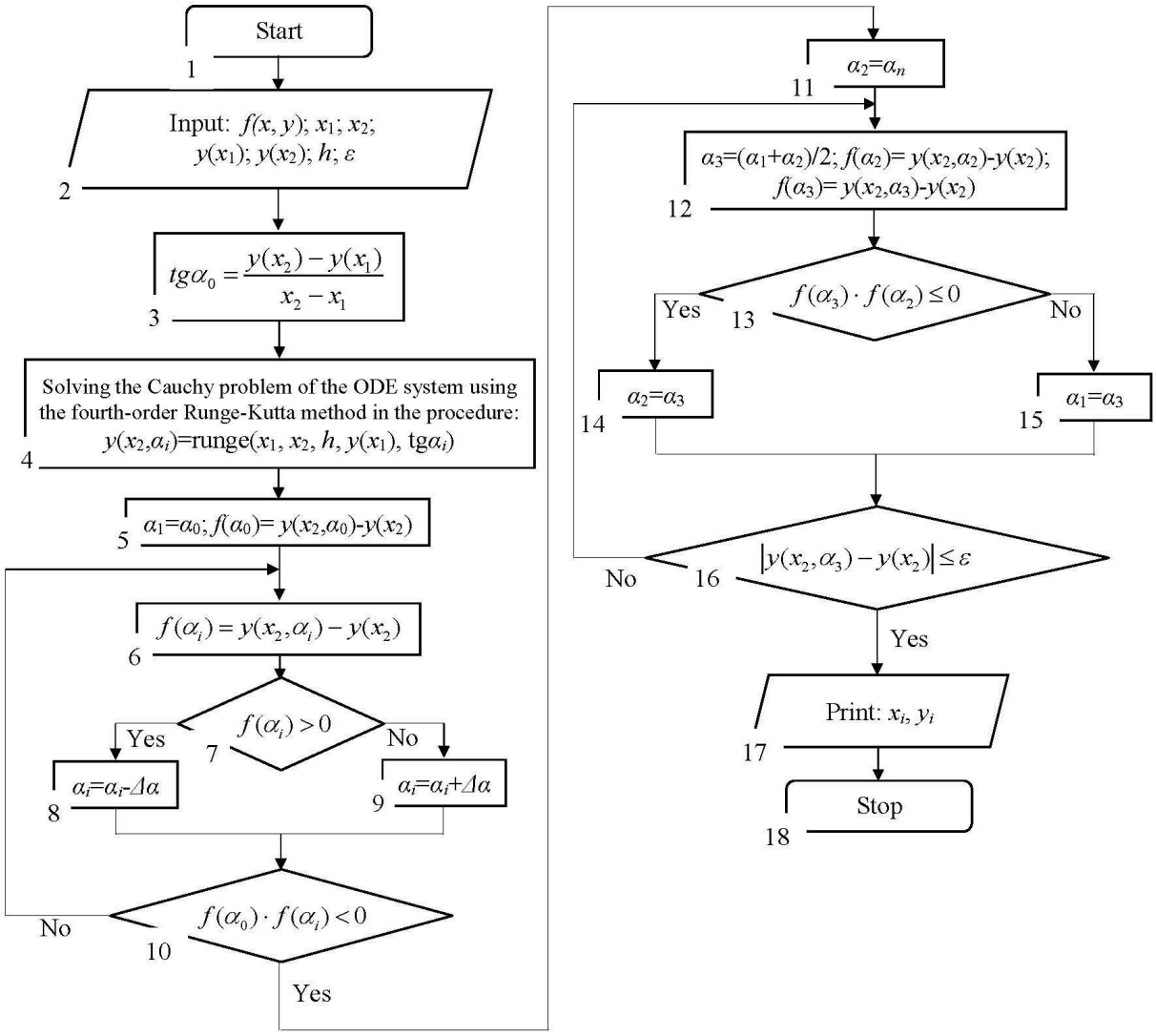


Figure 3.14 – Scheme of the «shooting» method algorithm

The equation $f(\alpha) = y(2, \alpha) - y(2, \alpha^*) = y(2, \alpha) - 1$, as a nonlinear equation with respect to the aiming angle α, can be solved using one of the numerical iterative methods considered, with parallel solving of the Cauchy problem at each iteration.

Let the initial aiming angle α_0 be determined using the relation:

$$tg\alpha_0 = \frac{B - A}{b - a} = \frac{1 - 0}{2 - 1} = 1;\ \alpha_0 = 45°.$$

Then the Cauchy problem for the second-order ODE (3.67) and the corresponding Cauchy problem for the normal system of the second-order ODE at the first iteration will have the form:

$$\begin{cases} y'' = \dfrac{1}{x}y' + x^2; \\ y(1,0) = 0; \\ y'(1,0) = tg\alpha_0 = 1; \end{cases} \rightarrow \begin{cases} y' = z; \\ z' = \dfrac{1}{x}z + x^2; \\ y(1,0) = 0; \\ z(1,0) = 1. \end{cases} \quad (3.68)$$

Solving the system (3.68) with a step size h=0,25 using the fourth-order Runge-Kutta numerical method, we obtain: $y(2, 45°) = y_4 = 2{,}625$ (Table 3.7). Since $f^1(45°) = y(2, 45°)–1 = 2{,}625–1{,}0 = 0{,}625 > 0$, it is necessary to reduce the shooting angle α (for example, $\Delta\delta = 44°$) so that the value of $f(\alpha)$ is less than zero in the next iteration. You can take $\alpha_1 = \alpha_0–\Delta\delta = 45°–44° = 1°$, and then the Cauchy problem at the second iteration will have the form:

$$\begin{cases} y'' = \dfrac{1}{x}y' + x^2; \\ y(1,0) = 0; \\ y'(1,0) = tg\alpha_1 = 0{,}0175; \end{cases} \rightarrow \begin{cases} y' = z; \\ z' = \dfrac{1}{x}z + x^2; \\ y(1,0) = 0; \\ z(1,0) = 0{,}0175. \end{cases} \quad (3.69)$$

Solving the system (3.69) again with a step size $h = 0{,}25$ using the fourth-order Runge-Kutta numerical method will yield: $y(2, 1°) = y_4 = 1{,}151$ (see Table 3.7). Since $f^2(1, 0°) = y(2, 1°) –1 = 1{,}151–1{,}0 = 0{,}151 > 0$, it is necessary to reduce the aiming angle α until the value of $f(\alpha)$ at the next iteration is less than zero. Therefore, taking $\alpha_2 = \alpha_1–\Delta\delta = 1°–44° = –43°$, the solution to the Cauchy problem at the third iteration will be $y(2, -43°) = y_4 = -0{,}274$ (see Table 3.7), i.e., $f^3(1°) = y(2, 1°) –1 = 1{,}151–1{,}0 = 0{,}151 > 0$.

Accordingly, the angle α^* is within the interval: $\alpha_1=1° < \alpha^* < \alpha_2= –43°$.

At the fourth iteration, the aiming angle is selected based on the iterative formula $\alpha_3 = (\alpha_1+\alpha_2)/2 = (1°+(–43°))/2 = -21°$. Then the Cauchy problem at the fourth iteration will have the form:

$$\begin{cases} y'' = \dfrac{1}{x}y' + x^2; \\ y(1) = 0; \\ y'(1) = tg\alpha_3 = –0{,}384; \end{cases} \rightarrow \begin{cases} y' = z; \\ z' = \dfrac{1}{x}z + x^2; \\ y(1) = 0; \\ z(1) = –0{,}384. \end{cases} \quad (3.70)$$

Solving the system (3.70) with a step $h = 0{,}25$ by the Runge-Kutta numerical method of the fourth order, we obtain: $y(2–21°) = y_4 = 0{,}549$ (see Table 3.7). Then $f^4(–21°) = y(2;–21°)–1 = 0{,}549–1{,}0 = -0{,}451 < 0$, and also $|f^4(–21°)| = 0{,}451 > \varepsilon = 0{,}01$.

Comparing the values of the function $f(\alpha)$ on the fourth ($f^4(–21°) < 0$) and on the third iterations ($f^3(1°) > 0$) means that $–21° < \alpha^* < 1°$.

Continuing by analogy with the solution of the given Cauchy problem in the following iterations ($\alpha_4 = (\alpha_3+\alpha_2)/2 = (1°+(–21°))/2 = -10°$, $–10° < \alpha^* < 1°$; $\alpha_5 = (\alpha_4+\alpha_3)/2 = (1°+(–10°))/2 = -4{,}5°$) by the fourth-order Runge-Kutta numerical method with a step $h = 0.25$, the results are summarized in Table 3.7. At the last, sixth, iterative step, $y(2,-4{,}5°) = 1{,}007$, $f^5(–4{,}5°) = y(2; –4{,}5°) – –1 = 1{,}007–1{,}0 = 0{,}007 > 0$, and $|f^5(–21°) = 0{,}007 < \varepsilon = 0{,}01$.

Thus, the aiming angle turns out to be $\alpha^* = -4{,}5°$. At this angle, the integral curve has the value y_i from the row of Table 3.7, which corresponds to the sixth iteration.

It is also possible to note the relatively high accuracy of the method for boundary value problems of the ODE compared to the exact solution $\tilde{y}(x) = \frac{1}{8}x^4 - \frac{7}{24}x^2 + \frac{1}{6}$, the results of which are shown in Table 3.7.

Table 3.7 – Calculation results of the solution for the differential equation of the boundary value problem

i	0	1	2	3	4
x_i	1,0	1,25	1,50	1,75	2,0
$\alpha_0=45^0$	1st iteration				
y_i	0,0	0,321	0,820	1,563	2,625
$\alpha_0=1^0$	2nd iteration				
y_i	0,0	0,044	0,206	0,550	1,151
$\alpha_0=-43^0$	3rd iteration				
y_i	0,0	– 0,223	– 0,388	– 0,430	– 0,274
$\alpha_0=-21^0$	4th iteration				
y_i	0,0	– 0,068	– 0,045	0,136	0,549
$\alpha_0=-10^0$	5th iteration				
y_i	0,0	– 0,010	0,085	0,350	0,860
$\alpha_0=-4{,}5^0$	6th iteration				
y_i	0,0	0,017	0,146	0,451	1,007
y_i	0,0	0,016	0,143	0,446	1,000

Let’s consider the implementation of the «shooting» method for the second-order boundary value problem in the PYTHON programming language:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import *
from sympy import *
import random
import math
#entering the integration step
```

```
h=float(input())
#entering the initial value of the integration argument (total integration
     interval) x1
x_a=int(input()); x_mtr=np.array(x_a)
#entering the initial value of the argument of the differential equation
     function y(x1)
A=float(input())
#entering the initial value of the argument of the differential equation
     function y(x2)
B=float(input())
#entering the final value of the total integration interval x2
x_b=int(input())
#entering the calculation error value
e=float(input())
# setting the functions of the 2nd-order differential equation in the form
     of a system of differential equations
def f1(z):
    return z
def f2(x,z):
    return (z/x)+(x**2)
# specification of the function of calculating the differential equation of
     the 2nd order by the Runge-Kutta method of the 4th order
def runge(x_a,x_b,h,y_0,z_0):
    # calculation of iterative formulas
    x=x_a; y=y_0; z=math.tan(z_0*math.pi/180)
    y_mas=[]; x_mas=[]; z_mas=[];
    del y_mas
    del z_mas
    del x_mas
    x_mas=[x]; y_mas=[y]; z_mas=[z]
    x_mtr=np.array(x)
    y_mtr=np.array(y_0)
    z_mtr=np.array(z_0)
    while x<x_b:
        #iterative formulas of the Runge-Kutta method
        K1_Z=f2(x,z); K1_Y=f1(z)
        K2_Z=f2(x+(h/2),z+(h/2*K1_Z)); K2_Y=f1(z+(h/2*K1_Z))
        K3_Z=f2(x+(h/2),z+(h/2*K2_Z)); K3_Y=f1(z+(h/2*K2_Z))
        K4_Z=f2(x+h,z+(h*K3_Z)); K4_Y=f1(z+(h*K3_Z))
        z=z+(h/6*(K1_Z+(2*K2_Z)+(2*K3_Z)+K4_Z))
        y=y+(h/6*(K1_Y+(2*K2_Y)+(2*K3_Y)+K4_Y))
        x=x+h
        x_mas.append(x)
        y_mas.append(y)
        z_mas.append(z)
        x_mtr=np.append(x_mtr, x)
    return y_mas, z_mas, x_mas, x_mtr
# determination of the data of the initial two aiming angles
t_an=(B-A)/(x_b-x_a)
ang=math.atan(t_an)*180/math.pi
ang_sec=ang
while          (runge(x_a,x_b,h,A,ang)[0][len(runge(x_a,x_b,h,A,ang)[0])-1]-
     B)*(runge(x_a,x_b,h,A,ang_sec)

     [0][len(runge(x_a,x_b,h,A,ang_sec)[0])-1]-B)>0:
    ang_frst=ang_sec
    if   (runge(x_a,x_b,h,A,ang_sec)[0][len(runge(x_a,x_b,h,A,ang_sec)[0])-
     1]-B)>0:
        ang_sec=ang_sec-44
    else:
        ang_sec=ang_sec+44
# determination of the exact value of the aiming angle by the method of
     halving
ang_new=ang_frst
while abs(runge(x_a,x_b,h,A,ang_new)[0][len(runge(x_a,x_b,h,A,ang_new)[0])-
     1]-B)>=e:
    ang_new=(ang_frst+ang_sec)/2
    if   (runge(x_a,x_b,h,A,ang_new)[0][len(runge(x_a,x_b,h,A,ang_new)[0])-
     1]-

     B)*(runge(x_a,x_b,h,A,ang_frst)[0][len(runge(x_a,x_b,h,A,ang_frst)[0])
```

```
     -1]-B)<=0:
        ang_sec=ang_new
    else:
        ang_frst=ang_new
print(' Shooting angle: alpha=',ang_new,'; The tangent of the shooting
    angle:tg(alpha)=',math.tan(ang_new*math.pi/180))
print('The solution of the ODE boundary value problem for
    yi=',runge(x_a,x_b,h,A,ang_new)[0])
# we construct graphs of the solutions of the differential equation
plt.figure(figsize=(7, 7))
plt.xlabel('t',fontsize=15, color='blue')
plt.ylabel('psi, dPsi(t)/dt',fontsize=15, color='blue')
plt.plot(runge(x_a,x_b,h,A,ang_new)[2], runge(x_a,x_b,h,A,ang_new)[0])
plt.plot(runge(x_a,x_b,h,A,ang_new)[2], runge(x_a,x_b,h,A,ang_new)[1])
plt.plot(runge(x_a,x_b,h,A,ang_new)[3],
    (runge(x_a,x_b,h,A,ang_new)[3]**4/8)-
        ((7*runge(x_a,x_b,h,A,ang_new)[3]**2)/24)+(1/6))
plt.legend([' Shooting method for y(x)','Shooting method for y`(x)','Exact
    solution'], loc=1)
plt.grid(True)
plt.xlim([1, 2])
plt.ylim([0, 3])
plt.show().
```

Finite difference method

Suppose it is necessary to solve a boundary value problem of this type:

$$y''(x) + p(x)y' + q(x)y = f(x),\ x\epsilon[a;\ b]; \tag{3.71}$$

$$y(a)=A,\ y(b)=B. \tag{3.72}$$

where $p(x)$, $q(x)$, $f(x)$ – known continuous values on the segment [a; b] functions; A і B – set constant values.

One of the most effective and popular numerical tools for solving ODE and partial differential equations is the apparatus of difference methods.

It is based on the presentation of an independent argument on the segment [a; b] in the form of a discrete set of points x_i (i=0, 1, …, n); $x_0=a$, $x_n=b$, which is called a grid.

The uniform grid (3.46) with a step was the most widely used $x_i–x_{i-1}=h$ (Fig. 3.15). In this case, instead of the continuous function $f(x)$, the grid function $y_i = f(x_i)$ is considered.

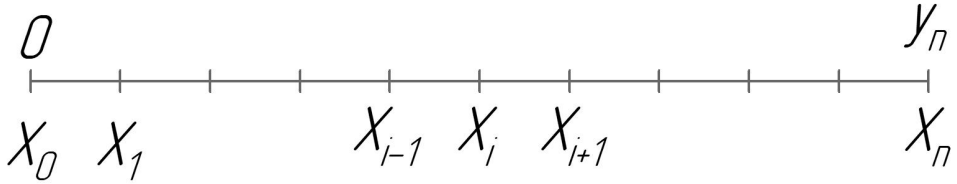


Figure 3.15 – Scheme of a one-dimensional calculation grid

A grid function can be thought of as a function with an integer argument

$$y(i) = y_i\ (i=0, \pm1, \pm2, \dots).$$

For y_i, you can introduce operations that are a discrete (difference) analogue of differentiation and integration operations.

An analogue of the first derivative is differences of the first order:

$\Delta y_i = y_{i+1} - y_i$ – right difference;

$\nabla y_i = y_i - y_{i-1}$ – left difference;

$\delta y_i = \frac{1}{2}(\Delta y_i + \nabla y_i) = \frac{1}{2}(y_{i+1} - y_{i-1})$ – central difference.

It should be noted that $\Delta y_i = \nabla y_{i+1}$.

Next, the second-order differences are recorded:

$$\Delta^2 y_i = \Delta(\Delta y_i) = \Delta(y_{i+1} - y_i) = y_{i+2} - 2y_{i+1} + y_i = \Delta\nabla y_{i+1};$$

$$\Delta\nabla y_i = \Delta(y_i - y_{i-1}) = (y_{i+1} - y_i) - (y_i - y_{i-1}) = y_{i+1} - 2y_i + y_{i-1}.$$

The difference of the m-th order is determined similarly:

$$\Delta^m y_i = \Delta(\Delta^{m-1} y_i).$$

It is obvious that:

$$\sum_{j=k}^{i} \Delta y_i = y_{i+1} - y_k; \qquad \sum_{j=k}^{i} \Delta y_i = y_i - y_{k-1}.$$

On the set of the calculation grid (see Fig. 3.15), which is called a template, the continuous differential operator L_y is replaced by the difference L_{hy}. Difference schemes are built by replacing derivatives in differential equations with difference relations. The general formula for approximating derivatives at some point x_i has the following form:

$$\frac{d^n y(x_i)}{dx^n} = \frac{1}{h^n} \sum_{s=-m}^{l} a_s y(x_i + sh) + O(h^p),$$

where the coefficients are selected a_s (s= $-m$, $-m$+1, …, 1) in such a way as to achieve the required order of approximation. The limit of the sum of m and l is subject to the condition $m+l \geq n+p-1$.

Very often, in practice, difference relations are used to approximate the first-order derivative with respect to h at three grid nodes:

– the right scheme

$$L_h^+ y = \frac{dy(x_i)}{dx} = \frac{y(x_i + h) - y(x_i)}{h} + O(h) = y_X^+;$$

– the left scheme

$$L_h^- y = \frac{dy(x_i)}{dx} = \frac{y(x_i) - y(x_i - h)}{h} + O(h) = y_X^-;$$

– the central scheme

$$L_h^0 y = \frac{dy(x_i)}{dx} = \frac{y(x_i + h) - y(x_i - h)}{2h} + O(h) = y_X^0;$$

– the central scheme with second-order accuracy in h

$$L_{hy} = \frac{d^2 y(x_i)}{dx^2} = \frac{y(x_i + h) - 2y(x_i) + y(x_i - h)}{h^2} + O(h^2) =$$

$$= \frac{y_X^+(x_i) - y_X^-(x_i)}{h} = \frac{y_X^-(x_i + h) - y_X^-(x_i)}{h} = y_{\overline{X}X}(x).$$

When obtaining a difference scheme, an important role is played by the requirement that the difference scheme best reflects the main properties of the original differential equation. The assessment of the accuracy of the difference scheme is reduced to the study of the approximation error and stability.

In the finite difference method, the solution of the boundary value problem (3.71) and (3.72) is reduced to a system of finite difference equations. For this, the main segment [a; b] is divided into n equal parts of length h (step), where h=(b–h)/n (Fig. 3.16). That is, the area of continuous change of the argument [a; b] is replaced by a discrete set of points called nodes x_i ($i = \overline{0,n}$).

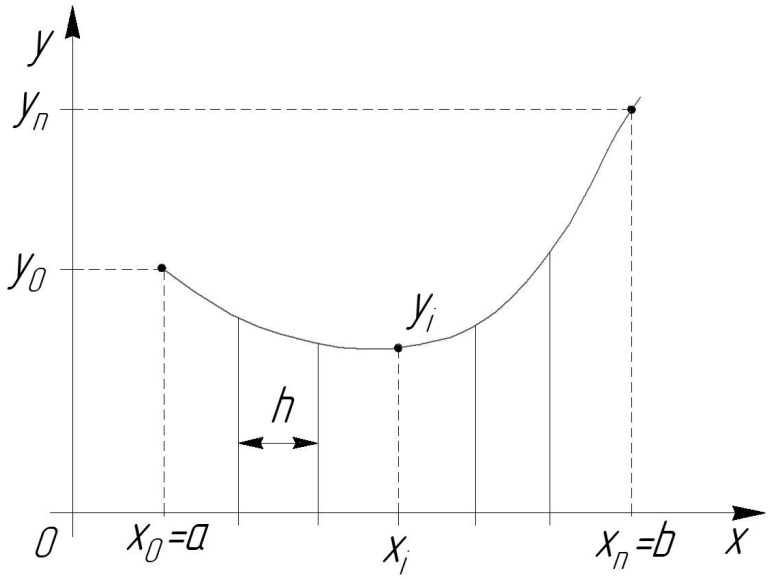


Figure 3.16 – Scheme of the calculation grid of the finite difference method

Breakpoints have abscissa coordinates:

$$x_i = x_0 + ih \ (i = 0, 1, 2, \ldots, n), \quad x_0 = a, x_n = b.$$

We denote the value at the division points x_i of the desired function y=y(x) and its derivatives $y' = y'(x)$, $y'' = y''(x)$ by $y_i = y(x_i)$, $y_i' = y'(x_i)$, $y_i'' = y''(x_i)$.

Notations are also introduced:

$$p_i = p(x_i), q_i = q(x_i), f_i = f(x_i).$$

By replacing the derivatives with symmetric finite-difference relations for the interior points x_i of the segment [a; b]:

$$y_i' = \frac{y_{i+1} - y_{i-1}}{2h}; \tag{3.73}$$

$$y_i'' = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}. \tag{3.74}$$

Substituting (3.73) and (3.74) into the original equation (3.71) for $x = x_i$ $(i = \overline{1, n-1})$, we obtain a system of difference equations:

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + p_i \frac{y_{i+1} - y_{i-1}}{2h} + q_i y_i = f_i \ (i = \overline{1, n-1}), \tag{3.75}$$

which can be presented in the form of such a LAES with a tridiagonal matrix:

$$\begin{cases} a_i y_{i-1} + b_i y_i + c_i y_{i+1} = d_i; \\ a_i = \dfrac{1}{h^2} - \dfrac{p_i}{2h}; b_i = -\dfrac{2}{h^2} + q_i; \\ c_i = \dfrac{1}{h^2} + \dfrac{p_i}{2h}; \quad d_i = f_i; \\ y_0 = A; \ y_n = B; \ i = \overline{1, n-1}. \end{cases} \tag{3.76}$$

In the case of a large value of n, the direct solution of the system (3.76) becomes cumbersome. To solve a system of this type, the running method is used (see Chapter 1).

The error estimate of the finite difference method for problem (3.71), (3.72) has the form $|y_i - y(x_i)| \le \dfrac{h^2 M_4}{96}(b-a)^2$, where $y(x_i)$ is the value of the exact solution for $x = x_i$, $M_4 = \max\limits_{[a,b]} |y^{(4)}(x)|$.

The LAES (3.76) has a tridiagonal matrix and the fulfillment of the precedence condition by the modulus of the diagonal elements ($|b_i| \ge |a_i| + |c_i|$) guarantees stable implementation of the sweep method for solving (3.75).

The numerical method of running for solving the LAES consists of a forward and a reverse run. For direct travel, the running coefficients are determined by the formulas:

$$\xi_i = \frac{-c_i}{b_i + a_i \xi_{i-1}}; \ \eta_i = \frac{d_i - a_i \eta_{i-1}}{b_i + a_i \xi_{i-1}}; \ i = \overline{1, n-1}, \tag{3.77}$$

moreover $\xi_1 = -c_1/b_1$, $\eta_1 = d_1/b_1$, since $\xi_0 = 0$ and $\eta_0 = 0$.

During the return stroke, the values of y_i $(i = \overline{n-1, 1})$ are determined using the expressions $y_i = \xi_i y_{i+1} + \eta_i$:

$$\begin{cases} i = n-1: \quad y_{n-1} = \xi_{n-1} y_n + \eta_{n-1} = \eta_{n-1}; \\ i = n-2: \quad y_{n-2} = \xi_{n-2} y_{n-1} + \eta_{n-2}; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\ i = 1: \quad y_1 = \xi_1 y_2 + \eta_1. \end{cases} \tag{3.78}$$

Thus, the solution of the boundary value problem for the differential equation is reduced to the solution of the system of n–1 linear algebraic equations of the form (3.76), with n–1 unknowns $y_1, \ldots, y_{n-1}$. After solving this system, we will get a table of values of the desired function y.

The algorithm of the finite difference method for solving ODE boundary value problems is presented in Figure 3.17.

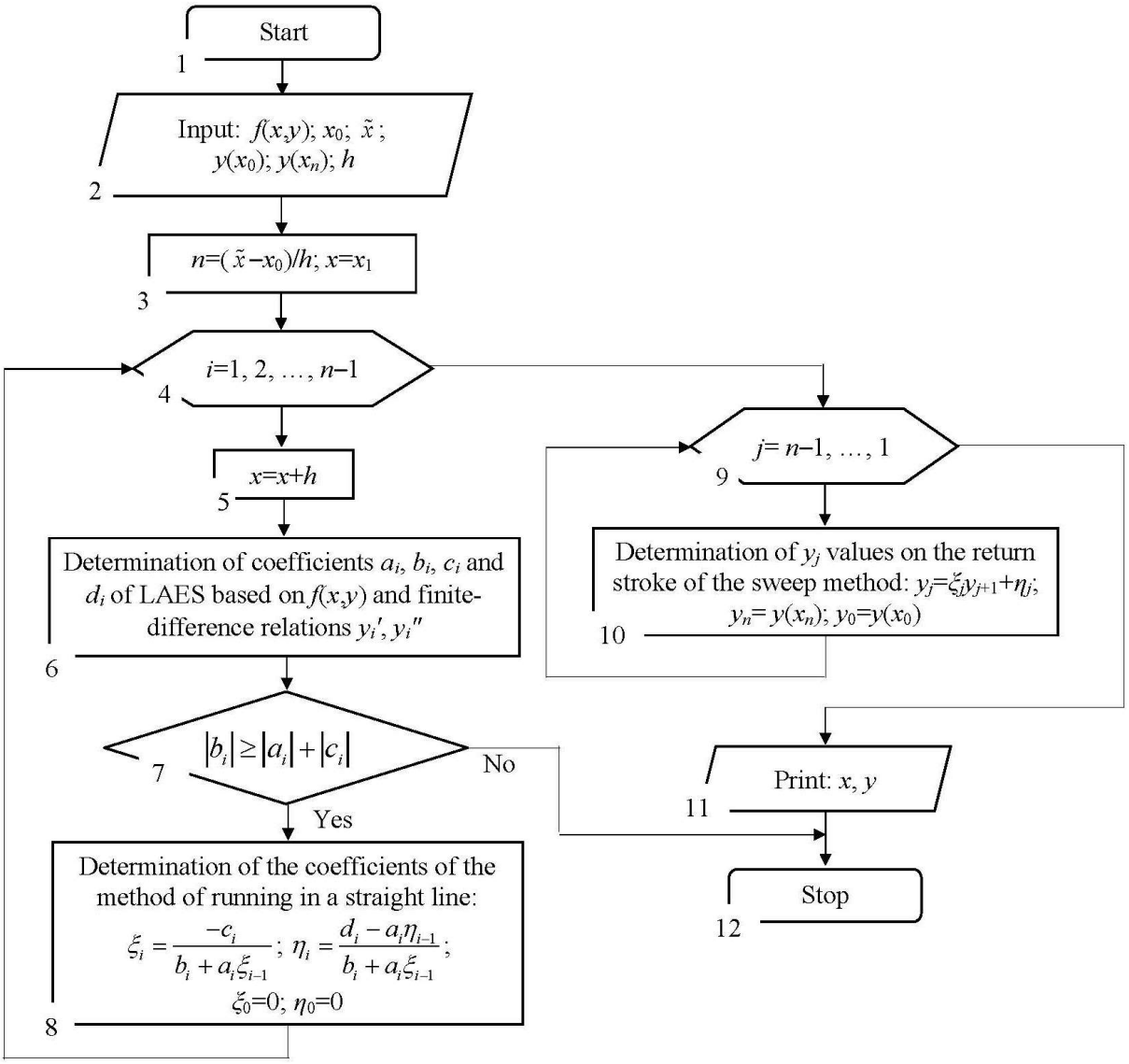


Figure 3.17 – Scheme of the finite difference method algorithm

Example 3.8. To solve the boundary value problem for the second-order ODE using the numerical method of finite differences:

$$x^2y'' + x^2y' = 1, \tag{3.79}$$

which satisfies the initial conditions for y(1, 0)=0; y(1, 4) = 0,0566.

Solution:

Using formulas (3.73) and (3.74) we replace the original equation (3.79) with a system of finite-difference equations:

$$x_i^2\left(\frac{y_{i+1}-2y_i+y_{i-1}}{h^2}\right)+x_i\left(\frac{y_{i+1}-y_{i-1}}{2h}\right)=1.$$

As a result of summing the free members, we get:

$$y_{i-1}(2x_i^2-hx_i)-4x_i^2y_i+y_{i+1}(2x_i^2+hx_i)=2h^2, \qquad (3.80)$$

where $a_i=2x_i^2-hx_i$, $b_i=-4x_i^2$, $c_i=2x_i^2+hx_i$, $d_i=2h^2$.

Choosing the step $h = 0.1$, three internal nodes will be obtained:

$x_i = 0, 1, i+1$ $(i=1, 2, 3)$ where $a_1=2x_1^2-hx_1=2\cdot1{,}1^2-0{,}1\cdot1{,}1=2{,}31;$

$b_1=-4x_1^2=-4\cdot1{,}1^2=-4{,}84;$ $c_1=2x_1^2+hx_1=2\cdot1{,}1^2+0{,}1\cdot1{,}1=2{,}53;$

$d_i=2h^2=2\cdot0{,}1^2=0{,}02.$

Similarly, the coefficients for a_j, b_j, c_j, d_j $(j=2, 3)$.

After writing equation (3.80) for each of these nodes we obtain the following system of equations:

$$\begin{cases}a_1y_0-b_1y_1+c_1y_2=d_1;\\ a_2y_1-b_2y_2+c_2y_3=d_2;\\ a_3y_2-b_3y_3+c_3y_4=d_3;\end{cases}\rightarrow\begin{cases}2{,}31y_0-4{,}84y_1+2{,}53y_2=0{,}02;\\ 2{,}76y_1-5{,}76y_2+3{,}00y_3=0{,}02;\\ 3{,}25y_2-6{,}76y_3+3{,}51y_4=0{,}02.\end{cases} \qquad (3.81)$$

Since the obtained tridiagonal matrix of the system of equations (3.81) fulfills the condition of preferring diagonal elements ($|b_i|\geq|a_i|+|c_i|$), the sweep method can be used.

Racing coefficients on a straight course:

$$\xi_1=\frac{-c_1}{b_1}=-2{,}53/(-4{,}84)=0{,}52273;\ \eta_1=\frac{d_1}{b_1}=0{,}02/(-4{,}84)=-0{,}00413;$$

$$\xi_2=\frac{-c_2}{b_2+a_2\xi_1}=\frac{-3}{-5{,}76+2{,}76\cdot0{,}52273}=0{,}69488;$$

$$\eta_2=\frac{d_2-a_2\eta_1}{b_2+a_2\xi_1}=\frac{0{,}02-2{,}76\cdot(-0{,}00413)}{-5{,}76+2{,}76\cdot0{,}52273}=-0{,}00727;$$

$$\xi_3=\frac{-c_3}{b_3+a_3\xi_2}=\frac{-3{,}51}{6{,}76+3{,}25\cdot0{,}69488}=0{,}77971;$$

$$\eta_3=\frac{d_3-a_3\eta_2}{b_3+a_3\xi_2}=\frac{-0{,}02-3{,}25\cdot(-0{,}00727)}{-6{,}76+3{,}25\cdot0{,}69488}=-0{,}0969.$$

On the reverse course, the values of y_i are determined using expressions $y_i = \xi_i y_{i+1} + \eta_i$:

$$\begin{cases} i=3: \quad y_3 = \xi_3 y_4 + \eta_3 = \eta_3 = 0,03446; \\ i=2: \quad y_2 = \xi_2 y_3 + \eta_2 = 0,69488 \cdot 0,03446 - 0,00727 = 0,01668; \\ i=1: \quad y_1 = \xi_1 y_2 + \eta_1 = 0,52273 \cdot 0,01668 - 0,00413 = 0,00459. \end{cases}$$

It is also possible to note the relatively high accuracy of the finite difference method for the boundary value problems of the ODE compared to the exact solution of $\tilde{y}(x) = 0,5\ln^2 x$ at the corresponding points:

$$y(x_1) = y(1,1) = 0,0047; \quad y(x_2) = y(1,2) = 0,0166; \quad y(x_3) = y(1,3) = 0,0344.$$

In particular, in points $x_1 = 1,1$ and $x_3 = 1,3$:

– relative error by comparing the values of the analytical solution and the finite difference method

$$\varepsilon_{x_1} = \left| \frac{\tilde{y}_{x_1} - y_{x_1}}{\tilde{y}_{x_1}} \right| \cdot 100\% = \left| \frac{0,00459 - 0,00470}{0,00459} \right| \cdot 100\% = 2,40\%,$$

$$\varepsilon_{x_3} = \left| \frac{\tilde{y}_{x_3} - y_{x_3}}{\tilde{y}_{x_3}} \right| \cdot 100\% = \left| \frac{0,03446 - 0,03440}{0,03446} \right| \cdot 100\% = 0,17\%.$$

Let's consider the implementation of the finite difference method for the ODE second-order boundary value problem in the PYTHON programming language:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import *
from sympy import *
#introducing an integration step
h=float(input())
#entering the initial value of the integration argument (total integration
     interval) x1
x_a=int(input()); x_mtr=np.array(x_a)
#entering the final value of the total integration interval x2
x_b=float(input())
#entering the initial value of the argument of the differential equation
     function y(x1)
A=float(input())
# entering the initial value of the argument of the differential equation
     function y(x2)
B=float(input())
# determination of the coefficients of the system of difference equations
n=round((x_b-x_a)/h)
x,a,b,c,d=[],[0],[0],[0],[0]
sig_ma,tet_ta=[0],[0]
x_tr=x_a
for i in range(0,n):
    x.append(x_tr)
    x_tr=x_tr+h
    x_mtr=np.append(x_mtr, x_tr)
for i in range(1,n):
    a.append(2*(x[i]**2)-(h*x[i]))
    b.append(-4*x[i]**2)
```

```
    c.append(2*(x[i]**2)+(h*x[i]))
    d.append(2*h**2)
#verification of the fulfillment of the precedence condition modulo the
     diagonal elements of the matrix
    if abs(b[i])<abs(a[i])+abs(c[i]):
        print('The system of finite difference equations has no solution')
        break
#determination of driving coefficients on a straight course
    sig_ma.append(-c[i]/(b[i]+(a[i]*sig_ma[i-1])))
    tet_ta.append((d[i]-(a[i]*tet_ta[i-1]))/(b[i]+(a[i]*sig_ma[i-1])))
# determination of the solution of the differential equation on the reverse
     course
y=[0]*(n+1)
y[0]=A; y[n]=B
for i in range(n-1,0,-1):
    y[i]=(sig_ma[i]*y[i+1])+tet_ta[i]
x.append(x_b)
print('x(i)=',x)
print('y(i)=',y)
# we construct graphs of the solutions of the differential equation
plt.figure(figsize=(7, 7))
plt.xlabel('x',fontsize=15, color='blue')
plt.ylabel('y',fontsize=15, color='blue')
plt.plot(x, y)
plt.plot(x_mtr, (0.5*(np.log(x_mtr)**2)))
plt.legend(['The Runge-Kutta method','f(x)=0.5*(ln(x))^2'],loc=1)
plt.grid(True)
plt.xlim([1, 1.4])
plt.ylim([0, 0.06])
plt.show().
```

3.4 Numerical methods for solving ordinary differential equations for «stiff» problems

There are ODE for which it is difficult to obtain a satisfactory solution to the problems using the numerical methods described above. The definition of such problems is related to the concept of the time constant of a differential equation, which is introduced in relation to the analytical solution. For equations of the first order, this is the time interval when the variable part of the solution decreases by *e* times. An equation of order *n* has, accordingly, *n* time constants; if any two of them differ greatly (in practice by a hundred or more times) or any of them is quite small compared to the time interval on which the solution is searched, then the problem is called «stiff», and its practical solution cannot be solved by conventional methods. The coefficients in such equations differ by several orders of magnitude.

It is appropriate to consider the «stiff» system using the example of the solution of the ODE:

$$\begin{cases} \dfrac{du}{dx} = \lambda u, \quad x \in (x_0; X]; \\ u(x_0) = u_0. \end{cases} \tag{3.82}$$

The analytical solution of ODE (3.82) is given by expression $u(t) = u_0 e^{\lambda(x-x_0)}$ in Figure 3.18.

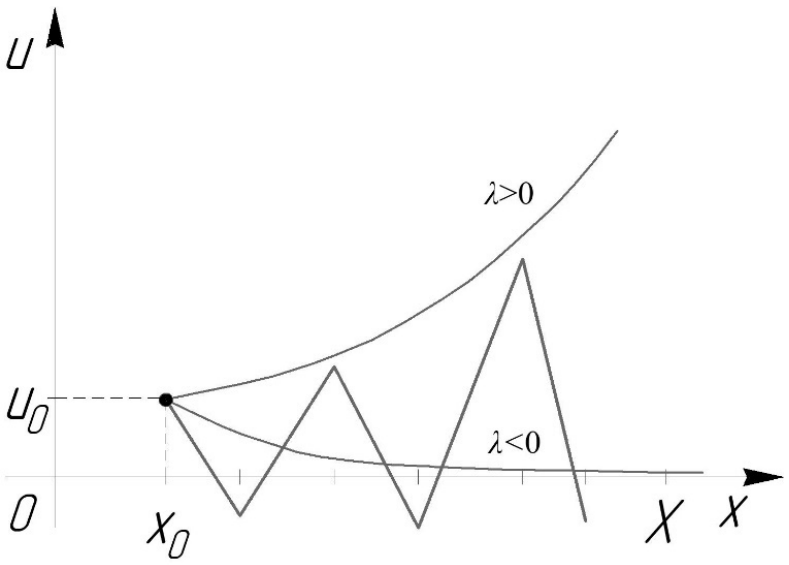


Figure 3.18 – The scheme of the analytical solution of the ODE for different values of λ

Assuming an error of δu in the input data, the exact solution will change over time:

$$u(x) = (u_0 + \delta u)e^{\lambda(x-x_0)} = u_0 e^{\lambda(x-x_0)} + \delta u e^{\lambda(x-x_0)}. \quad (3.83)$$

From equation (3.83), for $\lambda>0$, the error will only increase; in this case, the problem is ill-conditioned. For $\lambda<0$, the error always decreases, so in this case, the problem is stable with respect to input data errors (see Fig. 3.18). However, only if $\lambda<0$ do problems arise when solving the problem by numerical methods. During the solution of «stiff» problems by conventional numerical methods, the integration step should be small enough so that it is possible to take into account the growth of the most rapidly changing components of the solution even after their contribution becomes practically unnoticeable. But the reduction of the step leads to an increase in the consumption of machine time of computer systems, and the accumulation of errors. Moreover, even on a smooth part of the solution, increasing the step leads to an increase in rounding and discretization errors.

The simplest apparatus for solving «stiff» problems is the implicit Euler method, which has the first order of accuracy, where the solution is determined from the following iterative equation:

$$y_{n+1} = y_n + hf\left(y_{n+1}, x_{n+1}\right). \quad (3.84)$$

The algorithm of Euler implicit method for solving «stiff» problems is presented in Figure 3.19.

«Stiff» problems are frequently encountered in the theory of automatic control, such as when analyzing transient processes in a system containing high-order links with coefficients that significantly differ from each other. Differential equations describing the controlled motion of a manipulator robot also exemplify «stiff» systems, as transient processes in the drive control system decay faster than in the mechanical part of the robot.

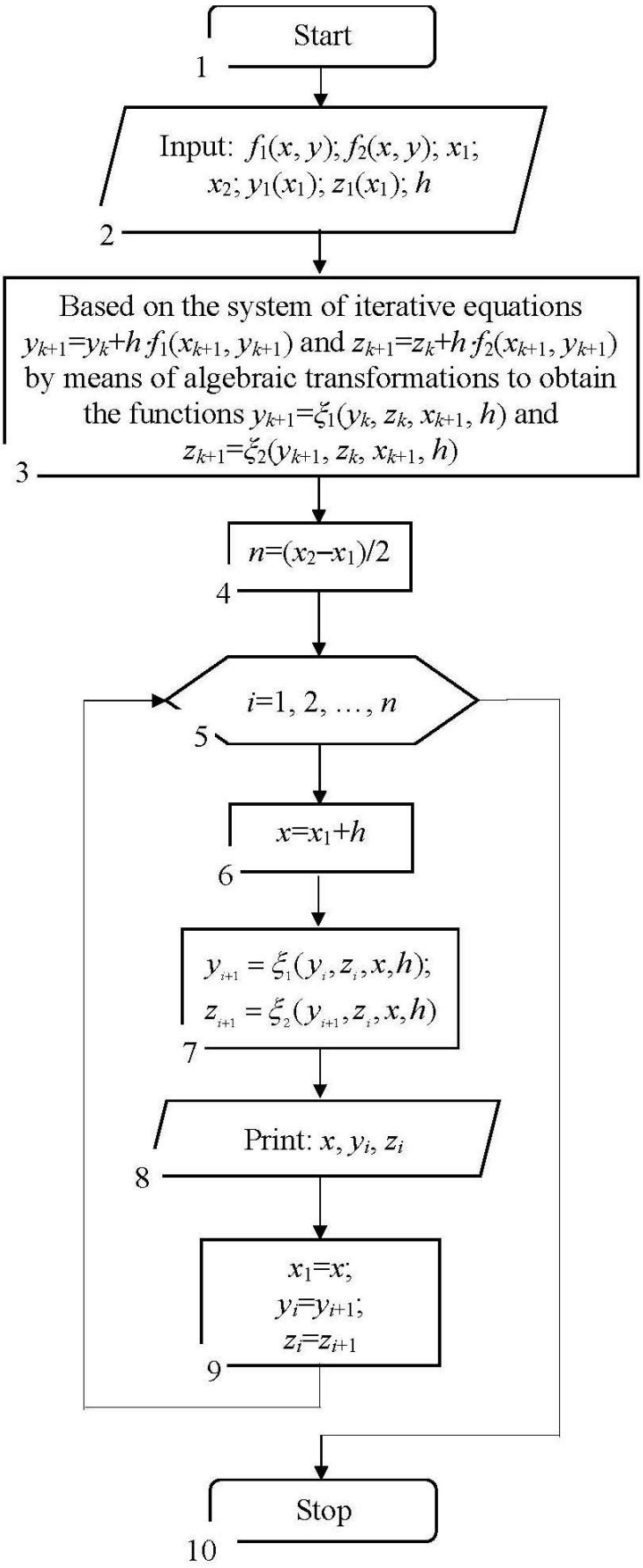


Figure 3.19 – Scheme of the Euler implicit method algorithm for ODE systems

Example 3.9. To solve the numerical method of ODE for the «stiff», problem of a mathematical model describing the behavior of the concentration of chemical substances in a mixture in which an exothermic chemical reaction takes place:

$$\begin{cases} \dfrac{du}{dt} = a_1 u(t) + b_1 v(t); \\ \dfrac{dv}{dt} = -a_2 u(t) - b_2 v(t), \end{cases} \tag{3.85}$$

which satisfies the initial conditions for $u(0) = v(0) = 1{,}0$ mol/l, where $a_1 = 998$ sec^{-1}, $a_2 = 999$ sec^{-1}, $b_1 = 1998$ sec^{-1}, $b_2 = 1999$ sec^{-1} – parameters determining the kinetics of the chemical reaction; $u(t)$, $v(t)$ are the concentrations of the initial chemical substance and the final product, respectively.

Compare the results of the numerical solution with the analytical solution of the ODEsystem for the «stiff» problem (3.84).

Solution:

Compare the results of the numerical solution with the analytical solution of the ODE system of the «stiff» problem (3.84):

$$\begin{cases} \dfrac{du}{dt} = a_1 u(t) + b_1 v(t); \\ \dfrac{dv}{dt} = -a_2 u(t) - b_2 v(t); \end{cases} \rightarrow \begin{cases} u_{n+1} = u_n + h\left(a_1 u_{n+1} + b_1 v_{n+1}\right); \\ v_{n+1} = v_n + h\left(-a_2 u_{n+1} - b_2 v_{n+1}\right), \end{cases} \tag{3.86}$$

where h – step of integration over the time course of a chemical reaction.

The system of iterative equations (3.86) can be solved using Newton's method (see Chapter 2) or the parameters can be expressed u_{n+1} and v_{n+1}:

$$\begin{cases} v_{n+1} = \dfrac{v_n - u_n \lambda h a_2}{1 + a_2 b_1 h^2 \lambda + b_2 h}; \\ u_{n+1} = u_n \lambda h + v_{n+1} \lambda h b_1, \end{cases} \tag{3.87}$$

where $\lambda = 1/\left(1 - a_1 h\right)$.

