West University of Timisoara
Faculty of Mathematics and Computer Science
Department of Computer Science

Liliana Braescu Stefan Balint
Nadia Bonchis Eva Kaslik

NUMERICAL METHODS

Timisoara 2007

Introduction

The substantial reforms of Higher Education systems, in agreement with the Bologna Process, include: three-cycle degree structure and qualification frameworks, quality assurance of higher education, the European credit transfer and accumulation system, mobility of students and staff, the social dimension of the European Higher Education area.
These determined changes in the Romanian Higher Education System and elaboration of new curricula and adequate syllabi.

The text book *Numerical methods* was elaborated in agreement with the requirements, based on the syllabus elaborated by the Department of Computer Science and approved by the Council of the Faculty of Mathematics and Computer Science from the West University of Timisoara.

This text book is written especially for students of the Computer Science Department, covering all the subjects of the syllabus, at the knowledge level of the fifth semester student. The lectures are tailored so that their presentation during the allocated time is made possible.

A distinguished element of this book consists of a large number of exercises, algorithms and their implementation in Borland C. This last characteristic is specific for computer science students.

The authors would like to thank to Prof. Dr. Gheorghe Bocsan for reading this manuscript, for comments and pertinent recommendations.

The authors

Contents

Chapter 1

Systems of Linear Equations

1.1 Gaussian Elimination and Pivoting

Definition 1.1.1. *A **linear system of** n **algebraic equations of** n **unknowns** is a system of the form:*

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases} \tag{1.1.1}$$

in which a_{ij}, b_i are given real numbers, $i = 1 \ldots n, j = 1 \ldots n$, and $x_1, x_2, \ldots, x_n$ are real unknown numbers.

Definition 1.1.2. *A solution of the system (1.1.1) is an ordered system of numbers $x_1, x_2, \ldots, x_n$ which, replaced into the system (1.1.1), verify the equations; this is considered as a vector $x = (x_1, x_2, \ldots, x_n)^T$ from $\mathbb{R}^n$.*

Remark 1.1.1. The system(1.1.1) can be written in the followings matrix form:

$$Ax = b \tag{1.1.2}$$

where: $A = (a_{ij})_{\substack{i=\overline{1,n} \\ j=\overline{1,n}}}$, $b = (b_1, b_2, \ldots, b_n)^T$, $x = (x_1, x_2, \ldots, x_n)^T$.

Theorem 1.1.1. *(Cramer)*
If det(A)$\neq 0$ (i.e. A is non-singular), then the system has a unique solution given by:

$$x = A^{-1}b \tag{1.1.3}$$

Definition 1.1.3. *The system (1.1.1) is called of **Cramer type** if $det(A) \neq 0$.*

Comment 1.1.1. In the case of a Cramer-type system, if n is large then the determination of the solution using formula (1.1.3) is difficult, and it conducts to many errors because the computation of the inverse matrix A^{-1} implies a large number of operations.

Comment 1.1.2. In order to solve system (1.1.1), another method is to obtain an equivalent upper-triangular system $Ux = y$ (all elements situated under main diagonal are null: $u_{ij} = 0$ for $i > j$), from the linear system (1.1.1).

Theorem 1.1.2. *(Gauss Elimination with Back Substitution)*
If A is a $n \times n$ non-singular matrix, then there exists a system $Ux = y$, equivalent to $Ax = b$, where U is an upper-triangular matrix with $u_{kk} \neq 0$. After U and y are constructed, back substitution can be used to solve $Ux = y$ for x.

The Gauss method with pivot for computing a numerical solution of a linear Cramer-type system consists of two stages (the first stage has $n-1$ steps, and the second has n steps), and conducts to an exact solution of the system (if the truncation errors are negligible).

Stage 1
In the first stage we construct a new system having the same solution as the initial system (1.1.1), and for which its matrix is upper-triangular. This stage consists of $n-1$ steps which will be described below.