To solve this problem, the time interval [0; 0,08], which is divided into the corresponding number of parts with a step of $h = 5{,}0 \cdot 10^{-6}$ sec. Then the values of v_1, v_2, …, v_n and u_1, u_2, …, u_n will be determined by Euler implicit method according to formula (3.87), namely:

$$\begin{cases} v_1 = \dfrac{v_0 - (u_0 \lambda h a_2)}{1 + (a_2 b_1 h^2 \lambda) + (b_2 h)} = \\ = \dfrac{1{,}0 - \left(1{,}0 \cdot 1{,}01 \cdot 5{,}0 \cdot 10^{-6} \cdot 999{,}0\right)}{1 + \left(999{,}0 \cdot 1998{,}0 \cdot \left(5{,}0 \cdot 10^{-6}\right)^2 \cdot 1{,}01\right) + \left(1999{,}0 \cdot 5{,}0 \cdot 10^{-6}\right)} = 0{,}985; \\ u_1 = u_0 \lambda h + v_1 \lambda h b_1 = 1{,}0 \cdot 1{,}01 \cdot 5{,}0 \cdot 10^{-6} + 0{,}985 \cdot 1{,}01 \cdot 5{,}0 \cdot 10^{-6} \cdot 1998{,}0 = 1{,}015; \end{cases}$$

$$\begin{cases} v_2 = \dfrac{v_1 - (u_1 \lambda h a_2)}{1 + (a_2 b_1 h^2 \lambda) + (b_2 h)} = \\ = \dfrac{0{,}985 - \left(1{,}015 \cdot 1{,}01 \cdot 5{,}0 \cdot 10^{-6} \cdot 999{,}0\right)}{1 + \left(999{,}0 \cdot 1998{,}0 \cdot \left(5{,}0 \cdot 10^{-6}\right)^2 \cdot 1{,}01\right) + \left(1999{,}0 \cdot 5{,}0 \cdot 10^{-6}\right)} = 0{,}970; \\ u_2 = u_1 \lambda h + v_2 \lambda h b_1 = 1{,}015 \cdot 1{,}01 \cdot 5{,}0 \cdot 10^{-6} + 0{,}970 \cdot 1{,}01 \cdot 5{,}0 \cdot 10^{-6} \cdot 1998{,}0 = 1{,}029; \end{cases}$$

……,

where $\lambda = \dfrac{1}{1 - a_1 h} = \dfrac{1}{1 - 998{,}0 \cdot 5{,}0 \cdot 10^{-6}} = 1{,}01$.

Also, for comparison, an analytical solution of the ODE system of the «stiff» problem was obtained $u(0)=v(0)=1{,}0$:

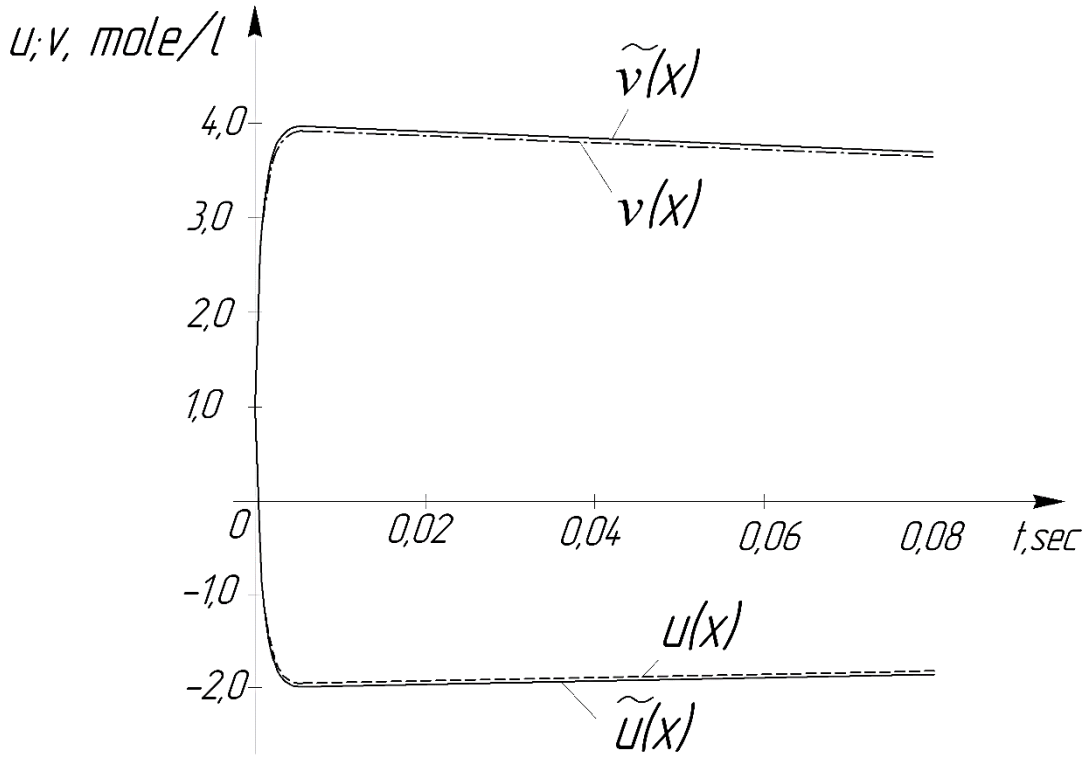
$$\begin{cases} \tilde{u}(t) = 4e^{-t} - 3e^{-1000t}, \\ \tilde{v}(t) = -2e^{-t} + 3e^{-1000t}. \end{cases} \tag{3.88}$$

The numerical results of the calculation of the solution of the ODE system of the «stiff» problem (3.85) for ten integration steps are presented in Table 3.8.

Table 3.8 – Calculation results of the solution of the differential equation

n	$t_n \cdot 10^{-6}$, sec	$v(t)$	$u(t)$	$\tilde{v}(t)$	$\tilde{u}(t)$
0	0,0	1,00000	1,00000	1,00000	1,00000
1	0,5	0,98508	1,01475	0,98505	1,01494
2	1,0	0,97024	1,02944	0,97017	1,02981
3	1,5	0,95548	1,04405	0,95537	1,04460
4	2,0	0,94079	1,05858	0,94064	1,05932
5	2,5	0,92617	1,07305	0,92598	1,07397
6	3,0	0,91163	1,08744	0,91140	1,08854
7	3,5	0,89715	1,10175	0,89689	1,10304
8	4,0	0,88276	1,11600	0,88244	1,11747
9	4,5	0,86843	1,13017	0,86808	1,13183
10	5,0	0,85418	1,14427	0,85379	1,14611

Also, for a visual comparison, it is convenient to present the results of the numerical solution on a graph (Fig. 3.20) for the time interval [0; 0,08].



$u(x)$, $v(x)$ – Euler implicit method; $\tilde{u}(x)$, $\tilde{v}(x)$ – the exact solution

Figure 3.20 – Diagram of the results of the numerical solution of the «stiff» problem ODE system

It can be seen from the graph (Fig. 3.20) that after a small period of time t=0,004 sec, the solution is very close to the functions:

$$\begin{cases} u \cong 4e^{-t}, \\ v \cong -2e^{-t}. \end{cases}$$

From the results obtained, as shown in Table 3.8, it is evident that errors in the process of determining the solution increase towards the end of the table, especially at certain points $x_5 = 2{,}5 \cdot 10^{-6}$ sec i $x_{10} = 5{,}0 \cdot 10^{-6}$ sec:

– relative error by comparing the values of the analytical solution and Euler's implicit method

$$\varepsilon^u_{x_5} = \left| \frac{\tilde{u}_{x_5} - u_{x_5}}{\tilde{u}_{x_5}} \right| \cdot 100\% = \left| \frac{1{,}07397 - 1{,}07305}{1{,}07397} \right| \cdot 100\% = 0{,}09\%,$$

$$\varepsilon^v_{x_5} = \left| \frac{\tilde{v}_{x_5} - v_{x_5}}{\tilde{v}_{x_5}} \right| \cdot 100\% = \left| \frac{0{,}92617 - 0{,}92598}{0{,}92617} \right| \cdot 100\% = 0{,}02\%,$$

$$\varepsilon^{u}_{x_{10}} = \left|\frac{\tilde{u}_{x_{10}} - u_{x_{10}}}{\tilde{u}_{x_{10}}}\right| \cdot 100\% = \left|\frac{1{,}14611 - 1{,}14427}{1{,}14611}\right| \cdot 100\% = 0{,}16\%,$$

$$\varepsilon^{v}_{x_{10}} = \left|\frac{\tilde{v}_{x_{10}} - v_{x_{10}}}{\tilde{v}_{x_{10}}}\right| \cdot 100\% = \left|\frac{0{,}85379 - 0{,}85418}{0{,}85379}\right| \cdot 100\% = 0{,}05\%.$$

Let's consider the implementation of the implicit Euler's method for the system of the ODE «stiff» problem in the PYTHON programming language:

```
# connection of computing libraries
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import *
from sympy import *
# introducing an integration step
h_0=float(input())
#entering the initial value of the integration argument (total integration
     interval) x0
a=int(input())
# entering the final value of the integration interval total
b=float(input())
#entering the initial value of the argument of the differential equation
     function v0
v_0=float(input())
#entering the initial value of the argument of the differential equation
     function u0
u_0=float(input())
#entering the permissible calculation error
e_0=float(input())
lambd_a=1/(1-(998*h_0))
#specification of the 1st iterative formula of the differential equation
     system
def f_1(u,v,h):
    return (v-(u*lambd_a*h*999))/(1+(999*1998*(h**2)*lambd_a)+(1999*h))
#specification of the 2nd iterative formula of the differential equation
     system
def f_2(u,v,h):
    return (u*lambd_a)+(f_1(u,v,h)*lambd_a*h*1998)
#calculation of iterative formulas by Euler's implicit method
def eyler_impl(a,b,u_0,v_0,h):
    v_mas=[v_0]; x=a; x_mas=[x]; u_mas=[u_0]
    u=u_0; v=v_0; x_mtr=[x]; x_mtr=np.array(x)
    while x<=b:
        x=x+h
        v=f_1(u,v,h)
        u=f_2(u,v,h)
        v_mas.append(v)
        u_mas.append(u)
        x_mas.append(x)
        x_mtr=np.append(x_mtr, x)
    return x_mas, v_mas, u_mas
#we construct graphs of the solutions of the differential equation
plt.figure(figsize=(7, 7))
plt.xlabel('x',fontsize=15, color='blue')
plt.ylabel('y',fontsize=15, color='blue')
plt.plot(eyler_impl(a,b,u_0,v_0,h_0)[0], eyler_impl(a,b,u_0,v_0,h_0)[1])
plt.plot(eyler_impl(a,b,u_0,v_0,h_0)[0], eyler_impl(a,b,u_0,v_0,h_0)[2])
plt.plot(eyler_impl(a,b,u_0,v_0,h_0)[3],                          (4*np.exp(-
```

```
     eyler_impl(a,b,u_0,v_0,h_0)[3]))-
         (3*np.exp(-eyler_impl(a,b,u_0,v_0,h_0)[3]*1000)))
plt.plot(eyler_impl(a,b,u_0,v_0,h_0)[3],                          (-2*np.exp(-
     eyler_impl(a,b,u_0,v_0,h_0)[3]))+
         (3*np.exp(-eyler_impl(a,b,u_0,v_0,h_0)[3]*1000)))
plt.legend(['Euler's implicit method for v(x)',' Euler's implicit method for
     u(x)','u(x)=4*e^(-x)-3*e^(-x*1000)','v(x)=-2*e^(-x)+3*e^(-x*1000)'],
     loc=1)
plt.grid(True)
plt.xlim([0, 0.015])
plt.ylim([-2.5, 4.5])
plt.show().
```

Conclusions on the application of numerical methods for solving ordinary differential equations

Numerical methods for solving the ODE do not allow finding a general solution; they can only provide a partial solution. However, these methods can be applied to a wide class of differential equations and all types of problems related to them. Numerical methods can be applied only to correctly posed (or adjustable) problems. However, it should be noted that for the successful application of numerical methods, the formal fulfillment of correctness conditions may not be sufficient. It is necessary for the problem to be well-conditioned, meaning that small changes in the initial conditions would result in sufficiently small changes in the integral curves. If this condition is not met, meaning the problem is ill-conditioned (weakly stable), then small changes in the initial conditions or, equivalent to these changes, small errors in the numerical method could significantly distort the result. To solve ordinary differential equations, it is advisable to use the following methods: Euler, Runge-Kutta of the fourth order, Adams («prediction and corrections»), «shooting», as well as finite differences. Before applying these methods, the type of problem is identified based on the type of ODE (Cauchy problem, boundary value problem, or «stiff» problem) and appropriate methods are applied. There are two types of numerical methods for solving Cauchy problems: one-step methods, where information on only one previous step is required to find the next point on the curve (Euler's and Runge-Kutta methods of the fourth order); multi-step methods, where information about more than one of the previous points is required to find the next point of the curve («prediction and correction» methods or the Adams method). When comparing the efficiency of single-step and multi-step methods, the following features are highlighted: multi-step methods require a large amount of memory in computing systems, as they operate with a large amount of initial data; when using multi-step methods, there is an opportunity to estimate the error per step, therefore, the step size is chosen optimally, which, compared to single-step methods, gives a certain margin, reducing the speed of calculations; with the same accuracy, one-step methods require a smaller number of calculations (for example, using the Runge-Kutta method of the fourth order, it is necessary to calculate four values of the function at each step, and to ensure the convergence of the «prediction and

correction» method of the same order of accuracy - two); one-step methods, unlike multi-step methods, allow you to start solving the problem (self-starting) and easily change the step during the solution process. If the Cauchy problem is very difficult, then the «prediction and correction» (Adams) method is usually preferred, which is also faster. The solution to the problem in this case starts with one-step methods, namely Euler or fourth-order Runge-Kutta. If more than two iterations are needed to calculate the next value y_i, or if the truncation error is significant, then it is necessary to decrease the calculation step h. On the other hand, in the case of a very small slice error, the step can be increased, which will increase the speed, but in this case, the entire solution process is performed first. Sometimes, in practice, it is necessary to minimize the time of preparing a problem for solving. Then it is advisable to use one-step methods. The «shooting» method or difference methods are used to solve boundary value problems. In the case of nonlinear differential equations, difference methods are preferred. When solving «stiff» problems, the easiest way is to use the «implicit» Euler method, where the step must be small enough to account for the growth of the most rapidly changing components of the solution even after their contribution becomes practically negligible. In general, for effective problem-solving, the experience, intuition, and qualification of the researcher are of great importance both in the process of setting the problem and in the process of choosing a method, developing an algorithm, and software solutions by means of computer systems.

Control questions and tasks

1. Give examples of the application of differential calculus in various areas of scientific research.
2. Formulate a generalized formulation of the problem for ordinary differential equations.
3. Formulate the Cauchy problem and the boundary value problem. What is the difference between these problems?
4. What types of numerical methods exist for solving ODE? Give them a comparative characteristic.
5. What is the difference between the usual Euler method and the refined one for the numerical solution of the Cauchy ODE problem?
6. Give a geometric interpretation of Euler method.
7. In the mathematical model of the problem of ballistics, namely the vertical fall of a body of mass m:

$$\frac{d\upsilon}{dt} = -\frac{\alpha}{m}\upsilon + g \ , \tag{3.89}$$

which is acted upon by the force of viscous friction F_{fr}, proportional to the velocity ($F_{fr} = -\alpha\upsilon$, where α is the coefficient of viscous friction). Using the usual Euler

numerical method and the refined one, determine the value of its velocity depending on time. Compare the obtained results of the numerical study with the analytical solution $\upsilon(t) = \frac{mg}{\alpha}\left(1 - e^{-\frac{\alpha}{m}t}\right)$ under the initial condition $\upsilon(0) = 0$ m/sec during the time interval $t \in [0; 10{,}0]$ seconds. The initial data for the calculation are presented in Table 3.9.

Table 3.9 – Output data for the task

Version	Body weight (m, kg)	Coefficient of viscous friction (α, N·sec/m)
1	2,0	0,00035
2	3,2	0,00027
3	3,5	0,00010
4	4,6	0,00004

8. The simplest mathematical model of the battle of two opposing armies is described by the Lanchester battle equations:

$$\begin{cases} \frac{dx(t)}{dt} = -by(t); \\ \frac{dy(t)}{dt} = -ax(t), \end{cases} \tag{3.90}$$

where a, b – the effectiveness of the weapons of armies X and Y, respectively, means that each combat unit of Army X destroys a soldier of Army Y in a unit of time, and vice versa. By using Euler numerical method, determine the value of the change in the number of opposing armies during the battle in the time interval $t \in [0; 100{,}0]$ seconds. How should the initial number of armies be altered to impact the battle's outcome? The initial data for the calculation are presented in Table 3.10.

Table 3.10 – Output data for the task

Version	Weapons effectiveness of the Army X (a, unit/hour)	Weapons effectiveness of the Army Y (b, unit/hour)	Initial size of the Army X ($x(0)$, units)	Initial size of the Army Y ($y(0)$, units)
5	2,0	2,8	100,0	180,0
6	3,2	4,1	220,0	280,0
7	3,5	2,4	300,0	240,0
8	4,6	3,1	410,0	390,0

9. Reveal the essence of the fourth-order Runge-Kutta numerical method for solving the Cauchy ODE problem.

10. Give a geometric interpretation of the Runge-Kutta method of the fourth order.

11. How is the calculation error determined by the Runge-Kutta numerical method of the fourth order?

12. Using the Runge-Kutta numerical method of the fourth order for the simplest mathematical model of the epidemic:

$$\frac{dx}{dt} = \alpha \cdot x(t) \cdot \left[N + 1 - x(t)\right], \tag{3.91}$$

where N – the number of people in the research group, and α is the proportionality coefficient. Determine the value of $x(t)$ – the number of cases of people depending on the units of time with the specified accuracy of the cut-off error R_n^h=0,001. Compare the obtained results of the numerical study with the analytical solution of ODE (3.91) for the time interval $t \in [0; 150{,}0]$ seconds, namely: $x(t) = \frac{N+1}{N \cdot e^{-\alpha(N+1)t} + 1}$. The initial data for the calculation are presented in Table 3.11.

Table 3.11 – Output data for the task

Version	Number of people in the research group (N, person)	Coefficient of proportionality (α, sec^{-2})
9	100,0	0,0010
10	80,2	0,0012
11	50,5	0,0022
12	60,6	0,0035

13. Using the Runge-Kutta numerical method of the fourth order for the mathematical model of the ballistics problem, namely the vertical fall of a body by mass m:

$$\frac{d^2x}{dt^2} = \left(-\frac{\alpha}{m}\right)\frac{dx}{dt} + g\ , \tag{3.92}$$

which is acted upon by the force of viscous friction F_{fr}, proportional to the velocity ($F_{fr} = -\alpha v$, where α is the coefficient of viscous friction), determine the value of its displacement depending on time t with the specified accuracy

R_n^h=0,001 under the initial condition $v(0)$=0 m/sec and drop height h. The initial data for the calculation are presented in Table 3.12.

Table 3.12 – Output data for the task

Version	Body weight (m, kg)	Coefficient of viscous friction (α, N·sec/m)	Initial height of fall (h, m)
13	2,0	0,00035	120,0
14	3,2	0,00026	110,0
15	3,5	0,00010	100,0
16	4,6	0,00004	150,0

14. Reveal the essence of the numerical method of «prediction and correction» (Adams') for solving the Cauchy problem of ODE.

15. How the calculation error is determined by the numerical method of «prediction and correction» (Adams') for solving the Cauchy problem of the ODE?

16. What is the «self-start» property? What methods for solving ODE have it?

17. What is the order of accuracy of the numerical methods of Euler, Runge-Kutta, and Adams for solving the Cauchy ODE problem?

18. Using the numerical method of «prediction and correction» (Adams') for a mathematical model of limited population growth (Ferhulst's logistic model):

$$\frac{dN}{dt} = rN - kN^2, \tag{3.93}$$

where r – the average growth rate of the population, k, is the meeting coefficient of competing individuals. Determine the value of the population size N depending on the time t with the specified accuracy of the error of the cut R_n^h=0,0001 for the time interval $t \in [0; 5,0]$. Determine the smallest population size in the presence of competition. The initial data for the calculation are presented in Table 3.13.

Table 3.13 – Output data for the task

Version	Initial population size (N, unit)	Average population growth rate (r, 1/time unit)	Coefficient of meeting of competing persons (k, person/unit of time)
17	1000,0	3,23	2,15
18	900,0	3,80	2,80
19	800,0	4,10	5,10
20	700,0	2,50	1,95

19. In the mathematical model of free oscillations of a sprung body of mass m under the linear resistance of the medium ($F_{fr}= -\alpha(dx/dt)$, where α is the coefficient of viscous friction)):

$$m\frac{d^2x}{dt^2} = -kx - \alpha\frac{dx}{dt}, \tag{3.94}$$

where k – stiffness coefficient of the elastic element, using the numerical method of «prediction and correction» (Adams'), determine the value of displacement x depending on time t with the specified accuracy Δ=0,0001 under the initial condition $x(0)$=0,02 m for the time interval $t\in$[0; 5,0] seconds. The initial data for the calculation are presented in Table 3.14.

Table 3.14 – Output data for the task

Version	Body weight (m, kg)	Coefficient of viscous friction (α, N·sec/m)	Stiffness coefficient of an elastic element (k, N/m)
21	10,0	45,0	$5{,}0\cdot10^3$
22	12,0	42,0	$8{,}0\cdot10^3$
23	8,0	52,0	$11{,}0\cdot10^3$
24	9,0	38,0	$13{,}0\cdot10^3$

20. A mathematical model for forecasting climatic conditions based on the Lorenz model (a model of the physical process of two-dimensional thermal convection):

$$\begin{cases} \dfrac{dx(t)}{dt} = \sigma\big(y(t) - x(t)\big); \\ \dfrac{dy(t)}{dt} = Rx(t) - y(t) - x(t)z(t); \\ \dfrac{dz(t)}{dt} = x(t)y(t) - bz(t), \end{cases} \tag{3.95}$$

where $x(t)$ – intensity of convection; $y(t)$ is the difference between the temperatures of the ascending and descending air flows; $z(t)$ – deviation of the vertical temperature profile from the linear dependence; R is the normalized Rayleigh number (reflects the behavior of the airflow under the influence of the temperature gradient); σ – Prandtl number (criterion of the similarity of thermal processes in liquids and gases); b – geometric parameters of the convective calculation cell. With the help of any one-step or multi-step numerical method, plot the dependence diagram of $x(y, z)$ on the time interval $t\in$[0; 100] seconds with a calculation step of t=0,01 sec. As initial parameters, select the following

values: $x(t)$=0,0; $y(0)$=1,0; $z(t)$=1,05. The initial data for the calculation are presented in Table 3.15.

Table 3.15 – Output data for the task

Version	Normalized Rayleigh number (R)	Prandtl number (σ)	Geometric parameters of the convective calculation cell (b)
25	28,0	10,0	10/3
26	30,0	12,0	11/3
27	31,0	13,0	13/5
28	27,0	9,0	9/2
29	29,0	11,0	14/3
30	33,0	14,0	15/4

21. What is the difference between the initial and boundary conditions of the problem statement during the solution of the ODE?

22. Reveal the essence of the numerical method of «shooting» for solving the boundary value problem of the ODE.

23. The problem of the brachystochrone (the optimal path that the body takes in the minimum time under the influence of the Earth's gravity) is given by a mathematical model:

$$2y\frac{d^2y}{dx^2}+1+\left(\frac{dy}{dx}\right)^2=0, \tag{3.96}$$

where y – coordinate of the path along the x axis; $0<x<L$ cm – coordinates of the path along the x axis; $\frac{dy(0)}{dx}=0$; $0<y<H$ cm – coordinates of the path along the y axis. Using the numerical «shooting» method, determine the value of the y coordinates of the optimal path curve depending on the x coordinate with an accuracy of ε=0,0001. The initial data for the calculation are presented in Table 3.16.

Table 3.16 – Output data for the task

Version	The initial position of the body along the vertical axis y (H, cm)	The initial position of the body along the horizontal axis x (L, cm)
1	35,0	45,0
2	30,0	20,0
3	40,0	50,0
4	25,0	38,0

24. Reveal the essence of the numerical method of finite differences for solving the boundary value problem of ODE. What are the stages of solving a problem using the finite difference method?

25. Write down the general formula for approximating derivatives at some point?

26. The problem of compressing a viscoplastic rod under longitudinal impact is given by a mathematical model:

$$\frac{d^2\delta}{dx^2} + ax\left(\frac{d\delta}{dx}\right)^b = 0, \tag{3.97}$$

where δ – compression value of the rod along the x axis; $0<x<L$ m – the coordinate of the length of the rod along the x axis; b=1,0 is a dimensionless parameter of the shock characteristic of the system; a is the stiffness of the rod under tension and compression. Using the numerical method of finite differences, determine the value of the local compression of the rod depending on the longitudinal coordinate x under the following boundary conditions $\delta(0)$=0 mm, $\delta(L) = \Delta$ mm. The initial data for the calculation are presented in Table 3.17.

Table 3.17 – Output data for the task

Version	Rod length (L, m)	Stiffness of the rod in tension and compression (a)	Deformation of the edge of the rod at the point of impact (Δ, mm)
5	1,0	0,0034	1,2
6	1,5	0,0038	1,5
7	1,8	0,0023	2,1
8	2,5	0,0048	2,8

27. What differential equations are called «stiff»? What are the peculiarities of their solution?

28. Reveal the essence of Euler numerical implicit method for solving «stiff» problems of ODE?

29. The problem of chemical reaction kinetics is given by a mathematical model:

$$\begin{cases} \dfrac{dx}{dt} = -0,5x + 30y; \\ \dfrac{dy}{dt} = -30y, \end{cases} \tag{3.98}$$

where x – the current value of the concentration of chemical substance A; y is the current value of the concentration of chemical substance B. Using Euler's implicit numerical method for this «stiff problem» of the ODE system (3.98), determine the value of the change in the concentration of chemical substances $x(t)$ and $y(t)$

during the time interval $t\epsilon[0; 2,0]$ seconds with initial data $x(0)=X_0$ та $y(0)=Y_0$. The initial data for the calculation are presented in Table 3.18.

Table 3.18 – Output data for the task

Version	The initial value of the concentration of chemical substance *A* (X_0, mol/l)	The initial value of the concentration of the chemical substance *B* (Y_0, mol/l)
9	1,6	2,8
10	5,5	3,5
11	3,8	2,9
12	4,5	5,1

Chapter 4. DIFFERENTIAL EQUATIONS OF MATHEMATICAL PHYSICS

One of the characteristic features of modern research is the mathematization of physical knowledge, which involves the intensive use of mathematical modeling methods in non-traditional and even «descriptive» sciences such as ecology and medicine. In today's practice, scientists are required to solve various kinds of problems, the thorough examination of which can often only be done numerically or through a carefully designed physical experiment. Therefore, the development of general numerical methods (algorithms) for solving problems in mathematical physics and nonlinear mechanics is highly significant.

The subject of mathematical physics is the construction and research of mathematical models of physical phenomena. The problems of classical mathematical physics are reduced to boundary value problems for partial differential equations. The main means of researching such problems are the theory of differential equations together with the theory of functions, calculus of variations, functional analysis, theory of probability, and computational mathematics.

If we denote by D the region of the n-dimensional space of R^n points $x=(x_1, x_2, \ldots, x_n)$; $x_1, x_2, \ldots, x_n$; $n\geq 2$ are Cartesian coordinates of point x, then equations of the form:

$$F\left(x, u, \frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial^k u}{\partial x_1^{i_1} \ldots \partial x_n^{i_n}}, \frac{\partial u}{\partial x_n^m}\right) = 0 \ \ (x\in D;\ \sum_{j=1}^{n} i_j = k\ ;\ k=0, 1, \ldots, m) \quad (4.1)$$

are called a partial differential equation of order m with respect to the unknown function $u=u(x)$, where $F = F\left(x, u, \frac{\partial u}{\partial x_1}, \ldots\right)$ is a given real function of the points $x\in D$, the unknown function u and its partial derivatives. The left part of equation (4.1) is called a differential operator with partial derivatives of order m.

The real function $u=u(x_1, x_2, \ldots, x_n)$, defined in the region D by equation (4.1), is continuous along with its partial derivatives included in this equation, and when it is inverted into the identity, it is called a classical (regular) solution in terms of equation (4.1).

The solution of equation (4.1) in the n+1-dimensional space of variables x_1, $x_2, \ldots, x_n$, u determines some smooth surface of dimension n, which is called the integral surface of equation (4.1).

Many problems in the physics of continuous media are reduced to solving differential equations with partial derivatives. In such cases, the desired functions are usually density, temperature, stress, and others, with the arguments being the coordinates of the point being considered in space, as well as time itself.

In particular, the thermal conductivity equation is used to describe the temperature distribution in a given area of space and its change over time:

$$\frac{\partial u}{\partial t} - \alpha\left(\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2} + \cdots + \frac{\partial^2 u}{\partial x_n^2}\right) = f(x,t), \tag{4.2}$$

where $x = (x_1, \ldots, x_n)$ are Cartesian coordinates, $f(x, t)$ is a function of heat sources, α is a coefficient of thermal conductivity, $u(x, t)$ is the desired temperature function at a point with coordinates x at time t. If $f(x, t) \equiv 0$, that is, there are no heat sources and «sinks» inside the system, then the heat conduction equation (4.2) is called homogeneous.

The complete mathematical formulation of the problem, along with the differential equations, also includes some additional conditions. If the search for a solution is conducted within a limited domain, boundary conditions are also defined. In such cases, the problem is referred to as a boundary value problem for equations with partial derivatives.

If one of the independent variables is time t, then the values of the sought functions at the initial moment of time t=0 are set, and they are called initial conditions. In particular, for the thermal conductivity equation (4.2), the boundary and initial conditions are $u|_{\Omega} = \psi(t)$ and $u|_{t=0} = \varphi(x)$, respectively, where $\Omega \in \mathbb{R}^n$ (n = 2, 3) is a two-dimensional or three-dimensional region.

The problem in which it is necessary to solve a partial differential equation under given initial conditions is called the Cauchy problem. At the same time, the problem is solved in an unbounded space, and boundary conditions are not set. Problems, in which both boundary and initial conditions are set at the same time, are called non-stationary (or mixed) boundary value problems. The solution that will be obtained in this case changes over time.

This section will only consider correctly posed problems, that is, problems whose solution exists and is unique in a certain class of initial and boundary conditions.

4.1 Classification of partial differential equations

Let $u(x, y)$ be an unknown function of two variables, x and y, which must be determined. Then, a sufficiently narrow class of problems for equations of the first and second orders, linear with respect to the derivatives, is expressed by the following form of the equation:

$$\begin{aligned} &A(x,y)\frac{\partial^2 u}{\partial x^2} + 2B(x,y)\frac{\partial^2 u}{\partial x \partial y} + C(x,y)\frac{\partial^2 u}{\partial y^2} + D(x,y)\frac{\partial u}{\partial x} + \\ &+E(x,y)\frac{\partial u}{\partial y} + F(x,y) = G, \end{aligned} \tag{4.3}$$

where A, B, C, D, E, F – functional coefficients that can depend on the arguments x, y, and on the function u.

Depending on this equation (4.3) can be:

a) a second-order equation in partial derivatives with constant coefficients;

b) linear, if the right-hand side of the equation depends linearly on the function u, and the coefficients depend only on x, y;

c) quasi-linear if the coefficients depend on u.

Similarly to ordinary differential equations, the unique solution of the equation can be obtained only by setting additional conditions. However, since there are two independent variables, x and y, in equation (4.2), the condition must be set for some curve in the x, y plane. This condition can be imposed on the function u or (and) on its derivatives, depending on the type of equation that determines its type and nature of change.

Different types of equations are distinguished depending on the ratio between the coefficients:

1) for $A = B = C = D = F = 0$, $D \neq 0$, $E \neq 0$ we have the transfer equation

$$\frac{\partial u}{\partial x} + P\frac{\partial u}{\partial y} = G \ (P=E/D). \tag{4.4}$$

If time is one of the independent variables in equation (4.3), then this equation is called evolutionary.

2) if at least one of the coefficients $A = B = C \neq 0$, then equation (4.3) is a second-order equation. In this case, depending on the discriminant $Ds = 4B^2 - 4AC$, equation (4.2) can belong to one of three types: hyperbolic ($Ds > 0$), parabolic ($Ds = 0$), or elliptic ($Ds < 0$)

Equations can change from one type to another depending on the values of the corresponding coefficients.

In the case when the coefficients A, B, C are constant, equation (4.3) has the same type at all points of the plane of variables x and y. In the event that the coefficients A, B, C depend continuously on x and y, the set of points in which this equation belongs to the hyperbolic (elliptic) type forms an open region on the plane, which is called hyperbolic (elliptic), and the set of points in whose equations belong to the parabolic type, is closed. Equation (4.3) is called mixed (of mixed type) if it is hyperbolic in some points of the plane, and elliptic in others. In this case, the parabolic points usually form a line called the line of change of type or the line of degeneration.

There are two types of methods for solving equations of this type: analytical (the result is derived by various mathematical transformations), and numerical, in which the obtained result corresponds to the real one with a given accuracy, but many algebraic calculations are necessary, which requires the use of computing power of computer systems.

This section is devoted to numerical methods, algorithms and their application for partial differential equations of the second order, which are most often used in scientific and applied engineering problems.

Examples of some partial differential equations that describe different types of problems are given in Table 4.1.

Table 4.1 – Differential equations in partial derivatives

Equation type	Mathematical form	Examples of problems
Laplace	$\Delta u = 0$	A steady flow of liquid. Stationary thermal fields
Poisson	$\Delta u = -k$	Heat transfer with an internal heat source
Diffusion	$\Delta u = \frac{1}{h^2}\frac{\partial^2 u}{\partial t^2}$	Unsteady thermal conductivity
Wavy	$\Delta u = \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2}$	Propagation of waves (sound, electromagnetic, etc.)
Biharmonic	$\Delta^2 u = F(x, y)$	Plate deformation

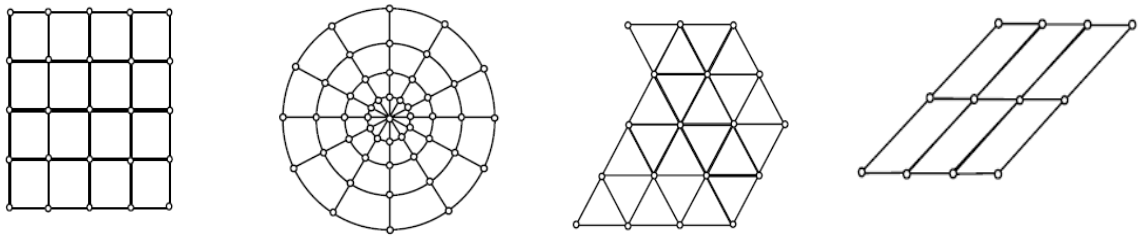
Table 4.1 uses accepted designations of the most common operators: $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ – Laplace, $\Delta^2 u = \frac{\partial^4 u}{\partial x^4} + 2\frac{\partial^4 u}{\partial x^2 \partial y^2} + \frac{\partial^4 u}{\partial y^4}$ – biharmonic.

There are two main methods of numerical solution for partial differential equations: the difference method (finite difference method) and the finite element method. In modern applied mathematics, both methods are considered as interpretations of the use of the general theory of difference schemes for solving partial differential equations.

The basis of the finite element method is variational calculus. Differential equations that describe the problem and corresponding boundary conditions are used to formulate the variational problem. In the finite element method, the physical problem is replaced by a piecewise smooth model. This method requires a complex formulation of the problem, high qualification, and experience. It is not universal, as each solution is only applicable to a specific problem. The finite element method has found wide use in solving special problems in theoretical mechanics, hydrodynamics, and field theory. It is complex and requires serious training and knowledge in a specific field of use.

4.2 Finite difference method

The apparatus of difference methods is an effective means of numerical solution for both ordinary and partial differential equations. In section 3.3, the main principles of constructing difference systems were discussed, which are based on representing an independent variable as a discrete set of points, known as a grid. In addition to the commonly used rectangular grid, other types such as polar, triangular, slanted, and others are also employed (Fig. 4.1). Multidimensional grids are used for solving partial derivative problems with multiple independent variables.



a) b) c) d)

Figure 4.1 – Schemes of calculation grids:
a) rectangular; b) polar; c) triangular; d) beveled

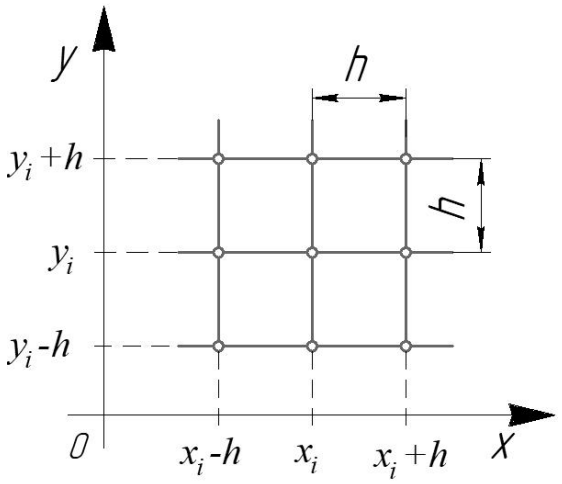


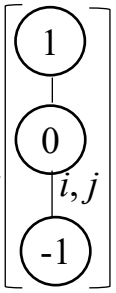
Figure 4.2 – Calculation scheme of a
two-dimensional square grid

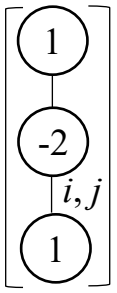
For differential equations of the second order in partial derivatives, a two-dimensional rectangular grid is most often used (Fig. 4.1, a). The central-difference templates, which are applied on a two-dimensional square grid with a step size h (Fig. 4.2), can be obtained similarly to the one-dimensional case (the index j refers to the independent variable y, and the index i refers to x).

For convenience, the notation $u(x_i+h, y_i)$ must be replaced by $u_{i+1,j}$. Using the following notations and performing an expansion in a Taylor series, we obtain expressions for partial derivatives, namely:

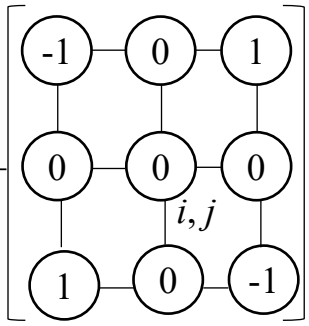
$$\frac{\partial u}{\partial x} \approx \frac{u_{i+1,j} - u_{i-1,j}}{2h} \approx \frac{1}{2h}\left[\underset{i,j}{(-1)\,(0)\,(1)}\right];$$

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} \approx \frac{1}{h^2}\left[\underset{i,j}{(1)\,(-2)\,(1)}\right];$$

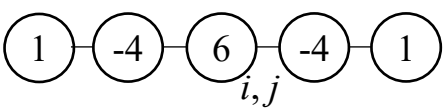
$$\frac{\partial u}{\partial y} \approx \frac{u_{i,j+1} - u_{i,j-1}}{2h} \approx \frac{1}{2h}$$ 

; $$\frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} \approx \frac{1}{h^2}$$ 

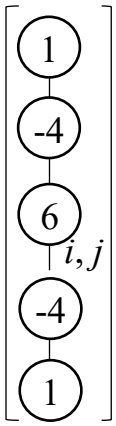
;

$$\frac{\partial^2 u}{\partial x \partial y} \approx \frac{u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1} + u_{i-1,j-1}}{4h^2} \approx \frac{1}{4h^2}$$ 

;

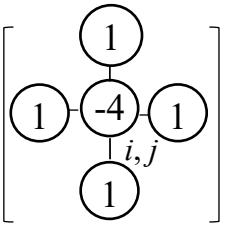
$$\frac{\partial^4 u}{\partial x^4} \approx \frac{u_{i-2,j} - 4u_{i-1,j} + 6u_{i,j} - 4u_{i+1,j} + u_{i+2,j}}{h^4} \approx \frac{1}{h^4}$$ 

;

$$\frac{\partial^4 u}{\partial y^4} \approx \frac{u_{i,j+2} - 4u_{i,j+1} + 6u_{i,j} - 4u_{i,j-1} + u_{i,j-2}}{h^4} \approx \frac{1}{h^4}$$ 

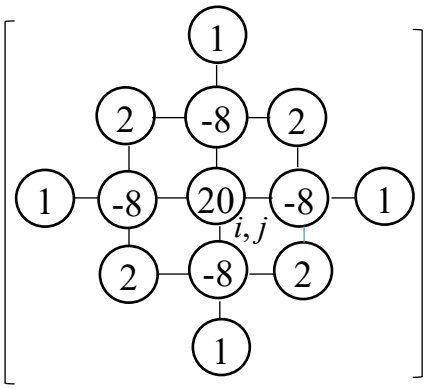
.

More complex computational templates for differential equations are built from these elements. The addition of derivatives is carried out by superposition of the corresponding calculation patterns. This method constructs templates for Δu and $\Delta^2 u$, which have an error of order h^2:

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i-1,j} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1} + u_{i,j-1}}{h^2} \approx \frac{1}{h^2}$$ 

;

$$\Delta^2 u = \frac{\partial^4 u}{\partial x^4} + 2\frac{\partial^4 u}{\partial x^2 \partial y^2} + \frac{\partial^4 u}{\partial y^4} \approx \frac{1}{h_4}$$



All of the computational patterns shown have second-order errors. It is possible to construct more accurate computational patterns if additional nodes are considered. The central-difference approximation is the basis of all the computational templates built above. Sometimes, left or right differences are used to minimize the spread of errors. When using computational templates, the difference equation (approximate partial differential equation) can become unstable. A difference scheme is considered unstable if the error does not decrease with each iteration step. Problems of instability of difference schemes especially arise in evolutionary problems.

By applying the computational template to each of the *n* nodes of the grid, we obtain a system of *n* equations, which can be linear if the initial differential equation has the appropriate structure. In this case, solving the problem is reduced to solving the system of equations in the form of.

$$\begin{bmatrix} coefficient \\ matrix \end{bmatrix} \begin{bmatrix} unknown\ value\ in \\ nodes\ (vector\text{ - }column) \end{bmatrix} = \begin{bmatrix} vector\text{ - }column \\ of\ \ free\ members \end{bmatrix},$$

which is most often solved by iterative methods (see Chapter 1).

4.3 Solving various types of partial differential equations

Practical methods and algorithms for solving various types of partial differential equations have certain features and require separate consideration. This can be demonstrated using the most common problems as examples.

Solving elliptic equations

Many different physical problems can be reduced to elliptic equations, such as the calculation of stresses arising during the elastic torsion of a long cylindrical rod, the distribution of electric stresses on the conductor plane, and the problem of stationary heat flows in a flat body, among others.

Most elliptic equations are described by Poisson's equation or its partial case – Laplace's equation.

One of the well-known problems is the classical Dirichlet problem for the Laplace equation in a rectangular domain. It is necessary to define a continuous function $u(x, y)$ that satisfies the Laplace equation $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ inside the rectangular domain $\Omega = \{(x;\ y)\ |0 \le x \le a,\ 0 \le y \le b\}$, and also acquires given values at the boundary of the domain: $x = 0$, $u(0,\ y) = u_1(y)$; $x = a$, $u(a,\ y) = u_2(y)$; $y = 0$, $u(x, 0) = u_3(x)$; $x = b$, $u(x, b) = u_4(x)$.

A two-dimensional grid with a pitch h along the x-axis and l along the y-axis is introduced into the solution domain. Then, using the given notation and approximating the Laplace equation by a difference equation (see Chapter 4.2), we obtain the following system of linear equations ($l = h$):

$$\begin{cases} u_{i,j} = \frac{1}{4}\left(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}\right); \\ u_{i,0} = u_3(x_i),\ u_{i,m} = u_4(x_i),\ u_{0,j} = u_1(y_i),\ u_{n,j} = u_2(y_i); \\ i = 1, 2, \ldots, n-1;\ j = 1, 2, \ldots, m-1. \end{cases} \tag{4.5}$$

Such a system of equations has a large number of zero elements and satisfies the condition of convergence when using iterative methods. To solve systems of equations of the type (4.5), the Gauss-Seidel method (see Chapter 1) is most often used, which, when applied to elliptic differential equations, is called the Liebmann method or the method of successive displacements. The algorithm for solving elliptic differential equations based on the Laplace equation and the difference scheme (4.5) by the Gauss-Seidel method is presented in Figure 4.3.

It should be noted that any elliptic equations that do not contain $\frac{\partial^2 u}{\partial x \partial y}$ reduce to systems of difference equations that can be solved by both the Liebman's method and other iterative methods, as long as sufficient convergence conditions are used. For elliptic equations that contain $\frac{\partial^2 u}{\partial x \partial y}$, in the general case, the issue of convergence of iterative methods has no theoretical solution. Therefore, it is necessary to consider the obtained system of equations separately in each case.

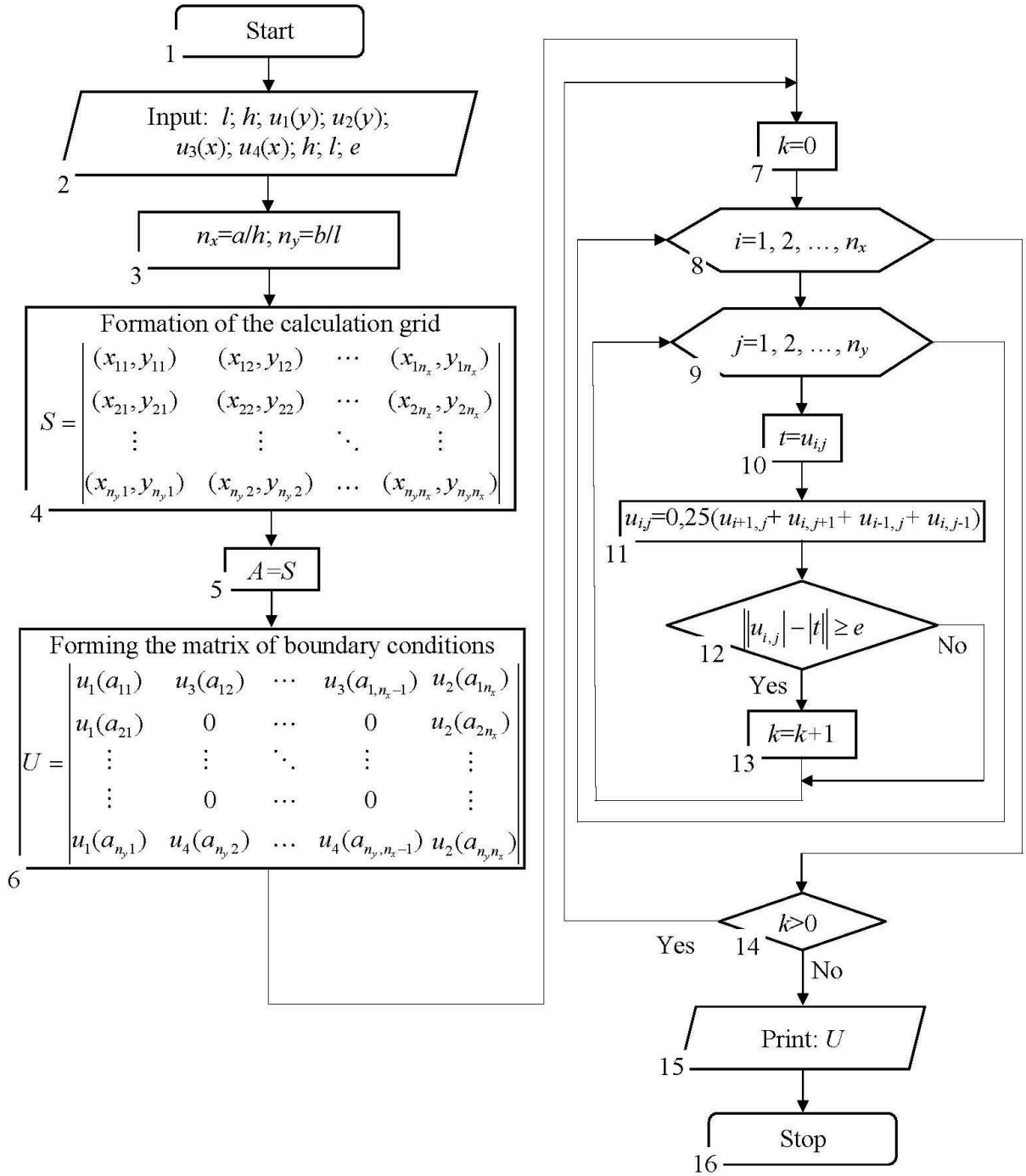


Figure 4.3 – Scheme of the algorithm for solving elliptic partial differential equations

Example 4.1. Determine the stationary temperature distribution in the plate by size $L \times H = 1{,}0 \times 1{,}0$ m, for which the following boundary conditions are specified: $u(0,y) = u_1(y) = 0^{\circ}$C; $u(L,y) = u_2(y) = 100\ ^{\circ}$C; $u(x,0)=u_3(x)=100x$ grad; $u(x, H) = u_4(x) = 100x^2$ grad.

Solution:

The steady-state temperature distribution for a flat body is described by the homogeneous heat conduction equation (4.1), namely the Laplace equation with two independent variables, x and y:

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0\,. \tag{4.6}$$

To formulate the problem, it is necessary to enter a two-dimensional grid on the plate with a distance between nodes of h=0,25 m (Fig. 4.4). The grid contains 25 nodes, out of which 16 have known temperature values based on the boundary conditions. It is necessary to determine the temperature in all 9 internal grid nodes. The serial number of the calculation node is indicated by the index i on the x-axis and by the index j on the y-axis. The new value of the node temperature $u_{i,j}$ can be calculated using the calculation template (see section 4.3) from equation (4.6), namely:

since

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} = 0\,,$$

then

$$u_{i,j} = 0{,}25\left(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}\right). \tag{4.7}$$

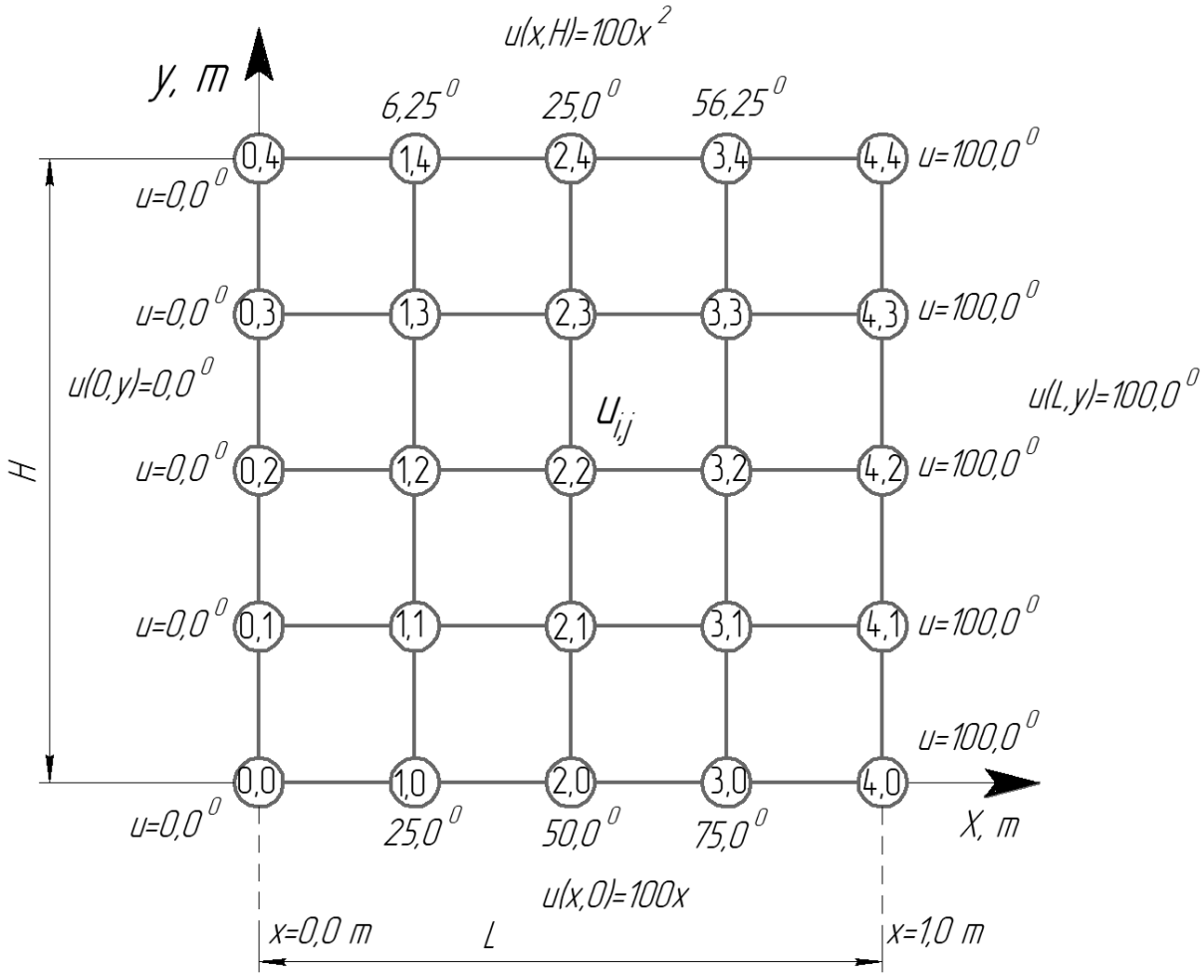


Figure 4.4 – Calculation scheme of stationary temperature distribution

Based on equation (4.7), we write down the system of equations for the temperature value of each node in the calculation grid of the plate:

$$\begin{cases} u_{11} = 0,25\left(u_{21} + u_{12} + u_{01} + u_{10}\right); \\ u_{21} = 0,25\left(u_{31} + u_{22} + u_{11} + u_{20}\right); \\ u_{31} = 0,25\left(u_{41} + u_{32} + u_{21} + u_{30}\right); \\ u_{12} = 0,25\left(u_{22} + u_{13} + u_{02} + u_{11}\right); \\ u_{22} = 0,25\left(u_{32} + u_{23} + u_{12} + u_{21}\right); \\ u_{32} = 0,25\left(u_{42} + u_{33} + u_{22} + u_{31}\right); \\ u_{13} = 0,25\left(u_{23} + u_{14} + u_{03} + u_{12}\right); \\ u_{23} = 0,25\left(u_{33} + u_{24} + u_{13} + u_{22}\right); \\ u_{33} = 0,25\left(u_{43} + u_{34} + u_{23} + u_{32}\right), \end{cases} \quad (4.8)$$

where $u_{00} = u_{01} = u_{02} = u_{03} = u_{04} = 0$ °C; $u_{40} = u_{41} = u_{42} = u_{43} = u_{44} = 100$ °C; since $u_{14} = 100x^2 = 100h^2 = 100 \cdot 0,25^2 = 6,25$ °C, then $u_{24} = 25,0$ °C; $u_{34} = 56,25$ °C; then $u_{14} = 100x = 100h = 100 \cdot 0,25 = 25,0$ °C, then $u_{20} = 50,0$ °C; $u_{30} = 75,0$ °C.

The actual temperature values in all nine internal nodes of the grid will be determined by the Gauss-Seidel method (see Chapter 1). For this, the system of equations (4.8) must be written in iterative form, assuming that the initial value of the temperature in the desired nodes is zero.

$$\begin{cases} u_{11}^{(n)} = 0,25\left(u_{21}^{(n-1)} + u_{12}^{(n-1)} + u_{01} + u_{10}\right); \\ u_{21}^{(n)} = 0,25\left(u_{31}^{(n-1)} + u_{22}^{(n-1)} + u_{11}^{(n)} + u_{20}\right); \\ u_{31}^{(n)} = 0,25\left(u_{41} + u_{32}^{(n-1)} + u_{21}^{(n)} + u_{30}\right); \\ u_{12}^{(n)} = 0,25\left(u_{22}^{(n-1)} + u_{13}^{(n-1)} + u_{02} + u_{11}^{(n)}\right); \\ u_{22}^{(n)} = 0,25\left(u_{32}^{(n-1)} + u_{23}^{(n-1)} + u_{12}^{(n-1)} + u_{21}^{(n)}\right); \\ u_{32}^{(n)} = 0,25\left(u_{42} + u_{33}^{(n-1)} + u_{22}^{(n)} + u_{31}^{(n)}\right); \\ u_{13}^{(n)} = 0,25\left(u_{23}^{(n-1)} + u_{14} + u_{03} + u_{12}^{(n)}\right); \\ u_{23}^{(n)} = 0,25\left(u_{33}^{(0)} + u_{24} + u_{13}^{(1)} + u_{22}^{(1)}\right); \\ u_{33}^{(n)} = 0,25\left(u_{43} + u_{34} + u_{23}^{(1)} + T_{32}^{(1)}\right), \end{cases} \quad (4.9)$$

where $u_{11}^{(0)} = u_{21}^{(0)} = u_{31}^{(0)} = u_{12}^{(0)} = u_{22}^{(0)} = u_{32}^{(0)} = u_{13}^{(0)} = u_{23}^{(0)} = u_{33}^{(0)} = 0$ – initial temperature values in the calculation nodes; $n = 1, 2, \ldots$ – computational iteration number.

Using the Gauss-Seidel calculation algorithm (see Fig. 4.3) with a calculation error of $\varepsilon = 0.001$, the following calculation result was obtained after 17 iterations:

$$u^{(17)} = \begin{vmatrix} u_{04} & u_{14} & u_{24} & u_{34} & u_{44} \\ u_{03} & u_{13} & u_{23} & u_{33} & u_{43} \\ u_{02} & u_{12} & u_{22} & u_{32} & u_{42} \\ u_{01} & u_{11} & u_{21} & u_{31} & u_{41} \\ u_{00} & u_{10} & u_{20} & u_{30} & u_{40} \end{vmatrix} = \begin{vmatrix} 0,0 & 6,25 & 25,0 & 56,25 & 100,0 \\ 0,0 & 16,35 & 38,06 & 66,35 & 100,0 \\ 0,0 & 21,09 & 44,53 & 71,09 & 100,0 \\ 0,0 & 23,49 & 47,88 & 73,49 & 100,0 \\ 0,0 & 25,0 & 50,0 & 75,0 & 100,0 \end{vmatrix}. \quad (4.10)$$

Based on the results of calculations (4.10), it is possible to construct a three-dimensional graph of the stationary temperature distribution in the plate (Fig. 4.5).

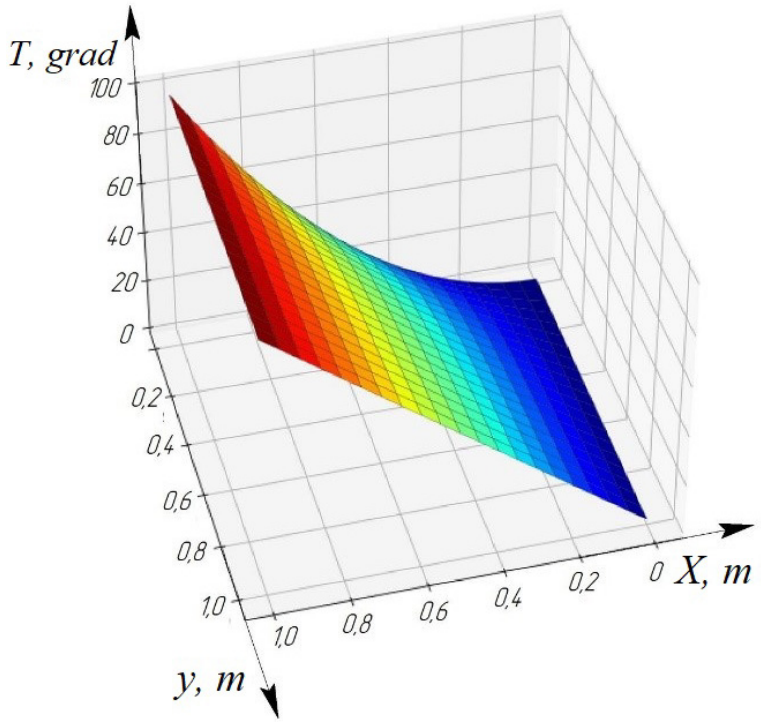


Figure 4.5 – Diagram of the stationary temperature distribution in the plate

The result of solving example 4.1 in the PYTHON programming language:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import *
from sympy import *
from matplotlib.colors import LinearSegmentedColormap
# entering the length of the plate
l=float(input())
# entering the width of the plate
h=float(input())
# entering the step value of the calculation grid along the x-axis
h_x=float(input())
# entering the step value of the calculation grid along the y axis
h_y=float(input())
```

```
# entering the value of the boundary conditions:
 # entering the temperature value on the left wall of the plate
t_left=float(input())
 # entering the temperature value on the right wall of the plate
t_right=float(input())
# introduction of the initial temperature distribution function on the upper
     wall of the plate
def t_top(x):
    return (t_right/(l*l))*x*x
# introduction of the initial temperature distribution function on the bottom
     wall of the plate
def t_down(x):
    return (t_right/l)*x
# entering the calculation error value
e=float(input())
# input of boundary condition values:
# for the upper wall
side_top, side_down=[],[]
x=0
for i in range(0,int(l/h_x)+1):
    side_top.append(t_top(x))
# for lower wall
    side_down.append(t_down(x))
    x=x+h_x
side_top, side_down=np.array(side_top),np.array(side_down)
# for the left wall and the right wall
side_left,
     side_right=[float(t_left)]*(int(h/h_y)+1),[float(t_right)]*(int(h/h_y)
     +1)
side_left, side_right=np.array(side_left),np.array(side_right)
side_right=side_right.reshape(-1,1); side_left=side_left.reshape(-1,1)
u=np.full((int(h/h_y)+1,int(l/h_x)+1),0.0)
u=np.append(u,side_right,axis=1); u=np.delete(u,int(l/h_x),axis=1)
u=np.insert(u,[0],side_left,axis=1); u=np.delete(u,1,axis=1)
u=np.insert(u,int(h/h_y)+1,side_down,axis=0);
     u=np.delete(u,int(h/h_y),axis=0)
u=np.insert(u,0,side_top,axis=0); u=np.delete(u,1,axis=0)
print(' Initial calculation grid with boundary conditions); print(u)
# calculation of the system of difference equations by the Gauss-Seidel method
k=0
while True:
    count=0
    for i in range(1,int(h/h_y)):
        for j in range(1,int(l/h_x)):
            u_last=u[i,j]
            u[i,j]=0.25*(u[i+1,j]+u[i,j+1]+u[i-1,j]+u[i,j-1])
            if abs(abs(u_last)-abs(u[i,j]))>=e:
                count=count+1
    k=k+1
    if count==0:
        break
u=np.around(u, decimals=2)
print(' The number of computational iterations k=',k)
print('The number of computational iterations'); print(u)
# construction of a three-dimensional graph of temperature distribution
x=np.linspace (0, l, int(l/h_x)+1)
y=np.linspace (0, h, int(h/h_y)+1)
x,y=np.meshgrid(x, y)
fig = plt.figure(figsize=(20, 20))
axes = fig.add_subplot(1, 2, 1, projection='3d')
axes.plot_surface(x,   y,   u,   rcount=1000,   ccount=1000,   linewidth=0.2,
     edgecolors='k', cmap='jet')
axes.view_init(elev=35, azim=75)
axes.set_xlabel('X')
axes.set_ylabel('Y')
axes.set_zlabel('T')
plt.show().
```

Solving hyperbolic equations

One of the most common types of hyperbolic equations with second-order partial derivatives in engineering practice is the wave equation, which describes various types of oscillations:

$$\frac{\partial^2 u}{\partial t^2} = a^2 \left(\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2} + \cdots + \frac{\partial^2 u}{\partial x_n^2} \right) + f(x, t), \tag{4.11}$$

where $x = (x_1, \ldots, x_n)$ – Cartesian coordinates;; $f(x, t)$ f(x, t) is a function of external influence (external force); $t \in R$ – time; a – phase speed; $u(x, t)$ is a function of the position of the wave at the point with coordinates x at time t.

Depending on the number of Cartesian coordinates, one-dimensional, two-dimensional, and three-dimensional wave equations are distinguished. The one-dimensional wave equation describes the longitudinal oscillations of a rod, in which the sections carry out plane-parallel oscillatory movements, as well as the transverse oscillations of a thin rod (string) and other tasks.

The two-dimensional wave equation is used to study the oscillations of a thin plate (membrane).

The three-dimensional wave equation describes the propagation of waves in space (for example, sound waves in a liquid, elastic waves in a continuous medium, etc.).

The one-dimensional homogeneous wave equation for the case of free oscillations, based on equation (4.11), is written in the following form:

$$\frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial x^2}, \tag{4.12}$$

where $u(x, t)$ – a function that describes the position of the string at time t; $a^2=T/\rho$ (T – the tension of the string, ρ – its linear (linear) density); $f(x, t) = 0$ – external influence function (see (4.11)).

Oscillations are assumed to be small, that is, their amplitude is small compared to the length of the string. The resistance of the medium to the oscillating process is not taken into account.

The simplest problem for equation (4.12) is the Cauchy problem: at the initial moment of time, two conditions are set (the number of conditions is equal to the order of the derivative with respect to time t):

$$u\big|_{t=0} = u(x, 0) = \varphi(x), \; \left.\frac{\partial u}{\partial t}\right|_{t=0} = \psi(x). \tag{4.13}$$

These conditions describe the initial shape of the string $u=\varphi(x)$ and the speed of movement of its points $\psi(x)$. In practice, one does not solve the Cauchy problem for an infinite string, but a mixed problem for a finite string of some length l. In this case, the boundary conditions at its ends $u(0, t) = \mu_1(t)$ and $u(l, t) = \mu_2(t)$.

For example, at fixed ends, their displacements are equal to zero, and the boundary conditions have the form:

$$u\big|_{t=0} = 0\,,\; u\big|_{x=l} = 0\,. \tag{4.14}$$

To solve the problem (4.12)–(4.14), a three-layer scheme is most often used, where the set of nodes according to t=const is called a layer. At the same time, a uniform rectangular grid is introduced: $x_i = i \cdot h$ ($i = 0,\ 1,\ \ldots,\ n$), $\tau_i = j \cdot \tau$ ($j = 0,\ 1,\ \ldots,\ m$). Based on basic difference schemes (see Section 4.3), equation (4.12) is represented by finite-difference relations:

$$\begin{cases} \dfrac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\tau^2} = a^2 \dfrac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}; \\ i = 1,\ 2,\ \ldots, n-1; j = 1,\ 2,\ \ldots, m-1. \end{cases} \tag{4.15}$$

An explicit expression for the value of the grid function on the j+1 layer is determined from equation (4.15):

$$u_{i,j+1} = \lambda(u_{i+1,j} + u_{i-1,j}) + 2(1-\lambda)u_{i,j} - u_{i,i-1}\,, \tag{4.16}$$

where $\lambda = \dfrac{a^2\tau^2}{h^2}$.

The solution scheme based on equation (4.16) is called a three-layer scheme because it connects the values of $u_{i,j}$ on three time layers: j–1, j, and j+1. To determine the unknown values on the j+1 layer, it is necessary to know the solutions on the j–th and (j–1)th layers. Therefore, calculations according to formulas (4.16) must start from the second layer, and the solutions on the zero and first layers must be known. In other words, they are determined using the initial conditions (4.13). Specifically, on the zero layer:

$$u_{i,0} = \varphi(x_i)\ \ (i=0,\ 1,\ \ldots,\ n). \tag{4.17}$$

To obtain the solution on the first layer, it is necessary to use the second initial condition (4.13), where the derivative $\partial u / \partial t$ is replaced by the finite-difference approximation:

$$\left.\frac{\partial u}{\partial t}\right|_{t=0} \approx \frac{u_{i,1} - u_{i,0}}{\tau} \approx \psi(x_i)\,. \tag{4.18}$$

The value of the grid function on the first time layer is determined from relation (4.18):

$$u_{i,1} = u_{i,0} + \tau\psi(x_i)\ \ (i=0,\ 1,\ \ldots,\ n;\ t=0). \tag{4.19}$$

It should be noted that the approximation of the initial condition in the form of (4.18) worsens the approximation of the original differential problem. The approximation error becomes of the order of $O(h^2+\tau)$, which means it is of the first order with respect to τ. However, scheme (4.16) itself has the second order of

approximation with respect to h and τ. Therefore, to increase the accuracy, a more accurate representation is chosen instead of equation (4.19):

$$u_{i,1} = u_{i,1} + \tau \left.\frac{\partial u}{\partial t}\right|_{t=0} + \left.\frac{\tau^2}{2}\frac{\partial^2 u}{\partial t^2}\right|_{t=0} . \qquad (4.20)$$

Instead of $\partial u / \partial t$, $\psi(x)$ is used. The expression for the second derivative of equation (4.20) can be determined using equation (4.12) and the first initial condition (4.13), namely:

$$\left.\frac{\partial^2 u}{\partial t^2}\right|_{t=0} = a^2 \left.\frac{\partial^2 u}{\partial t^2}\right|_{t=0} = a^2 \frac{\partial^2 \varphi}{\partial x^2} .$$

Then equation (4.19) takes the following form:

$$u_{i,1} = u_{i,0} + \tau\psi(x_i) + a^2 \frac{\tau^2}{2}\varphi''(x_i) \quad (i=0, 1, \ldots, n). \qquad (4.21)$$

The difference scheme (4.15) taking into account (4.21) has an approximation error of the second order of accuracy $O(h^2+\tau^2)$.

When solving a mixed problem with boundary conditions of the form (4.14), that is, when the value of the function itself is set at the ends of the considered segment, the second order of approximation is preserved. In this case, the extreme nodes of the grid are located at the limit points ($x_0 = 0$, $x_1 = l$). However, boundary conditions can also be specified for the derivative. For example, in the case of free longitudinal oscillations of a rod at its unfixed end, the condition is set:

$$\left.\frac{\partial u}{\partial x}\right|_{x=l} = 0 . \qquad (4.22)$$

If this condition is written in the difference form with the first order of approximation, then the approximation error of the scheme will be of the order of $O(h^2+\tau^2)$. Therefore, to preserve the second order of this scheme with respect to h, it is necessary to approximate the boundary condition (4.22) with the second order of accuracy.

The difference scheme (4.16) for the solution of problem (4.12) – (4.14) is conditionally stable. Therefore, the necessary and sufficient condition for stability is in the form:

$$r = a\frac{\tau}{h} < 1 . \qquad (4.23)$$

Condition (4.23) ensures the acceptable accuracy of obtaining the solution $u(x, t)$, which has continuous derivatives of the fourth order. Moreover, under the condition $r > 1$, the solution is unstable, and under $r < 1$, the solution is stable, but its accuracy decreases as r decreases. Provided that $r = 1$, the difference solution is stable and coincides with the exact one.

The algorithm for modeling free string vibrations based on the wave equation (4.12) is presented in Figure 4.6.

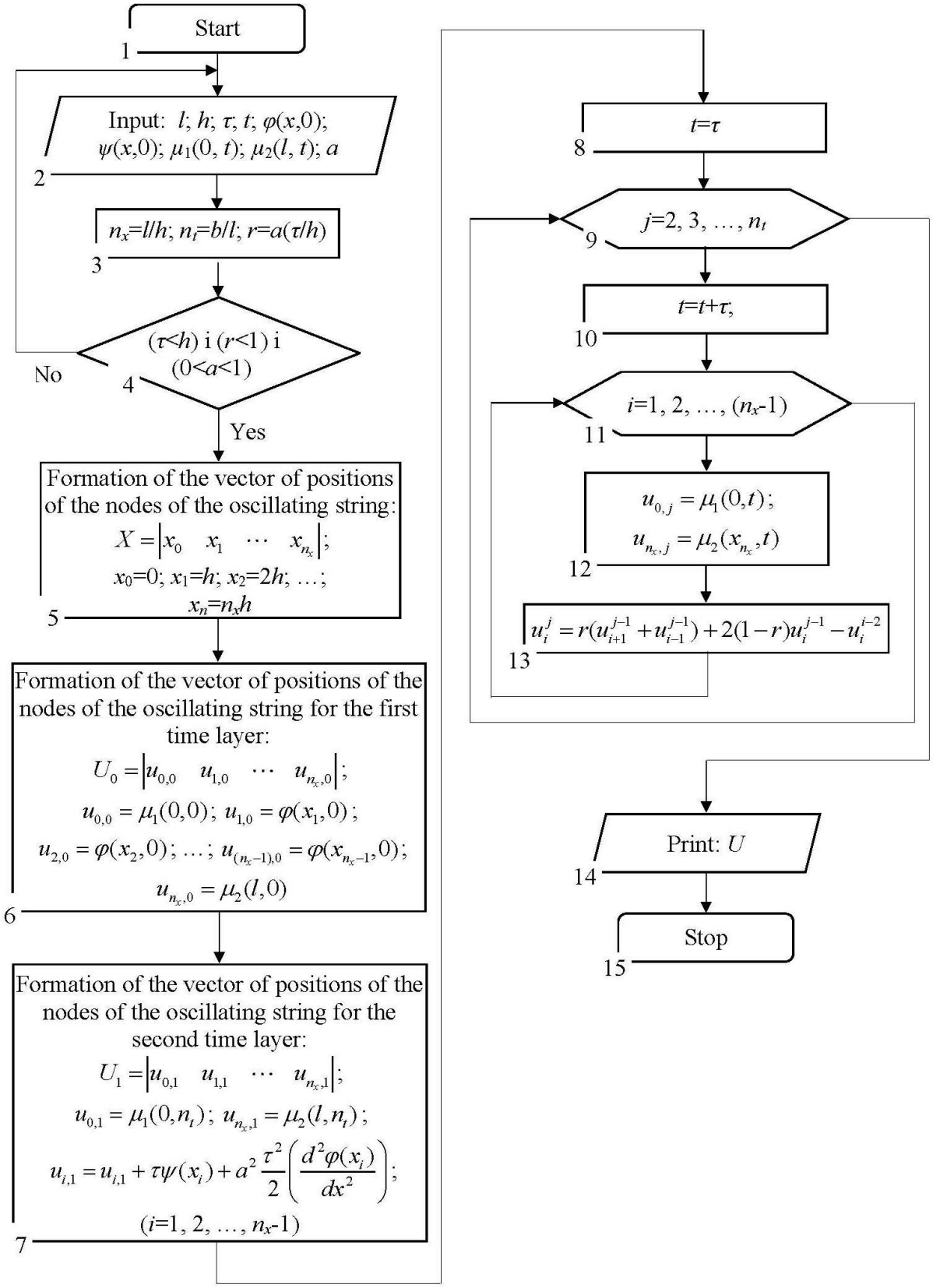


Figure 4.6 – Scheme of the algorithm for solving hyperbolic partial differential equations

Example 4.2. Determine the position of the string as a function of time, which undergoes free oscillations during the time $t = 2{,}0$ sec with fixed ends (the distance between the points of fixing the ends L = 4,0 m) and for which the following conditions are set:

– phase velocity coefficient a = 1,0 m/sec;

– initial

$$\varphi(x) = u(x, 0) = \cos^2\left(\frac{x}{2}\right),\ \psi(x) = \frac{\partial u(x,0)}{\partial t} = -\frac{\sin(x)}{2}\ (0 \le x \le L);$$

– borderline

$$u(0, t) = \cos^2\left(\frac{t}{2}\right),\ u(L, t) = \cos^2\left(\frac{t}{2} + 2{,}0\right)\ (0 \le t \le 2{,}0).$$

Compare the obtained solution with the exact solution: $u(x, t) = \cos^2\left(\frac{t+x}{2}\right)$.

Solution:

The free oscillations of the string are described by one of the types of hyperbolic equations, namely, the uniform wave equation (4.12) with two independent variables x, t:

$$\frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial x^2}. \qquad (4.24)$$

To formulate the problem and fulfill the condition of stability of the solution of the difference scheme (4.23), it is necessary to introduce a two-dimensional calculation grid, namely: along the horizontal axis x with a distance between nodes h=0,2 m (Fig. 4.7), as well as for time (variable t) with a distance between nodes τ=0,1 sec. The grid along the horizontal axis x contains $n_x=(L/h)+1=(4{,}0/0{,}2)+1=21$ nodes, and along the time variable τ also contains $n_\tau=(t/\tau)+1=(2{,}0/0{,}1)+1=21$ nodes. The serial number of the calculation node is indicated by the index i on the x-axis, and by the index j on the time axis.

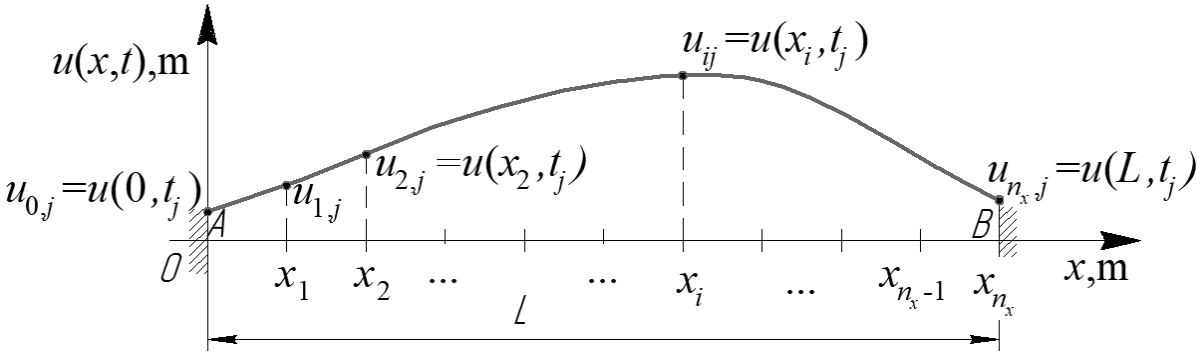


Figure 4.7 – Calculation scheme for the free oscillations of the string

The value of the vector of positions of the oscillating string for the first time layer at the instant of time t_0 = 0,0 sec (j = 0):

$$U_0 = \left(u_{0,0} \quad u_{1,0} \quad \ldots \quad u_{i,0} \quad \ldots \quad u_{n_x,0}\right) = (1{,}0 \quad 0{,}990 \quad 0{,}961 \quad 0{,}913 \quad 0{,}848$$
$$0{,}770 \quad 0{,}681 \quad 0{,}585 \quad 0{,}485 \quad 0{,}386 \quad 0{,}292 \quad 0{,}206 \quad 0{,}131 \quad 0{,}716 \quad 0{,}289$$
$$0{,}005 \quad 0{,}00085 \quad 0{,}017 \quad 0{,}052 \quad 0{,}105 \quad 0{,}173),$$

where $u_{0,0} = u(0,0) = \cos^2\left(\frac{t}{2}\right) = \cos^2\left(\frac{0,0}{2}\right) = 1{,}00$ m; $u_{i,0} = u(x_i,0) = \cos^2\left(\frac{x_i}{2}\right)$;

$u_{1,0} = u(h,0) = \cos^2\left(\frac{x_1}{2}\right) = \cos^2\left(\frac{0,2}{2}\right) = 0{,}99$ m; $u_{20,0} = u(4,0;0) = \cos^2\left(\frac{t}{2} + 2{,}0\right) =$

$= \cos^2\left(\frac{0,0}{2} + 2{,}0\right) = 0{,}173$ m.

The value of the position vector of the oscillating string for the second time layer, at the moment of time $t_1 = \tau = 0{,}1$ sec ($j = 1$) based on equation (4.21):

$$U_1 = \left(u_{0,1} \quad u_{1,1} \quad \ldots \quad u_{i,1} \quad \ldots \quad u_{n_x,1}\right) = (0{,}998 \quad 0{,}978 \quad 0{,}939 \quad 0{,}882 \quad 0{,}811$$
$$0{,}727 \quad 0{,}634 \quad 0{,}535 \quad 0{,}435 \quad 0{,}338 \quad 0{,}248 \quad 0{,}167 \quad 0{,}099 \quad 0{,}048 \quad 0{,}014$$
$$0{,}0004 \quad 0{,}0063 \quad 0{,}032 \quad 0{,}076 \quad 0{,}137 \quad 0{,}213),$$

where $u_{0,1} = u(0;0,1) = \cos^2\left(\frac{t_1}{2}\right) = \cos^2\left(\frac{0,1}{2}\right) = 0{,}998$ m; $u_{20,1} = u(4;0,1) =$

$= \cos^2\left(\frac{t_1}{2} + 2{,}0\right) = \cos^2\left(\frac{0,1}{2} + 2{,}0\right) = 0{,}173$ m; $u_{i,1} = u_{i,0} + \tau\psi(x_i) + a^2\frac{\tau^2}{2} \times$

$\times\left(\frac{d^2\varphi(x)}{dx^2}\right) = u_{i,0} + \tau\left(-\frac{\sin(x_i)}{2}\right) + a^2\frac{\tau^2}{2}\left(-\frac{\cos(x_i)}{2}\right)$; $u_{1,1} = u(h;0,1) = u_{1,0} +$

$+\tau\left(-\frac{\sin(x_1)}{2}\right) + a^2\frac{\tau^2}{2}\left(-\frac{\cos(x_1)}{2}\right) = 0{,}99 + 0{,}1\left(-\frac{\sin(0,1)}{2}\right) + 1^2\frac{0{,}1^2}{2}\left(-\frac{\cos(0,2)}{2}\right) =$

$=0{,}978$ m.

The value of the position vector of the oscillating string for the third time layer, at the moment of time $t_2=2\tau=2\cdot0{,}1$ sec based on equation (4.16) for the same value of $j=1$, for $\lambda = \frac{a^2\tau^2}{h^2} = \frac{1{,}0^2 \cdot 0{,}1^2}{0{,}2^2} = 0{,}25$:

$$U_2 = \left(u_{0,2} \quad u_{1,2} \quad \ldots \quad u_{i,2} \quad \ldots \quad u_{n_x,2}\right) = (0{,}990 \quad 0{,}961 \quad 0{,}913 \quad 0{,}848 \quad 0{,}770$$
$$0{,}681 \quad 0{,}585 \quad 0{,}485 \quad 0{,}386 \quad 0{,}292 \quad 0{,}206 \quad 0{,}131 \quad 0{,}071 \quad 0{,}029 \quad 0{,}005$$
$$0{,}0008 \quad 0{,}0167 \quad 0{,}052 \quad 0{,}105 \quad 0{,}173 \quad 0{,}255),$$

where $u_{0,2} = u(0;0,2) = \cos^2\left(\frac{t_2}{2}\right) = \cos^2\left(\frac{0,2}{2}\right) = 0{,}990$ m; $u_{i,j+1} = \lambda(u_{i+1,j} + u_{i-1,j}) +$

$+2(1-\lambda)u_{i,j} - u_{i,i-1}$; $u_{1,2} = \lambda(u_{2,1} + u_{0,1}) + 2(1-\lambda)u_{1,1} - u_{1,0} = 0{,}25(0{,}939 + 0{,}998) +$

$+2(1-0,25)0,978-0,990=0,961$ m; $u_{20,2}=u(4,0;0,2)=\cos^2\left(\frac{t_2}{2}+2,0\right)=$

$=\cos^2\left(\frac{0,2}{2}+2,0\right)=0,255$ m.

Similarly, formula (4.16) is used to determine the position vectors of the oscillating string for the remaining time layers (j = 3, 4, …, 20). Based on the results of the calculations, it is possible to construct a diagram of the positions of the string for different moments of time during free oscillation (Fig. 4.8), where the movement of the wave (point A) can be clearly determined.

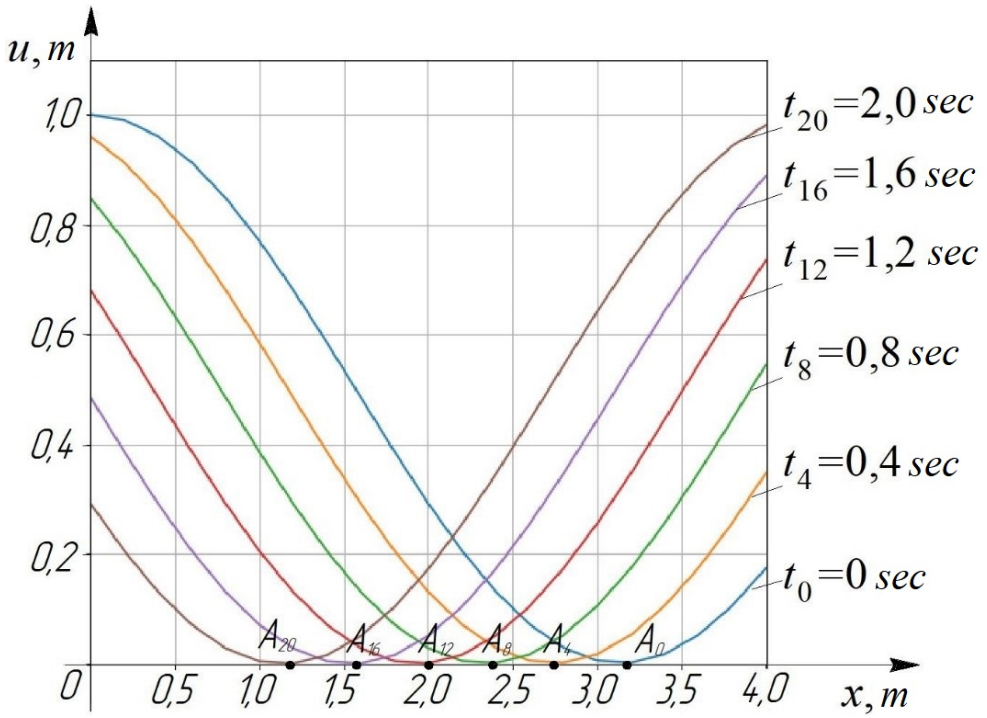


Figure 4.8 – Diagram of the positions of the string at different moments of time during free oscillation

The relative error of calculating the wave equation by comparing the values of the analytical solution and the numerical method is:

– of the third time layer at the nodal point i=10 (x_{10}=2,0 m)

$$\varepsilon_{10,2}=\left|\frac{u^{an}_{10,2}-u^{num}_{10,2}}{u^{an}_{10,2}}\right|\cdot 100\%=\left|\frac{0,2058-0,2060}{0,2058}\right|\cdot 100\%=0,01\%,$$

where $u^{an}_{10,2}=\cos^2\left(\frac{t_2+x_{10}}{2}\right)=\cos^2\left(\frac{0,2+2,0}{2}\right)=0,2058$ m, $u^{num}_{10,2}=0,2060$ m.

– of the fourth temporal layer at the nodal point i=15 (x_{10}=3,0 m)

$$\varepsilon_{15,3}=\left|\frac{u^{an}_{15,3}-u^{num}_{15,3}}{u^{an}_{15,3}}\right|\cdot 100\%=\left|\frac{0,00630-0,00626}{0,00626}\right|\cdot 100\%=0,64\%,$$

where $u_{15,3}^{an} = \cos^2\left(\frac{t_3 + x_{15}}{2}\right) = \cos^2\left(\frac{0,3+3,0}{2}\right) = 0,00630\,\text{m},$

$u_{15,3}^{num} = \lambda(u_{16,2} + u_{14,2}) + 2(1-\lambda)u_{15,2} - u_{15,1} = 0,25(0,0167 + 0,0050) +$

$+2(1-0,25)0,0008 - 0,0004 = 0,00626$ m.

The value of the relative errors in the calculation using the numerical method of finite differences demonstrates the high accuracy of solving partial differential equations of the hyperbolic type. This is achieved by employing the explicit difference scheme for evaluating the grid function.

The result of solving example 4.2 in the programming language PYTHON:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from scipy import *
from sympy import *
#entering the value of the length of the string
l=float(input()) #l=4.0
# entering the step value of the calculation grid along the length of the
     string
h_x=float(input()) #h_x=0.2
# entering the step value of the calculation grid over time
h_t=float(input()) #h_t=0.1
#entering the value of the boundary conditions:
#entering the coordinate value of the position of the left end of the string
def u_left(t):
    return np.cos(0.5*t)*np.cos(0.5*t)
#entering the coordinate value of the position of the right end of the string
def u_right(t):
   return np.cos((0.5*t)+2)*np.cos((0.5*t)+2)
#entering the function of the initial position of the string
def u_x0(x):
    return np.cos(0.5*x)*np.cos(0.5*x)
#entering the function of the initial velocity of the points of the string
def v_first(x):
    return -0.5*np.sin(x)
#entering the value of the total oscillation time
t=float(input()) #t=2.0
#entering the calculation error value
e=float(input()) #e=0.005
#entering the string tension indicator
a=float(input()) #a=1
lamb_da=(a*a*h_t*h_t)/(h_x*h_x)
#control of the solution's stability condition
if np.sqrt(lamb_da)>1:
    print("The solution condition is not stable. Change the resolution
     parameters!")
#entering the values of the boundary conditions:
u_first,u_second,x_coord=[u_left(0)],[u_left(h_t)],[0]
x=h_x
for i in range(0,int(l/h_x)):
#for the first temporal layer
    u_first.append(u_x0(x))
#for the second time layer
    if i==int(l/h_x):
        u_first.append(u_right(0))
    else:
     u_second.append(u_x0(x)+(v_first(x)*h_t)+(0.5*(a*a)*(h_t*h_t)*(0.5*(-
```

```
     np.cos(x)))))
    x_coord.append(x)
    x=x+h_x
u_second[int(l/h_x)]=u_right(h_t)
u_first, u_second=np.array(u_first),np.array(u_second)
u=np.array([u_first,u_second])
#solution of a three-layer system of finite-difference equations
t_0=h_t
for j in range(2,int(t/h_t)+1):
    t_0=t_0+h_t; u_next=[u_left(t_0)]
    for i in range(0,int(l/h_x)-1):
        u_next.append(lamb_da*(u[j-1,i+2]+u[j-1,i])+(2*(1-lamb_da)*u[j-
     1,i+1])-u[j-2,i+1])
    u_next.append(u_right(t_0)); u=np.insert(u,j,u_next,axis=0)
    del u_next
print('The matrix of values of the position of the oscillating string for
     each moment in time:')
print(u)
#graphing the solutions of the hyperbolic (wave) equation
k=0
plt.figure(figsize=(8, 8))
plt.xlabel('x',fontsize=15, color='blue')
plt.ylabel('y',fontsize=15, color='blue')
for i in range(0,6):
    plt.plot(x_coord, u[k]); k=k+4
plt.grid(True)
plt.xlim([0, 4])
plt.ylim([-0.1, 1.1])
plt.show()
#calculating the values of the exact solution of the hyperbolic (wave)
     equation
x_an=0; t_an=0
u_an=[]
for i in range(0,int(l/h_x)+1):
    u_an.append(np.cos(0.5*(x_an+t_an))*np.cos(0.5*(x_an+t_an)))
    x_an=x_an+h_x
t_an=t_an+h_t
u_an=np.array([u_an])
for j in range(1,int(t/h_t)+1):
    u_tran=[]; x_an=0
    for i in range(0,int(l/h_x)+1):
        u_tran.append(np.cos(0.5*(x_an+t_an))*np.cos(0.5*(x_an+t_an)))
        x_an=x_an+h_x

    t_an=t_an+h_t
    u_an=np.insert(u_an,j,u_tran,axis=0)
    del u_tran
print('The matrix of exact values of the position of the oscillating string
     for each instant of time:')
print(u_an)
#determination of calculation error
e_tran=[]; count=0
for j in range(0,int(t/h_t)+1):
    for i in range(0,int(l/h_x)-1):
        e_tran.append(abs(u_an[j,i]-u[j,i]))
        if abs(u_an[j,i]-u[j,i])>=e:
            count=count+1
if count>0:
    print("The calculation error exceeds the specified accuracy. Please
     change the initial parameters of the calculation.").
```

Solving parabolic equations

An example of a problem that can be reduced to a parabolic equation in partial derivatives is the problem of non-stationary thermal conductivity described by the homogeneous equation (4.2). In particular, in the homogeneous thermal conductivity problem, it is necessary to determine the function $u(x, t)$ that satisfies the equation in the domain $\Omega = \{(x, t), 0 \le x \le l, 0 \le t \le T)\}$.