In the *first step* the unknown x_1 is eliminated from the equations $2, 3, 4, \ldots, n$. This step is realized by successive multiplications of the first equation with $m_{21} = -\frac{a_{21}}{a_{11}}, m_{31} = -\frac{a_{31}}{a_{11}}, \ldots, m_{n1} = -\frac{a_{n1}}{a_{11}}$ and by successive sums of the obtained equations with equations $2, 3, 4, \ldots, n$. This new system looks like:

$$\begin{cases} a_{11}^1 x_1 + a_{12}^1 x_2 + \ldots a_{1n}^1 x_n = b_1^1 \\ \qquad\quad a_{22}^2 x_2 + \ldots a_{2n}^2 x_n = b_2^2 \\ \qquad\quad \ldots\ldots\ldots\ldots\ldots\ldots \\ \qquad\quad a_{n2}^2 x_2 + \ldots a_{nn}^2 x_n = b_n^2 \end{cases} \tag{1.1.4}$$

where the superscript 1 indicates that the first equation remains unchanged, the superscript 2 indicates that the first elimination has been carried out and:

$$\begin{array}{lll} a_{ij}^1 = a_{ij} & b_i^1 = b_i & \text{for } i, j = 1, 2, \ldots, n \\ a_{ij}^2 = a_{ij}^1 + m_{i1} \cdot a_{1j}^1 & b_i^2 = b_i^1 + m_{i1} \cdot b_1^1 & \text{for } i, j = 2, 3, \ldots, n. \end{array}$$

At the *second step* the first equation will be ignored, and the unknown x_2 will be eliminated from the equations $3, 4, \ldots, n$ by successive multiplications of the second equation with $m_{i2} = -\frac{a_{i2}^2}{a_{22}^2}$; $i = 3, \ldots, n$, and by successive sums of the obtained equation with the equations $3, 4, \ldots, n$. After this step we obtain the followings system:

$$\left\{\begin{array}{r} a_{11}^1 x_1 + a_{12}^1 x_2 + a_{13}^1 x_3 + \ldots + a_{1n}^1 x_n = b_1^1 \\ a_{22}^2 x_2 + a_{23}^2 x_3 + \ldots + a_{2n}^2 x_n = b_2^2 \\ a_{33}^3 x_3 + \ldots + a_{3n}^3 x_n = b_3^3 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n3}^3 x_2 + \ldots + a_{nn}^3 x_n = b_n^3 \end{array}\right. \tag{1.1.5}$$

where the superscript 3 indicates that the second elimination has been carried out and:

$$a_{ij}^3 = a_{ij}^2 + m_{i2} \cdot a_{2j}^2 \quad b_i^3 = b_i^2 + m_{i2} \cdot b_2^2 \quad m_{i2} = -\frac{a_{i2}^2}{a_{22}^2} \quad \text{for } i, j = 3, 4, \ldots, n.$$

At the k - *step* the first $k-1$ equations will be ignored, and the unknown x_k will be eliminated from the equations $k+1, \ldots, n$ by successive multiplications of the equations k with the ratio $m_{ik} = -\frac{a_{ik}^k}{a_{kk}^k}$, and by successive sums of the obtained equation with the equation i. The obtained system is:

$$\left\{\begin{array}{r} a_{11}^1 x_1 + a_{12}^1 x_2 + a_{13}^1 x_3 + \ldots + a_{1k}^1 x_k + a_{1\,k+1}^1 x_{k+1} + \ldots + a_{1n}^1 x_n = b_1^1 \\ a_{22}^2 x_2 + a_{23}^2 x_3 + \ldots + a_{2k}^2 x_k + a_{2\,k+1}^2 x_{k+1} + \ldots + a_{2n}^2 x_n = b_2^2 \\ a_{33}^3 x_3 + \ldots + a_{3k}^3 x_k + a_{3\,k+1}^3 x_{k+1} + \ldots + a_{3n}^3 x_n = b_3^3 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{kk}^k x_k + a_{k\,k+1}^k x_{k+1} + \ldots + a_{kn}^k x_n = b_k^k \\ a_{k+1\,k+1}^{k+1} x_{k+1} + \ldots + a_{k+1\,n}^{k+1} x_n = b_{k+1}^{k+1} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n\,k+1}^{k+1} x_{k+1} + \ldots + a_{n\,n}^{k+1} x_n = b_n^{k+1} \end{array}\right. \tag{1.1.6}$$

where:

$$a_{ij}^{k+1} = a_{ij}^k + m_{ik} \cdot a_{kj}^k, \quad b_i^{k+1} = b_i^k + m_{ik} \cdot b_k^k, \quad m_{ik} = -\frac{a_{ik}^k}{a_{kk}^k} \quad \text{for } i, j = k+1, \ldots, n.$$