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} \ (k = \text{const} > 0), \tag{4.25}$$

initial condition $u(x,0) = u_0(x)$ and boundary conditions of the first kind $u(0, t) = \mu_1(t)$ and $u(l, t) = \mu_1(t)$.

There are two possible options for constructing the difference equation on the calculation grid with steps h along x and τ along t.

A variant of approximation using a four-point template, a central difference scheme of the second kind (see Section 4.3), and a left difference scheme $\frac{\partial u}{\partial t} \approx \frac{u_{i,j+1} - u_{i,j}}{\tau} \approx \frac{1}{\tau}$ [1 — -1 — 0] (i,j) leads to an explicit two-layer scheme, namely: $\frac{u_{i,j+1} - u_{i,j}}{\tau} \approx \alpha \left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} \right)$ ($i = 1, 2, \ldots, n-1$; $j = 0, 1, \ldots, m-1$), then

$$u_{i,j+1} = \delta u_{i+1,j} + (1 - 2\delta) u_{ij} + \delta u_{i-1,j}, \tag{4.26}$$

where $\delta = \alpha \frac{\tau}{h^2}$.

The explicit two-layer difference scheme (4.26) is stable only for $\delta \le 0.5$, which leads to the need to perform calculations with a very small step in t ($\tau \le 0{,}5h^2$). This also limits the speed of operation and requires a large expenditure of machine time for computer systems.

The algorithm for solving parabolic differential equations based on the heat conduction equation and the explicit difference scheme (4.26) is presented in Figure 4.9.

Therefore, for parabolic equations, the most widely used implicit scheme is the one where the approximation is performed using a four-point template, a central difference scheme of the second kind (see Section 4.3), and a right-hand difference scheme. $\frac{\partial u}{\partial t} \approx \frac{u_{i,j} - u_{i,j-1}}{\tau} \approx \frac{1}{\tau}$ [0 — 1 — -1] (i,j) leads to an implicit two-layer scheme, namely: $\frac{u_{i,j} - u_{i,j-1}}{\tau} \approx \alpha \left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} \right)$ ($i = 1, 2, \ldots, n\text{-}1$; $j = 1, 2, \ldots, m$), then

$$-u_{i,j-1} = \delta u_{i+1,j} - (1 + 2\delta) u_{ij} + \delta u_{i-1,j}. \tag{4.27}$$

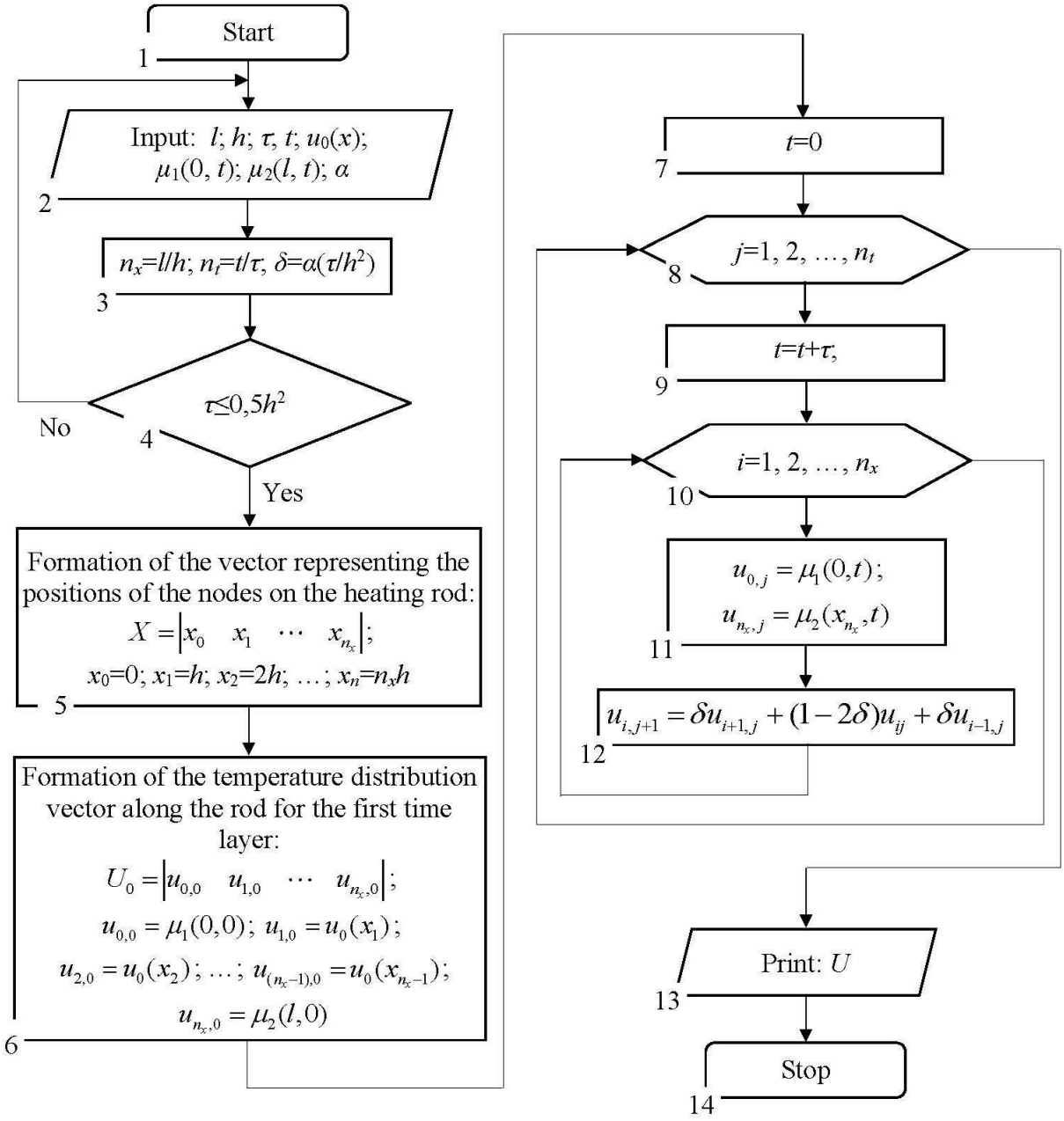


Figure 4.9 – Scheme of the algorithm for solving parabolic partial differential equations using the explicit difference method

The implicit two-layer difference scheme (4.27), supplemented with equations from the boundary conditions $u_{0,j} = \mu_1(t_j)$ and $u_{n,j} = \mu_2(t_j)$, leads to a system of equations that has a stable solution for any values of δ.

The algorithm for solving parabolic differential equations based on the heat conduction equation and the implicit difference scheme (4.27) by the sweep method is presented in Figure 4.10.

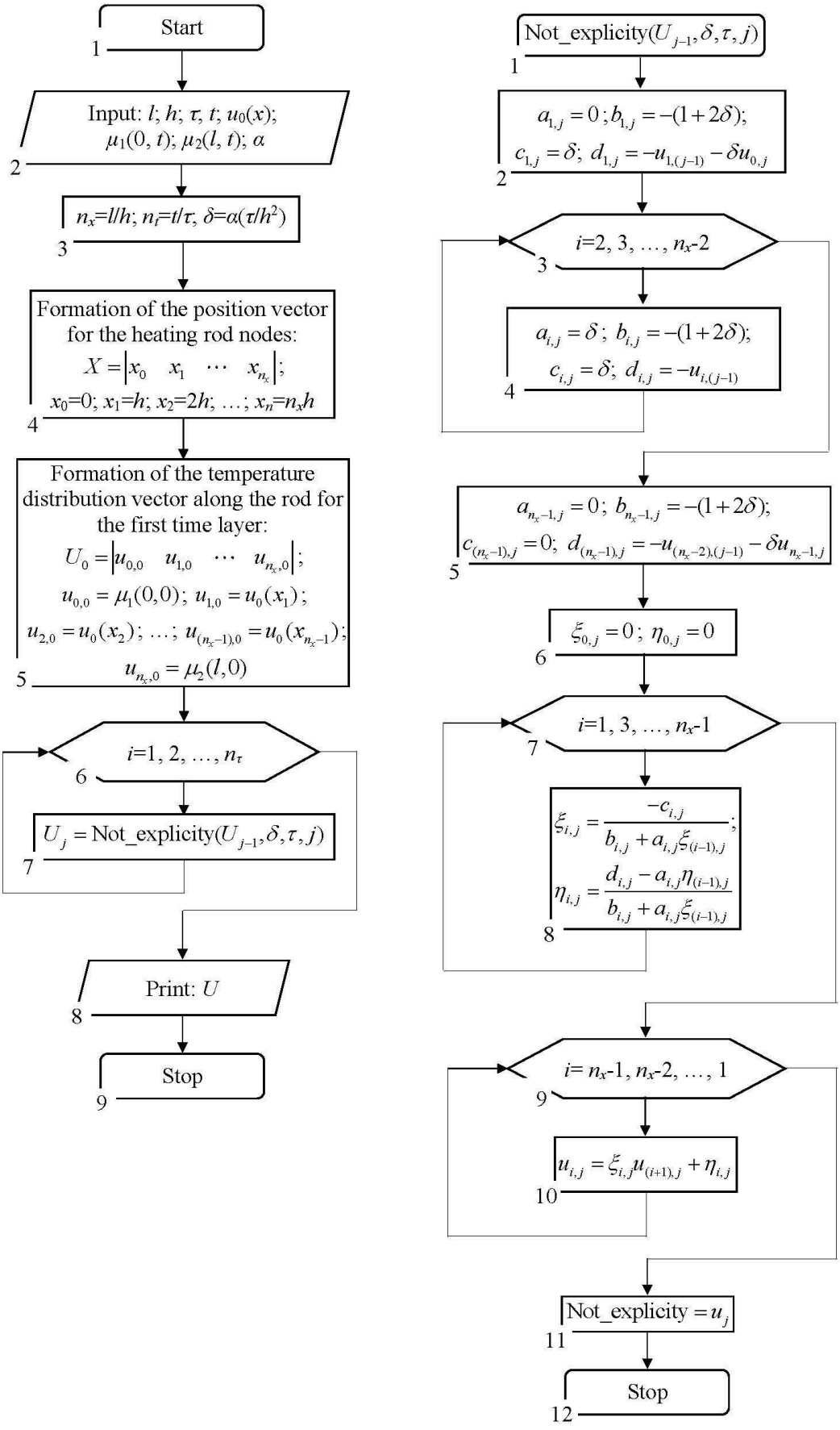


Figure 4.10 – Scheme of the algorithm for solving parabolic partial differential equations using the implicit difference method

Example 4.3. A copper rod of length $L = 2{,}0$ m with a constant cross-sectional area along its length is placed in an insulated material in such a way that only its extreme right and left ends interact with the environment. At the initial moment of time, the rod has a balanced temperature $T = 0$ °C, and its left and right ends are constantly maintained at temperatures $T_l = 80{,}0$ °C and $T = 10{,}0$ °C, respectively. It is necessary to determine the change in temperature distribution along the rod depending on time during one hour (t=3600,0 sec). Compare the results of calculations with the help of explicit and implicit difference schemes.

Solution:

The non-stationary temperature distribution along the length of the body is described by the homogeneous heat conduction equation (4.1), namely:

$$\frac{\partial^2 u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}. \tag{4.28}$$

To formulate the problem and fulfill the condition of stability of the solution of the difference scheme (4.26), it is necessary to introduce a two-dimensional calculation grid: along the horizontal axis x with the distance between nodes $h = 0{,}2$ m (Fig. 4.11); for the explicit difference scheme, the distance along the vertical axis for the time variable t with the distance between the nodes $\tau = 0{,}5h^2 = 0{,}5 \cdot 0{,}2^2 = 0{,}02$ sec; for implicit – τ=0,1 sec. The grid along the horizontal axis x contains $n_x = (L/h)+1 = (2{,}0/0{,}2)+1 = 11$ nodes, and the grid along the vertical axis τ contains: for the explicit difference seven – $n_\tau^e = (t/\tau)+1 =$ $=(3600{,}0/0{,}02)+1=180001$ nodes; for implicit – $n_\tau^i = (t/\tau)+1 = (3600{,}0/0{,}02)+1 =$ $=180001$. The serial number of the calculation node is denoted by the index i on the x-axis, and by the index j on the time axis.

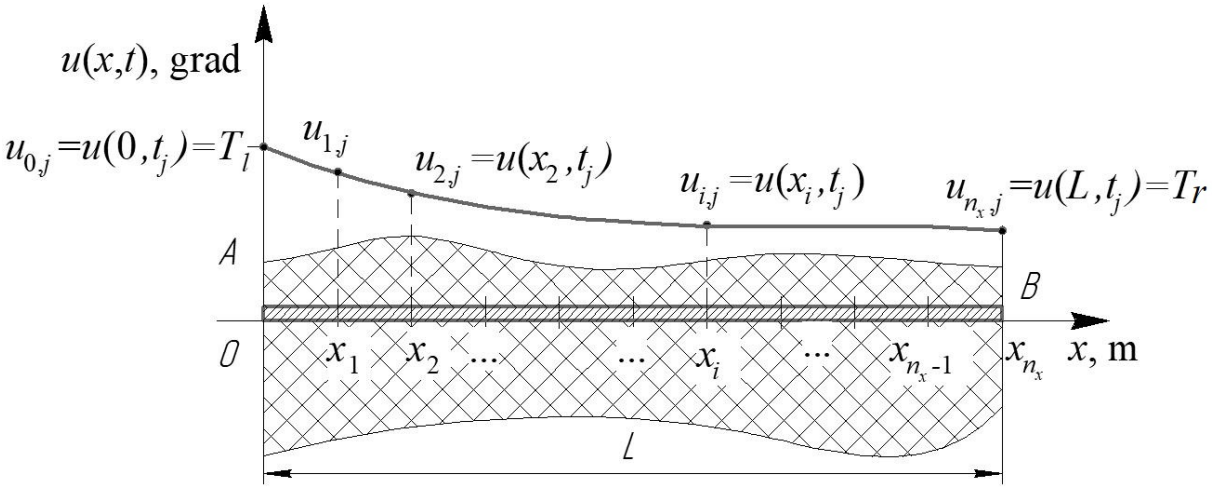


Figure 4.11 – Calculation diagram of heat transfer along the rod

The value of the temperature vector along the rod for the first time layer at the moment of time $t_0 = 0$ sec ($j = 0$) for both explicit and implicit difference schemes:

$U_0 = \left(u_{0,0} \quad u_{1,0} \quad \ldots \quad u_{i,0} \quad \ldots \quad u_{10,0}\right) = \left(20,0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 10,0\right)$,

where $u_{0,0} = u(0,0) = 20,0$ grad; $u_{i,0} = u(x_i,0) = 0,0$ grad;

$u_{10,0} = u(2,0;0) = 20,0$ grad.

The value of the temperature distribution vector for the second time layer, at the moment of time $t_1 = 0{,}02$ sec based on the equation of the explicit difference scheme (4.26) for the same value of $j = 0$, for $\delta = \alpha\frac{\tau}{h^2} = 0,00011\frac{0,02}{0,2} =$

$= 5,49 \cdot 10^{-5}$ ($\alpha = 0,00011$ m/sec^2 – coefficient of thermal conductivity for copper):

$U_1^e = \left(u_{0,1}^e \quad u_{1,1}^e \quad \ldots \quad u_{i,1}^e \quad \ldots \quad u_{10,1}^e\right) = \left(20,0 \quad 0,0011 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \right.$

$\left. 0,00055 \quad 10,0\right)$, where $u_{0,1}^e = u(0;0,02) = 20,0$ grad; $u_{10,1}^e = u(2,0;0,02) =$

$=20,0$ grad; $u_{i,j+1} = \delta u_{i+1,j} + (1-2\delta)u_{ij} + \delta u_{i-1,j}$; $u_{1,1}^e = \delta u_{2,0} + (1-2\delta)u_{1,0} + \delta u_{0,0} =$

$= 5,49 \cdot 10^{-5} \times 0,0 + (1 - 2 \cdot 5,49 \cdot 10^{-5}) \cdot 0,0 + 5,49 \cdot 10^{-5} \cdot 20,0 = 0,00011$ grad,

$u_{9,1}^e = \delta u_{10,0} + (1-2\delta)u_{9,0} + \delta u_{8,0} = 5,49 \cdot 10^{-5} \cdot 10,0 + (1 - 2 \cdot 5,49 \cdot 10^{-5}) \cdot 0,0 +$

$+5,49 \cdot 10^{-5} \cdot 0,0 = 0,00055$ grad.

Similarly, based on the equation of the explicit difference scheme (4.26), the temperature distribution vectors for the remaining time layers are determined ($j = 2, 3, \ldots, n_\tau^e - 1$), in particular for the last time layer:

$$U_{180000}^e = \left(u_{0,180000}^e \quad u_{1,180000}^e \quad \ldots \quad u_{i,180000}^e \quad \ldots \quad u_{10,180000}^e\right) =$$

$$= \left(20,0 \quad 16,74 \quad 13,71 \quad 11,12 \quad 9,12 \quad 7,81 \quad 7,21 \quad 7,25 \quad 7,84 \quad 8,82 \quad 10,0\right).$$

Using the equation of the implicit difference scheme (4.27), the number of calculations can be significantly reduced, specifically for the second time layer and the value of $j = 1$ for each calculation node:

$$\begin{cases} \delta u_{0,1}^i - (1+2\delta)u_{1,1}^i + \delta u_{2,1}^i = -u_{1,0} \quad (i=1); \\ \delta u_{1,1}^i - (1+2\delta)u_{2,1}^i + \delta u_{3,1}^i = -u_{2,0} \quad (i=2); \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\ \delta u_{(k-1),1}^i - (1+2\delta)u_{k,1}^i + \delta u_{(k+1),1}^i = -u_{k,0}; \quad (i=k); \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\ \delta u_{(n_x-3),1}^i - (1+2\delta)u_{(n_x-2),1}^i + \delta u_{(n_x-1),1}^i = -u_{(n_x-2),0} \quad (i=n_x-2). \end{cases} \tag{4.29}$$

If the system of equations (4.29) is reduced to the following form:

$$\begin{cases} a_{1,1}u^i_{0,1} + b_{1,1}u^i_{1,1} + c_{1,1}u^i_{2,1} = d_1 \quad (i=1); \\ a_{2,1}u^i_{1,1} + b_{2,1}u^i_{2,1} + c_{2,1}u^i_{3,1} = d_{2,1} \quad (i=2); \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\ a_{k,1}u^i_{(k-1),1} + b_{k,1}u^i_{k,1} + c_{k,1}u^i_{(k+1),1} = d_{k,1} \quad (i=k); \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\ a_{n_x-2,1}u^i_{(n_x-3),1} + b_{n_x-2,1}u^i_{(n_x-2),1} + c_{n_x-2,1}u^i_{(n_x-1),1} = d_{n_x-2,1} \ (i=n_x-2). \end{cases} \tag{4.30}$$

where $a_1 = \left(a_{1,1},\ a_{2,1},\ \ldots,\ a_{k,1},\ \ldots,\ a_{(n_x-2),1}\right) = \left(0,\ \delta,\ \ldots,\ \delta,\ \ldots,\ \delta\right)$; $b_1 = \left(b_{1,1},\ b_{2,1}, \ldots,\ b_{k,1},\ \ldots, b_{(n_x-2),1}\right) = \left(-(1+2\delta),\ -(1+2\delta),\ \ldots,\ -(1+2\delta),\ \ldots,\ -(1+2\delta)\right)$; $c_1 = \left(c_{1,1},\ c_{2,1},\ \ldots,\ c_{k,1},\ \ldots,\ c_{(n_x-2),1}\right) = \left(\delta,\ \delta,\ \ldots,\ \delta,\ \ldots,\ 0\right)$; $d_1 = \left(d_{1,1},\ d_{2,1},\ \ldots,\ d_{k,1},\ \ldots,\ d_{(n_x-2),1}\right) = \left((-u_{1,0} - \delta u_{0,1}),\ (-u_{2,0}),\ \ldots,\ (-u_{k,0}),\ \ldots,\ (-u_{(n_x-2),0} - \delta u_{(n_x-1),1})\right)$, then, in the obtained tridiagonal matrix of the system of equations (4.30), the condition of preferring diagonal elements is fulfilled $\left(\left|b_{i,1}\right| \geq \left|a_{i,1}\right| + \left|c_{i,1}\right|\right)$, which allows us to use the run-off method (see Chapter 1) to solve this LAES.

Then the value of the driving coefficients on the straight line:

– $\xi_{i,1} = \dfrac{-c_{i,1}}{b_{i,1} + a_{i,1}\xi_{(i-1),1}}$ (i = 1, 2, …, n_x–2)

$$\xi_{1,1} = \frac{-c_{1,1}}{b_{1,1} + a_{1,1}\xi_{0,1}} = \frac{-\delta}{-(1+2\delta) + 0 \cdot 0} = \frac{-5{,}49 \cdot 10^{-5}}{-\left(1 + 2 \cdot (5{,}49 \cdot 10^{-5})\right)} = 0.000275,$$

$$\xi_{2,1} = \frac{-c_{2,1}}{b_{2,1} + a_{2,1}\xi_{1,1}} = \frac{-5{,}49 \cdot 10^{-5}}{-\left(1 + 2 \cdot (5{,}49 \cdot 10^{-5})\right) + 5{,}49 \cdot 10^{-5} \cdot 0{,}000275} = 0.000275,$$

…… ,

$\xi_1 = \left(\xi_{1,1},\ \xi_{2,1},\ \ldots,\ \xi_{k,1},\ \ldots,\ \xi_{9,1}\right) = \left(0{,}000275\ \ 0{,}000275\ \ 0{,}000275\ \ 0{,}000275\ \ 0{,}000275\ \ 0{,}000275\ \ 0{,}000275\ \ 0{,}000275\ \ 0{,}0\right)$;

– $\eta_{i,1} = \dfrac{d_{i,1} - a_{i,1}\eta_{(i-1),1}}{b_{i,1} + a_{i,1}\xi_{(i-1),1}}$ (i = 1, 2, …, n_x–2)

$$\eta_{1,1} = \frac{d_{1,1} - a_{1,1}\eta_{0,1}}{b_{1,1} + a_{1,1}\xi_{0,1}} = \frac{(-u_{1,0} - \delta u_{0,1}) - 0 \cdot 0}{-(1+2\delta) + 0 \cdot 0} = \frac{-20{,}0 - (5{,}49 \cdot 10^{-5}) \cdot 20{,}0}{-\left(1 + 2(5{,}49 \cdot 10^{-5})\right)} = 0.0055,$$

$$\eta_{2,1} = \frac{d_{2,1} - a_{2,1}\eta_{1,1}}{b_{,12} + a_{2,1}\xi_{1,1}} = \frac{(-u_{2,0}) - \delta\eta_{1,1}}{-(1+2\delta) + \delta\xi_{1,1}} =$$

$$= \frac{(-0,0) - (5,49 \cdot 10^{-5} \cdot 0,0055)}{-\left(1 + 2 \cdot 5,49 \cdot 10^{-5}\right) + (5,49 \cdot 10^{-5} \cdot 0,000275)} = 1,51 \cdot 10^{-6} ,$$

.... ,

$\eta_1 = \left(\eta_{1,1},\ \eta_{2,1},\ \ldots,\ \eta_{k,1},\ \ldots,\ \eta_{9,1}\right) = \left(0,0055\ \ 1,51 \cdot 10^{-6}\ \ 4,15 \cdot 10^{-10}\ \ 1,14 \cdot 10^{-13}\right.$ $\left.3,14 \cdot 10^{-17}\ \ 8,62 \cdot 10^{-21}\ \ 2,37 \cdot 10^{-24}\ \ 6,51 \cdot 10^{-28}\ \ 0,00275\right)$.

On the return stroke, the $u_{i,1}^{i}$ values are determined using recursive equations $u_{i,1}^{i} = \xi_{i,1} u_{(i+1),1}^{i} + \eta_{i,1}$ (i=n_x–2, n_x–3, …, 1):

$$\begin{cases} i = n_x - 2 = 11 - 2 = 9: \quad u_{(n_x-2),1}^{i} = u_{9,1}^{i} = \xi_{(n_x-1),1} u_{(n_x-1),1}^{i} + \eta_{(n_x-2),1} = \\ \qquad = \xi_{9,1} u_{10,1}^{i} + \eta_{9,1} = 0 \cdot 10,0 + 0,00275 = 0,0028 \text{ grad}; \\ i = n_x - 3 = 11 - 3 = 8: \quad u_{(n_x-3),1}^{i} = u_{8,1}^{i} = \xi_{(n_x-3),1} u_{(n_x-2),1}^{i} + \eta_{(n_x-3),1} = \\ \qquad = \xi_{8,1} u_{9,1}^{i} + \eta_{8,1} = 0,000275 \cdot 0,00275 + 6,51 \cdot 10^{-28} = \\ \qquad = 7,55 \cdot 10^{-7} \text{ grad}; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots ; \\ i = n_x - 10 = 11 - 10 = 1: \quad u_{(n_x-10),1}^{i} = u_{1,1}^{i} = \xi_{(n_x-10),1} u_{(n_x-9),1}^{i} + \eta_{(n_x-10),1} = \\ \qquad = \xi_{1,1} u_{2,1}^{i} + \eta_{1,1} = 0,000275 \cdot 1,51 \cdot 10^{-6} + 0,0055 = \\ \qquad = 0,0055 \text{ grad}, \end{cases}$$

$U_1^{i} = \left(u_{0,1}^{i}\ \ u_{1,1}^{i}\ \ \ldots\ \ u_{i,1}^{i}\ \ \ldots\ \ u_{10,1}^{i}\right) = \left(20,0\ \ 0,0055\ \ 1,51 \cdot 10^{-6}\ \ 4,15 \cdot 10^{-10}\right.$ $\left.1,14 \cdot 10^{-13}\ \ 4,71 \cdot 10^{-17}\ \ 5,71 \cdot 10^{-14}\ \ 2,08 \cdot 10^{-10}\ \ 7,55 \cdot 10^{-7}\ \ 0,0027\ \ 10,0\right)$.

Similarly, based on the equation of the implicit difference scheme (4.27), the temperature distribution vectors are determined for the remaining time layers (j = 2, 3, …, $n_\tau^{i} - 1$), particularly for the last time layer:

$$U_{36000}^{i} = \left(u_{0,36000}^{i}\ \ u_{1,36000}^{i}\ \ \ldots\ \ u_{i,36000}^{i}\ \ \ldots\ \ u_{10,36000}^{i}\right) =$$
$$= \left(20,0\ \ 16,74\ \ 13,71\ \ 11,12\ \ 9,12\ \ 7,81\ \ 7,21\ \ 7,25\ \ 7,84\ \ 8,82\ \ 10,0\right).$$

Based on the results of the calculations, it is possible to construct a diagram of the temperature distribution along the rod for different moments of time (Fig. 4.12).

Comparing the temperature values in the nodes of the rod, determined by the explicit and implicit methods in the corresponding time layers, it is possible to observe a high level of convergence in the calculation results. It is important to note the high efficiency of using the implicit calculation method, particularly as

the number of calculations using the implicit difference method is reduced by three times compared to the explicit method.

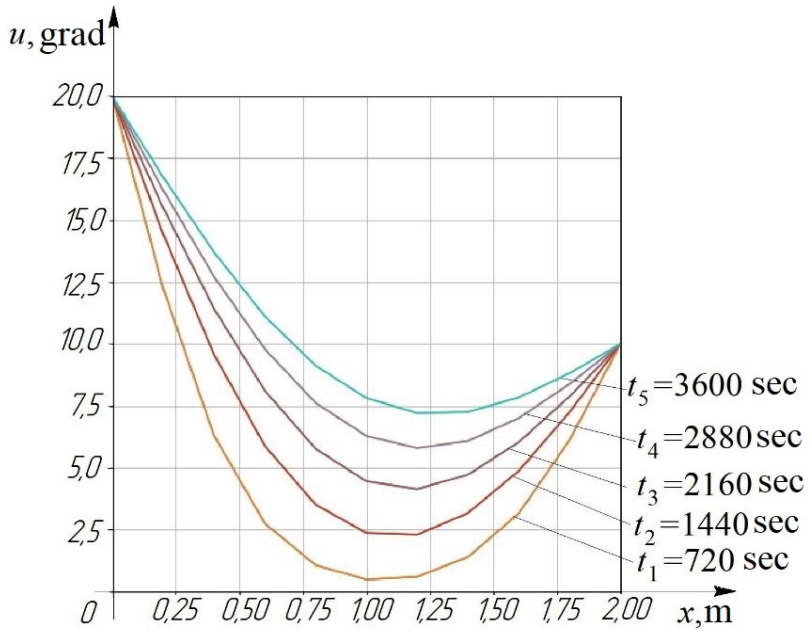


Figure 4.12 – Diagram of temperature distribution along the rod

The result of solving example 4.3 in the PYTHON programming language:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from scipy import *
from sympy import *
#entering the length of the rod
l=float(input()) #l=2.0
#entering the time value of the heat exchange duration
t=float(input()) #t=3600.0
# entering the step value of the calculation grid along the length of the rod
h_x=float(input()) #h_x=0.2
#entering the step value of the calculation grid over time
print("Enter the value of the step of the calculation grid in time for the
     explicit difference scheme (the step should not be more)", 0.5*h_x*h_x,
     "c")
h_texp=float(input()) #h_texp=0.02
print("Enter the time grid step value for the implicit difference scheme")
h_tnexp=float(input()) #h_tnexp=0.1
#entering the value of the coefficient of thermal conductivity
a=0.00011 #a=float(input()) a=0.00011
del_tae=(a*h_texp)/(h_x*h_x); del_tane=(a*h_tnexp)/(h_x*h_x)
#entering the value of the boundary conditions:
#entering the value of the temperature of the left end of the rod
def u_left(t):
    return 20.0
#entering the temperature value of the right end of the rod
def u_right(t):
    return 10.0
#introduction of the initial temperature distribution function along the
     length of the rod
def u_x0(x):
    return 0
```

```
#entering the values of boundary and initial conditions:
u_first,x_coord=[u_left(0)],[0]
x=h_x
for i in range(0,int(l/h_x)):
# for the first time layer
    u_first.append(u_x0(x)); x_coord.append(x)
    x=x+h_x
u_first[int(l/h_x)]=u_right(0)
ru_first=np.array(u_first)
u_exp=np.array([u_first]); u_nexp=np.array([u_first])
#calculation of the temperature distribution along the rod using an explicit
     difference scheme
t_0=0
for j in range(1,int(t/h_texp)+1):
    t_0=t_0+h_texp; u_next=[u_left(t_0)]
    for i in range(1,int(l/h_x)):
        u_next.append((del_tae*u_exp[j-1,i-1])+((1-(2*del_tae))*u_exp[j-
     1,i])+(del_tae*u_exp[j-1,i+1]))
    u_next.append(u_right(t_0))
    u_exp=np.insert(u_exp,j,u_next,axis=0)
    del u_next
print("Matrix of temperature distribution along the rod by time layers
     determined by the implicit difference method:")
print(u_exp)
# determination of the temperature distribution vector along the rod in the
     current time layer by the implicit difference method
def u_rawnexp(del_tane,h_tnexp, k):
    a_kof=[0];    b_kof=[-1-(2*del_tane)];    c_kof=[del_tane];    d_kof=[-
     u_nexp[k,1]-(del_tane*u_left(h_tnexp))]
    for i in range(2,int(l/h_x)):
        a_kof.append(del_tane)
        b_kof.append((-1-(2*del_tane)))
        if i==int(l/h_x)-1:
            c_kof.append(0)
            d_kof.append((-u_nexp[k,int(l/h_x)-1]-
     (del_tane*u_right(h_tnexp))))
        else:
            c_kof.append(del_tane)
            d_kof.append(-u_nexp[k,i])
    et_ta,tet_ta=[-c_kof[0]/b_kof[0]],[d_kof[0]/b_kof[0]]
    for i in range(1,int(l/h_x)-1):
        et_ta.append((-c_kof[i])/(b_kof[i]+(a_kof[i]*et_ta[i-1])))
        tet_ta.append((d_kof[i]-(a_kof[i]*tet_ta[i-
     1]))/(b_kof[i]+(a_kof[i]*et_ta[i-1])))
    u=[tet_ta[int(l/h_x)-2]]
    for i in range(int(l/h_x)-3,-1,-1):
        u.append((et_ta[i]*u[int(l/h_x)-3-i])+tet_ta[i])
    u=list(reversed(u))
    u.append(u_right(h_tnexp))
    u.insert(0,u_left(h_tnexp))
    return u
#determination of the distribution matrix of the temperature distribution
     along the rod using an implicit difference scheme
for k in range(0,int(t/h_tnexp)-1):
    u_nexp=np.insert(u_nexp,k+1,u_rawnexp(del_tane,h_tnexp, k),axis=0)
print(u_nexp)
print(len(u_nexp))
# construction of the diagram of temperature distribution along the rod by
     time layers
plt.figure(figsize=(8, 8))
plt.xlabel('x',fontsize=15, color='blue')
plt.ylabel('y',fontsize=15, color='blue')
for k in range(1,6,2):
    plt.plot(x_coord, u_exp[36000*k])
    plt.plot(x_coord, u_nexp[1440*k])
plt.grid(True)
plt.xlim([0, 2])
plt.ylim([0, 20])
plt.show().
```

Conclusions on the application of numerical methods for solving differential equations in mathematical physics

Solving engineering problems in many areas of science and technology is connected with partial differential equations. They contain partial derivatives, and the sought function depends on several variables at the same time.

The complete mathematical formulation of the problem, along with the partial differential equations, also includes some additional conditions. If the search for a solution is conducted within a limited domain, then boundary conditions are set, and the problem is then referred to as a boundary value problem for equations with partial derivatives. A problem in which it is necessary to solve a partial differential equation under given initial conditions is called a Cauchy problem. The problem is solved in an unbounded space, and boundary conditions are not specified. Problems in which both boundary and initial conditions are set simultaneously are referred to as non-stationary (or mixed) boundary value problems. The resulting solution changes over time.

Depending on the values of the functional coefficients and their ratios, different types of differential equations of mathematical physics are distinguished: transport, evolutionary, hyperbolic, parabolic, and elliptic.

There are two types of methods for solving equations of the following types:

– analytical (the result is derived through various mathematical transformations);

– numerical, where the obtained result corresponds to reality with a given accuracy, but requires a lot of algebraic calculations and the use of computing power of computer systems.

When comparing the methods of solving partial differential equations, it is necessary to remember that the finite element method approximates the solution of the problem, while the finite difference method approximates the derivatives of the sought functions. The finite element method, unlike the difference method, requires a time-consuming formulation of the problem, high qualification, and experience of the researcher, but it is convenient when solving a problem with a complex boundary shape and a non-uniform distribution of parameters.

In the initial stage of solving partial differential equations, the method for solving the problem is selected. Usually, it is easier to use the finite difference method, as it requires simpler preparation of the problem for solution. However, in certain cases, such as problems in mechanics that have a well-developed theory, it is advisable to turn to the finite element method.

When determining the steps to solve a problem, accuracy is the main factor. If accuracy is high, then either a very fine mesh or a very fine decomposition is required. At the same time, it is necessary to take into account that the error of finite-difference methods is of the first order, meaning it is proportional to h^2.

In the case of symmetry in the solution area, the number of nodes can be reduced by two or four times due to symmetry along both coordinate axes. This allows you to save time and the amount of memory of the computer system.

The choice of the initial values of the variables is of great importance for the effective solution of the problem. The speed of convergence of the calculation results depends significantly on this in the process of using iterative methods. It is often advisable to solve the problem in several stages: at the first stage, with the help of a coarse grid (or division into large elements), an initial approximation is obtained, after which an exact solution is performed on a fine grid.

To solve partial differential equations, a number of modern and effective software tools have been developed, which are used in automating the design of technological systems.

Control questions and tasks

1. Provide the definition of partial differential equations.
2. What kinds of necessary conditions must be met in order to obtain a solution to partial differential equations?
3. Provide examples of engineering problems that can be described by partial differential equations.
4. What generalized functional equation characterizes all types of partial differential equations?
5. What are the types of second-order partial differential equations, depending on the input functional coefficients?
6. Provide examples of engineering problems that are described by the corresponding types of differential equations in mathematical physics.
7. Determine the type of partial differential equations given in Table 4.2.

Table 4.2 – Output data for the task

Version	Differential equation
1	$\frac{\partial^2 u}{\partial t^2}+6\frac{\partial^2 u}{\partial t\partial x}=0$
2	$\frac{\partial^2 u}{\partial t^2}-6\frac{\partial^2 u}{\partial t\partial x}+9\frac{\partial^2 u}{\partial^2 x}=0$
3	$\frac{\partial^2 u}{\partial y^2}-6\frac{\partial^2 u}{\partial x\partial y}+9\frac{\partial^2 u}{\partial^2 x}=0$
4	$\frac{\partial^2 u}{\partial t\partial x}-3\frac{\partial u}{\partial t}+5\frac{\partial u}{\partial x}=u$
5	$\frac{\partial^2 u}{\partial x^2}-2x\frac{\partial^2 u}{\partial x\partial y}+\frac{\partial^2 u}{\partial y^2}=xu^2$
6	$\frac{\partial^2 u}{\partial x^2}+10\frac{\partial^2 u}{\partial x\partial y}+25\frac{\partial^2 u}{\partial y^2}-xy^2u=0$

8. What are the methods for solving partial differential equations? Provide a comparative analysis of each method.

9. Describe the heat conduction equation.

10. How are computational templates for partial derivatives constructed?

11. Compile calculation templates for the Laplace operator and the biharmonic operator.

12. Create calculation templates for the partial differential equations provided in Table 4.3.

Table 4.3 – Output data for the task

Version	Differential equation
7	$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y \partial x}=0$
8	$\frac{\partial u}{\partial t}-\frac{\partial^2 u}{\partial t \partial x}+\frac{\partial^2 u}{\partial^2 x}=0$
9	$\frac{\partial^2 u}{\partial y^2}-\frac{\partial^3 u}{\partial^3 x}=0$
10	$\frac{\partial^2 u}{\partial t \partial x}-\frac{\partial^4 u}{\partial x^4}=u(x,t)$
11	$\frac{\partial^2 u}{\partial x^2}-x\frac{\partial^2 u}{\partial x \partial y}+\frac{\partial^2 u}{\partial y^2}=xu(x,y)$
12	$\frac{\partial u}{\partial x}+\frac{\partial^2 u}{\partial x^2}+\frac{\partial^3 u}{\partial y^3}+\frac{\partial^2 u}{\partial x \partial y}=0$

13. What different methods are used to calculate the classical Dirichlet problem for the Laplace equation in a rectangular domain?

14. Describe the wave equation.

15. Develop an algorithm for solving elliptic partial differential equations.

16. Create a three-layer scheme for solving hyperbolic equations.

17. What necessary and sufficient stability condition must be met for the numerical solution of hyperbolic partial differential equations?

18. Develop an algorithm for solving hyperbolic equations with partial derivatives.

19. Compare the effectiveness of using explicit and implicit schemes for solving parabolic equations with partial derivatives.

20. Develop an algorithm for solving parabolic equations with partial derivatives using the explicit difference method.

21. Develop an algorithm for solving parabolic equations with partial derivatives using the implicit difference method.

22. Under what conditions are explicit and implicit two-layer difference schemes for solving parabolic partial differential equations stable?

23. How is the step chosen when solving partial differential equations?

24. Provide a general algorithm for solving partial differential equations using the difference method.

25. Determine the steady-state temperature distribution in a rectangular plate measuring $l \times h = 2{,}0 \times 1{,}5$ m, for which the following boundary conditions are set (Table 4.4): $u(0, y) = \varphi(y)$, $u(l, y) = \Psi(y)$, $u(x, 0) = \mu_1(x)$, $u(x,h) = \mu_2(x)$ ($0 \le x \le l$, $0 \le y \le h$). Construct a diagram of the stationary temperature distribution in the plate.

Table 4.4 – Output data for the task

Version	Functions of initial conditions		Functions of boundary conditions	
	$\varphi(y)$	$\Psi(y)$	$\mu_1(x)$	$\mu_2(x)$
13	$\cos(0{,}5y)$	$e^{-4}\cos(0{,}5y)$	e^{-x}	$e^{-x}\cos(0{,}5x)$
14	$e^{2y}(2y^2+1)$	$e^{2y}(2y^2+3)$	$e^{6}(x+19)$	$x+1{,}0$
15	$\mathrm{tg}y$	$\mathrm{tg}^2(y+0{,}9)$	$\mathrm{tg}^2(x+0{,}4)$	$\mathrm{tg}^2(x)$
16	0,0	$\cos y$	$5\cos x+\sin x$	$10\sin x+x\sin(1{,}0)$
17	$-\cos y$	$62\cos(y^2)$	$2x^3+3x-1$	$(15x^2+1)\cos(x)$
18	$5y(y-1)$	$11y(y-1)$	$12(x^2-x)+60$	0,0

26. It is necessary to determine the position of the string depending on time (the total time of the oscillating process T), which performs free oscillations with fixed ends (the distance between the points of fixing the ends l = 4,0 м) with initial conditions $u(x, 0) = \varphi(x)$, $\dfrac{\partial u(x,0)}{\partial t} = \psi(x)$ $(0 \le x \le l)$ and boundary conditions $u(0, t) = \mu_1(t)$, $u(l, t) = \mu_2(t)$ $(0 \le t \le T)$ (Table 4.5). The phase velocity coefficient a=1,2 m/sec. Plot the position of the oscillating string at different times.

Table 4.5 – Output data for the task

Ver-sion	Functions of initial conditions		Total oscillation time, T (sec)	Functions of boundary conditions	
	$\varphi(x)$	$\Psi(x)$		$\mu_1(t)$	$\mu_2(t)$
19	$0{,}5(x+1)^2$	$(x+0{,}5)\cos(\pi x)$	1,0	0,5	$2{,}0-3t$
20	$x^2\cos(\pi x)$	$x^2(x+1)$	0,2	$0{,}5t$	$t-1{,}0$
21	$x+1$	0,0	1,2	$0{,}5t^2$	$t+1{,}0$
22	$e^x\sin x$	$e^x(\cos x+\sin x)$	2,0	$\cos^2 t+\sin t$	$\cos^2(t+6{,}0)+$ $+\sin x$
23	$\cos^2 x+\sin x$	$2x\sin(2x)$	2,0	$e^t\sin t$	$e^{t+2}\sin(t+2{,}0)$
24	$\cos^2(0{,}5x)$	$-0{,}5\sin x$	6,0	$\cos^2(0{,}5t)$	$\cos^2(0{,}5t+3{,}0)$

27. It is necessary to determine the change in temperature distribution along the rod depending on time during the time $t_0 = 30{,}0$ min. A rod of length L with a cross-sectional area constant along its length is placed in an insulated material in such a way that only its extreme right and left ends interact with the environment. At the initial moment of time, the rod has a balanced temperature $u(x, 0) = \varphi(x)$, and the temperature $u(0, t) = \mu_1(t)$, $u(l, t) = \mu_2(t)$ $(0 \le t \le T)$ remains constant at its left and right ends (Table 4.6). Construct a diagram of the temperature distribution of the rod for different moments of time.

Table 4.6 – Output data for the task

Version	Functions of initial conditions $\varphi(x)$	Rod length, l(m)	Functions of boundary conditions		Rod material	A type of difference method
			$\mu_1(t)$	$\mu_2(t)$		
25	x^2	1,3	$50\sin t$	$40\cos^2 t$	Steel	Implicit
26	15,0	2,2	$20\cos t+$ $+10\sin t$	80,0	Copper	Explicit
27	$2x+5$	1,5	$0{,}05t+5{,}0$	$0{,}01t-15{,}0$	Silver	Implicit
28	$10\sin x$	1,4	$2e^{0{,}001t}+3{,}0$	$0{,}01t+20\sin t$	Graphite	Explicit
29	$5e^x$	2,5	–30,0	$70\sin t$	Cast iron	Implicit
30	$\cos(x)+$ $+10\sin(x)$	3,0	$35\sin^2(0{,}3t)$	–20,0	Glass	Explicit

28. Create a computational template of the Laplace operator for three coordinates.

29. The transverse deformation w of a thin rectangular plate, $l \times h =$ $=3{,}0\times1{,}7$ m in size and uniformly loaded with pressure p, is determined by the equation:

$$\Delta^2 w = p/D, \tag{4.31}$$

where $D = \dfrac{Et^3}{12(1-\nu^2)}$ – bending stiffness, $E = 2{,}1\cdot10^{11}$ MPa – elasticity modulus, $\nu = 0{,}3$ – Poisson's ratio, $t = 0{,}005$ m – plate thickness, $p = 100{,}0$ Pa – the pressure with which the plate is uniformly loaded. Determine the plate deformation distribution $w(x, y)$ depending on the given load p.

30. The Navier-Stokes equation, which describes the steady motion of a viscous fluid in a pipe of arbitrary cross-section, has the form:

$$\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = \frac{1}{\mu}\frac{dp}{dt}, \tag{4.32}$$

where v – fluid velocity modulus (it is zero at the pipe walls), μ – viscosity coefficient, dp/dz – derivative of pressure along the length of the pipe. Taking dp/dz=-5000,0 Pa/m and μ=1,5·10^{-4} N·sec/m^2, determine the distribution of the velocity modulus in the cross section of the pipe shown in Figure 4.13.

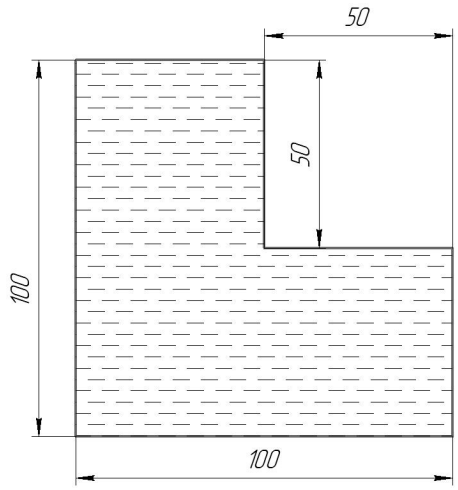


Figure 4.13 – Pipeline cross-section diagram

Chapter 5. DATA PROCESSING TASKS

A very important task in the process of identifying mathematical models is the processing of experimental data obtained during active or passive identification experiments. Various methods and algorithms are used for this purpose, and the choice of method depends on the type of modeling object, model, and the available computing power of computer tools. Among the main methods of data processing, we highlight interpolation, approximation, and statistical data processing.

5.1 Interpolation

The purpose of interpolation is to construct the function $F(x)$ (interpolant) from a given class of functions, which takes on values at individual points $x_i \epsilon [a; b]$ (i=0, 1, 2, …, n) (interpolation nodes, and their set is an interpolation grid):

$$F(x_0) = y_0,\ F(x_1) = y_1,\ ..., F(x_i) = y_i,\ ..., F(x_n) = y_n, \qquad (5.1)$$

which coincide with the previously specified values (for example, obtained from the experiment) at these points of the unknown function $y = f(x)$. Geometrically, this means that it is necessary to define a curve $y = F(x)$ of a certain type, which passes through the system of points $M(x_i, y_i)$ (i = 0, 1, 2, ..., n).

In general, this problem has both an infinite set of solutions and no solutions at all. However, it becomes unique if, instead of an arbitrary function $F(x)$, we look for a polynomial $P_n(x)$ of no higher than the n-th power that satisfies the condition (5.1), i.e.:

$$P_n(x_0) = y_0,\ P_n(x_1) = y_1\ ,\ \ldots,\ P_n(x_i) = y_i\ , \ldots,\ P_n(x_n) = y_n\ .$$

The interpolation formula $Y = F(x)$ is used for the approximate calculation of the values of the function $f(x)$ for $x \neq x_i$ (i = 0, 1, 2, …, n). It should be noted that there is interpolation in the narrow sense when $x \epsilon [x_0; x_n]$, and extrapolation when x is outside the interval $[x_0; x_n]$ i.e. $x < x_0$ or $x > x_n$.

When analyzing the interpolation procedure, it is necessary to specify the constraints that are imposed on the set of base points (x_i, y_i). The initial interpolation grid of points should only describe a smooth function. According to the conditions of a specific problem, the values of the derivative function must be set at the edge points of the input interpolation grid to obtain an unambiguous result.

The main methods of interpolation include:

1. Linear interpolation (linear interpolation). The simplest and fastest method, in which the specified nodal points (x_i, y_i) are connected by straight lines.

2. Interpolation using polynomials. A polynomial of the n-th order is used, which in the general case has the form:

$$P(x) = P_n(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n,$$

where a_i (i=0, 1, …, n) – constant coefficients.

All methods of finding the interpolation polynomial are reduced to obtaining constant coefficients. Such methods include:

a) interpolation by difference methods;

b) interpolation according to Lagrange;

c) Hermit polynomial interpolation.

3. Polynomial spline interpolation. Nodal points are connected using a polynomial of a given order, which is chosen depending on the method. The most common spline interpolation methods include:

a) classical cubic splines;

b) Hermit splines;

c) B-splines;

d) Bezier curves.

This section considers the algorithmization and application of the most common interpolation methods in engineering practice.

5.1.1 Different methods

There are many well-known finite-difference interpolation methods. The most common is Newton's method for «forward» interpolation (Newton–Gregory method). The interpolation polynomial in this case has the form:

$$P_n(x) = C_0 + C_1(x - x_0) + C_2(x - x_0)(x - x_1) + \ldots$$

$$\ldots + C_n(x - x_0)(x - x_1) \ \ldots \ (x - x_{n-1}).$$

Coefficients C_i are determined from equations:

$$P_n(x_i) = y_i \ (i = 0, 1, 2, \ldots, n),$$

which allows you to record the system:

$$\begin{cases} C_0 = y_0, \\ C_0 + C_1(x_1 - x_0) = y_1, \\ C_0 + C_1(x_2 - x_0) + C_2(x_2 - x_0)(x_2 - x_1) = y_2, \\ \ldots \\ C_0 + C_1(x_n - x_0) + \ldots + C_n(x_n - x_0)(x_n - x_1)\ldots(x_n - x_{n-1}) = y_n. \end{cases} \quad (5.2)$$

Equation (5.2) is a LAES with a triangular matrix. If you take a step $x_{i+1} - x_i = h$, then in the region of change of values of interpolation nodes $x \in [x_0; x_n]$, a one-dimensional uniform interpolation grid will be obtained. This will allow us

to use the difference image of the system (5.2), resulting in difference expressions for determining the coefficients:

$$C_0 = y_0,\ C_1 = \frac{y_1 - y_0}{h} = \frac{\Delta y_0}{h},$$

where Δy_0 – right difference of the first order at the point y_0;

$$C_2 = \frac{y_2 - 2y_1 + y_0}{2h^2} = \frac{\Delta^2 y_0}{2h^2},$$

where $\Delta^2 y_0$ – right difference of the second order;

$$C_j = \frac{\Delta^j y_0}{(j!)h^j},$$

where $\Delta^j y_0$ – right difference of the j-th order.

Then equation (5.2) can be written as:

$$P_n(x) = y_0 + \frac{\Delta y_0}{1!h}(x - x_0) + \frac{\Delta^2 y_0}{2!h^2}(x - x_0)(x - x_1) + \ldots$$

$$\ldots + \frac{\Delta^n y_0}{n!h^n}(x - x_0)(x - x_1)\ \ldots\ (x - x_{n-1}). \qquad (5.3)$$

From a practical standpoint, the expression is used to determine higher order differences:

$$\Delta^j y_i = \Delta(\Delta^{j-1} y_i) = \Delta^{j-1} y_{i+1} - \Delta^{j-1} y_i,\ (i=0,\ 1,\ 2,\ \ldots,\ n–j).$$

In the case when n=1, from (5.3) we obtain the formula for linear interpolation:

$$P_1(x) = y_0 + \frac{\Delta y_0}{h}(x - x_0),$$

and if n=2 – parabolic or quadratic interpolation formula:

$$P_2(x) = y_0 + \frac{\Delta y_0}{h}(x - x_0) + \frac{\Delta^2 y_0}{2h^2}(x - x_0)(x - x_1).$$

If an unlimited number of values of the function x is specified, then n can be any number. In practice, n is chosen in such a way that the difference $\Delta^n y_i$ is constant within a given level of accuracy. Any tabular value of the argument x can be used as the initial value of x_0. When the number of function values is finite, the number of n is limited and cannot exceed the number of function values y reduced by one.

The scheme of the algorithm for implementing interpolation according to Newton's first interpolation formula will take the form shown in Figure 5.1.

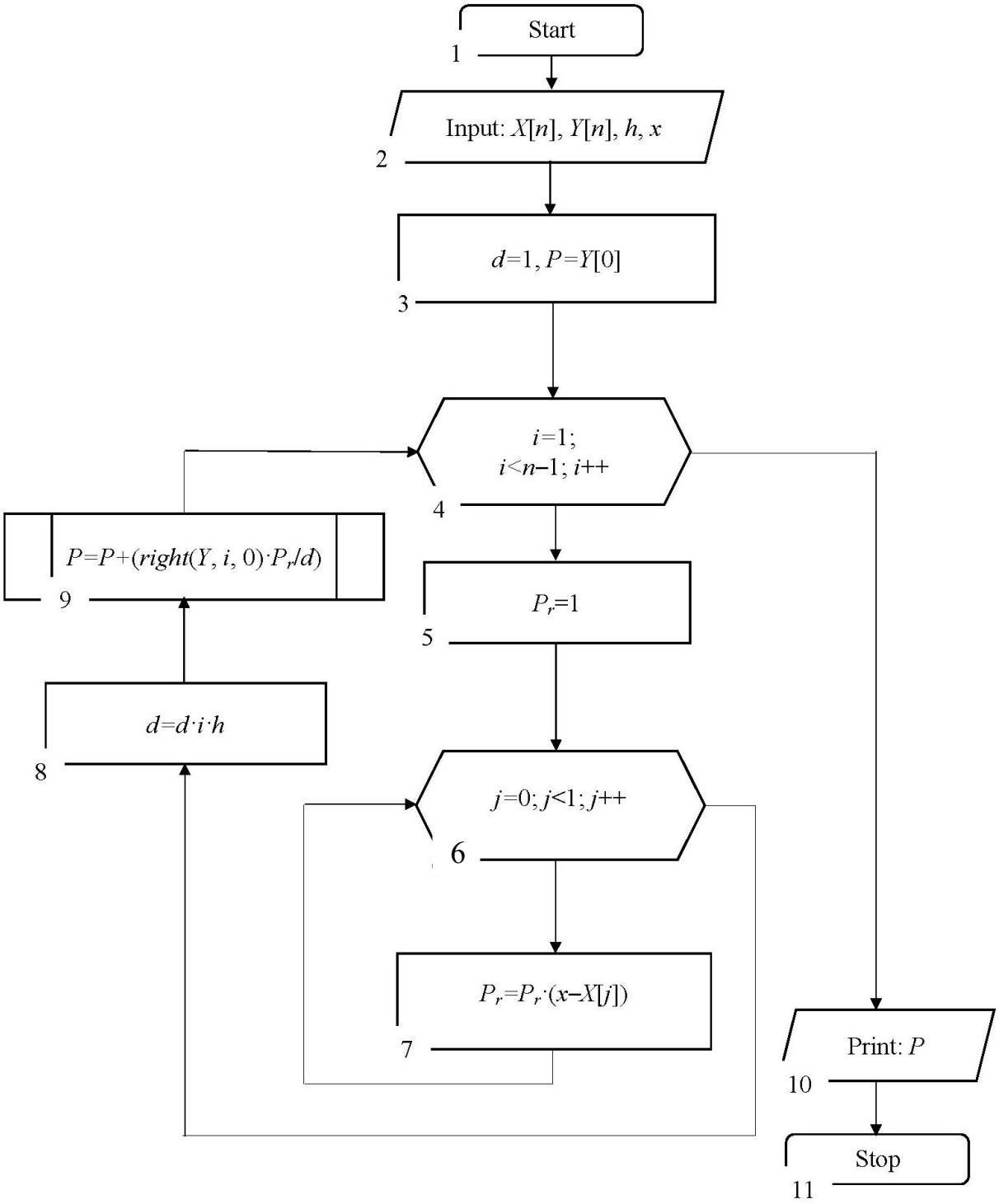


Figure 5.1 – Algorithm of the first interpolation method Newton's interpolation formula

To find the differences, you need to use recursion. For this, the function *float right(float y[], int p, int i)* is created. The algorithm for determining finite differences for a given function is shown in Figure 5.2.

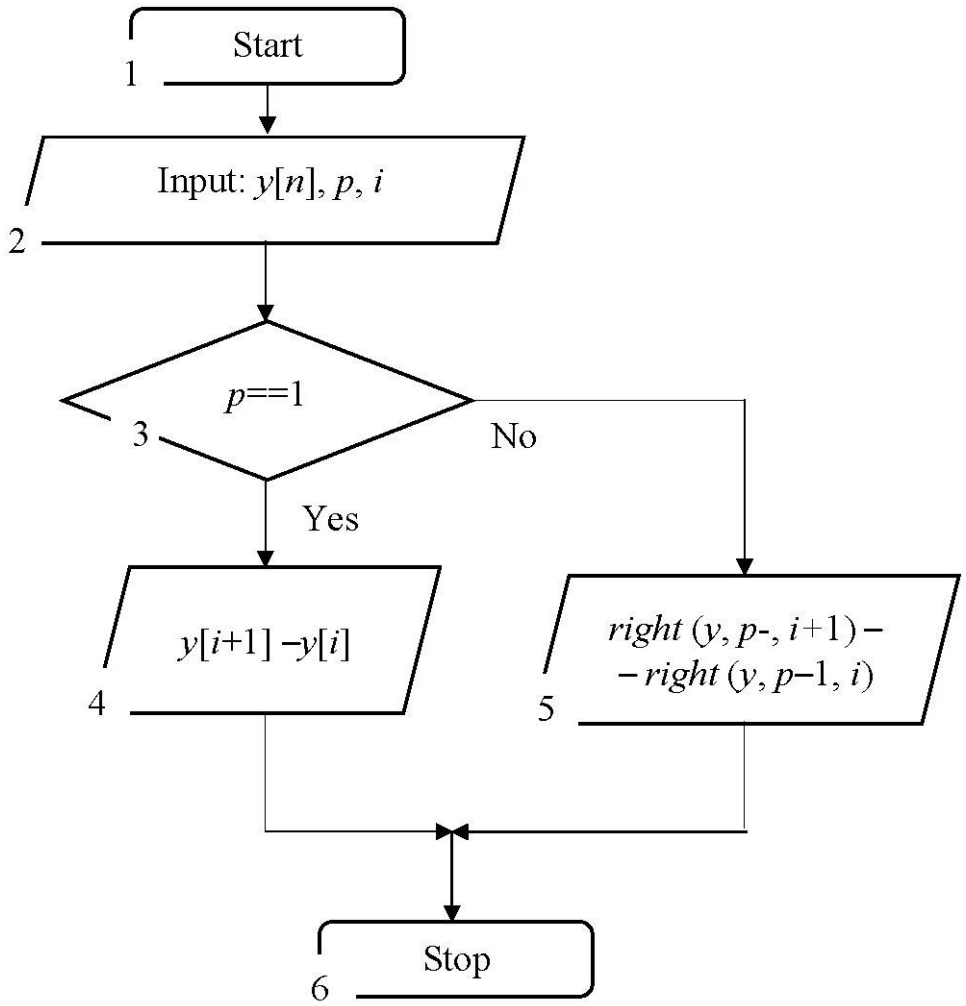


Figure 5.2 – Algorithm for determining finite differences:
y[] is an array of y values; *p* is the order of difference; *i* is the ordinal number of the variable *y* for which the difference is calculated

Consider the procedure for finding differences in the C++ programming language:

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
const int n = 6;
float y (float x)
{return (1/(pow(x,4)+5));}
float right(float y[], int p, int i)
{
 if (p == 1)
return y[i+1]-y[i];
 else
return right(y,p-1,i+1)-right(y,p-1,i);
}
void main()
{
float x[6] = {1, 1.1, 1.2, 1.3, 1.4, 1.5};
float y[6] = {0.16667, 0.15470, 0.14137, 0.12729, 0.11310, 0.09938};
```

```
 for (int i = 1; i<n; i++) {
for (int j = 0; j<n-i; j++)
printf("%9.5f ", right(y, i, j));
printf("\n");
}  }
```

```
Result:
-0.01197    -0.01333    -0.01408    -0.01419    -0.01372
-0.00136    -0.00075    -0.00011     0.00047
 0.00061          0.00064           0.00058
 0.00003          -0.00006
-0.00009
```

Implementation of the Newton's method algorithm according to the first interpolation formula in the C++ programming language:

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
const int n = 6;
float y (float x)
{return (1/(pow(x,4)+5));}
float right(float y[], int p, int i) // auxiliary function for finding
     differences from array Y, order of difference p, coefficient i
{
if (p == 1)
 return y[i+1]-y[i];
 else
{
return right(y,p-1,i+1)-right(y,p-1,i);
 }
};
void main()
{
float X[6] = {1, 1.1, 1.2, 1.3, 1.4, 1.5};
float Y[6] = {0.16667, 0.15470, 0.14137, 0.12729, 0.11310, 0.09938};
float h = 0.1, d = 1, Pr;
float x = 1.07;
float P = Y[0];
for (int i = 1; i<n-1; i++)
{
Pr = 1;
for (int j = 0; j<i; j++)
Pr = Pr * (x-X[j]);
d = d * i * h;
P = P + (right(Y, i, 0)*Pr/d);
}
printf("x = %.5f y = %.5f", x, P);
getch();
}.
```

Formula (5.3) is called Newton's first interpolation formula. This expression is inconvenient for interpolation around the last values of y_i. In this case, as a rule, Newton's second interpolation formula is used, which is obtained by using the left differences from the last value (x_n, y_n) («backward» interpolation). Then the interpolation polynomial will look like this:

$$P_n(x) = C_0 + C_1(x - x_n) + C_2(x - x_n)(x - x_{n-1}) + C_3(x - x_n)(x - x_{n-1})(x - x_{n-2}) + \ \ldots$$
$$\ldots + C_n(x - x_n)(x - x_{n-1}) \ \ldots \ (x - x_1).$$

The coefficients C_j are determined as follows:

$$C_0 = y_n,\ C_1 = \frac{\Delta y_{n-1}}{h} = \frac{\nabla y_n}{h},$$

where ∇y_n – «left» difference of the first order at the point y_n;

$$C_2 = \frac{\Delta^2 y_{n-2}}{2!h^2} = \frac{\nabla^2 y_n}{2h^2},$$

where $\nabla^2 y_n$ – «left» difference of the second order;

$$\ldots,\ C_j = \frac{\Delta^j y_{n-j}}{j!h^j} = \frac{\nabla^j y_n}{j!h^j},$$

where $\nabla^j y_n$ – j-th order left difference.

The final expression for Newton's second interpolation formula:

$$P_n(x) = y_n + \frac{\Delta y_{n-1}}{1!h}(x - x_n) + \frac{\Delta y_{n-2}^2}{2!h^2}(x - x_n)(x - x_{n-1}) + \ldots$$
$$\ldots + \frac{\Delta^n y_0}{n!h^n}(x - x_n)(x - x_{n-1}) \ldots (x - x_1).$$

Newton's interpolation formulas can be used to extrapolate a function. If $x<x_0$, then it is convenient to use Newton's first interpolation formula, and $\frac{x - x_0}{h} < 0$.

If $x>x_n$, then Newton's second interpolation formula is used, where $\frac{x - x_n}{h} > 0$.

Thus, the first Newton's interpolation formula is generally used for forward interpolation and backward extrapolation, and the second for backward interpolation and forward extrapolation.

Newton's formulas use «left» and «right» differences. Using «central» differences to obtain interpolation formulas leads to the Gaussian's, Stirling's, and Bessel's formulas.

It should be noted that the «central» differences are not used in the usual way, but by applying the «right» differences with a gradual shift of the indices to the left.

It is convenient to consider these Newton's formulas on $(2n+1)$ equidistant interpolation nodes:

$$x_{-n},\ x_{-(n-1)},\ \ldots,\ x_{-1},\ x_0,\ x_1,\ \ldots,\ x_{n-1},\ x_n,$$

and $\Delta x_i = x_{i+1} - x_i = h = \text{const}$ ($i = -n, -(n-1), \ldots, n-1$), and for the function $y = f(x)$, its values at these nodes $y_i = f(x_i)$ are known.

Let it be necessary to construct a polynomial $P(x)$ of degree no higher than $2n$ such that $P(x_i) = y_i$. Then the polynomial $P(x)$ is defined as:

$$P(x) = C_0 + C_1(x - x_0) + C_2(x - x_0)(x - x_1) + C_3(x - x_{-1}) \times$$
$$\times (x - x_0)(x - x_1) + \ldots + C_{2n-1}(x - x_{-(n-1)}) \ldots \times \qquad (5.4)$$
$$\times \ldots (x - x_{-1})(x - x_0)(x - x_1) \ldots (x - x_{n-1})(x - x_n).$$

Similarly to Newton's interpolation formulas, using (5.4), are determined: $C_0 = y_0$; $C_1 = \dfrac{\Delta y_0}{h}$; $C_2 = \dfrac{\Delta^2 y_{-1}}{2!h^2}$; …, $C_{2n-1} = \dfrac{\Delta^{2n-1} y_{-(n-1)}}{(2n-1)!h^{2n-1}}$; $C_{2n} = \dfrac{\Delta^{2n} y_{-n}}{(2n)!h^{2n}}$.

Substituting the found values of the coefficients in (5.4), we obtain the first Gaussian's interpolation formula, which contains the differences (Table 5.1):

$$\Delta y_0,\ \Delta^2 y_{-1},\ \Delta^3 y_{-1},\ \Delta^4 y_{-2},\ \Delta^5 y_{-2},\ \Delta^6 y_{-2},\ \ldots.$$

Similarly, a second Gaussian's interpolation formula, containing central differences, can be obtained:

$$\Delta y_{-1},\ \Delta^2 y_{-1},\ \Delta^3 y_{-2},\ \Delta^4 y_{-2},\ \Delta^5 y_{-3},\ \Delta^6 y_{-3},\ \ldots.$$

Using the arithmetic mean of the first and second interpolation formulas of Gauss, we obtain Stirling's formula. These formulas enable us to derive Bessel's interpolation formula. In general, it is recommended to use interpolation formulas with central differences within the interval, while at its boundary nodes, Newton's formulas are typically used (refer to Table 5.1).

The interpolation errors for Newton's formulas can be estimated, respectively, for the first and second formulas as:

$$\Delta_n(x) = \frac{q(q-1)\ldots(q-n)}{(n+1)!}\Delta^{n+1} y_0;$$

$$\Delta_n(x) = \frac{q(q+1)\ldots(q+n)}{(n+1)!}\Delta^{n+1} y_n,$$

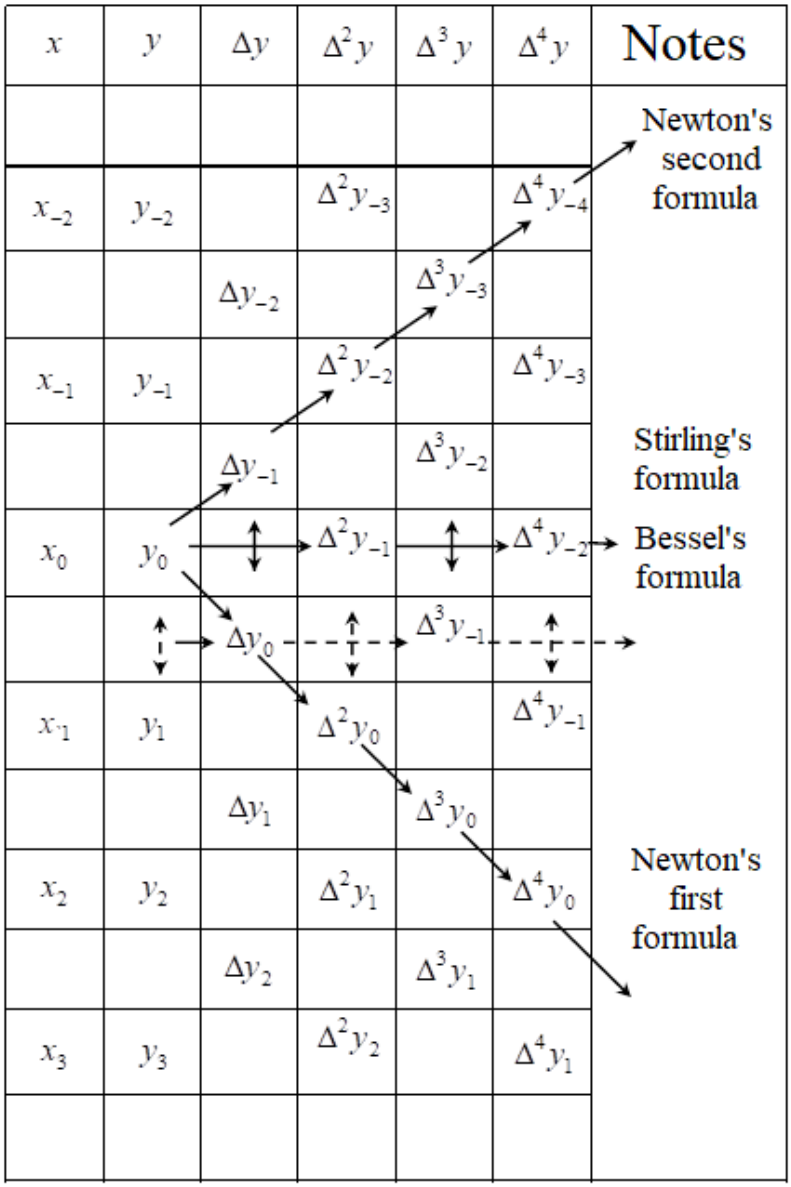
where $q = \dfrac{x - x_n}{h}$.

For the Stirling's formula:

$$\Delta(x_n) = \frac{\Delta^{2n+1} y_{-(n-1)} + \Delta^{2n+1} y_{-n}}{2(2n+1)!} q(q^2 - 1)(q^2 - 2^2)\ \ldots\ (q^2 - n^2).$$

For the case of unequally spaced values of the argument, interpolation formulas can be obtained using the definition of divided differences.

Table 5.1 – Application of difference interpolation formulas



For example, the operation $[x_i, x_{i+1}] = \dfrac{y_{i+1} - y_i}{x_{i+1} - x_i}$ is referred to as the divided difference of the first order, while the operation $[x_i, x_{i+1}, x_{i+2}] = \dfrac{[x_{i+1}, x_{i+2}] - [x_i, x_{i+1}]}{x_{i+2} - x_i}$ is referred to as the divided difference of the second order.

Divided differences of order *n* are obtained from the recurrence relation:

$$[x_i, x_{i+2}, \ldots, x_{i+n}] = \frac{[x_{i+1}, \ldots, x_{i+n}] - [x_{i+1}, \ldots, x_{i+n-1}]}{x_{i+n} - x_i}.$$

Newton's interpolation formula for unequally spaced values of the argument can also be obtained:

$$P(x) = y_0 + [x_0, x_1](x - x_0) + [x_0, x_1, x_2](x - x_0)(x - x_1) + \ldots$$
$$+ [x_0, x_1, \ldots, x_n](x - x_0)(x - x_1)\ldots(x - x_{n-1}).$$

Example 5.1. The production site manufactures construction metal structures from sheet and profile rolled metal. The site operates consistently, and the orders are generally the same type, with a slight variation in the number of workers. There is data available on the total output of products (in thousand units) for the previous four months. The goal is to determine the function that represents the dependence of output (in thousand units) over the entire production period. To solve this, we will use Newton's first interpolation formula, taking into account the known values of the interpolation nodes (as shown in Table 5.2). Finally, we will find the value of the total output in one and a half months (x = 1.5) by performing the interpolation.

Table 5.2 – Output data

i	0	1	2	3
x_i, month number	1	2	3	4
y_i, thousand units	2	5	10	17

Solution:

We determine the values of the differences of different orders:

1) x_0=1, y_0=2;

2) x_1=2, y_1=5, $\Delta y_0 = y_1 - y_0$=5–2=3;

3) x_2=3, y_2=10, $\Delta y_1 = y_2 - y_1$=10–5=5;

4) x_2=4, y_3=17, $\Delta y_2 = y_3 - y_2$=17–10=7;

5) $\Delta^2 y_0 = \Delta y_1 - \Delta y_0$=5–3=2;

6) $\Delta^3 y_0 = \Delta(\Delta^2 y_0) = \Delta^2 y_1 - \Delta^2 y_0 = (\Delta y_2 - \Delta y_1) - (\Delta y_1 - \Delta y_0)$=(7–5)–(5–3)=0.

We determine the values of the coefficients of the interpolation polynomial (5.3):

$$C_0 = y_0 = 2; C_1 = \frac{\Delta y_0}{1!h^1} = \frac{3}{1 \cdot 1} = 3;\ C_2 = \frac{\Delta^2 y_0}{2!h^2} = \frac{2}{1 \cdot 2 \cdot 1^2} = 1;$$

$$C_3 = \frac{\Delta^3 y_0}{3!h^3} = \frac{0}{1 \cdot 2 \cdot 3 \cdot 1^3} = 0.$$

Interpolation polynomial function:

$$P(x) = C_0 + C_1(x-1) + C_2(x-1)(x-2) + C_3(x-1)(x-2)(x-3),$$

$$P(x) = 2 + 3(x-1) + (x-1)(x-2).$$

The value of the interpolation function at a point x = 1,5:

$$P(1,5) = 2 + 3(1,5-1) + (1,5-1)(1,5-2) = 3,25 \text{ thousand units.}$$

Example 5.2. Table 5.3 shows the results of measuring beam deflection from a uniformly distributed vertical force load. Determine the deflection function along the length of the beam. Solve the interpolation problem using Newton's formula based on the known values of the interpolation nodes (Table 5.3) and find the value of the function at the point x = 0,4 m.

Table 5.3 – Output data

i	0	1	2	3
x_i, m	0,0	0,1	0,3	0,5
y_i, mm	– 0,5	0,0	0,2	1,0

Solution:

Using Newton's interpolation polynomial, based on the initial data given in Table 5.3, we have the case of unequally spaced nodes for $n = 3$. Then, the value of the divided differences:

$$[x_0, x_1] = \frac{y_1 - y_0}{x_1 - x_0} = \frac{0,0 - (-0,5)}{0,1} = 5,0;$$

$$[x_1, x_2] = \frac{y_2 - y_1}{x_2 - x_1} = \frac{0,2 - 0,0}{0,3 - 0,1} = 1,0;$$

$$[x_2, x_3] = \frac{y_3 - y_2}{x_3 - x_2} = \frac{1,0 - 0,2}{0,5 - 0,3} = 4,0;$$

$$[x_0, x_1, x_2] = \frac{[x_1, x_2] - [x_0, x_1]}{x_2 - x_0} = \frac{1,0 - 5,0}{0,3 - 0,0} = -\frac{40}{3};$$

$$[x_1, x_2, x_3] = \frac{[x_2, x_3] - [x_1, x_2]}{x_3 - x_1} = \frac{4,0 - 1,0}{0,5 - 0,1} = \frac{15}{2};$$

$$[x_0, x_1, x_2, x_3] = \frac{[x_1, x_2, x_3] - [x_0, x_1, x_2]}{x_3 - x_0} = \frac{\frac{15}{2} - \left(-\frac{40}{3}\right)}{0,5 - 0,0} = \frac{125}{3}.$$

The results of the calculations are given in Table 5.4.

Table 5.4 – Results of the calculation of split differences

n	x_n	y_n	$[x_n, x_{n+1}]$	$[x_n, x_{n+1}, x_{n+2}]$	$[x_n, x_{n+1}, x_{n+2}, x_{n+3}]$
0	0	– 0,5			
1	0,1	0,0	5,0	– 40/3	125/3
2	0,3	0,2	1,0	15/2	
3	0,5	1,0	4,0		

Newton's interpolation formula for unequally spaced values of the argument:

$$P(x) = y_0 + [x_0, x_1](x - x_0) + [x_0, x_1, x_2](x - x_0)(x - x_1) + [x_0, x_1, x_2, x_3](x - x_0) \times$$

$$\times (x - x_1)(x - x_2) = -0,5 + 5(x - 0) + \left(-\frac{40}{3}\right)(x - 0)(x - 0,1) +$$

$$+ \frac{125}{3}(x - 0)(x - 0,1)(x - 0,3) = \frac{125}{3}x^3 - 30x^2 + \frac{91}{12}x - 0,5.$$

For $x = 0,4$ m; $y = P(0,4) = \frac{125}{3}(0,4)^3 - 30 \cdot (0,4)^2 + \frac{91}{12}(0,4) - 0,5 = 0,36$ mm.

5.1.2 Lagrangian interpolation

The Lagrangian interpolation is used in general cases with arbitrarily located nodes.

The interpolation polynomial for the Lagrange's method is given in the form:

$$P_n(x) = y_0 b_0(x) + y_1 b_1(x) + \ldots + y_n b_n(x),$$

where all $b_j(x)$ (j=0, 1, 2, …, n) are polynomials of degree n, the coefficients of which can be found using the (n+1)-th equation.

$$P_n(x_i) = y_i,$$

as a result, a system of equations will be obtained:

$$\begin{cases} y_0 b_0(x_0) + y_1 b_1(x_0) + \ \ldots\ + y_n b_n(x_0) = y_0; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\ y_0 b_0(x_n) + y_1 b_1(x_n) + \ \ldots\ + y_n b_n(x_n) = y_n. \end{cases}$$

If the value of $b_j(x_i)$ is determined so that:

$$b_j(x) = \begin{cases} 1, \ i = j; \\ 0, \ i \neq j, \end{cases}$$

then the system of equations will be defined.

This condition means that any polynomial $b_j(x)$ is zero for every x_i except when x_i is equal to x_j. Therefore, in the general case, the polynomial $b_j(x)$ has the following form:

$$b_j(x) = C_j(x - x_0)(x - x_1)\ldots(x - x_{j-1})(x - x_{j+1})\ldots(x - x_n).$$

If $b_j(x)$ = 1, then the coefficients C_j are determined from the expression:

$$C_j = 1/(x_j - x_0)\ldots(x_j - x_{j-1})(x_j - x_{j+1})\ldots(x_j - x_n). \qquad (5.5)$$

Then for the defined polynomial, we get:

$$P_n(x) = \sum_{j=0}^{n} y_j \frac{(x - x_0)(x - x_1)\ldots(x - x_{j-1})(x - x_{j+1})\ldots(x - x_n)}{(x_j - x_0)(x_j - x_1)\ldots(x_j - x_{j-1})(x_j - x_{j+1})\ldots(x_j - x_n)}. \qquad (5.6)$$

Then for the defined polynomial, we get:

$$L_j(x) = (x - x_0)(x - x_1)\ldots(x - x_{j-1})(x - x_{j+1})\ldots(x - x_n),$$

can be written down

$$P_n(x) = \sum_{j=0}^{n} y_j \frac{L_j(x)}{L_j(x_j)}. \qquad (5.7)$$

It is necessary to note two main properties of Lagrange polynomials:

1. $\sum_{j=0}^{n}\left(L_j(x)/L_j(x_j)\right)=1.$

2. If $P_n(x)$ depends linearly on y_j, then the principle of superposition is valid: the interpolation polynomial of the sum of several functions is equal to the sum of the interpolation polynomials of the components.

The Lagrangian interpolation error is estimated by the residual term of the interpolation formula.

If the interpolation nodes are different from each other, and the function $f(x)$ is such that it has a continuous derivative of order $(n+1)$ on the interval $[a; b]$, where the interpolation nodes are located, it is possible to write the residual term of the interpolation formula $R_n(x)=f(x)-P_n(x)$ in the form:

$$R_n(x)=\frac{f^{(n+1)}(\xi)}{(n+1)!}\,\omega_{n+1}(x),$$

where $\xi\in[\alpha_1;\alpha_2]$, $\alpha_1=\min\left(x, x_0, x_1, \ldots, x_n\right)$, $\alpha_2=\max\left(x, x_0, x_1, \ldots, x_n\right)$.

Then

$$\left|R_n(x)\right|\le\frac{M_{n+1}}{(n+1)!}\left|\omega_{n+1}(x)\right|,$$

where $M_{n+1}=\max\limits_{x\in[\alpha_1,\alpha_2]}\left|f^{(n+1)}(x)\right|$.

The scheme of the algorithm for implementing the Lagrangian interpolation is presented in Figure 5.3.

Consider the implementation of finding differences in the C++ programming language:

```
#include <stdio.h>
#include <math.h>
#include <conio.h>

const int n = 4;

void main()
{
  float X[n] = {1, 2, 3, 4};
  float Y[n] = {15, 17, 7, 21};
  float x = 2.5;
  float yx = 0, Pr;
  for (int i=0; i<n; i++)
  {
    Pr = 1;
        for (int j=0; j<n; j++)
          if (i!=j)
            Pr = Pr * ((x-X[j])/(X[i]-X[j]));
        yx = yx + Y[i]*Pr;
  }
  printf("y = %.4f\n", yx);
  getch();
}
Result:
y = 11.2500.
```

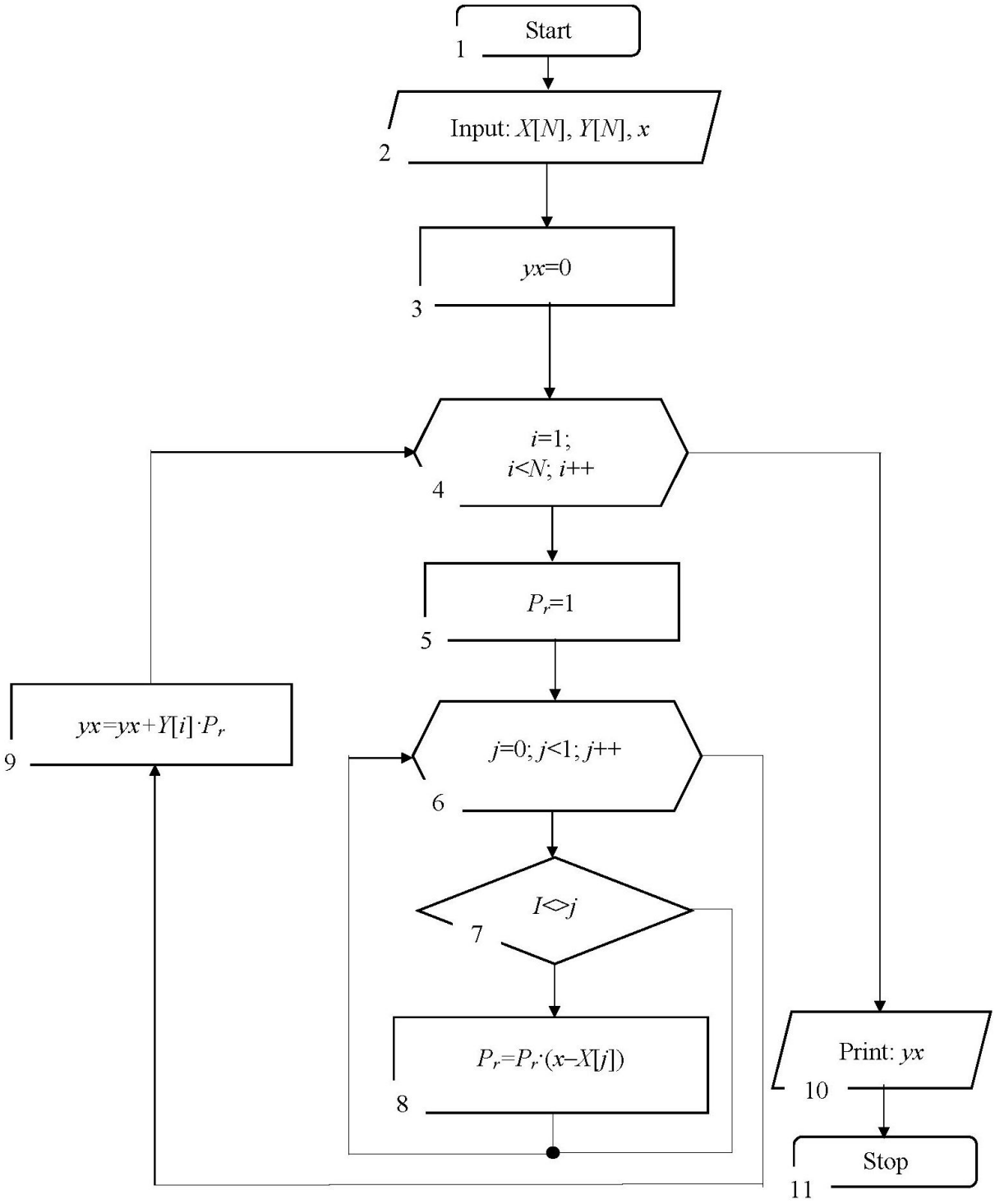


Figure 5.3 – Scheme of the interpolation algorithm according to the Lagrangian interpolation

Example 5.3. Find the value of the function of the change of motion acceleration depending on the time of the body's movement $y=f(x)$ based on its experimental table values (Table 5.5) and find the value of the function at the point x = 0,4 sec.

Table 5.5 – Output data

i	0	1	2	3
x_i, sec	0,0	0,1	0,3	0,5
y_i, m/sec^2	– 0,5	0,0	0,2	1,0

Solution:

Using the Lagrange interpolation polynomial formula:

$$P_3(x)=\sum_{i=0}^{n} y_i \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j}=\sum_{i=0}^{3} y_i \prod_{\substack{j=0 \\ j\neq i}}^{3} \frac{x-x_j}{x_i-x_j}=y_0\frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)}+$$

$$+y_1\frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)}+y_2\frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)}+$$

$$+y_3\frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}=-0,5\frac{(x-0,1)(x-0,3)(x-0,5)}{(0,0-0,1)(0,0-0,3)(0,0-0,5)}+$$

$$+0\frac{(x-0,0)(x-0,3)(x-0,5)}{(0,1-0,0)(0,1-0,3)(0,1-0,5)}+0,2\frac{(x-0,0)(x-0,1)(x-0,5)}{(0,3-0,0)(0,3-0,1)(0,3-0,5)}+$$

$$+1,0\frac{(x-0,0)(x-0,1)(x-0,3)}{(0,5-0,0)(0,5-0,1)(0,5-0,3)}=\frac{125}{3}x^3-30x^2+\frac{91}{12}x-0,5.$$

For x=0,4 sec; $y\approx L(0,4)=\frac{125}{3}0,4^3-30\cdot 0,4^2+\frac{91}{12}0,4-0,5=$

=0,3999 m/sec^2.

Example 5.4. Determine the Lagrange interpolation polynomial based on tabular data (Table 5.6) obtained from the function $y=x+\sin x$, with an interpolation error Δ=0,2·10^{-4}.

Table 5.6 – Output data

i	0	1	2	3
x_i	1,40	1,50	1,70	1,80
y_i	2,38545	2,49749	2,69166	2,77385

Solution:

We find the Lagrange interpolation polynomial in the form:

$$P_3(x)=\sum_{i=0}^{n} y_i \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j}=\sum_{i=0}^{3} y_i \prod_{\substack{j=0 \\ j\neq i}}^{3} \frac{x-x_j}{x_i-x_j}=y_0\frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)}+$$

$$+y_1\frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)}+y_2\frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)}+$$

$$+y_3\frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}=2,38548\frac{(x-1,5)(x-1,7)(x-1,8)}{(1,4-1,5)(1,4-1,7)(1,4-1,8)}+$$

$$+2,49749\frac{(x-1,4)(x-1,7)(x-1,8)}{(1,5-1,4)(1,5-1,7)(1,5-1,8)}+$$

$$+2,69166\frac{(x-1,4)(x-1,5)(x-1,8)}{(1,7-1,4)(1,7-1,5)(1,7-1,8)}+$$

$$+2,77385\frac{(x-1,4)(x-1,5)(x-1,7)}{(1,8-1,4)(1,8-1,5)(1,8-1,7)}=$$

$$=-198,79(x-1,5)(x-1,7)(x-1,8)+416,2483(x-1,4)(x-1,7)(x-1,8)-$$

$$-448,61(x-1,4)(x-1,5)(x-1,8)+231,1542(x-1,4)(x-1,5)(x-1,7).$$

Lagrangian interpolation error estimation:

$$\left|R_n(x)\right|=\left|R_3(x)\right|\le\frac{M_{n+1}}{(n+1)!}\left|\omega_{n+1}(x)\right|=\frac{M_4}{4!}\left|\omega_4(x)\right|=\frac{0,98545}{4!}\cdot 0,0004=1,64\cdot 10^{-5},$$

де $\omega_4(1,6)=(x-x_0)(x-x_1)(x-x_2)(x-x_3)=(1,6-1,4)(1,6-1,5)(1,6-1,7)\times$

$\times(1,6-1,8)=0,2\cdot 0,1\cdot(-0,1)\cdot(-0,2)=0,0004$; α_1=min(x, x_0, x_1, x_2, x_3)=min(1,6; 1,4; 1,5; 1,7; 1,8)=1,4; α_2=max(x, x_0, x_1, x_2, x_3)=max(1,6; 1,4; 1,5; 1,7; 1,8)=1,8;

$$M_{n+1}=M_4=\max_{x\in[\alpha_1,\alpha_2]}\left|f^{(4)}(x)\right|=\max_{x\in[1,4;\,1,8]}\left|\sin(x)\right|=0,98545.$$

Since $\left|R_n(x)\right|=1,64\cdot 10^{-5}<\Delta=0,2\cdot 10^{-4}$, which corresponds to the condition of the problem.

To determine the value of the Lagrangian interpolation polynomial at the point $x=1,6$, the formula $P_3(x)=\sum_{i=0}^{3}y_i\prod_{\substack{j=0\\ j\ne i}}^{3}\frac{x-x_j}{x_i-x_j}$ is used. Calculations are performed step by step based on the formula, and the results are summarized in Table 5.7.

Table 5.7 – Calculation results of the interpolation polynomial

i	$x-x_i$	x_i-x_j, $i\ne j$			$\Pi(x_i-x_j)$, $i\ne j$	$y_i\prod_{\substack{j=0\\ j\ne i}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)$	$\sum_{i=0}^{n}y_i\prod_{\substack{j=0\\ j\ne i}}^{n}\left(\frac{x-x_j}{x_i-x_j}\right)$
0	0,2	– 0,1	– 0,3	– 0,4	– 0,012	– 0,3978500	– 0,3978500
1	0,1	0,1	– 0,2	– 0,3	0,006	1,6649930	1,2671430
2	– 0,1	0,3	0,2	– 0,1	– 0,006	1,7944400	3,0615830
3	– 0,2	0,4	0,3	0,1	0,012	– 0,4623083	2,5992747

Based on the calculation results (see Table 5.6), the value of the Lagrange interpolation polynomial $P_3(1,6)=2,5992747$. To compare the value of the function $y=x+\sin x$ at point x=1,6 to seven decimal places, it is y(1,6)=2,5995736.

5.1.3 Spline interpolation

Interpolation with a Lagrange or Newton polynomial over the entire segment using a large number of interpolation nodes often leads to an inaccurate approximation, which is explained by the significant accumulation of errors in the calculation process. In addition, due to the divergence of the interpolation process, increasing the number of nodes does not always lead to increased accuracy. In order to avoid large errors, the entire segment is divided into partial segments, and on each of the partial segments, the function $f(x)$ is approximately replaced by a polynomial of low degree (the so-called piecewise polynomial interpolation).

One way to interpolate over the entire segment is to use spline functions. A spline function, or spline, is a piecewise polynomial function that is defined on a segment and has a certain number of continuous derivatives on this segment.

The word «spline» means a «flexible ruler» used to draw smooth curves through given points in a plane. The main advantage of splines is the ability to locally change the shape of the curve on a selected range of values.

Classic cubic spline

Consider the most well-known and common interpolation spline of the third order. In machine-building drawings, these splines are widely used in the form of patterns (flexible rulers), which are deformed so that with their help it is possible to draw a curve through given points (x_i, y_i). It can be shown (using the theory of elastic transverse bending of a beam under small deformations) that a spline is a group of combined cubic polynomials where the first and second derivatives of the corresponding functions of the polynomials are continuous at the junctions. Such functions are called cubic splines (Fig. 5.4), for the construction of which it is necessary to set the coefficients that uniquely determine the polynomial in the interval between two points.

For a mathematical description of cubic splines, consider the segment $[a; b]$ of the real OX axis. The interpolation grid, $a = x_0 < x_1 < \ldots < x_n = b$, consists of nodes where the values of the function $f(x)$ are determined as $y_i = f(x_i)$ $(i = 0, 1, \ldots, n)$. It is necessary to construct a continuous function – spline $S(x)$ on the segment $[a; b]$ that satisfies the following requirements:

1. On each segment $[x_{i-1}; x_i]$, the spline $S(x)$ is a polynomial $S_i(x)$ of degree not higher than three (Fig. 5.4):

$$S_i(x) = k_{1i} + k_{2i}x + k_{3i}x^2 + k_{4i}x^3, \tag{5.8}$$

where k_{ij} – constant coefficients to be determined.

2. At nodes x_i, the spline $S_i(x)$ acquires given values $y_i = f(x_i)$ $(i = 0, 1, \ldots, n)$, which is:

$$\begin{cases} S_i(x_i) = y_i \quad (i = 1, 2, \ldots, n); \\ S_{i+1}(x_i) = y_i \quad (i = 0, 1, \ldots, n-1). \end{cases} \tag{5.9}$$

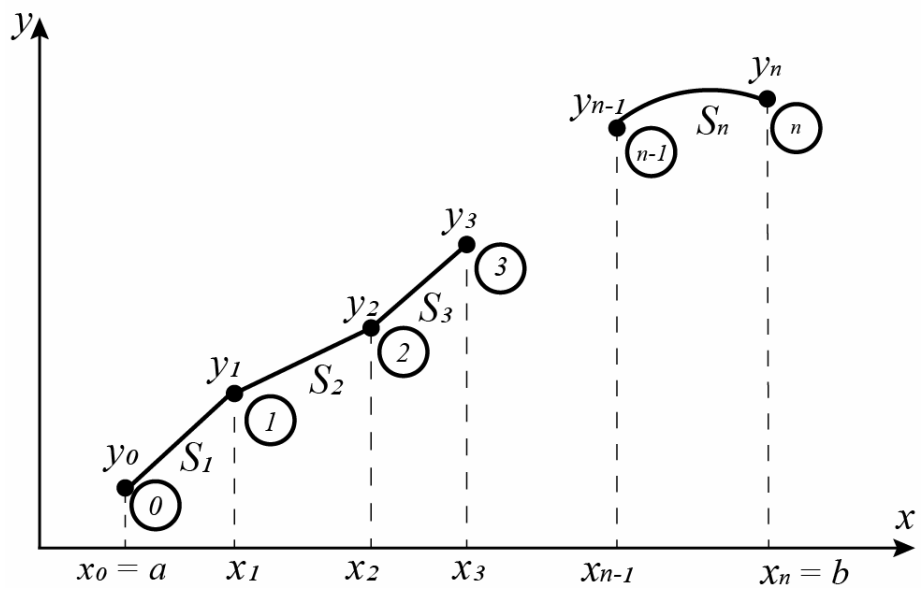


Figure 5.4 – Spline interpolation scheme

Condition (5.19) is necessary for passing splines through the nodes of a given interpolation grid $a = x_0 < x_1 < \ldots < x_n = b$, $y_i = f(x_i)$ $(i = 0, 1, \ldots, n)$. This condition (5.19) forms $2n$ equations.

3. At internal nodes x_i $(i = 1, 2, \ldots, n–1)$, the spline has continuous first and second derivatives, namely:

$$\begin{cases} S'_{i+1}(x_i) = S'_i(x_i) \ (i = \overline{1, n-1}); \\ S''_{i+1}(x_i) = S_i''(x_i) \ (i = \overline{1, n-1}). \end{cases}$$

That is, the following expressions will be obtained:

$$\begin{cases} S_i(x) = k_{4i}x^3 + k_{3i}x^2 + k_{2i}x + k_{1i} \quad (i = \overline{1, n}); \\ S_i'(x) = 3k_{4i}x^2 + 2k_{3i}x + c_i \quad (i = \overline{1, n-1}); \\ S_i''(x) = 6k_{4i}x + 2k_{3i} \quad (i = \overline{1, n-1}). \end{cases}$$

At the points of spline conjugation, their first and second derivatives must be equal to each other. The number of such conditions should be $2n–2$. To find the spline, it is necessary to determine the coefficients k_{ij} of the polynomials $S_i(x)$ $(i = 1, 2, \ldots, n–1)$ with $4n$ unknowns, which satisfy the system of $4n–2$ equations.

Two additional equations are needed to obtain the solution of the system of equations. They are obtained by determining the curvature value of the spline graph at the ends of the general interpolation curve, namely: $S''(x_0) = \sigma_1$, $S''(x_n) = \sigma_2$.

The algorithm for solving the interpolation problem using a third-order polynomial is presented in Figure 5.5.

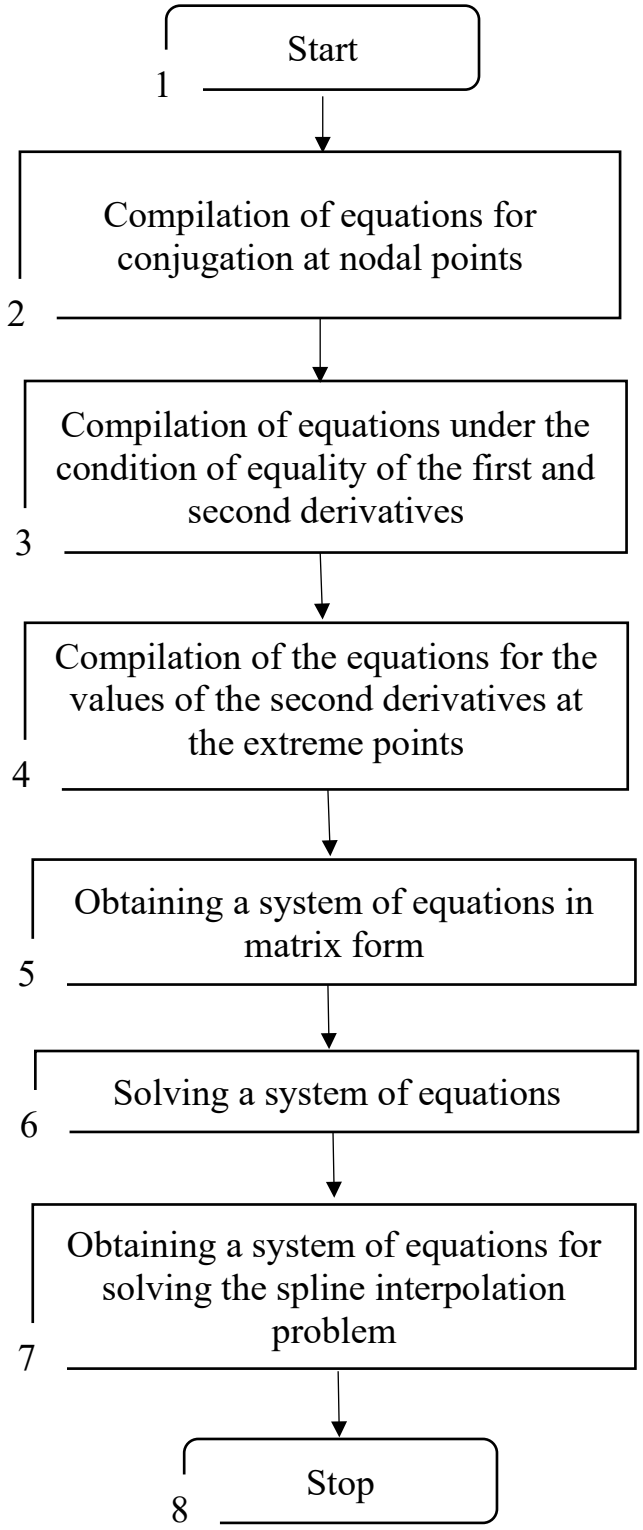


Figure 5.5 – Scheme of the algorithm for solving the interpolation problem using a third-order polynomial

If $\sigma_1 = \sigma_2 = 0$, then such a spline is called natural. If there is additional information about the behavior of the function at the ends of the interpolation interval, then additional boundary conditions are recorded. Thus, the obtained

cubic spline, which is «glued» from cubic parabolas, passes through the given points, is a piecewise smooth and continuous function.

To construct the curve function (5.18), it is necessary to determine four coefficients. The expression (5.18) can be written in the form proposed by Charles Hermite, which allows us to reduce the number of computational operations. For this, individual cubic equations can be written in the form:

$$S_i(x) = ty_i + \overline{t}y_{i-1} + \Delta x_i\left[(k_{i-1} - d_i)\overline{t}t^2 - (k_i - d_i)t^2\overline{t}\right] \quad (i = \overline{1,n}), \qquad (5.10)$$

where $\Delta x_i = x_i - x_{i-1}$, $t = \dfrac{x - x_{i-1}}{\Delta x_i}$, $\bar{t} = 1 - t$, $\Delta y_i = y_i - y_{i-1}$, $\dfrac{\Delta y_i}{\Delta x_i} = d_i$, Δx_i, Δy_i – length of the interval; t and $\overline{t}$ are auxiliary variables; x is an intermediate point on the segment $[x_{i-1}; x_i]$.

Each of the $S_i(x)$ equations (5.20) contains only two constant unknown coefficients. After the first equation $S_i(x)$ is written, only one new unknown coefficient is added with each subsequent equation. Then for $x=x_{i-1}$ $t=0$, $\overline{t} = 1$, and for $x=x_i$ $t=1$, $t=0$.

Accordingly, all conditions, except for the conditions for the second derivatives, are satisfied. Second derivatives for interior points are expressed as ratios:

$$k_{i-1}\Delta x_{i+1} + 2k_i(\Delta x_i + \Delta x_{i+1}) + k_{i+1}\Delta x_i = 3(d_i\Delta x_{x+1} + d_{i+1}\Delta x_i),$$

and for the two outer ones – $2k_0+k_1=3d_1$ і $k_{m-1}+2k_m=3d_m$.

Thus, the system of equations to be solved is linear, and its matrix is tridiagonal (see Chapter 1):

$$\begin{vmatrix} 2 & 1 & 0 & 0 & 0 \\ \Delta x_2 & 2(\Delta x_1 + \Delta x_2) & \Delta x_1 & 0 & 0 \\ 0 & \Delta x_3 & 2(\Delta x_2 + \Delta x_3) & \Delta x_2 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 1 & 2 \end{vmatrix} \cdot \begin{vmatrix} k_0 \\ k_1 \\ k_2 \\ \cdots \\ k_n \end{vmatrix} = 3 \begin{vmatrix} d_1 \\ d_1\Delta x_2 + d_2\Delta x_1 \\ d_2\Delta x_3 + d_3\Delta x_2 \\ \cdots\cdots\cdots\cdots \\ d_{m-1}\Delta x_m + d_m\Delta x_{m-1} \\ d_m \end{vmatrix}.$$

In many cases, the spline interpolation method is the most convenient, as it allows you to obtain an analytical piecewise polynomial function. There are higher-order splines. The application of this method is also possible in other areas of computational mathematics, for example, in numerical integration or for solving differential equations.

Example 5.5. Determine the deflection function of a normally loaded rod along its length using spline interpolation based on a third-order polynomial for the function $y=f(x)$, which is given in the table (Table 5.10), and also determine the approximate value of the deflection of the rod at the point x=2,5 m.

Table 5.8 – Output date

i	0	1	2	3
x_i, m	1,0	2,0	3,0	4,0
y_i, 10^{-2} mm	15,0	17,0	7,0	21,0

Solution:

According to three ranges of values of the argument x_i (see Table 5.10), on the basis of condition (5.18), the following system of m=3 equations will be obtained:

$$y = \begin{cases} A_1x^3 + B_1x^2 + C_1x + D_1, & x \in [1; 2]; \\ A_2x^3 + B_2x^2 + C_2x + D_2, & x \in [2; 3]; \\ A_3x^3 + B_3x^2 + C_3x + D_3, & x \in [3; 4]. \end{cases} \quad (5.11)$$

In order to determine the unknown coefficients коефіцієнти A_1, B_1, C_1, D_1, A_2, B_2, C_2, D_2, A_3, B_3, C_3, D_3, it is necessary to create a system with the number of $m \cdot 4$=12 equations and 12 unknowns.

The first 2m equations are composed based on the requirement that the splines must converge at the specified nodal points:

1) $A_1x_0^3 + B_1x_0^2 + C_1x_0 + D_1 = y_0$;

2) $A_1x_1^3 + B_1x_1^2 + C_1x_1 + D_1 = y_1$;

3) $A_2x_1^3 + B_2x_1^2 + C_2x_1 + D_2 = y_1$;

4) $A_2x_2^3 + B_2x_2^2 + C_2x_2 + D_2 = y_2$;

5) $A_3x_2^3 + B_3x_2^2 + C_3x_2 + D_3 = y_2$;

6) $A_3x_3^3 + B_3x_3^2 + C_3x_3 + D_3 = y_3$.

The following 2m–2 equations are formed under the condition that the first and second derivatives are equal at the points of conjugation of the splines:

7) $3A_1x_1^2 + 2B_1x_1 + C_1 = 3A_2x_1^2 + 2B_2x_1 + C_2$;

8) $3A_2x_2^2 + 2B_2x_2 + C_2 = 3A_3x_2^2 + 2B_3x_2 + C_3$;

9) $6A_1x_1 + 2B_1 = 6A_2x_1 + 2B_2$;

10) $6A_2x_2 + 2B_2 = 6A_3x_2 + 2B_3$.

For the last two equations, an additional condition is used that the value of the second derivative at the extreme points must be equal to zero, namely:

11) $6A_1x_0 + 2B_1 = 0$;

12) $6A_3x_3 + 2B_3 = 0$.

Substituting the initial data (see Table 5.10), we obtain a system of 12 equations with 12 unknowns:

$$\begin{cases} A_1 + B_1 + C_1 + D_1 = 15; \\ 8A_1 + 4B_1 + 2C_1 + D_1 = 17; \\ 8A_2 + 4B_2 + 2C_2 + D_2 = 17; \\ 27A_2 + 9B_2 + 3C_2 + D_2 = 7; \\ 27A_3 + 9B_3 + 3C_3 + D_3 = 7; \\ 64A_3 + 16B_3 + 4C_3 + D_3 = 21; \\ 12A_1 + 4B_1 + C_1 = 12A_2 + 4B_2 + C_2; \\ 27A_2 + 6B_2 + C_2 = 27A_3 + 6B_3 + C_3; \\ 12A_1 + 2B_1 = 12A_2 + 2B_2; \\ 18A_2 + 2B_2 = 18A_3 + 2B_3; \\ 6A_1 + 2B_1 = 0; \\ 24A_3 + 2B_3 = 0. \end{cases} \quad (5.12)$$

Submitting the system of equations (5.22) in matrix form, we obtain:

$$\begin{array}{c|cccccccccccc|c} & A_1 & B_1 & C_1 & D_1 & A_2 & B_2 & C_2 & D_2 & A_3 & B_3 & C_3 & D_3 & \\ \hline 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15 \\ 2 & 8 & 4 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 17 \\ 3 & 0 & 0 & 0 & 0 & 8 & 4 & 2 & 1 & 0 & 0 & 0 & 0 & 17 \\ 4 & 0 & 0 & 0 & 0 & 27 & 9 & 3 & 1 & 0 & 0 & 0 & 0 & 7 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 27 & 9 & 3 & 1 & 7 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 64 & 16 & 4 & 1 & 21 \\ 7 & 12 & 4 & 1 & 0 & -12 & -4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 27 & 6 & 1 & 0 & -27 & -6 & -1 & 0 & 0 \\ 9 & 12 & 2 & 0 & 0 & -12 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 & 0 & 18 & 2 & 0 & 0 & -18 & -2 & 0 & 0 & 0 \\ 11 & 6 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 24 & 2 & 0 & 0 & 0 \end{array} . \quad (5.13)$$

After solving the system of equations (5.22), for example, using the Gaussian elimination, we substitute the found coefficients in (5.21), which allows us to obtain a solution to the given problem.

When solving the problem using the Gaussian elimination, it is necessary to first transform the system (5.23) into the form so that there are no zero elements in the main diagonal (it is necessary to change the corresponding equations), namely:

$$
\begin{array}{r|cccccccccccc|c}
 & A_1 & B_1 & C_1 & D_1 & A_2 & B_2 & C_2 & D_2 & A_3 & B_3 & C_3 & D_3 & \\
11 & 6 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15 \\
7 & 12 & 4 & 1 & 0 & -12 & -4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 8 & 4 & 2 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 17 \\
8 & 0 & 0 & 0 & 0 & 27 & 6 & 1 & 0 & -27 & -6 & -1 & 0 & 0 \\
9 & 12 & 2 & 0 & 0 & -12 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 8 & 4 & 2 & 1 & 0 & 0 & 0 & 0 & 17 \\
4 & 0 & 0 & 0 & 0 & 27 & 9 & 3 & 1 & 0 & 0 & 0 & 0 & 7 \\
10 & 0 & 0 & 0 & 0 & 18 & 2 & 0 & 0 & -18 & -2 & 0 & 0 & 0 \\
12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 24 & 2 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 27 & 9 & 3 & 1 & 7 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 64 & 16 & 4 & 1 & 21
\end{array}
\quad (5.14)
$$

Let's solve the system of equations using the Gaussian elimination. The C/C++ program looks like this:

```
#include <iostream>
#include <stdlib.h>

int main(int argc, char *argv[])
{
 int n = 12;
 float a[12][12] = {6, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                    1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                    12, 4, 1, 0, -12, -4, -1, 0, 0, 0, 0, 0,
                    8, 4, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                    0, 0, 0, 0, 27, 6, 1, 0, -27, -6, -1, 0,
                    12, 2, 0, 0, -12, -2, 0, 0, 0, 0, 0, 0,
                    0, 0, 0, 0, 8, 4, 2, 1, 0, 0, 0, 0,
                    0, 0, 0, 0, 27, 9, 3, 1, 0, 0, 0, 0,
                    0, 0, 0, 0, 18, 2, 0, 0, -18, -2, 0, 0,
                    0, 0, 0, 0, 0, 0, 0, 0, 24, 2, 0, 0,
                    0, 0, 0, 0, 0, 0, 0, 0, 27, 9, 3, 1,
                    0, 0, 0, 0, 0, 0, 0, 0, 64, 16, 4, 1};
 float b[12] = {0, 15, 0, 17, 0, 0, 17, 7, 0, 0, 7, 21};
 float d;

for (int i=0; i<n-1; i++)
    for (int j=i+1; j < n; j++)
    {
       d = a[j][i] / a [i][i];
       for (int m=0; m < n; m++)
           a[j][m] = a[i][m] * d - a[j][m];
       b[j] = b[i]*d - b[j];
    }

    float x[12];

    for (int i=n-1; i>=0; i--)
    {
```

```
        float D = 0;
        for (int j=n-1; j>i; j--)
             D = D + x[j]*a[i][j];
        x[i] = (b[i]-D) / a[i][i];
    }
    for (int i=0; i<n; i++)
       cout <<  "x[" << i << "]=" << x[i] << endl;

system("PAUSE");
return 0; }
```

In the process of solving the system of equations (5.24), the following coefficient values were obtained: $A_1 = -4{,}8$; $B_1=14{,}4$; $C_1 = -7{,}6$; $D_1 = 13{,}0$; $A_2 = 12{,}0$; $B_2 = -86{,}4$; $C_2 = 194{,}0$; $D_2 = -121{,}4$; $A_3 = -7{,}2$; $B_3 = 86{,}4$; $C_3 = -324{,}4$; $D_3 = 397{,}0$.

Accordingly, the system of equations (5.21) will have the form:

$$y = \begin{cases} -4{,}8x^3 + 14{,}4x^2 - 7{,}6x + 13, & x \in [1;\ 2]; \\ 12x^3 - 86{,}4x^2 + 194x - 121{,}4, & x \in [2;\ 3]; \\ -7{,}2x^3 + 86{,}4x^2 - 324{,}4x + 397, & x \in [3;\ 4]. \end{cases}$$

Then $y(2{,}5) = 12 \cdot 2{,}5^3 - 86{,}4 \cdot 2{,}5^2 + 194 \cdot 2{,}5 - 121{,}4 = 11{,}1 \cdot 10^{-2}$ mm.

5.2 Data approximation

Approximation is an approximate description provided by one function (the approximating function) of a given form of another function (the approximated function), which can be given in any form (for data approximation, it is provided in the form of data arrays).

There are two main approaches to data approximation. For one of them, it is necessary that the approximating curve (perhaps piecewise smooth) passes through all the points specified in the table. This is implemented using the interpolation methods discussed in the previous subsection. Another approach is to approximate the data with a simple function that is applied to all table values, but not necessarily through all points. This approach is called curve fitting, which is sought to be drawn so that its deviation from tabular data is minimal. As a rule, the least squares (LS) is used, that is, the sum of the squares of the differences between the value of the function determined by the selected approximating curve and the tabular data is minimized.

At the same level as the most common LS, the Chebyshev's method is also used, in which the maximum distance of the approximating curve from the approximated one is minimized. In general, the criterion of closeness can be any measure justified by the statement of the problem, which leads to the use of

various mathematical methods and models of approximation, as well as algorithms for finding the parameters of the approximating function.

Let the table specify $(n+1)$ points (x_0, y_0), (x_1, y_1), …, (x_n, y_n) and it is necessary to determine the approximating curve $g(x)$ in the range $x_0 \le x \le x_n$ (Fig. 5.6). In this case, the error in each table point:

$$\varepsilon_i = g(x_i) - y_i \,. \tag{5.15}$$

Then the sum of squared errors is determined by the expression:

$$E = \sum_{i=0}^{n} \left[g(x_i) - y_i \right]^2 . \tag{5.16}$$

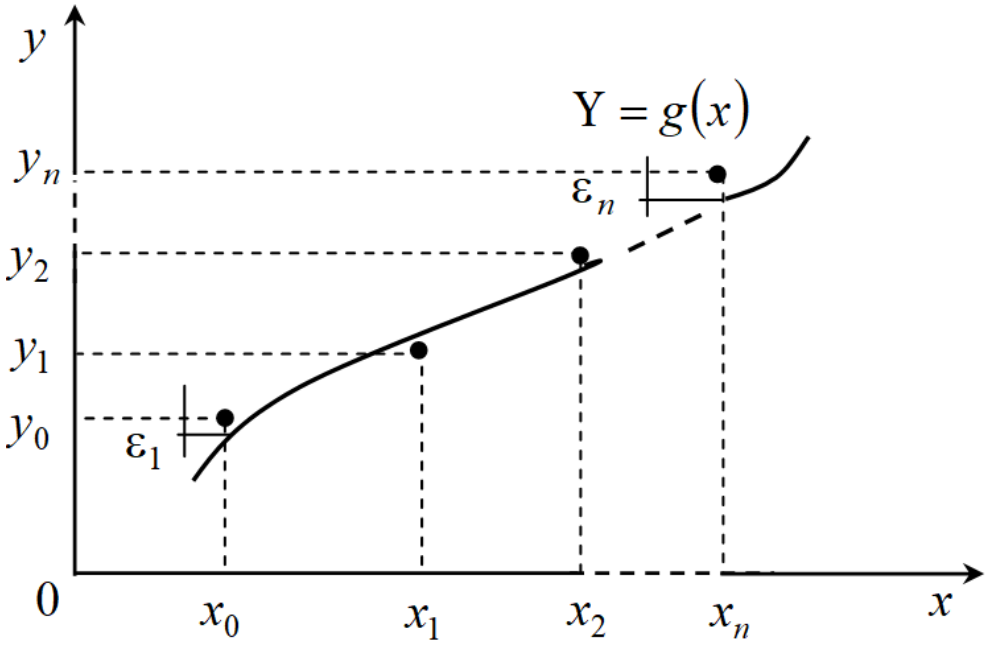


Figure 5.6 – Schematic diagram of data approximation

As a rule, the function $g(x)$ is chosen as a linear combination of typical functions $g_k(x)$:

$$g(x) = C_1 g_1(x) + C_2 g_2(x) + \ldots + C_k g_k(x) \,. \tag{5.17}$$

The minimum condition of the function E is determined by the equations:

$$\frac{\partial E}{\partial C_1} = \frac{\partial E}{\partial C_2} = \ldots = \frac{\partial E}{\partial C_k} = 0 \,. \tag{5.18}$$

Since

$$E = \sum_{i=0}^{n} \left[C_1 g_1(x_i) + C_2 g_2(x_i) + \ \ldots \ + C_k g_k(x_i) - y_i \ \right]^2 ,$$

then from formula (5.25) we obtain the following system of equations:

$$\begin{cases} \dfrac{\partial E}{\partial C_1} = 2\sum\left[C_1 g_1(x_1) + ... + C_k g_k(x_i) - y_i\right] g_1(x_1) = 0; \\ \dfrac{\partial E}{\partial C_2} = 2\sum\left[C_1 g_i(x_1) + ... + C_k g_k(x_i) - y_i\right] g_1(x_i) = 0; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\ \dfrac{\partial E}{\partial C_k} = 2\sum\left[C_1 g_1(x_i) + ... + C_k g_k(x_i) - y_i\right] g_k(x_i) = 0. \end{cases}$$

This system of k equations can be written in matrix form, namely:

$$\begin{vmatrix} \sum g_1^2(x_i) & \sum g_1(x_i)g_2(x_i) & \cdots & \sum g_1(x_i)g_k(x_i) \\ \sum g_1(x_i)g_2(x_i) & \sum g_2^2(x_i) & \cdots & \sum g_2(x_i)g_k(x_i) \\ \ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots & \cdots & \ldots\ldots\ldots\ldots \\ \sum g_1(x_i)g_k(x_i) & \sum g_2(x_i)g_k(x_i) & \cdots & \sum g_k^2(x_i) \end{vmatrix} \begin{vmatrix} C_1 \\ C_2 \\ \cdots \\ C_3 \end{vmatrix} = \begin{vmatrix} \sum g_1(x_i)y_i \\ \sum g_2(x_i)y_i \\ \cdots \\ \sum g_k(x_i)y_i \end{vmatrix}. \quad (5.19)$$

Since the elements of the matrix on the left and the column vector on the right parts of (5.29) are determined by tabular data, the resulting system of k linear equations with k unknowns has a solution.

If we switch to the matrix form of the record, then the general formula of LS takes the following form:

$$\boldsymbol{C} = [\boldsymbol{U}^T \cdot \boldsymbol{U}]^{-1} \cdot \boldsymbol{U}^T \cdot \boldsymbol{Y}, \quad (5.20)$$

where $\boldsymbol{C} = \begin{vmatrix} C_1 \\ C_2 \\ \cdots \\ C_k \end{vmatrix}$, $\boldsymbol{Y} = \begin{vmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{vmatrix}$, $\boldsymbol{U} = \begin{vmatrix} g_1(x_1) & \cdots & g_k(x_1) \\ g_1(x_2) & \cdots & g_k(x_2) \\ \cdots & \cdots & \cdots \\ g_1(x_k) & \cdots & g_k(x_n) \end{vmatrix}$.

The choice of the type of function $g(x)$ must be made taking into account the nature of tabular data (periodicity, symmetry properties, existence of asymptotics, etc.).

For example, by approximation, a quadratic function will be obtained:

$$g(x) = C_0 + C_1 x + C_2 x^2 \implies E = \sum_{i=1}^{n}\left[(C_0 + C_1 x + C_2 x^2) - y_i\right]^2;$$

$$\begin{cases} \dfrac{\partial E}{\partial C_0} = 2\sum_{i=0}^{n}\left[(C_0 + C_1 x + C_2 x^2) - y_i\right] = 0; \\ \dfrac{\partial E}{\partial C_1} = 2\sum_{i=0}^{n}\left[(C_0 + C_1 x + C_2 x^2) - y_i\right] x_i = 0; \\ \dfrac{\partial E}{\partial C_2} = 2\sum_{i=0}^{n}\left[(C_0 + C_1 x + C_2 x^2) - y_i\right] x_i^2 = 0; \end{cases}$$

$$\begin{bmatrix} (n+1) & \sum x_i & \sum x_i^2 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i^2 & \sum x_3^i & \sum x_i^4 \end{bmatrix} \begin{bmatrix} C_0 \\ C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \end{bmatrix}.$$

Sometimes tabular data is divided into several parts and a separate approximating curve is selected for each part. This approach satisfies those cases when the data correspond to different physical states of the system.

The residual mean square error of approximation is estimated using the following expression:

$$\Delta = \sqrt{E/(n+1)}. \tag{5.21}$$

The algorithm for solving the approximation problem using the method of least squares will take the form (Fig. 5.7).

If orthogonal polynomials for which $\sum g_j(x_i) g_k(x_i) = 0$ ($j \neq k$), are used during the construction of the approximating function $g_i(x)$, then the system (5.29) is simplified and the matrix becomes diagonal. For their part, the coefficients are determined from such ratios:

$$C_j = \sum_{i=0}^{n} g_j(x_i) y_i \Big/ \sum_{i=0}^{n} g_j^2(x_i). \tag{5.22}$$

This approach simplifies the task, which allows orthogonal polynomials to be used in many standard curve fitting programs.

Example 5.6. The distribution function of the company's profit rate $f(x)$ in thousand USD by calendar months, presented in tabular form (Table 5.9), it is necessary to approximate the type function:

$$\phi(x) = C_1 x^2 + C_2 x + C_3 + C_4 \sin 2x. \tag{5.23}$$

Table 5.9 – Output data

i	1	2	3	4	5
X, month	0	1	2	3	4
$f(x)$, thousand USD	–1,0	2,0	3,0	–2,0	5,0

Solution:

Let's introduce functional notation based on the approximating function (5.33): $g_1(x) = x^2$, $g_2(x) = x$, $g_3(x) = 1$, $g_4(x) = \sin 2x$.

Based on system (5.29) written in matrix form, we obtain:

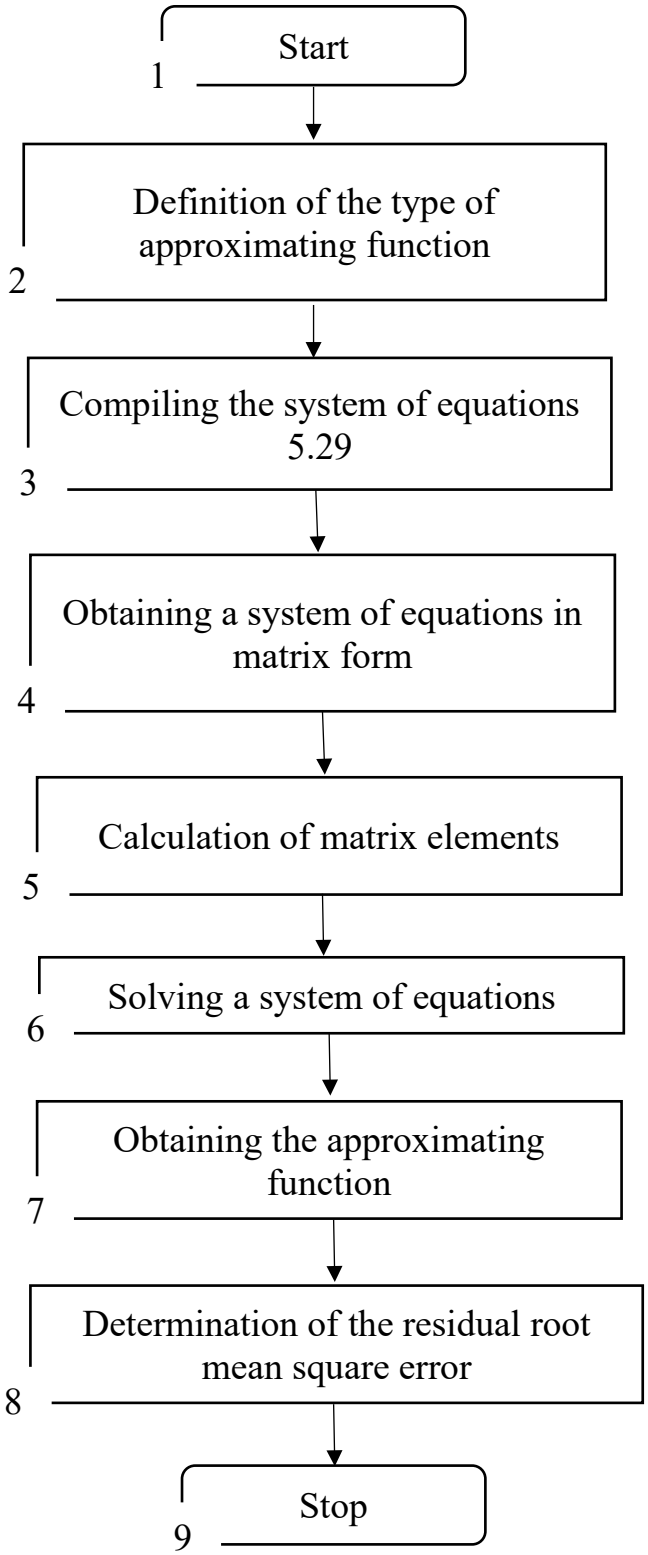


Figure 5.7 – Scheme of the algorithm for solving the approximation problem using the method of least squares

$$\begin{bmatrix} \sum_{i=1}^{n}(x_i^2)^2 & \sum_{i=1}^{n}x_i^2 x_i & \sum_{i=1}^{n}x_i^2 & \sum_{i=1}^{n}x_i^2 \sin 2x_i \\ \sum_{i=1}^{n}x_i^2 x_i & \sum_{i=1}^{n}x_i^2 & \sum_{i=1}^{n}x_i & \sum_{i=1}^{n}x_i \sin 2x_i \\ \sum_{i=1}^{n}x_i^2 & \sum_{i=1}^{n}x_i & 1 & \sum_{i=1}^{n}\sin 2x_i \\ \sum_{i=1}^{n}x_i^2 \sin 2x_i & \sum_{i=1}^{n}x_i \sin 2x_i & \sum_{i=1}^{n}\sin 2x_i & \sum_{i=1}^{n}\sin^2 2x_i \end{bmatrix} \times \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n}f(x_i)x_i^2 \\ \sum_{i=1}^{n}f(x_i)x_i \\ \sum_{i=1}^{n}f(x_i) \\ \sum_{i=1}^{n}f(x_i)\sin 2x_i \end{bmatrix}.$$

Let's calculate all elements of the matrix:

$$\sum_{i=1}^{n}(x_i^2)^2 = \sum_{i=1}^{5}x_i^4 = 0+1^4+2^4+3^4+4^4 = 354;$$

$$\sum_{i=1}^{n}x_i^2 x_i = \sum_{i=1}^{5}x_i^3 = 0+1^3+2^3+3^3+4^3 = 100;$$

$$\sum_{i=1}^{n}x_i^2 = 0+1^2+2^2+3^2+4^2 = 30;$$

$$\sum_{i=1}^{n}x_i^2 \sin 2x_i = 0+\sin 2+4\sin 4+9\sin 6+16\sin 8 == 0{,}909-3{,}027+2{,}515+$$
$$+15{,}83=11{,}197;$$

$$\sum_{i=1}^{n}x_i^2 f(x_i) = -1\cdot 0+2\cdot 1^2+3\cdot 2^2-2\cdot 3^2+5\cdot 4^2 = 2+12-18+80 = 76.$$

Similarly, calculations are performed for the remaining elements of the matrix:

$$\begin{bmatrix} 354{,}0 & 100{,}0 & 30{,}0 & 11{,}197 \\ 100{,}0 & 30{,}0 & 10{,}0 & 2{,}515 \\ 30{,}0 & 10{,}0 & 1{,}0 & 0{,}862 \\ 11{,}197 & 2{,}515 & 0{,}862 & 2{,}456 \end{bmatrix} \times \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{bmatrix} = \begin{bmatrix} 76{,}0 \\ 22{,}0 \\ 7{,}0 \\ 5{,}054 \end{bmatrix}. \qquad (5.24)$$

Solving the system of equations (5.34) (see Chapter 1) will yield unknown coefficients $C_{1,2,3,4}$: $C_1 = -0{,}0376$; $C_2 = 0{,}6645$; $C_3 = 0{,}2115$; $C_4 = 1{,}4745$.

So, the approximating function will look like this:

$$\varphi(x) = -0{,}0376x^2 + 0{,}6645x + 0{,}2115 + 1{,}4745\sin 2x.$$

For a complete solution, it is necessary to determine the residual error. According to formula (5.31), it is necessary to determine the sum of squares of deviations of the obtained function $\varphi(x)$ from the given $f(x)$. First, the value of the

function $\varphi(x)$ at the specified points is determined, after which the square of the difference between the two functions is calculated (Table 5.10).

Table 5.10 – The results of calculating the squared differences between two functions

i	0	1	2	3	4
x	0,000	1,000	2,000	3,000	4,000
$f(x)$	– 1,000	2,000	3,000	– 2,000	5,000
$\varphi(x)$	0,211	2,179	0,274	1,455	3,727
$\varphi(x) - f(x)$	1,212	0,197	– 2,726	3,455	– 1,273
$(\varphi(x) - f(x))^2$	1,468	0,032	7,430	11,934	1,621

From Table 5.12, the value of the sum of squared errors:

$$E = \sum_{i=0}^{4} \left[\varphi(x_i) - f(x_i)\right]^2 = 1,468 + 0,032 + 7,430 + 11,934 + 1,621 = 22,485.$$

Then the value of the residual root mean square error:

$$\Delta = \sqrt{\frac{E}{n+1}} = \sqrt{\frac{22,485}{5+1}} = 1,936.$$

From a practical point of view, the resulting root mean square error Δ=1,936 of the approximation of the function $f(x)$ is quite large. In order to reduce the approximation error, it is necessary to first select the optimal type of $\varphi(x)$ function

5.3 Statistical data processing

When processing the results of experimental data, there is a need to evaluate the characteristics of a random variable.

The arithmetic mean of the results of N independent tests is used to estimate $\overline{X}$ the unknown value of the mathematical expectation m_X of the random variable X:

$$\overline{\mathrm{X}} = \frac{\sum_{i=1}^{n} x_i}{N}, \tag{5.25}$$

and to estimate the value of the dispersion D_x for a sufficiently large number of experimental data ($N \geq 30$) – the ratio:

$$D_x = \sigma_X^2 = \frac{\sum_{i=1}^{N} (x_i - \overline{x})^2}{N-1}. \tag{5.26}$$

Consider the implementation of finding the mathematical expectation and variance in the C++ programming language:

```
#include <cmath>
#include <iostream>
using namespace std;

double sum(int size, double sample[]){
    double sum = 0;
    for (int i = 0; i < size; i++){
        sum+=sample[i];
    }
    return sum;
}

double mExp(int size, double sample[]){
    return sum(size, sample)/size;
}

double variance(int size, double sample[]){
    double mean = mExp(size, sample);
    double diffsSquared[size];
    for (int i=0; i<size;i++) {
        diffsSquared[i] = pow(sample[i]-mean,2);
    }

    return sum(size,diffsSquared)/(size-1);
}

int main(int argc, char *argv[]) {
    int size = argc-2;
    double sample [size];

    for (int i = 0; i < size; i++){
        sample[i]=stod(*(argv+2+i));
    }
    string mode(*(argv+1));
    cout << "Mode: " << mode << endl;
    if ("expectation" == mode) {
        cout << "Expectation: " << mExp(size,sample) << endl;
    } else if ("variance" == mode) {
        cout << "Variance: " << variance(size,sample) << endl;

    } else {
        cout << "Unknown mode" << endl;
    }
    return 0;
}.
```

If the normal distribution law of the quantity X is considered, then it can be shown that the quantity:

$$T = \frac{\overline{X} - m_X}{\sigma_X / \sqrt{N}},$$

has a Student's t-distribution with $k=N-1$ degrees of freedom. The degree of freedom in statistics is defined as the difference between the number of experiments and the number of model coefficients that can be calculated based on

the results of these experiments independently of each other. For example, in a normal distribution, there are two parameters, and Poisson distribution has one. From here, you can determine the confidence interval for the true value of x: from the known values of the confidence probability P, namely from Table 5.11, the value of ε is determined, from where $\Delta = \varepsilon \frac{D_X}{\sqrt{N}}$.

Thus, if a random variable is normally distributed with expectation m_X and variance D_X, then the true value of x lies in the interval (m_X–Δ, m_X+Δ) with confidence probability P.

To evaluate the type of distribution law under the confidence distribution law, the Kolmogorov–Smirnov and Pearson's chi-squared test are most widely used (Student's t-test is used only under the normal distribution law), which allow, based on the comparison of the empirical distribution function $f_X^*(x)$, obtained in the form of a histogram as a result of processing experimental data with hypothetical $f_X(x)$, corresponding to the proposed hypothesis, to conclude about their coincidence at the level of significance α, which is defined as the probability that a reliable hypothesis will be rejected.

Table 5.11 – Values for the interval $-\varepsilon<t<\varepsilon$, where the value t has the Student's distribution depending on the reliable probability P and the number of degrees of freedom k

k	*P*=0,90	*P*=0,95	*P*=0,99
1	*2*	*3*	*4*
1	6,310	12,71	63,7
2	2,920	4,30	9,92
3	2,350	3,18	5,84
4	2,130	2,77	4,60
5	2,020	2,57	4,03
6	1,943	2,45	3,71
7	1,895	2,36	3,50
8	1,860	2,31	3,36
9	1,833	2,26	3,25
10	1,812	2,23	3,17
11	1,796	2,20	3,11
12	1,782	2,18	3,06
13	1,771	2,16	3,01
14	1,761	2,14	1,98
15	1,753	2,13	2,95
16	1,746	2,12	2,92
17	1,740	2,11	2,90
18	1,734	2,10	2,86

Continuation of Table 5.11

1	*2*	*3*	*4*
19	1,729	2,09	2,86
20	1,725	2,08	2,84
22	1,717	2,07	2,82
24	1,711	2,06	2,80
26	1,706	2,06	2,78
28	1,701	2,05	2,76
30	1,697	2,04	2,75
40	1,684	2,02	2,70
60	1,671	2,00	2,66
120	1,658	1,98	2,62
240	1,645	1,96	2,58

In the Kolmogorov–Smirnov test, the measure is the magnitude:

$$\lambda = \left| f_X(x) - f_X^*(x) \right|_{\max} \sqrt{n},$$

which is compared with the critical value given in Table 5.12.

Table 5.12 – Critical values of λ_0 depending on the level of significance

α	0,500	0,400	0,300	0,200	0,100	0,050	0,020	0,001	0,001
λ_0	0,828	0,895	0,974	1,073	1,224	1,358	1,520	1,627	1,950

Under the condition $\lambda < \lambda_{cr}$, the hypothesis of the coincidence of $f_X(x)$ and $f_X^*(x)$ is accepted.

In the Pearson's chi-squared test, the value is calculated:

$$x^2 = \sum_{i=0}^{k} \frac{\left[f_X(x_i) - f_X^*(x_i) \right]^2}{f_X(x_i)}, \qquad (5.27)$$

where k – the number of histogram digits (discrete $f_X(x_i)$ values).

From Table 5.13, the critical value of χ^2 is determined, taking into account the value of α and the number of degrees of freedom:

$$r = k - l - 1,$$

where l – the number of parameters contained in the distribution law (for normal l=2, Poisson's l=1, etc.).

For $\chi^2 < \chi^2_{cr}$, the hypothesis is accepted.

If we compare analytically obtained probability distribution laws, the mean squared error is a measure of their closeness.

To assess the interdependence of random variables that have a stochastic relationship, the correlation coefficient is used:

$$r_{xy} = \frac{1}{n-1} \frac{\sum_{i=1}^{n} (x_i - m_x)(y_i - m_Y)}{\sigma_x \sigma_y}, \quad (5.28)$$

where n – sample size.

When determining the interdependence of the values of random variables at different points in time, the correlation coefficient is estimated by the formula:

$$r_X(\tau) = \frac{1}{n-m-1} \frac{\sum_{i=1}^{n-m} [x(t_i) - m_x][x(t_i + \tau) - m_X]}{D_X}, \quad (5.29)$$

where $x(t_i)$ – the value of the random variable X at the moment of time t_i, and $x(t_i+\tau)$ – at the moment of time that differs from t_i at the time interval τ. Thus, $x(t_i) = x_i$, $x(t_i+\tau) = x_j$, τ is the time interval between i and j values of x ($i–j = m$).

Table 5.13 – Critical points of the distribution x is a random variable that is distributed according to the χ^2 law with k degrees of freedom

Number of degrees of freedom k	α=0,010	α=0; 0,025	α=0,050	α=0,950	α=0,975	α=0,990
1	6,6	6,0	3,8	0,0039	0,00098	0,00016
2	9,2	7,4	6,0	0,103	0,051	0,020
3	11,3	9,4	7,8	0,352	0,216	0,115
4	13,3	11,1	9,5	0,711	0,484	0,297
5	15,1	12,8	11,1	1,15	0,831	0,554
6	16,8	14,4	12,6	1,64	1,24	0,872
7	18,5	16,0	14,1	2,17	1,69	1,24
8	20,1	17,5	15,5	2,73	2,18	1,65
9	21,7	19,0	16,9	3,33	2,70	2,09
10	23,2	20,5	18,3	3,94	3,25	2,56
11	24,7	21,9	19,7	4,57	3,82	3,05
12	26,2	23,3	21,0	5,23	4,40	3,57
13	27,7	24,7	22,4	5,89	5,01	4,11
14	29,1	26,1	23,7	6,57	5,63	4,66
15	30,6	27,5	25,0	7,26	6,26	5,23
16	32,0	28,8	26,3	7,96	6,91	5,81

The correlation interval is defined as the time interval during which the correlation function decreases by 95%.

Determining the correlation coefficient (normalized correlation function) and correlation function based on known data arrays x and y on the basis of the above formulas does not cause difficulties, and the approximation of the form of the

correlation function by typical correlation functions (Table 5.14) can be carried out by the method of least squares.

Table 5.14 – Typical correlation functions

Type	Parameters
$R_x(\tau)=\sigma_X^2(1-\alpha\lvert\tau\rvert)$, $\tau<1/\alpha$	$\alpha=(\sigma_X^2-R_x(\tau^*))/\sigma_X^2\tau^*$, $R_x(\tau^*)$ – known value of correlation function
$R_x(\tau)=\sigma_X^2 e^{-\alpha\lvert\tau\rvert}$	$\alpha=\frac{1}{\tau^*}ln\frac{\sigma_X^2}{R_x(\tau^*)}$
$R_x(\tau)=\sigma_X^2 e^{-\alpha^2\tau^2}$	$\alpha=\frac{1}{\tau^*}\sqrt{ln\frac{\sigma_x^2}{R_x(\tau^*)}}$
$R_x(\tau)=\sigma_x^2 e^{-\alpha\lvert\tau\rvert}(1+\alpha\lvert\tau\rvert)$	$\alpha\approx 4{,}5/\tau_k^{max}$
$R_x(\tau)=\sigma_x^2 e^{-\alpha\lvert\tau\rvert}cos\beta\tau$	$\alpha=\frac{ln}{\tau_2^*}*\frac{\sigma_x^2 cos\frac{\pi\tau_2^*}{2\tau_1^*}}{R_x(\tau_2^*)}$, $\beta=\pi/2\tau_1^*$ for two known values of the correlation function $R_x(\tau)$, and $R_x(\tau_1^*)=0$.

Conclusions regarding the application of data processing problem solving methods

Data processing tasks combine a number of questions that arise for researchers during the processing of experimental data or in the process of tests with the objects under study. Problems of approximating unknown functions are generally approximation problems. If obtaining a function of the given form is the final result, then the process of solving the problem itself is determined by the choice of the criterion of proximity of the approximated and approximating functions. Most often, this method is the method of least squares, where the measure of closeness is the sum of the squares of the deviations of these functions, but there are also other formulations of the problem. In general, the criterion of closeness is chosen by the researcher himself. When the approximation is used to estimate the value of an unknown function at a certain point, the interpolation problem is considered, which is solved by the known methods of Lagrange, difference, splines, etc. The choice of an effective algorithm depends on the answers to the following questions: how accurate is the chosen method, how much machine time is spent on its use, how smooth is the interpolation function, how many data points does it require, etc. The errors of interpolation methods depend on whether all the points obtained from the experiment are used. The Lagrangian polynomial explicitly contains the function values at the interpolation nodes, so it

is useful when the function values change and the interpolation nodes are fixed. The effectiveness of the application of different interpolation methods depends on the location of the point at which the function value is being searched. If it is located at the beginning of the interval, it is more convenient to use Newton's first interpolation formula; at the end, the second formula is more suitable; and in the middle, formulas based on central differences are preferred. Splines are effectively used when multiple interpolation calculations are required and in the case of a large fixed database. They are also effective in the case of multidimensional interpolation and are widely used (spatial interpolation based on Bezier curves) for the reconstruction and processing of spatial images in computer graphics. Extrapolation is the procedure of interpolating beyond the specified interval. During statistical data processing, it is necessary to evaluate the main characteristics of a random variable. These procedures are included in the set of classic problems of processing experimental data in combination with algorithms and methods for their solution, which must be in the arsenal of an engineer and a researcher.

Control questions and tasks

1. Formulate the problem of interpolation. In which cases is interpolation impossible?
2. What restrictions are placed on the set of base points when solving an interpolation problem?
3. Compile an algorithm and program for solving the problem of interpolation using the Lagrangian interpolation.
4. Compile an algorithm and program for solving the problem of interpolation using Newton’s first interpolation formula.
5. Compile an algorithm and program for solving the problem of interpolation using Newton’s second interpolation formula.
6. Compile an algorithm and program for solving the interpolation problem using interpolation formulas with central differences.
7. Derive Newton’s interpolation formula for equidistant nodes.
8. What is spline interpolation? How are spline coefficients determined?
9. What is extrapolation?
10. What is the more generalized concept – extrapolation or interpolation?
11. What is an approximation? What is the difference between approximation and interpolation?
12. In what ways is the approximation problem solved?
13. Give the main formula for the method of least squares.
14. Derive the system of equations for determining the coefficients of the approximating polynomial in the least squares method.

15. What are polynomials called orthogonal? Please provide examples.

16. Construct the *n*-th degree Lagrangian interpolation polynomial for the function $y(x)$ on the given interval, and calculate its value at the given point. Divide the interval into 10 and 20 points, and compare the errors:

a) $y = \dfrac{2x}{3x^2 + 4x + 1}$ on the interval $x \in [2; 4]$, x=3,0;

b) $y = \dfrac{1}{3\sqrt{x} + 2}$ on the interval $x \in [1; 3]$, x=2,0;

c) $y = \dfrac{1}{2x^2 + 3x + 5}$ on the interval $\epsilon[2; 3]$, x=2,2;

d) $y = \dfrac{x}{\lg(2 + x)}$ on the interval $x \in [4; 5]$, x=4,3.

17. Solve the problem of interpolation using the Lagrange's method for the function given in the table and calculate the value of the function at the point x=2,2:

a)

i	0	1	2	3
x_i	2	4	5	8
y_i	10	15	9	25

b)

i	0	1	2	3
x_i	1	3	5	8
y_i	11	5	9	12

c)

i	0	1	2	3
x_i	– 5	– 3	0	3
y_i	– 24	– 12	4	22

18. The calibration table below for a thermocouple gives the voltmeter readings when the temperature changes with a constant step. Using Lagrangian interpolation polynomial, find the voltmeter readings at T=55 ^{0}C.

i	0	1	2	3	4	5
T, ^{0}C	0	20	40	60	80	100
U, mV	– 0,670	– 0,254	0,171	0,609	1,057	1,517

19. Construct a diagonal table of finite differences for the given function using Newton's first interpolation formula and calculate the interpolation value at the given point by dividing the interval into 10 and 20 points. Compare the calculation errors:

a) $y(x)=\dfrac{1}{2x^3+3x^2+1}$ on the interval $x\in[4; 5]$, x=4,2;

b) $y(x)=\dfrac{1}{\sqrt[3]{3x^2+5}}$ on the interval $x\in[6; 7]$, x=6,4;

c) $y(x)=\dfrac{1}{\sqrt{x^3+\sqrt{3x}}}$ on the interval $x\in[1; 3]$, x=1,2.

20. Solve the interpolation problem using Newton's first interpolation formula and calculate the value of the function at the point x=1,5.

a)

i	0	1	2	3
x_i	0	2	4	6
y_i	– 2	15	7	24

b)

i	0	1	2	3
x_i	– 2	0	2	4
y_i	10	5	7	12

c)

i	0	1	2	3
x_i	– 5	0	5	10
y_i	– 24	– 12	4	22

21. Construct a diagonal table of finite differences for the given function using Newton's second interpolation formula and calculate the interpolation value at the given point by dividing the interval into 10 and 20 points, compare the calculation errors:

a) $y(x)=\dfrac{1}{x^4+3x^3+x}$ on the interval $x\in[1; 3]$, x=1,5;

b) $y(x)=\dfrac{1}{5x\sqrt{x+x^5}}$ on the interval $x\in[3; 5]$, x=4,0;

c) $y(x)=\dfrac{1}{4x^3+5x+\sqrt{x}}$ on the interval $x\in[6; 8]$, x=7,6.

22. Solve the interpolation problem using Newton's second interpolation formula and calculate the value of the function at the point x=6,2.

a)

i	0	1	2	3
x_i	2	4	6	8
y_i	10	– 2	9	5

b)

i	0	1	2	3
x_i	1	3	5	7
y_i	11	2	9	10

c)

i	0	1	2	3
x_i	– 4	0	4	8
y_i	– 24	– 12	4	22

23. Construct a diagonal table of finite differences for the given function using Stirling's formula and calculate the interpolation value at the given point by dividing the interval into 10 and 20 points, compare the calculation errors:

a) $y(x) = \dfrac{1}{2x^3 + 3x^2 + 1}$ on the interval $x \in [4; 6]$, x=5,0;

b) $y = \sin(x) + x^2 - x^3$ on the interval $x \in [6; 7]$, x=6,8;

c) $y(x) = \dfrac{1}{\sqrt{x^3 + \sqrt{3x}}}$ on the interval $x \in [1; 3]$, x=1,2.

24. Solve the interpolation problem using Stirling's formula and calculate the value of the function at the point x=2,2:

a)

i	0	1	2	3
x_i	0	2	4	6
y_i	– 2	15	7	24

b)

i	0	1	2	3
x_i	– 3	0	3	6
y_i	10	5	7	12

c)

i	0	1	2	3
x_i	– 4	0	4	8
y_i	– 15	– 12	4	10

25. Construct third-order splines for the function $y_i(x_i)$ given in the table:

a)

i	0	1	2	3
x_i	2	4	5	8
y_i	10	15	9	25

b)

i	0	1	2	3
x_i	1	3	5	8
y_i	11	5	9	12

c)

i	0	1	2	3
x_i	– 5	– 3	0	3
y_i	– 24	– 12	4	22

26. Using the method of least squares, approximate the given tabular data with a function of the following form:

a) $\varphi(x) = C_1 + C_2 x$;

b) $\varphi(x) = C_1 + C_2 x + C_3 x^2$;

c) $\varphi(x) = C_1 + C_2 x + C_3 \sin(3x)$.

Determine the corresponding coefficients C_1, C_2, C_3 and the residual error.

i	0	1	2	3	4
x	– 4,0	1,0	3,0	5,0	12,0
$f(x)$	2,4	– 5,0	1,2	7,0	4,0

27. Develop an algorithm and compile a program for evaluating the statistical characteristics of the results of measuring the random variable X.

28. Write an algorithm and a program for estimating the correlation coefficient of two random variables.

29. Write an algorithm and program to construct a histogram of a random variable, and also construct a histogram for a random variable obtained from a standard random number generator.

30. Compose an algorithm and program to estimate the autocorrelation coefficient of a random variable and approximate the resulting function with one of the typical correlation functions from Table 5.14.

Chapter 6. NUMERICAL INTEGRATION AND DIFFERENTIATION

In many tasks related to the development, analysis, identification and quality assessment of various methods and means of mathematical modeling, as well as information technologies, there is a need to calculate certain integrals.

The function $F(x)$ on a given interval D is called the original function for the function $f(x)$ or the integral of $f(x)$, provided that $f(x)$ is the derivative of the function $F(x)$ throughout this interval, or the same, that $f(x)dx$ serves as a differential for $F(x)$:

$$F'(x) = f(x) \text{ or } dF(x) = f(x)dx.$$

If the function $f(x)$ is continuous on the interval $[a; b]$ and its original function $F(x)$ is known, then the definite integral from a to b can be calculated using the fundamental theorem of calculus:

$$I = \int_a^b f(x)dx = F(b) - F(a). \quad (6.1)$$

The graphic interpretation of the integral is the area of the curved trapezoid bounded by the curve $y = f(x)$, two ordinates $x_1 = a$ and $x_2 = b$ and the line segment x.

Very often, calculating the value of an integral is not only a difficult process (due to the complexity of analytical transformations), but also impossible altogether (due to the presence of improper integrals), especially when the integral function is given by a set of numerical data (experimental data).

Therefore, the task of numerical integration (numerical integration) of the function consists in calculating the value of the definite integral based on a number of values of the integral function (replacing the original integral function with a certain approximating function). Numerical integration formulas are often called quadrature.

The most famous methods of finding definite integrals are:

- rectangle formulas;
- Newton-Cotes, Gaussian quadrature, Chebyshev polynomials formulas, which are based on the use of so-called quadrature formulas obtained by replacing $f(x)$ with interpolation polynomials;
- Monte Carlo methods, which are based on the use of statistical models.

6.1 Riemann sum

Let it be necessary to determine the value of the integral of the function $f(x)$ on the segment $[a; b]$. The idea of the Riemann sum is to divide the segment of integration $[a; b]$ into elementary segments $[x_{i-1}; x_i]$ by points $a = x_0 < x_1 < \ldots < x_n = b$, based on which rectangles with height $f(\xi_i)$ are

constructed. With a uniform division of the segment $x_i = a+i \cdot h$ (h is a step), therefore $h = \frac{(a-b)}{n}$.

The value of the integral of the function $f(x)$ is approximately expressed as the sum of the areas of the constructed rectangles. The generalized quadrature formula for rectangles has the form:

$$I = \int_a^b f(x)dx \approx \sum_{i=1}^{n} f(\xi_i)(x_i - x_{i-1}), \tag{6.2}$$

where the point $\xi_i \in [x_i; x_{i-1}]$.

Depending on the selection of the position of the ξ_i point, the formulas of the left, right and middle rectangles are distinguished

For $\xi_i = x_{i-1}$, the formula for left rectangles with first-order accuracy is $O(h)$:

– for non-equidistant nodes

$$I \approx \sum_{i=1}^{n} f(\xi_i)(x_i - x_{i-1}); \tag{6.3}$$

– for equidistant nodes

$$I \approx h\sum_{i=1}^{n} f(x_{i-1}). \tag{6.4}$$

The geometric interpretation is given in Figure 6.1, a).

For $\xi_I = x_i$ the formula of right rectangular quadrants with first order accuracy is $O(h)$:

– for non-equidistant nodes

$$I \approx \sum_{i=1}^{n} f(x_i)(x_i - x_{i-1}); \tag{6.5}$$

– for equidistant nodes

$$I \approx h\sum_{i=1}^{n} f(x_i). \tag{6.6}$$

The geometric interpretation is given in Figure 6.1, b).

In the case of $\xi_i = \frac{x_{i-1} + x_i}{2}$, the formula for averaging rectangles with a different order of accuracy is $O(h^2)$:

– for non-equidistant nodes

$$I \approx \sum_{i=1}^{n} f(\frac{x_i + x_{i-1}}{2})(x_i - x_{i-1}); \tag{6.7}$$

– for equidistant nodes

$$I \approx h\sum_{i=1}^{n} f\left(x_{i-1} + \frac{h}{2}\right). \tag{6.8}$$

The geometric interpretation is given in Figure 6.1, c).

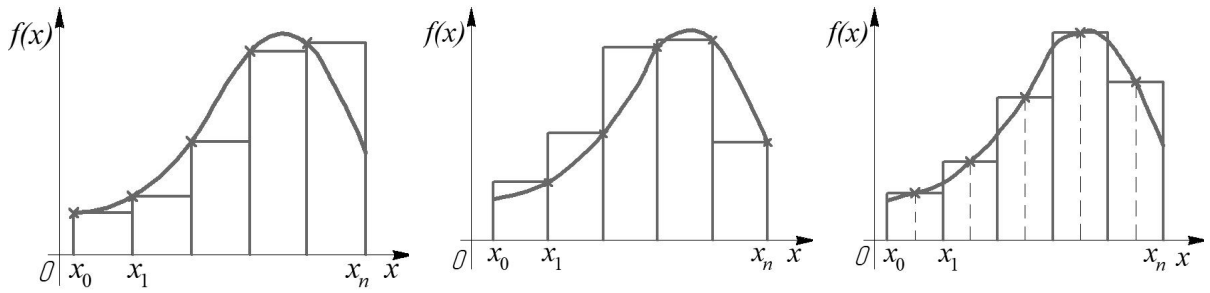


Figure 6.1 – Scheme of numerical integration by the Riemann sum:
a) left; b) right; c) average

Formulas for left and right rectilinear equations can be used for both analytically defined functions and functions defined in tables. The method of average rectilinear equations (Riemann sum) can only be used to find integrals from analytically defined functions.

Example 6.1 Determine the value of the integral $I = \int_0^1 \frac{dx}{1+x^2}$ by the method of left, right, and middle rectangles (Riemann sum) to calculate with step size h=0,2.

Solution:

According to the formula for left-handed rectangles (6.4), we obtain:

$$I_{lr} = h\sum_{i=1}^{5} f(x_{i-1}) = h\cdot(f(0)+f(0,2)+f(0,4)+f(0,6)+f(0,8)) =$$

$$= 0,2\left(\frac{1}{1,0+0^2}+\frac{1}{1,0+0,2^2}+\frac{1}{1,0+0,4^2}+\frac{1}{1,0+0,6^2}+\frac{1}{1,0+0,8^2}\right) = 0,833732.$$

According to the formula for right-handed rectangles (6.6), we obtain:

$$I_{rr} = h\sum_{i=1}^{5} f(x_i) = h\cdot(f(0,2)+f(0,4)+f(0,6)+f(0,8)+f(1,0)) =$$

$$= 0,2\left(\frac{1}{1,0+0,2^2}+\frac{1}{1,0+0,4^2}+\frac{1}{1,0+0,6^2}+\frac{1}{1,0+0,8^2}+\frac{1}{1,0+1,0^2}\right) = 0,733732.$$

According to the formula for average rectilinear curves (6.8), we obtain:

$$I_{mr} = h\sum_{i=1}^{5} f\left(x_{i-1}+\frac{h}{2}\right) = h\left[\begin{array}{l} f\left(0+\frac{0,2}{2}\right)+f\left(0,2+\frac{0,2}{2}\right)+f\left(0,4+\frac{0,2}{2}\right)+ \\ +f\left(0,6+\frac{0,2}{2}\right)+f\left(0,8+\frac{0,2}{2}\right) \end{array}\right] =$$

$$= 0,2\left(\frac{1}{1,0+0,1^2}+\frac{1}{1,0+0,3^2}+\frac{1}{1,0+0,5^2}+\frac{1}{1,0+0,7^2}+\frac{1}{1,0+0,9^2}\right)=0,786826.$$

The exact value of the integral based on analytical decoupling:

$$I=\int_0^1\frac{dx}{1+x^2}=arctg(x)\Big|_0^1=\frac{\pi}{4}=0,785398.$$

The relative difference in calculation by means of equalization between the values of the analytical decoupling of the integral and the numerical method according to the formula:

– left-handed rectangular

$$\varepsilon_л=\left|\frac{I-I_{lr}}{I}\right|\cdot 100\%=\left|\frac{0,785398-0,833732}{0,785398}\right|\cdot 100\%=6,15\%,$$

– right-winged rectangular

$$\varepsilon_n=\left|\frac{I-I_{rr}}{I}\right|\cdot 100\%=\left|\frac{0,785398-0,733732}{0,785398}\right|\cdot 100\%=6,58\%,$$

– medium-sized rectangular

$$\varepsilon_c=\left|\frac{I-I_{mr}}{I}\right|\cdot 100\%=\left|\frac{0,785398-0,786826}{0,785398}\right|\cdot 100\%=0,18\%.$$

The method of left and right triangles (Riemann sum) for a given alignment introduces a fundamental calculation error. The most precise result is achieved by using the method of average rectilinearity. As the number of intervals increases (as the values of *h* decrease), the accuracy of the integral calculation will also increase.

6.2 Newton-Cotes formulas

To derive the Newton-Cotes formulas, the integral is written in the form:

$$\int_a^b f(x)dx=\sum_{i=0}^{n}A_i f(x_i)+\Delta, \qquad (6.9)$$

where x_i – interpolation nodes;

A – coefficients that depend on the type of formula;

Δ – error of the quadrature formula.

By replacing the integrand function in equation (6.9) with the corresponding Lagrange polynomial for n equidistant nodes with a step $h=\frac{b-a}{n}$, one can obtain

the following formula for calculating the A_i coefficients for an arbitrary number of nodes:

$$A_i = \left(\frac{b-a}{n}\right)\frac{(-1)^{n-1}}{i!\,(n-1)!}\int_0^m \frac{q(q-1)\ldots(q-n)}{(q-i)}dq \quad (i=0, 1, 2, \ldots, n), \qquad (6.10)$$

where $q = \dfrac{x-a}{h}$ – cast variable.

Usually, the coefficients $H_i = \dfrac{A_i}{b-a}$ are called Cotes' coefficients. Formula (6.9) will take the form:

$$\int_a^b f(x)dx = (b-a)\sum_{i=0}^{n} H_i f(x_i), \qquad (6.11)$$

and has the following properties $\sum_{i=0}^{n} H_i = 1$ i $H_i = H_{n-i}$.

For n = 1 and n = 2, from (6.10) and (6.11) we obtain the trapezoidal and Simpson's rules. Table 6.1 shows the values of the Cotes' coefficients for n = 1, 2, …, 8. Since the Cotes coefficients for a large number of ordinates are complex, in practice, for approximating definite integrals, the integration interval is divided into a large number of small intervals and Newton's quadrature formula is applied to each of them – Cotes with a small number of ordinates. After that, formulas of simpler structure will be obtained, which can have sufficiently high accuracy.

Table 6.1 – Values of the coefficients of the Newton-Cotes formula

$\hat{H}_i = H_i$	$\hat{H}_0$	$\hat{H}_1$	$\hat{H}_2$	$\hat{H}_3$	$\hat{H}_4$	$\hat{H}_5$	$\hat{H}_6$	$\hat{H}_7$	$\hat{H}_8$	Common denominator N
1	1	1								2
2	1	4	1							6
3	1	3	3	1						8
4	7	32	12	32	7					90
5	19	75	50	50	75	19				288
6	41	216	27	272	27	216	41			840
7	751	3577	1223	2989	2989	1223	3577	751		17280
8	989	5888	–928	10496	–4540	10496	–928	5888	989	28350

For example, the trapezoidal and Simpson's rules obtained in this way have the following form:

$$I = \frac{h}{2}\left[f(x_0) + 2f(x_1) + 2f(x_2) + ... + 2f(x_{n-1}) + f(x_n) \right], \qquad (6.12)$$

$$I = \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + ... + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n) \right]. \quad (6.13)$$

Moreover, the errors of the component formulas are, respectively:

$$\Delta = -n\frac{h^3}{12}M_2 \text{ та } \Delta = -n\frac{h^5}{180}M_4.$$

Similarly, it is possible to obtain Newton-Cotes component formulas for higher orders.

To estimate the calculation error in practice, the Runge's method (Richardson extrapolation) is used, similar to the use of one-step numerical methods for solving the Cauchy problem (see Chapter 3).

Example 6.2. Determine the value of the integral $I = \int_0^1 \frac{dx}{1+x^2}$ using the Newton-Cotes formulas, namely the trapezoid formula with a step of h = 0,2 and Simpson's rule with a step of h = 0,25.

Solution:

Since the value of the integral function is $f(x) = \frac{1}{1+x^2}$, then for h = 0.2, the interval $x\epsilon[0; 11,0]$ will be divided into five segments. Using the trapezoid formula (6.13), we have:

$$I_t = \frac{h}{2}\left[f(x_0) + 2f(x_1) + 2f(x_2) + 2f(x_3) + 2f(x_4) + f(x_5) \right] =$$

$$= \frac{h}{2}\left[f(0) + 2 \cdot f(0,2) + 2 \cdot f(0,4) + 2 \cdot f(0,6) + 2 \cdot f(0,8) + f(1,0) \right] =$$

$$= \frac{0,2}{2}\left[\frac{1}{1,0+0^2} + 2 \cdot \frac{1}{1,0+0,2^2} + 2 \cdot \frac{1}{1,0+0,4^2} + 2 \cdot \frac{1}{1,0+0,6^2} + 2 \cdot \frac{1}{1,0+0,8^2} + \right.$$

$$\left. + \frac{1}{1,0+1,0^2} \right] = 0,783730.$$

Also, using Simpson's formula (6.13) with a step size of h = 0.25, we obtain:

$$I_s = \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + f(x_4)\right] =$$

$$= \frac{h}{3}\left[f(0) + 4f(0,25) + 2f(0,5) + 4f(0,75) + f(1,0)\right] =$$

$$= \frac{0,2}{3}\left[\frac{1}{1,0+0^2} + 4\cdot\frac{1}{1,0+0,25^2} + 2\cdot\frac{1}{1,0+0,5^2} + 4\cdot\frac{1}{1,0+0,75^2} + \frac{1}{1,0+1,0^2}\right] =$$

$$= 0,785392.$$

The relative error of the calculation can be determined by comparing the values of the analytical solution of the integral (refer to Example 6.1) and the numerical method, using the formula:

– trapezium

$$\varepsilon_t = \left|\frac{I - I_m}{I}\right| \cdot 100\% = \left|\frac{0,785398 - 0,783730}{0,785398}\right| \cdot 100\% = 0,21\%,$$

– Simpson's (parabola)

$$\varepsilon_s = \left|\frac{I - I_c}{I}\right| \cdot 100\% = \left|\frac{0,785398 - 0,785392}{0,785398}\right| \cdot 100\% = 7,63 \cdot 10^{-4}\%.$$

The most accurate result of the solution can be obtained using the Simpson's (parabola) formula, given that the calculation step is much larger than that of the trapezoidal formula for calculating the integral.

6.3 Chebyshev's integral

If you replace $x_i = \frac{a+b}{2} + \frac{b-a}{2}t_i$ in expression (6.9), the corresponding expression will be reduced to the following form:

$$\int_{-1}^{1} f(t)dt = \sum_{i=1}^{n} A_i f(t_i)\ . \qquad (6.14)$$

During the derivation of the Chebyshev's formula, the following conditions are used: the coefficients A_i are equal to each other; the quadrature formula (6.14) has a high degree of accuracy for all polynomials up to and including the n-th power.

Given that $A_1=A_2=\ldots=A_n=A$ and $f(t)=1$, then $\sum_{i=1}^{n} A_i = nA = 2$, whence $A=2/n$.

Under these conditions, formula (6.14) will have the following form:

$$\int_{-1}^{1} f(t)dt = \frac{2}{n}\sum_{i=1}^{n} f(t_i)\,. \qquad (6.15)$$

To determine t_i, the second condition is used, according to which it is required that the formula (6.15) has a high degree of calculation accuracy for the function of the form:

$$f(t) = t^k \ (k=1, 2, \ldots, n). \tag{6.16}$$

After substituting these functions into (6.15), we obtain a system of equations:

$$\begin{cases} t_1 + t_2 + \ldots + t_n = 0; \\ t_1^2 + t_2^2 + \ldots + t_n^2 = \dfrac{n}{3}; \\ \ldots\ldots\ldots\ldots\ldots\ldots; \\ t_1^n + t_2^n + \ldots + t_n^n = \dfrac{n\left[1-(-1)^{n+1}\right]}{2(n+1)}. \end{cases} \tag{6.17}$$

The system of equations (6.17) has a solution for $n < 8$ and $n = 9$, which imposes limitations on the accuracy of the calculation, as a drawback of using Chebyshev's integral. The values of the abscissa t_i in Chebyshev's integral for different values of n are given in Table 6.2.

Table 6.2 – The value of the abscissa t_i in Chebyshev's integral

n	i	t_i	n	i	t_i
2	1; 2	∓0,5773500	6	1; 6	∓0,866247
3	1; 3	∓0,7071070		2; 5	∓0,422519
	2	0,0		3; 4	∓0,266635
4	1; 4	∓0,7946540	7	1; 7	∓0,883862
	2; 3	∓0,1875920		2; 6	∓0,529657
5	1; 5	∓0,8324980		3; 5	∓0,323912
	2; 4	∓0,3745413		4	0,0
	3	0,0			

For an arbitrary interval [a; b], formula (6.15) takes the form:

$$I = \frac{b-a}{n}\sum_{i=1}^{n} f(x_i), \tag{6.18}$$

where $x_i = \dfrac{a+b}{2} + \dfrac{b-a}{2}t_i$.

Calculation error by the Chebyshev's integral:

$$\Delta = \int_a^b \frac{\left(x - \dfrac{a+b}{2}\right)^{n+1}}{(n+1)!} f^{(n+1)}(x)dx = \frac{b-a}{n(n+1)!}\sum_{i=1}^{n}\left(x_i - \frac{a+b}{2}\right)^{n+1} f^{n+1}(x). \tag{6.19}$$

Example 6.3. Determine the value of the integral $I=\int_0^1 \frac{dx}{1+x^2}$ using the Chebyshev's integral (the order of the method is $n = 3$).

Solution:

For order $n = 3$, we obtain the abscissa value from Table 6.2:

$t_1 = -0{,}707107$; $t_2 = 0$; $t_3 = 0{,}707107$.

If we substitute the obtained abscissa values into formula 6.19, we get:

$$I_{ch}=\frac{b-a}{n}\sum_{i=1}^{n} f(x_i)=\frac{b-a}{n}\left[f(x_1)+f(x_2)+f(x_3)\right]=\frac{1{,}0-0}{3}\left[0{,}9790+0{,}80+\right.$$

$$\left.+0{,}57852=0.78584,\right.$$

where $x_1=\frac{a+b}{2}+\frac{b-a}{2}\cdot t_1=\frac{0+1{,}0}{2}+\frac{1{,}0-0}{2}\cdot(-0{,}707107)=0{,}14645$;

$$x_2=\frac{a+b}{2}+\frac{b-a}{2}\cdot t_2=\frac{0+1{,}0}{2}+\frac{1{,}0-0}{2}\cdot 0=0{,}5;$$

$$x_3=\frac{a+b}{2}+\frac{b-a}{2}\cdot t_3=\frac{0+1{,}0}{2}+\frac{1{,}0-0}{2}\times 0{,}707107=0{,}85355;$$

$$f(x_1)=\frac{1}{1+x_1^2}=\frac{1}{1{,}0+0{,}14645^2}=0{,}9790;\ f(x_2)=\frac{1}{1+x_2^2}=\frac{1}{1+0{,}5^2}=0{,}8;$$

$$f(x_3)=\frac{1}{1+x_3^2}=\frac{1}{1+0{,}85355^2}=0{,}57852.$$

Example 6.4. Determine the value of the integral $I=\int_0^1 \frac{dx}{1+x^2}$ using the Simpson's rule with a calculation step of $h = 0{,}5$ (the order of Chebyshev's integral method is $n = 3$).

Solution:

For the calculation step $h = 0.5$, the interval $x\epsilon[0; 1{,}0]$ is divided into two equal intervals, for which $I = I_1 + I_2$, respectively.

The values of the integrals I_1 and I_2 for each of the intervals are determined by Chebyshev's integral. Therefore, from Table 6.2, we obtain the values of the abscissa: $t_1 = -0{,}707107$; $t_2 = 0$; $t_3 = 0{,}707107$.

The value of the integral I_1 for the interval [0; 0,5]:

$$I_1=\frac{b-a}{n}\sum_{i=1}^{n} f(x_i)=\frac{b-a}{n}\left[f(x_1)+f(x_2)+f(x_3)\right]=\frac{0{,}5-0}{3}\left[0{,}99467+0{,}94118+\right.$$

$$\left.+0{,}84592\right]=0.46363,$$

where $x_1 = \frac{a+b}{2} + \frac{b-a}{2} \cdot t_1 = \frac{0+0,5}{2} + \frac{0,5-0}{2} \cdot (-0,707107) = 0,07322$;

$x_2 = \frac{a+b}{2} + \frac{b-a}{2} \cdot t_2 = \frac{0+0,5}{2} + \frac{0,5-0}{2} \cdot 0 = 0,250$;

$x_3 = \frac{a+b}{2} + \frac{b-a}{2} \cdot t_3 = \frac{0+0,5}{2} + \frac{0,5-0}{2} \cdot (0,707107) = 0,42678$;

$f(x_1) = \frac{1}{1+x_1^2} = \frac{1}{1+0,07322^2} = 0,99467$;

$f(x_2) = \frac{1}{1+x_2^2} = \frac{1}{1+0,25^2} = 0,94118$; $f(x_3) = \frac{1}{1+x_3^2} = \frac{1}{1+0,42678^2} = 0,84592$.

The value of the integral I_2 for the interval [0,5; 1,0]:

$I_2 = \frac{b-a}{n} \sum_{i=1}^{n} f(x_i) = \frac{b-a}{n} \left[f(x_4) + f(x_5) + f(x_6) \right] = \frac{1,0-0,5}{3} [0,75268 + 0,64 +$

$+0,53795] = 0.32177$,

where $x_4 = \frac{a+b}{2} + \frac{b-a}{2} \cdot t_1 = \frac{0,5+1}{2} + \frac{1-0,5}{2} \cdot (-0,707107) = 0,57322$;

$x_5 = \frac{a+b}{2} + \frac{b-a}{2} \cdot t_2 = \frac{0,5+1}{2} + \frac{1-0,5}{2} \cdot 0 = 0,750$;

$x_6 = \frac{a+b}{2} + \frac{b-a}{2} \cdot t_3 = \frac{0+0,5}{2} + \frac{0,5-0}{2} \cdot (0,707107) = 0,92678$;

$f(x_4) = \frac{1}{1+x_4^2} = \frac{1}{1+0,57322^2} = 0,75268$;

$f(x_5) = \frac{1}{1+x_5^2} = \frac{1}{1+0,75^2} = 0,640$; $f(x_6) = \frac{1}{1+x_6^2} = \frac{1}{1+0,92678^2} = 0,53795$.

Then the full value of the integral:

$$I_{ch} = \int_0^1 \frac{dx}{1+x^2} = I_1 + I_2 = 0,46363 + 0,32177 = 0,78540.$$

The relative error of the calculation can be obtained by comparing the values of the analytical solution of the integral (see Example 6.1) with the Chebyshev's integral:

– for the whole interval (see Example 6.3)

$$\varepsilon_{wi} = \left| \frac{I - I_{ch}}{I} \right| \cdot 100\% = \left| \frac{0,785398 - 0,785840}{0,785398} \right| \cdot 100\% = 0,056\ \%,$$

– by the method of curved trapezoids (see Example 6.4)

$$\varepsilon_{ct} = \left| \frac{I - I_{ch}}{I} \right| \cdot 100\% = \left| \frac{0,785398 - 0,785400}{0,785398} \right| \cdot 100\% = 2,55 \cdot 10^{-4}\ \%.$$

The use of the Chebyshev's integral, compared to the trapezoidal and Simpson's rule, is the most productive, as it allows you to consider the entire integration interval in general, while providing higher calculation accuracy. In order to further increase the accuracy of the calculation, based on the Chebyshev's integral, it is advisable to reduce the calculation intervals within the general integration interval.

6.4 Gaussian quadrature

The Gaussian quadrature is known as the formula of the highest algebraic accuracy. For a formula of the form (6.14), the highest accuracy can be achieved for polynomials of degree (2*n*–1), which are defined by 2*n* constants t_i and A_i (i = 1, 2, …, n).

To ensure this condition, it is necessary and sufficient that it is fulfilled for functions of the type:

$$f(t) = t^k \ (k = 0, 1, \ldots, 2n-1).$$

Given that the function $f(t)$ can be approximated by polynomials of degree (2*n*–1), namely $f(t) = \sum_{k=0}^{2n-1} c_k t^k$, it is possible to obtain:

$$\int_{-1}^{1} f(t)dt = \sum_{k=0}^{2n-1} C_k \int_{-1}^{1} t_k dt = \sum_{k=0}^{2n-1} C_k \sum_{i=1}^{n} A_i t_i^k = \sum_{i=1}^{n} A_i \sum_{k=0}^{2n-1} C_k t_i^k A_i f(t_i).$$

The task is to determine the coefficients A_i and abscissa points t_i. To determine these constants, it is necessary to consider the implementation of formula (6.14) for functions of the form $f(t) = t^k$ (k = 0, 1, …, 2*n*–1).

Given that:

$$\int_{-1}^{1} t^k dt = \begin{Bmatrix} 2/(k+1) \\ 0 \end{Bmatrix},$$

we get a system of equations:

$$\begin{cases} \sum_{i=1}^{n} A_i = 2; \\ \sum_{i=1}^{n} A_i t_i = 0; \\ \sum_{i=1}^{n} A_i t_i^2 = 1; \\ \sum_{i=1}^{n} A_i t_i^{2n-2} = \dfrac{2}{2n-1}; \\ \sum_{i=1}^{n} A_i t_i^{2n-1} = 0. \end{cases} \qquad (6.20)$$

The system of equations (6.20) is nonlinear, its solution is associated with significant computational difficulties. But if you use the system for polynomials of the form:

$$f(t) = t^k P_n(t) \ \ (k = 0, 1, \ldots, n-1), \tag{6.21}$$

where $P_n(t)$ – the Legendre polynomials can then be reduced to a linear system with respect to the coefficients A_i, using the given points t_i.

Legendre polynomials are called polynomials of the form

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} \left[(x^2 - 1) \right],$$

which have the following basic properties:

1) $P_n(1) = 1$, $P_n(-1) = (-1)^n$ for any integer n;

2) orthogonality

$$\int_{-1}^{1} P_n(x) Q_k(x) dx = 0, \tag{6.22}$$

where $Q_k(x)$ – any polynomial of degree $k < n$;

3) presence of n real roots on the interval [–1; 1].

Since the powers of the polynomials in the ratio (6.21) do not exceed the value $2n$–1, the system (6.20) must be fulfilled, and the formula (6.14) takes the following form:

$$\int_{-1}^{1} t^k P_n(t) dt = \sum_{i=1}^{n} A_i t_i^k P_n(t_i). \tag{6.23}$$

As a result of the orthogonality property (6.22), the left part of the expression (6.23) is equal to zero. Then:

$$\sum_{i=1}^{n} A_i t_i^k P_n(t_i) = 0, \tag{6.24}$$

which is always ensured for any values of A_i at the points t_i, corresponds to the roots of the corresponding Legendre polynomials.

By substituting these t_i values into the system (6.20) and considering the first n equations, it is possible to determine the coefficients A_i.

The formula (6.14), where t_i are the zeros of the Legendre polynomials $P_n(t)$, and A_i (i = 1, 2, …, n) are determined from the system (6.20), is called the Gaussian quadrature.

Table 6.3 shows the values of t_i and A_i in the Gaussian quadrature for different values of n ranging from 1 to 8.

For an arbitrary interval [a; b] the formula for the Gaussian quadrature takes the form:

$$I = \frac{b-a}{2} \sum_{i=1}^{n} A_i f(x_i), \tag{6.25}$$

where $x_i = \frac{a+b}{2} + \frac{b-a}{2} t_i$.

The estimate of the error of the Gaussian quadrature with n nodes is determined from the ratio:

$$\Delta \le \frac{(b-a)^{2n+1}(n!)^4 M_{2n}}{\left[(2n)!\right]^3 (2n+1)}, \tag{6.26}$$

where M_{2n} – the maximum value of the 2nd derivative on the interval [a; b].

Table 6.3 – Elements of the Gaussian quadrature

n	i	t_i	A_i
1	1	0,0	2,0
2	1; 2	∓0,57735027	1,0
3	1; 3	∓0,77459667	$\frac{5}{9}$=0,55555556
	2	0,0	$\frac{8}{9}$=0,88888889
4	1; 4	∓0,86113631	0,34785484
	2; 3	∓0,33998104	0,65214516
5	1; 5	∓0,90617985	0,23692689
	2; 4	∓0,53846931	0,47862867
	3	0,0	0,56888889
6	1; 6	∓0,93246951	0,17132450
	2; 5	∓0,66120939	0,36076158
	3; 4	∓0,238619119	0,46791394
7	1; 7	∓0,94910791	0,12948496
	2; 6	∓0,74153119	0,27970540
	3; 5	∓0,40584515	0,38183006
	4	0,0	0,41795918
8	1; 8	∓0,96028986	0,10122854
	2; 7	∓0,79666648	0,22238104
	3; 6	∓0,52553142	0,31370664
	4; 5	∓0,18343464	0,36268378

Example 6.5. Determine the value of the integral $I = \int_0^1 (1+x^2)dx$ using the Gaussian quadrature (the order of the method is n = 3).

Solution:

For order n=3, we choose the abscissa value from Table 6.3: $t_1 = -0{,}77459667$; $A_1 = 0{,}55555556$; $t_2 = 0$; $A_2 = 0{,}88888889$; $t_3 = 0{,}77459667$; $A_2 = 0{,}55555556$.

If we substitute the obtained abscissa values into formula (6.25), we will have:

$$I_1 = \frac{b-a}{2}(A_1 f(x_1) + A_2 f(x_2) + A_3 f(x_3)) = \frac{1{,}0-0}{2}(1{,}012702 \cdot 0{,}55555556 +$$
$$+1{,}250 \cdot 0{,}88888889 + 1{,}787298 \cdot 0{,}55555556) = 1.333333,$$

where $x_1 = \frac{a+b}{2} + \frac{b-a}{2} \cdot t_1 = \frac{0+1{,}0}{2} + \frac{1{,}0-0}{2} \cdot (-0{,}77459667) = 0{,}112702$;

$x_2 = \frac{a+b}{2} + \frac{b-a}{2} \cdot t_2 = \frac{0+1{,}0}{2} + \frac{1{,}0-0}{2} \cdot 0 = 0{,}50$;

$x_3 = \frac{a+b}{2} + \frac{b-a}{2} \cdot t_3 = \frac{0+1{,}0}{2} + \frac{1{,}0-0}{2} \cdot (0{,}77459667) = 0{,}887298$;

$f(x_1) = 1 + x_1^2 = 1 + 0{,}112702^2 = 1{,}012702$; $f(x_2) = 1 + x_2^2 = 1 + 0{,}5^2 = 1{,}250$;

$f(x_3) = 1 + x_3^2 = 1 + 0{,}887298^2 = 1{,}787298$.

The exact value of the integral is based on the analytical solution:

$$I = \int_0^1 (1 + x^2)dx = \frac{1}{3}x(x^2 + 3)\Big|_0^1 = \frac{4}{3} = 1{,}333333. \qquad (6.27)$$

Example 6.6. Determine the value of the integral $I = \int_0^1 (1 + x^2)dx$ using the trapezoidal rule with a calculation step of h = 0,5 (the order of the Gaussian quadrature is n = 3).

Solution:

For the calculation step h = 0.5, the interval $x \in [0; 1]$ is divided into two equal intervals, resulting in $I = I_1 + I_2$, respectively.

The values of the integrals I_1 and I_2 for each of the intervals are determined by Gaussian quadrature. Therefore, from Table 6.3, we obtain the values of the abscissa:
$t_1 = -0{,}77459667$; $A_1 = 0{,}55555556$; $t_2 = 0$; $A_2 = 0{,}88888889$; $t_3 = 0{,}77459667$; $A_2 = 0{,}55555556$.

The value of the integral I_1 for the interval [0; 0,5]:

$$I_1 = \frac{b-a}{2}\left[A_1 f(x_1) + A_2 f(x_2) + A_3 f(x_3)\right] = \frac{0{,}5-0}{2}(1{,}003175 \cdot 0{,}55555556 +$$
$$+1{,}0625 \cdot 0{,}88888889 + 0{,}196825 \cdot 0{,}55555556) = 0.541667,$$

where $x_1 = \frac{a+b}{2} + \frac{b-a}{2} \cdot t_1 = \frac{0+0,5}{2} + \frac{0,5-0}{2} \cdot (-0,77459667) = 0,056351;$

$x_2 = \frac{a+b}{2} + \frac{b-a}{2} \cdot t_2 = \frac{0+0,5}{2} + \frac{0,5-0}{2} \cdot 0 = 0,250;\ x_3 = \frac{a+b}{2} + \frac{b-a}{2} \cdot t_3 =$

$= \frac{0+0,5}{2} + \frac{0,5-0}{2} \cdot (0,77459667) = 0,443649;\ f(x_1) = 1 + x_1^2 = 1 + 0,056351^2 =$

$= 1,003175;\ f(x_2) = 1 + x_2^2 = 1 + 0,25^2 = 1,06250;\ f(x_3) = 1 + x_3^2 = 1 +$

$+0,443649^2 = 1,196825.$

The value of the integral I_2 for the interval [0.5; 1.0]:

$$I_2 = \frac{b-a}{2}\left(A_1 f(x_4) + A_2 f(x_5) + A_3 f(x_6)\right) = \frac{0,5-0}{2}(1,309526 \cdot 0,55555556 +$$

$$+1,5625 \cdot 0,88888889 + 1,890474 \cdot 0,55555556) = 0.791667,$$

where $x_1 = \frac{a+b}{2} + \frac{b-a}{2} \cdot t_1 = \frac{0,5+1,0}{2} + \frac{1,0-0,5}{2} \cdot (-0,77459667) = 0,556351;$

$x_2 = \frac{a+b}{2} + \frac{b-a}{2} \cdot t_2 = \frac{0,5+1,0}{2} + \frac{1,0-0,5}{2} \cdot 0 = 0,750;\ x_3 = \frac{a+b}{2} + \frac{b-a}{2} \cdot t_3 =$

$= \frac{0,5+1,0}{2} + \frac{1,0-0,5}{2} \cdot (0,77459667) = 0,943649;\ f(x_4) = 1 + x_4^2 = 1 +$

$+0,556351^2 = 1,309526;\ f(x_5) = 1 + x_5^2 = 1 + 0,750^2 = 1,5625;\ f(x_6) = 1 + x_6^2 =$

$= 1 + 0,943649^2 = 1,890474.$

Then, the full value of the integral:

$$I = \int_0^1 (1 + x^2)dx = I_1 + I_2 = 0,541677 + 0,791667 = 1,333333.$$

Comparing the results of the numerical and analytical solutions, one can note the high accuracy of the calculation of the integral value.

Example 6.7. Determine the value of the integral $I = \int_0^1 \frac{dx}{1+x^2}$ using the Gaussian quadrature method (with an order of n = 5).

Solution:

Let's replace the variable $x = \frac{a+b}{2} + \frac{b-a}{2}\xi = \frac{1}{2} + \frac{1}{2}\xi$, де a=0 і b=1,0. Also $dx = \frac{d\left[\frac{1}{2} + \frac{1}{2}\xi\right]}{d\xi} = \frac{1}{2}$. Based on the formula (6.25), the new value of the

integration interval is for $x\in[x_1;\ x_2]=[0;\ 1]$: $\xi_1=2t_1-1=2{,}0\cdot 0-1{,}0=-1{,}0$; $\xi_2=2t_2-1=2\cdot 1{,}0-1{,}0=1{,}0$.

Also, the function $f(x)=\dfrac{1}{1+x^2}$ will look like this after substitution:

$$\varphi(\xi)=\frac{4}{4+(\xi+1)^2}.$$

After substituting and replacing all components, the new integral expression will look like this:

$$I=2\int_{-1}^{1}\frac{d\xi}{4+(\xi+1)^2}.$$

According to the Gauss quadrature (6.25):

$$I=2\int_{-1}^{1}\varphi(\xi)d\xi=\frac{\xi_2-\xi_1}{2}\sum_{i=1}^{n}A_i f(\varepsilon_i)=2\left(\frac{\xi_2-\xi_1}{2}\right)\Big[A_1\varphi(\varepsilon_1)+A_2\varphi(\varepsilon_2)+A_3\varphi(\varepsilon_3)+$$

$$+A_4\varphi(\varepsilon_4)+A_5\varphi(\varepsilon_5)\Big]=2\left(\frac{1{,}0-(-1{,}0)}{2}\right)\Big[0{,}236926885\cdot\varphi(-0{,}9061179846)+$$

$$+0{,}478628670\cdot f(-0{,}538469310)+0{,}568888889\cdot f(0)+0{,}478628670\times$$

$$\times f(0{,}538469310)+0{,}236926885\cdot f(0{,}906179846)\Big]=0{,}78539816,$$

where $\varepsilon_i=\dfrac{\xi_1+\xi_2}{2}+\dfrac{\xi_2-\xi_1}{2}t_i=\dfrac{1{,}0+(-1{,}0)}{2}+\dfrac{1{,}0-(-1{,}0)}{2}t_i=t_i$ and, accordingly, the values of the parameters A_i and $f(t_i)$ are given in Table 6.4.

Table 6.4 – Results of calculating the integral of the function

i	t_i	$f(t_i)$	A_i
1	– 0,9061179846	0,24945107	0,236926885
2	– 0,538469310	0,23735995	0,478628670
3	0,0	0,20000000	0,568888889
4	0,538469310	0,15706211	0,478628670
5	0,906179846	0,13100114	0,236926885

Comparing the results of the numerical and analytical (see Example 6.1) solutions, it is possible to observe the high accuracy of the calculation of the value of the integral.

6.5 Algorithms for the application of numerical methods

The sequence of application of Newton-Cotes formulas:

1. Selection of the formula and definition (see Table 6.1) for the coefficients H_i.
2. Algorithm and program compilation, moreover:

– in the case of setting discrete values $y_i = f(x_i)$ through step h, these values are substituted into the selected formula, for example, in (6.12) or (6.13);

– in the case of setting the function $y = f(x)$, the value $y_i = f(x_i)$ is determined, and $x_i = x_0 + ih = a + ih$ ($a \le x \le b$).

3. Error estimation.

The algorithm for solving the integral using the trapezoidal rule is presented in Figure 6.5.

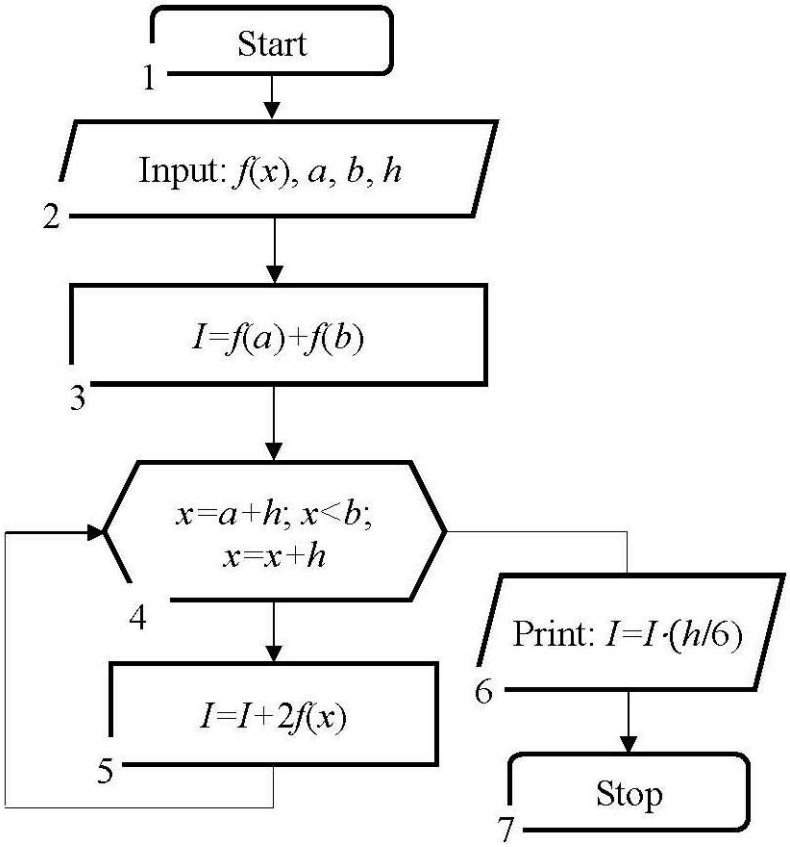


Figure 6.5 – Scheme of the algorithm for solving integrals using the trapezoidal rule

Consider the implementation of the trapezoidal rule in the C++ programming language:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

FILE *fp;
const int n_h = 2;

float f(float x)
{   return (sin(x)/(x*x + 1)); }

float Metod_Trapets (float a, float b, float h)
{
    float I, x = a + h;
     fprintf (fp, «\n--- Metod trapets - h = %.3f ---\n», h);
```

```
     I = f(a) + f(b);
     for (x; x<b; x+=h)
       I = I + 2*f(x);

   I = h*I/2;
   fprintf (fp, «I = %9.7f\n», I);
   return 0;
}

int main()

{
     float a = 0, b = 1;
     float h[n_h] = {0.1, 0.05};
    if ( (fp = fopen(«Data.txt», «w»)) == NULL)
    {
        printf(«Error opening file\n»);
        return 0;
    }
     for (int i=0; i<n_h; i++)
     {
       Metod_Trapets (a, b, h[i]);
     }
     printf(«End!»);
     getch();
     fclose(fp);
     return 0;
}.
```

When solving a problem using Simpson's rule for an odd number of intervals, it is suggested to additionally split each interval in half. The algorithm of the Simpson's rule for an odd number of intervals is presented in Figure 6.6.

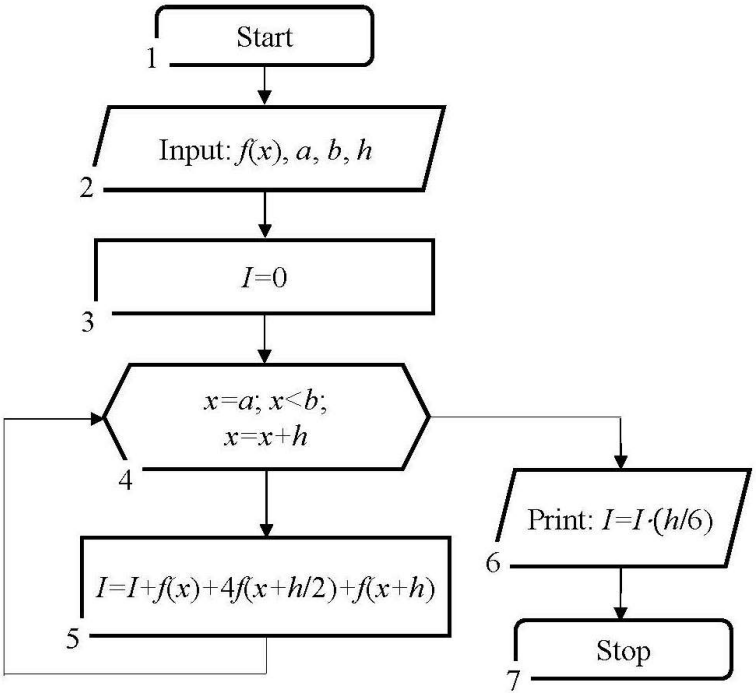


Figure 6.6 – Scheme of the algorithm for solving integrals using the Simpson's rule for an odd number of intervals

The sequence of application of the Gaussian quadrature:

1. Selection of the order of the method and determination (see Table 6.3) of coefficients A_i and values of t_i ($-1 \le t \le 1$).

2. Division of the interval $a \le x \le b$ into l segments (Fig. 6.7).

3. Determination of integral values for each interval ($j = 1, 2, \ldots, l$)

$$I = \sum_{j=1}^{l} I_j .$$

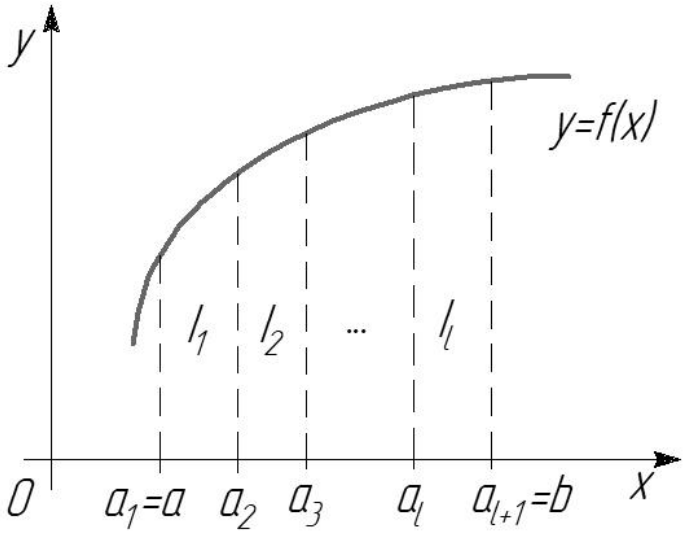


Figure 6.7 – The diagram illustrating the division of the integration interval into segments in the Gaussian quadrature

In this case, the values of the abscissa x_i within each interval j are determined by the formula:

$$x_i = \frac{a_{j+1} + a_i}{2} + \frac{a_{j+1} - a_j}{2} t_i , \qquad (6.28)$$

where $a_{j+1} = a_j + h$, moreover $a_1 = a$, $a_{l+1} = b$, $h = \frac{b-a}{n}$ ($j = 1, 2, \ldots, l$).

The values of the integral I_j are determined by the formula:

$$I_j = \int_{a_j}^{a_{j+1}} f(x)dx = \frac{(a_{j+1} - a_j)}{2} \sum_{i=1}^{n} A_i f(x_i) . \qquad (6.29)$$

The algorithm of the Gaussian quadrature is presented in Figure 6.8, b).

Consider the implementation of the Gaussian quadrature function in the C++ programming language:

```
const int n_G = 4; // method order for the Gaussian method
float t_G[n_G]= {-0.86113631, -0.33998104, 0.33998104, 0.86113631}; //
    coefficients t [i] for the Gaussian method n=4
float A_G[n_G]= {0.34785484, 0.65214516, 0.65214516, 0.34785484}; //
    coefficients A[i] for the Gaussian quadrature n=4
```

```
float Method_Gaussa (float a, float b)
{
     float I;
     fprintf (fp, "\n--- Metod Gaussa_simple \n");
     float F = 0;
     for (int i=0; i<n_G; i++)
     {
       float x = (a+b)/2 + (b-a)*t_G[i]/2;
       F = F + A_G[i]*f(x);
     }
     I =  (b-a)*F/2;
    fprintf (fp, "I = %9.7f\n", I);
    return 0;
}
```

Accordingly, you need to add to the main function:

```
Method_Gaussa (a, b);
```

In the Chebyshev's integral, the sequence of actions is similar to the Gaussian quadrature, but in step 1, the coefficients t_i are taken from Table 6.2, and in step 3, the formula is used to determine the jth integral:

$$I_j = \int_{a_j}^{a_{j+1}} f(x)dx = \frac{(a_{j+1} - a_j)}{n} \sum_{i=1}^{n} f(x_i), \qquad (6.30)$$

where x_i is estimated in a similar way as in the Gaussian quadrature, according to formula (6.25).

The algorithm of the Chebyshev's integral is presented in Figure 6.8, a).

Let's consider the implementation of the Chebyshev's integral function in the C++ programming language:

```
const int n = 4; // order of the method for the Chebyshev method
float t[n]= {-0.794654, -0.187592, 0.187592, 0.794654};

float Method_Chebisheva (float a, float b)
{
    float I;
     fprintf (fp, "\n--- Metod Chebisheva_simple \n");
     float F = 0;
     for (int i=0; i<n; i++)
     {
       float x = (a+b)/2 + (b-a)*t[i]/2;
       F = F + f(x);
     }
     I =  (b-a)*F/n;
    fprintf (fp, "I = %9.7f\n", I);
    return 0;
}
```

To the main function, respectively, should be added:

```
Method_Chebisheva (a, b);
```

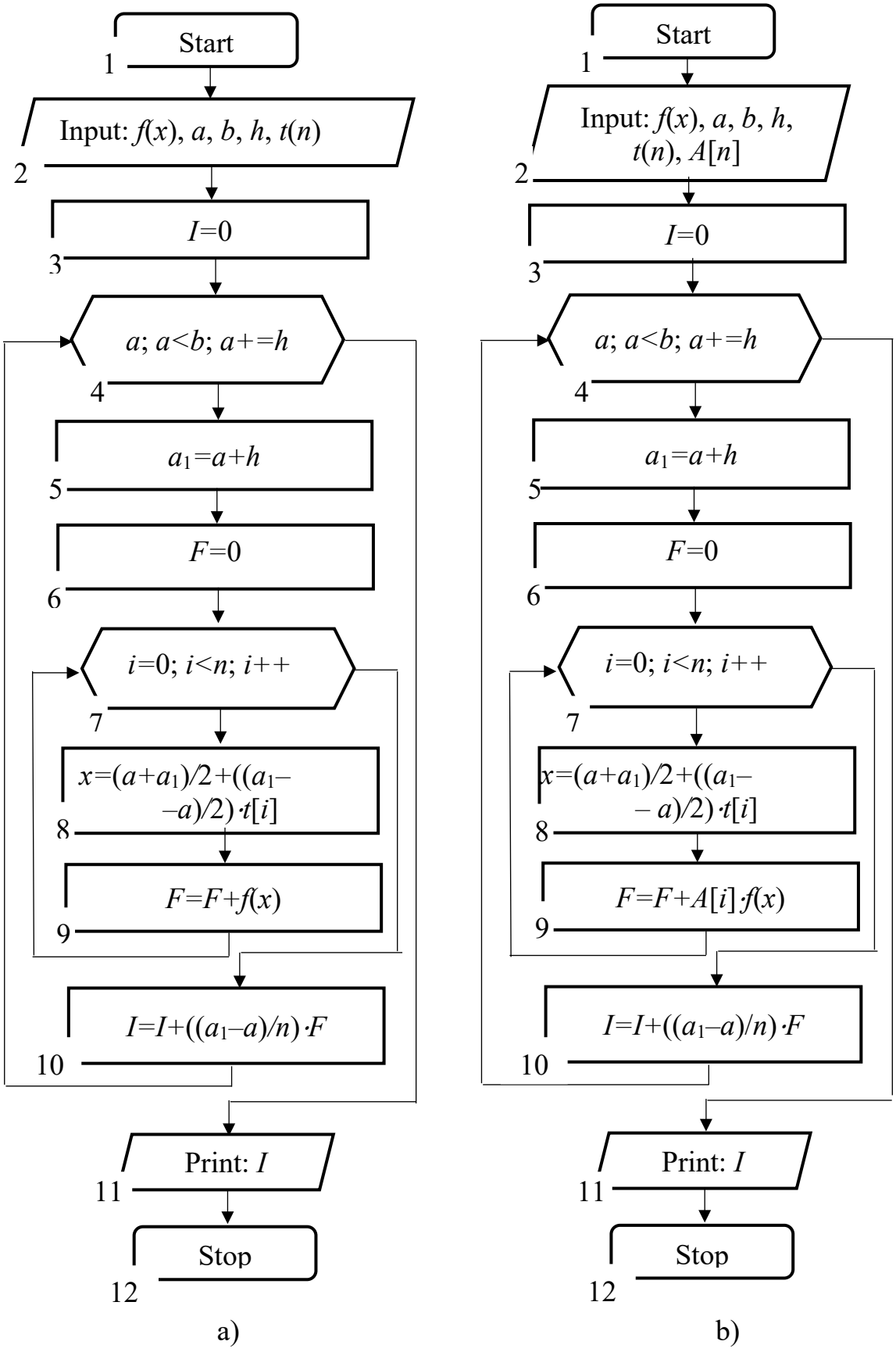


Figure 6.8 – Schemes of algorithms for solving problems by:

a) Chebyshev's integral; b) Gaussian quadrature

6.6 Monte Carlo integration

The Monte Carlo integration is the most famous application of statistical modeling for solving applied mathematical problems.

If the sequence of random numbers $\{x_i\} \in X$ with the probability distribution law $f_x(x)$ is subjected to the functional transformation $y_i = \phi(x_i)$, then the mathematical expectation of the sequence of random numbers $\{y_i\} \in Y$:

$$m_y = \int_{-\infty}^{\infty} \varphi(x) f_x(x) dx \tag{6.31}$$

for a sample size ($n > 1000$) with sufficiently high accuracy ($\Delta_{relative} < 0{,}1$), the estimate can be obtained using the formula:

$$m_y = \frac{1}{n} \sum_{i=1}^{n} y_i \,. \tag{6.32}$$

If expressions (6.31) and (6.32) include the area indicator function [a; b]:

$$\mathbf{1}[a, b, x] = \begin{cases} 1, & a \le x \le b; \\ 0, & x < a, x > b, \end{cases}$$

and choose $\varphi(x) = \dfrac{f(x)}{f_X(x)}$, where $f(x)$ is the integrand function, and the values of a and b are the limits of integration from formula (6.14). Then the final expression will have the form:

$$I = m_y = \int_a^b f(x) dx = \frac{1}{n} \sum_{i=1}^{n} \frac{f(x_i)}{f_x(x_i)} \mathbf{1}[a, b, x_i].$$

The scheme of the numerical integration algorithm using the Monte Carlo integration is presented in Figure 6.9.

The error of the Monte Carlo integration is determined by the error of generating a pseudorandom sequence of numbers generated by a computer system and the size of the sample. It can be estimated from this ratio:

$$\Delta = \frac{1}{2\sqrt{n(1-P)}} \cdot \tag{6.33}$$

where P – guaranteed probability of hitting an error in the [$-\Delta$; $+\Delta$] interval.

The number of trials does not depend on the dimension of the integral I. Therefore, the Monte Carlo integration is advantageous to use for calculating multiple integrals, where the use of other methods of numerical integration is time-consuming. For example, calculating a ten-fold integral in a unit volume with a step of h=0,1 requires calculating the sum of approximately 10^{10} components.

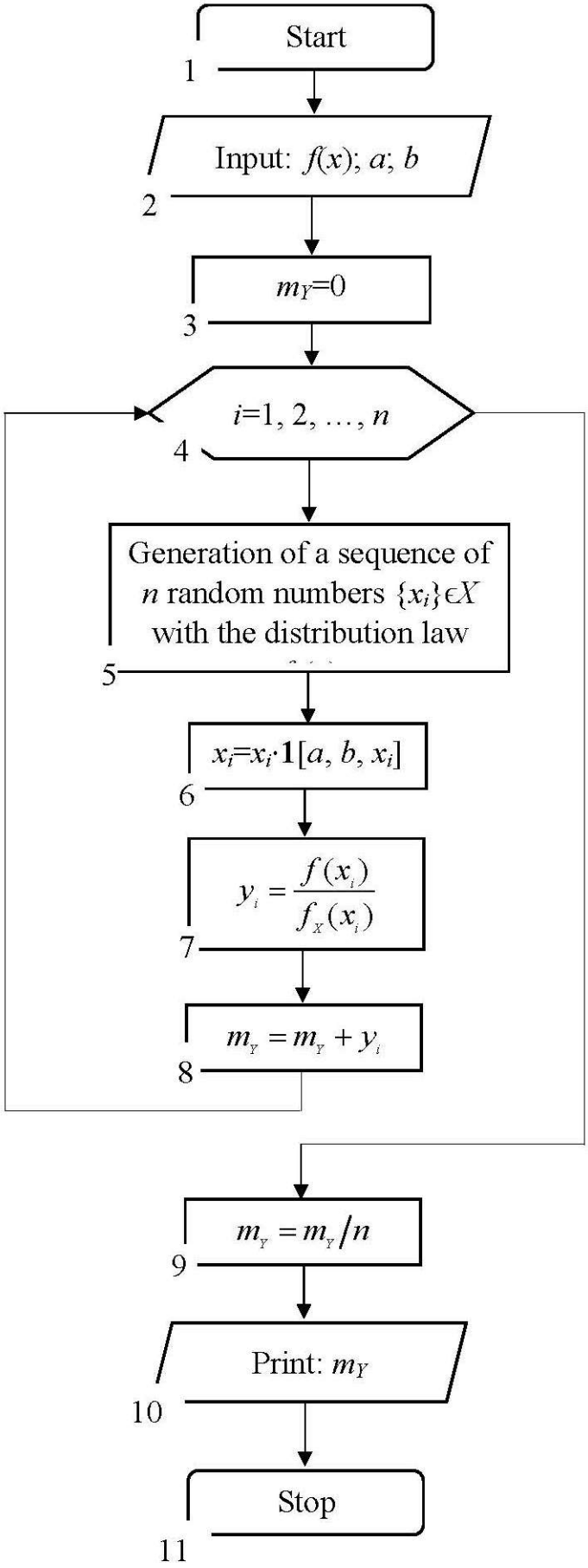


Figure 6.9 – Scheme of the Monte Carlo integration algorithm

6.7 Estimation of the error in numerical integration

In the case when the integrand function is given analytically, the task of determining the integral with a predetermined accuracy can be set. Since the accuracy of the quadrature formulas discussed above depends on the calculation step h, the accuracy of the result can be increased by reducing the step. For example, by dividing the calculation step in half.

To determine the value of the integral with the specified accuracy ε, it is necessary to choose the appropriate quadrature formula and the initial step $h=h_0$. Then, calculate the integral with a step of $h/2$. It is necessary to reduce the calculation step by dividing it in half until the condition is fulfilled:

$$\left|I_{h/2} - I_h\right| < \varepsilon .$$

If this condition is fulfilled, the value of the integral is equal to $I_{h/2}$.

It is also possible to use the Runge method (in order to avoid introducing additional errors into the calculation, it is necessary to choose steps that are close in value in the first and second calculations):

$$c = \frac{I_{h_1} - I_{h_2}}{h_2^{p+1} - h_1^{p+1}},$$

where $I^* = I_{h_1} + ch_1^{p+1} = I_{h_2} + ch_2^{p+1}$.

Also, the errors in calculating integrals using numerical methods can be calculated using the following formulas:

– trapezoidal rule

$$\Delta = \frac{nh^3}{12} M_2,$$

where M_2 – the maximum value of the second derivative of $f(x)$ in the computational domain $x \in [a; b]$;

– Simpson's rule

$$\Delta = \frac{nh^5}{180} M_4,$$

where M_5 – the maximum value of the fourth derivative of $f(x)$ in the computational domain $x \in [a; b]$;

– Chebyshev's integral

$$\Delta = \frac{b-a}{n(n+1)!} \sum_{i=1}^{n} \left(x_i - \frac{a+b}{2} \right)^{n+1} f^{n+1}(x);$$

– Gaussian quadrature

$$\Delta \le \frac{(b-a)^{2n+1}(n!)^4 M_{2n}}{\left[(2n)!\right]^3 (2n+1)};$$

– method of rectangles (left, right)

$$\Delta = \frac{nh^2}{2} M_1,$$

where M_1 – the maximum value of the first derivative $f(x)$ in the computational domain $x \in [a; b]$;

– Riemann sum (averages)

$$\Delta = \frac{nh^3}{24} M_2.$$

6.8 Numerical differentiation

The task is to find the derivative at a point for a given differentiable function $f(x)$, which is given by tabular data or an analytical expression x_0.

6.8.1 Numerical differentiation of analytically given functions

Approximate differentiation of analytically given functions is necessary when developing a universal procedure for finding the derivative for a large number of different functions, or in the case when the analytical form of the derivative function is too cumbersome and leads to a loss of accuracy.

The basis of numerical differentiation of analytically given functions is the definition of the derivative:

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}.$$

Since it is not known what value h to take, it is necessary to construct the sequence $\{h_k\}$ such that $h_k \to 0$ (for example, $h_k = 0{,}5^k$) and, accordingly, the sequence $\{D_k\}$ is formed, where:

$$D_k = \frac{f(x + h_k) - f(x)}{h_k} \quad (k=1, 2, \ldots, n). \tag{6.34}$$

The sequence elements are calculated as long as the condition is met:

$$\left|D_{n+1} - D_n\right| < \left|D_n - D_{n-1}\right|.$$

If the accuracy ε with which it is necessary to determine the derivative is known, then the condition for completing the calculation of the derivative by this method with the order of accuracy h:

$$\left|D_{n+1} - D_n\right| < \varepsilon .$$

Let $f \in C^3[a; b]$, then the order of accuracy of the previous method can be increased by using other expressions instead of formula (6.34).

Formula for the second-order $O(h^2)$ accuracy in calculating the derivative:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}. \tag{6.35}$$

Formula (6.35) can be obtained by expanding the function $f(x)$ into a Taylor series:

$$f(x+h) = f(x) + f'(x) \cdot h + \frac{f^{(2)}(x) \cdot h^2}{2!} + \frac{f^{(3)}(C_1) \cdot h^3}{3!}; \tag{6.36}$$

$$f(x-h) = f(x) - f'(x) \cdot h + \frac{f^{(2)}(x) \cdot h^2}{2!} - \frac{f^{(3)}(C_1) \cdot h^3}{3!}. \tag{6.37}$$

If expression (6.37) is subtracted from equation (6.36), we have:

$$f(x+h) - f(x-h) = 2f'(x) \cdot h + \frac{(f^{(3)}(C_1) + f^{(3)}(C_2)) \cdot h^3}{3!},$$

or

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2). \tag{6.38}$$

Similarly, formulas similar to those can be obtained for higher derivatives of the order of accuracy $O(h^2)$:

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}; \tag{6.39}$$

$$f'''(x) \approx \frac{f(x+2h) - 2f(x+h) + 2f(x-h) - f(x-2h)}{2h^3}; \tag{6.40}$$

$$f^{(4)}(x) \approx \frac{f(x+2h) - 4f(x+h) + 6f(x) - 4f(x-h) + f(x-2h)}{h^4}. \tag{6.41}$$

The derivative calculation algorithm itself remains unchanged, namely: the sequence $\{h_k\}$ is formed so that the value $h_k \to 0$ and the elements of the sequence $\{D_k\}$ are calculated accordingly, for the calculation of which one of the obtained formulas (6.38)–(6.41) is used instead of expression (6.34).

There are also higher-order accuracy formulas that can be found in specialized literature.

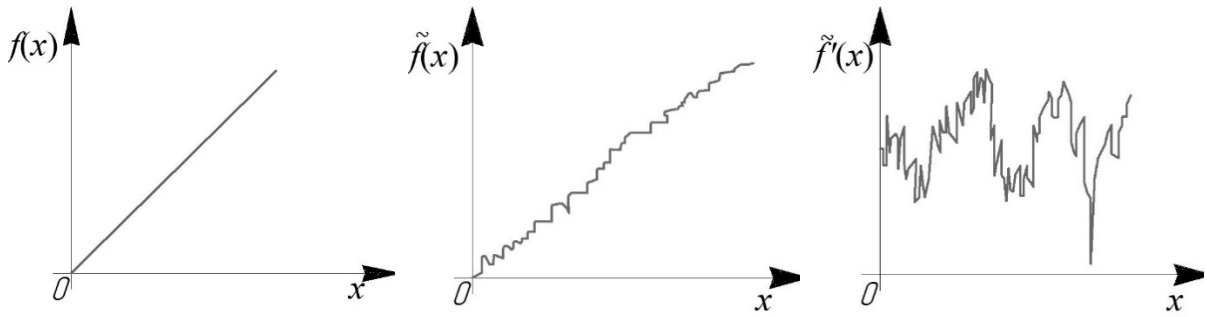
6.8.2 Numerical differentiation of experimental data

During the solution of many practical problems, there is a need to determine the derivatives of functions that are given by arrays of experimentally obtained data. In this case, direct differentiation can give false results due to unseparated «noise» that greatly distorts the value of the derivative. An example is shown in Figure 6.10, where $f(x)$ is the true signal, and $\tilde{f}(x)$ is the measured signal (with «noise»).

It is clear that the value of the derivative of the function $f(x)$ contains a significant random component, which introduces a large error in determining the real derivative. A practical solution to this problem is to use the smoothed signal obtained from the experiment through interpolation or approximation, followed by differentiation of the interpolation polynomial $P(x)$ (or approximation function). Additionally, in order to obtain higher-order derivatives, it is necessary to establish the condition for their convergence to the signal and its interpolation polynomial. The error value of the derivative of the interpolation function $\Delta^*(x)$ is equal to the derivative of the error of this function $\Delta'(x)$:

$$\Delta(x) = \tilde{f}(x) - P(x);\ \Delta^*(x) = \tilde{f}'(x) - P'(x),$$

that is, due to the linearity of the differentiation and subtraction operations, we get $\Delta^*(x) = \Delta'(x)$.



a) b) c)

Figure 6.10 – Diagrams of signal functions:
a) – valid; b) – measured; c) – a derivative

In the handbooks, there are special tables that allow you to determine the value of derivatives using various difference interpolation formulas. Such formulas are easy to obtain for any interpolation method by differentiating the interpolation formulas in the general form.

Example 6.8. Determine the acceleration and speed of the car as the value of the derivative of the displacement function $f(x)$ over time, as provided in tabular data (Table 6.5).

Table 6.5 – Output data

i	0	1	2	3	4	5
x, sec	0	1,0	2,0	3,0	4,0	5,0
y, m	3,0	6,0	9,0	10,0	11,0	12,0

Solution:

For an approximate estimation of the derivatives of the function $f(x)$, the left finite differences are used. Namely, the value of the differentiation step and:

$$h = x_i - x_{i-1} = x_1 - x_0 = 1{,}0 - 0{,}0 = x_2 - x_1 = x_3 - x_2 = x_4 - x_3 = x_5 - x_4 = 1{,}0 \text{ sec}.$$

The speed of the car is the value of the first derivative, as given by the formula (6.34):

$$\begin{cases} y_0' = \dfrac{y_1 - y_0}{h} = \dfrac{6{,}0 - 3{,}0}{1{,}0} = 3{,}0 \text{ m / sec}; \\ y_1' = \dfrac{y_2 - y_1}{h} = \dfrac{9{,}0 - 6{,}0}{1{,}0} = 3{,}0 \text{ m / sec}; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\ y_4' = \dfrac{y_5 - y_4}{h} = \dfrac{12{,}0 - 11{,}0}{1{,}0} = 1{,}0 \text{ m / sec}. \end{cases}$$

The values of the second (car acceleration) and third derivatives are based on formula (6.34):

$$\begin{cases} y_0'' = \dfrac{y_1' - y_0'}{h} = \dfrac{3{,}0 - 3{,}0}{1{,}0} = 0 \text{ m / sec}^2; \\ y_1'' = \dfrac{y_2' - y_1'}{h} = \dfrac{1{,}0 - 3{,}0}{1{,}0} = -2{,}0 \text{ m / sec}^2; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots; \\ y_3'' = \dfrac{y_5' - y_4'}{h} = \dfrac{1{,}0 - 1{,}0}{1{,}0} = 0 \text{ m / sec}^2. \end{cases}$$

$$\begin{cases} y_0''' = \dfrac{y_1'' - y_0''}{h} = \dfrac{(-2{,}0) - 0{,}0}{1{,}0} = -2{,}0; \\ y_1''' = \dfrac{y_2'' - y_1''}{h} = \dfrac{0{,}0 - (-2{,}0)}{1{,}0} = 2{,}0; \\ y_2''' = \dfrac{y_3'' - y_2''}{h} = \dfrac{0{,}0 - 0{,}0}{1{,}0} = 0. \end{cases}$$

The results of the derivative calculation are entered in Table 6.6.

Table 6.6 – Results of calculating the derivatives of the function $f(x)$

i	0	1	2	3	4	5
x, sec	0,0	1,0	2,0	3,0	4,0	5,0
y, m	3,0	6,0	9,0	10,0	11,0	12,0
y', m/sec	3,0	3,0	1,0	1,0	1,0	
y'', m/sec^2	0,0	–2,0	0,0	0,0		
y'''	–2,0	2,0	0,0			

Conclusions on the application of numerical integration and differentiation methods

Numerical integration of the functions given by the array of experimental data is carried out using Newton-Cotes methods. In the case of an analytical integration task, it is possible to use more accurate methods such as Gaussian quadrature and Chebyshev's integral, or the Monte Carlo integration. The Gaussian quadrature and Chebyshev's integral, also known as methods of the highest algebraic accuracy, require the ability to calculate the integral function at any point within the integration interval, which is determined by the order of the method and the integration intervals. This is only possible in the case of an analytical task because any approximation methods introduce additional errors. The Monte Carlo integration, based on the generation and processing of random numbers, utilizes the fact that the mathematical expectation of a random variable is estimated by the average value of a sequence of random numbers. This allows for the construction of an integral approximation algorithm as a sequence of operations using a generated set of random (or pseudo-random) numbers. The advantages of this method are particularly evident when calculating multiple integrals. Due to the complexity of analytical calculations, the estimation of errors in the application of numerical integration methods is often carried out by performing multiple calculations with different step sizes. In the process of numerical differentiation, it is necessary to understand the nature of the data being differentiated. In the presence of significant random errors, preliminary smoothing of the data should be performed.

Control questions and tasks

1. What is the process of integrating a function called? What is the issue with numerical integration of a function?

2. What are the methods of numerical integration? Provide a comparative analysis of them.

3. What is the essence of the rectangular method for numerical integration of functions? What is the peculiarity of using the methods of left, right, and middle rectangles for numerical integration of functions?

4. How does the use of the left, right, and middle rectangle methods affect the accuracy of numerical integration of functions? Define the calculation error using the appropriate numerical methods.

5. How are Cotes coefficients determined? What are the properties of Cotes coefficients when using them to determine the value of the integral of a function?

6. What is the sequence of application of Newton-Cotes methods? Develop appropriate algorithms. How do simple Newton-Cotes formulas differ from complex ones?

7. How are the Newton-Cotes formulas obtained? Derive the simple trapezoidal and Simpson's rules for numerical integration, as well as the components of the formulas.

8. What is the peculiarity of using Simpson's rule for an even and odd number of intervals on the entire integration interval?

9. How are errors estimated using Newton-Cotes methods of numerical integration of functions?

10. Calculate the integral using the appropriate numerical method based on the initial data given in Table 6.7. Compile an algorithm and a calculation program. Estimate the calculation error.

Table 6.7 – Output data for the task

Variant	Integral function (I)	Calculation step (h)	Type of calculation method
1	$\int_{1,0}^{10,0} x^2 \ln x \, dx$	0,20	Trapezoidal rule
2	$\int_{1,0}^{10,0} \frac{x}{\sqrt{1+x^2}} dx$	0,25	Riemann sum
3	$\int_{1,0}^{10,0} \frac{x \sin x}{\ln x} dx$	0,20	Simpson's rule
4	$\int_{1,0}^{10,0} \frac{arctg\, x}{x} dx$	0,25	Riemann sum
5	$\int_{1,0}^{10,0} \sin x \lg x \, dx$	0,50	Trapezoidal rule

Continuation of Table 6.7

Variant	Integral function (I)	Calculation step (h)	Type of calculation method
6	$\int_{1,0}^{10,0} \frac{\sin x}{x} dx$	0,30	Simpson's rule
7	$\int_{1,0}^{10,0} x^3 \ln x dx$	0,50	Trapezoidal rule
8	$\int_{1,0}^{10,0} \frac{x}{\ln x} dx$	0,30	Simpson's rule
9	$\int_{1,0}^{10,0} \cos x \ln x dx$	0,50	Riemann sum
10	$\int_{1,0}^{10,0} \frac{\sqrt{x^3+4}}{x} dx$	0,60	Simpson's rule
11	$\int_{1,0}^{10,0} \frac{\cos x}{x^2} dx$	0,20	Riemann sum
12	$\int_{0}^{1,0} \frac{\cos x}{\ln x} dx$	0,20	Simpson's rule
13	$\int_{1,0}^{10,0} x^4 \lg x dx$	0,25	Trapezoidal rule

11. Derive the Chebyshev's integral for numerical integration of a function given at three points.

12. What is the fundamental drawback of Chebyshev's integral for numerical integration of a function?

13. What are Legendre polynomials and what are their main properties? Obtain the expressions for the first five Legendre polynomials.

14. Derive the Gaussian quadrature for the numerical integration of a function.

15. How are the coefficients in the Gaussian quadrature for numerical integration determined? Why is it called the formula of the highest algebraic accuracy?

16. How are the methods of Gaussian quadrature and Chebyshev's integral used, and what is the fundamental difference between them? Describe the methodology, sequence of actions, and develop an algorithm.

17. Calculate the integral using the appropriate numerical method based on the initial data given in Table 6.8. Compile an algorithm and a calculation program. Estimate the calculation error.

Table 6.8 – Output data for the task

Variant	Integral function (I)	Method order (n)	Type of calculation method
14	$\int_{1,0}^{10,0} \frac{x \ln x}{\sqrt{1+x^2}} dx$	3	Chebyshev's integral
15	$\int_{1,0}^{10,0} \frac{x^2}{\ln x} dx$	5	Gaussian quadrature
16	$\int_{1,0}^{10,0} \frac{\lg x}{x^4} dx$	4	Chebyshev's integral
17	$\int_{1,0}^{10,0} \ln\lvert x \sin x \rvert \, dx$	6	Gaussian quadrature
18	$\int_{1,0}^{10,0} \frac{x^2}{\sqrt{x+1}} dx$	3	Chebyshev's integral
19	$\int_{1,0}^{10,0} \frac{\ln x}{x^2} dx$	5	Gaussian quadrature
20	$\int_{1,0}^{10,0} x \cdot arctg(x) dx$	4	Chebyshev's integral
21	$\int_{1,0}^{10,0} \frac{arcctg\, x}{x+1} dx$	3	Gaussian quadrature
22	$\int_{1,0}^{10,0} \sin 3x \lg(x+1) \, dx$	6	Chebyshev's integral
23	$\int_{1,0}^{10,0} \frac{\sin x}{x} dx$	7	Gaussian quadrature
24	$\int_{1,0}^{10,0} x^x \ln(x+1) dx$	3	Chebyshev's integral
25	$\int_{1,0}^{10,0} \frac{x+1}{\ln x^2} dx$	5	Gaussian quadrature
26	$\int_{1,0}^{10,0} \frac{\cos x}{\ln x} dx$	4	Chebyshev's integral
27	$\int_{1,0}^{10,0} \frac{\sqrt{x^2+1}}{x-2} dx$	6	Gaussian quadrature
28	$\int_{1,0}^{10,0} \frac{\cos 2x}{x^x} dx$	5	Chebyshev's integral

Continuation of Table 6.8

Variant	Integral function (I)	Method order (n)	Type of calculation method
29	$\int_0^{1,0} x\frac{\cos x}{\ln(x+2)}dx$	3	Gaussian quadrature
30	$\int_{1,0}^{10,0} x^2 \lg(x+2)\sin x dx$	3	Chebyshev's integral

18. Give a comparative analysis of Chebyshev's integral, Gaussian quadrature and Newton-Cotes methods.

19. What is the use of the Monte Carlo integration for the numerical integration of functions? How is the calculation error determined? How can the accuracy of the method be increased?

20. Determine the value of the integral $I = \int_0^{10,0} x^2 dx$ using the numerical Monte Carlo integration. Compare the obtained value with the value of the analytical expression for different sample sizes of the sequence of random numbers.

21. Develop a Monte Carlo numerical integration program for the double integral $I = \iint_{(\sigma)} (x^2 + y^2)dxdy,$ where the region of integration σ is determined by such inequalities as: $0,5 \geq x \geq 1,0$ and $0 \geq y \geq 2x - 1$. Will the points with coordinates (0,55; 0,75), (0,25; 0,75), (0,25; 0,25), (0,99; 0,70) fall into this region of integration?

22. How are errors estimated when using the Chebyshev's integral and Gaussian quadrature of numerical integration of functions?

23. How is numerical differentiation of analytically given functions carried out? How can the accuracy of numerical differentiation of a function be increased?

24. How is the error determined during the numerical differentiation of analytically given functions?

25. Differentiate the function $f(x)=x^2\sin(x)$ on the interval $x\in[0; 10,0]$. Estimate the calculation error by comparing the values of the derivative function calculated using analytical and numerical methods.

26. Determine the second and third derivatives of the function $f(x)=x\ln(x)$ on the interval $x\in[0; 5.0]$. Estimate the calculation error by comparing the values of the derivative function calculated using analytical and numerical methods.

27. How is numerical differentiation of functions determined using experimental data carried out?

28. Determine the values of the first and second derivatives of the function $f(x)$, which are given by tabular data (Table 6.9).

Table 6.9 – Output data for the task

i	0	1	2	3	4	5	6	8
x	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8
y	2,5	3,4	5,2	5,8	5,9	6,3	6,8	7,1

30. How is the error of numerical differentiation of an experimentally determined function determined?

SOURCES:

Abramovitz M. Handbook of mathematical functions. With Formulas, Graphs, and Mathematical Tables/ Abramowitz M., Stegun I. A. – National Bureau of Standards, N.Y., 1972. – 1044 p.

Brezinski C.Numerical Analysis: Historical Developments in the 20th Century / C. Brezinski, L. Wuytack, Elsevier, 2001. – 504 p.

Collatz L. Functional Analysis and Numerical Mathematics. / L. Collatz – Academic Press, New York., 1966. – 494 p.

Forsythe G. E. Computer Methods for Mathematical Computations. Englewood Cliffs / G. E. Forsythe, M. A. Malcolm, C. B. Moler – New Jersey 07632. Prentice Hall, Inc., 1977. – 259 p.

Gander W. Solving Problems in Scientific Computing Using Maple and MATLAB / Walter Gander, Jiri Hrebicek, Springer, 2011. – 479 p.

Greenbaum A. Numerical Methods: Design, Analysis, and Computer Implementation of Algorithms / A. Greenbaum, T. P. Chartier, – Princeton University Press, 2012. – 464 p.

Greenspan D. Numerical analysis for applied mathematics, science, and engineering / Donald Greenspan and Vincenzo Casulli. CRC Press, Boca Raton, 2018. – 352p.

Kvyetnyy R. Basics of Modelling and Computational Methods / R. Kvyetnyy. – Вінниця : ВДТУ, 2007. – 147 p.

Kong Q. Python Programming and Numerical Methods: A Guide for Engineers and Scientists/ Kong, Qingkai; Siauw, Timmy; Bayen, Alexandre – Elsevier, 2020. – 480 p.

Mitsotakis D. Computational Mathematics: An introduction to Numerical Analysis and Scientific Computing with Python/ Dimitrios Mitsotakis CRC-Press, 2023. – 530 p.

Judd K.L. Numerical Methods in Economics / Kenneth L. Judd, MIT Press, 1998. – 656 p.

Stoer J. Introduction to Numerical Analysis /Josef Stoer, R. Bulirsch,– Springer Science & Business Media, 2002. – 746 p.

SOURCES IN UKRAINIAN:

Задачин В. М. Чисельні методи: навчальний посібник / В. М. Задачин, І. Г. Конюшенко. – Х. : Вид. ХНЕУ ім. С. Кузнеця, 2014. – 180 с.

Квєтний Р. Н. Методи комп'ютерних обчислень: навчальний посібник / Квєтний Р. Н. – Вінниця : ВНТУ, 2001. – 218 с.

Комп'ютерне моделювання систем та процесів. Методи обчислень: навчальний посібник / під заг. ред. Р. Н. Квєтного – Вінниця : ВНТУ, 2012. – Ч. 1 – 196 с.; Ч. 2 – 230 с.

Ляшенко М. Я. Чисельні методи: підручник / Ляшенко М. Я., Головань М. С. – К. : Либідь, 1996. – 288 с.

Моделювання та оптимізація систем: підручник / Дубовой В. М., Квєтний Р. Н., Михальов О. І., Усов А. В. – Вінниця : ПП «ТД«Едельвейс», 2017 – 804 с.

Самборська О.М. Чисельні методи: навчальний посібник./ Самборська О. М., Шелестовський Б. Г. – Тернопіль: ТНТУ імені Івана Пулюя, 2010. – 164с.

Усов А. В. Чисельні методи та їх реалізація у середовищі SCILAB6 навчальний посібник / Усов А. В., Шпинковський О. А., Шпинковська М. І. – Київ : Освіта України, 2013. − 192 с.

Чабан В. Чисельні методи: навчальний посібник / В. Чабан. – Львів : Вид. Нац. ун-ту "Львівська політехніка", 2001. – 186 с.

Фельдман Л. П. Чисельні методи в інформатиці: підручник / Фельдман Л. П., Петренко А. І., Дмитрієва О. А. – К. : Вид. група BHV, 2006. – 480 с.

Електронне навчальне видання

Квєтний Роман Наумович

Іванчук Ярослав Володимирович

Computational Methods and Algorithms

(Обчислювальні методи та алгоритми)

(англ. мовою)

Підручник

Рукопис оформлено *Я. Іванчук*

Оригінал-макет підготовлено в *редакційно-видавничому відділі ВНТУ*

Підписано до видання 09.09.2024 р.
Гарнітура Times New Roman, Lucida Console.
Зам. № Р2024-155.

Видавець та виготовлювач –
Вінницький національний технічний університет,
Редакційно-видавничий відділ.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95, м. Вінниця, 21021.
press.vntu.edu.ua;
Email: rvv.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009.