The *last step* of the first stage is the step $n-1$, in which the unknown x_{n-1} will be eliminated from the equation n. These $n-1$ steps give the

system:

$$\begin{cases} a_{11}^1 x_1 + a_{12}^1 x_2 + a_{13}^1 x_3 + \ldots + a_{1k}^1 x_k + \ldots + a_{1n}^1 x_n = b_1^1 \\ a_{22}^2 x_2 + a_{23}^2 x_3 + \ldots + a_{2k}^2 x_k + \ldots + a_{2n}^2 x_n = b_2^2 \\ a_{33}^3 x_3 + \ldots + a_{3k}^3 x_k + \ldots + a_{3n}^3 x_n = b_3^3 \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ a_{kk}^k x_k + \ldots + a_{kn}^k x_n = b_k^k \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ a_{nn}^n x_n = b_n^n \end{cases} \tag{1.1.7}$$

Stage 2
In the second stage we solve the obtained system (1.1.7), by back substitution. From the last equation we obtain:

$$x_n = \frac{b_n^n}{a_{nn}^n}. \tag{1.1.8}$$

Introducing the computed x_n in the equation $n-1$ we obtain the unknown x_{n-1}. Introducing x_n and x_{n-1} in the equation $n-2$ we obtain the unknown x_{n-2}, etc, the process is repeated until we are left with one equation and one unknown x_1. The unknowns $x_{n-1}, x_{n-2}, \ldots, x_1$ are given by the formula:

$$x_k = \left(b_k^k - \sum_{j=k+1}^{n} a_{kj}^k \cdot x_j \right) \cdot \frac{1}{a_{kk}^k}, \quad k = n-1, n-2, \ldots, 1. \tag{1.1.9}$$

Remark 1.1.2. The method described above (Gauss with trivial pivot) assumes that at each step the element a_{kk}^k from the main diagonal of the system (1.1.7) is different of zero. These coefficients are called pivots or pivotal elements. If there is a pivotal element equal to zero then the procedure breaks down since m_{ik} cannot be defined. To ensure that zero pivots are not used to compute the multipliers at the step k, a search is made in the column k for nonzero elements from rows $k+1, \ldots, n$. If some entry is nonzero, for example $i > k$ then we switch rows i and k. This interchange of rows does not change the solution of the system. This interchange is possible because A is non-singular (if all elements from the pivot down in column k including the pivot are zero, then the matrix is singular).

Because the computer uses fixed-precision arithmetic, it is possible that a small error will be introduced each time that an arithmetic operation is performed. Hence, the trivial pivoting strategy in Gaussian elimination can lead to significant error in the solution of a linear system of equations.

Remark 1.1.3. The purpose of the pivoting strategy is to move the entry of greatest magnitude to the main diagonal and then use it to eliminate the remaining entries in the column. This implies an interchange between row k and the row which contains the element with the largest magnitude from the column. This modified method is called **Gauss method with partial pivot**.

Remark 1.1.4. In the **Gauss method with partial pivot** we can improve errors by choosing in each step a pivot given by the element with the largest magnitude from the column and row, simultaneously. This is called **Gauss method with total pivot**.

Remark 1.1.5. The Gauss method with pivot has the following matrix representation:

$$A^{(n-1)} = M \cdot A \tag{1.1.10}$$

where: $A^{(n-1)}$ is the matrix of the system (1.1.7) and M is the matrix:

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ m_{21} & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ m_{n1} & m_{n2} & m_{n3} & \dots & 1 \end{pmatrix} = M. \tag{1.1.11}$$

Remark 1.1.6. *Denoting by $A^{(n-1)} = U$, $M^{-1} = L$ and supposing that the system (1.1.1) is compatible determinate, the equality (1.1.10) becomes:*

$$A = L \cdot U. \tag{1.1.12}$$

This shows that the Gauss method with pivot conducts to the existence of the factorization $A = L \cdot U$, where L and U are lower-triangular and upper-triangular matrices.

Exercises

Solve the following systems using the Gauss method:

a) with pivot

$$\begin{cases} 2x - 4y + 2z = 20 \\ x - 4y - z = 2 \\ -x + y + z = -2 \end{cases}$$

b) with partial pivot

$$\begin{cases} & - & 8y & - & 2z & = & -1 \\ 6x & + & 4y & & & = & 4 \\ 8x & & & - & 6z & = & 7 \end{cases}$$

c) with total pivot

$$\begin{cases} & x & - & y & - & 2z & = & 2 \\ - & 2x & - & 4y & + & z & = & 8 \\ & x & + & y & + & 2z & = & 2 \end{cases}$$

The algorithm of implementation of the Gauss method is:

$k = 1 \dots n-1$
$\quad i = k+1 \dots n$
$\quad\quad m = \dfrac{a_{ik}}{a_{kk}}$
$\quad\quad j = k+1 \dots n$
$\quad\quad\quad a_{ij} = a_{ij} - m \cdot a_{kj}$
$\quad\quad b_i = b_i - m \cdot b_k$
$x_n = \dfrac{b_n}{a_{nn}}$
for $i = n-1 \dots 1$
$\quad x_i = \left(b_i - \sum_{j=i+1}^{n} a_{ij} \cdot x_j \right) \cdot \dfrac{1}{a_{ii}}$

Input data:

- n - dimension of the space

- $(a_{ij})_{\substack{i=1,\dots,n \\ j=1,\dots,n}}$ - matrix of the system

- $(b_i)_{i=1,\dots,n}$ - the column vector form the right hand side

Output data:

- $(x_i)_{i=1,\dots,n}$ - the solution of the system $Ax = b$

Implementation in Borland C language:

```
#include<stdio.h>
#include<math.h>
#include<malloc.h>
```

```
float **Matrice(int imin, int imax, int jmin, int jmax);
float *Vector(int imin, int imax);
void CitireMat(float **a, int n, int m);
void CitireVect(float *a, int init, int n);
void ScriereMat(float **a, int n, int m);
void ScriereVect(float *a, int init, int n);
void gauss(float **a, float *b, int n);
void main()
{
    float **a, *b;
    int n;
    printf("n= "); scanf("%d", &n);
    a=Matrice(1,n,1,n);
    b=Vector(0,n);
    CitireMat(a,n,n);
    ScriereMat(a,n,n);
    CitireVect(b,1,n);
    ScriereVect(b,1,n);
    printf("----------SOLUTIE----------\n");
    gauss(a,b,n);
}
#define Swap(a,b) {t=a; a=b; b=t;}
void gauss(float **a, float *b, int n)
{
    float amax, suma, m,t, *x;
    int i, imax, j, k;
    x=Vector(0,n);
    for(k=1;k<=n;k++)
    {
        amax=0.0;
        for(i=k;i<=n;i++)
            if(amax<fabs(a[i][k]))
            {
                amax=fabs(a[i][k]);
                imax=i;
            }
        if(amax==0)
        {
            printf("matrice nesingulara! \n");
        }
        if(imax!=k)
```

```
            {
                for(j=k;j<=n;j++) Swap(a[imax][j], a[k][j]);
                Swap(b[imax], b[k]);
            }
            m=1.0/a[k][k];
            for(j=1;j<=n;j++) a[k+1][j]*=m;
            b[k+1]*=m;
            for(i=k+1;i<=n;i++)
            {
                m=a[i][k]/a[k][k];
                for(j=k;j<=n;j++) a[i][j]-=a[k][j]*m;
                b[i]-=b[k]*m;
            }
        }
        x[n]=b[n]/a[n][n];
        for(k=n-1;k>=1;k–)
        {
            suma=0.0;
            for(j=k+1;j<=n;j++)
                suma+=a[k][j]*x[j];
            x[k]=(b[k]-suma)/a[k][k];
        }
        ScriereVect(x,1,n);
    }
```

1.2 LU Factorization

In the Gaussian elimination method, for solving a Cramer-type system

$$Ax = b, \tag{1.2.1}$$

the system was reduced to a triangular form and then solved by back substitution. It is much easier to solve triangular systems. Let us exploit this idea and assume that a given $n \times n$ matrix A can be written as a product of two matrices L and U so that

$$A = L \cdot U \tag{1.2.2}$$

where L is an $n \times n$ lower-triangular matrix and U is an $n \times n$ upper-triangular matrix:

$$L = \begin{pmatrix} \lambda_{11} & 0 & 0 & \dots & 0 \\ \lambda_{21} & \lambda_{22} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ \lambda_{n1} & \lambda_{n2} & \lambda_{n3} & \dots & \lambda_{nn} \end{pmatrix} \tag{1.2.3}$$

$$U = \begin{pmatrix} \mu_{11} & \mu_{12} & \mu_{13} & \dots & \mu_{1n} \\ 0 & \mu_{22} & \mu_{23} & \dots & \mu_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \mu_{nn} \end{pmatrix} \tag{1.2.4}$$

Theorem 1.2.1. *If the matrix A can be written in the form (1.2.2) then the system (1.2.1) is decomposed into two triangular systems:*

$$Ly = b \tag{1.2.5}$$

$$Ux = y \tag{1.2.6}$$

which are equivalent with:

$$\begin{cases} \lambda_{11}\, y_1 = b_1 \\ \lambda_{21}\, y_1 + \lambda_{22}\, y_2 = b_2 \\ \quad \dots\dots\dots\dots\dots\dots \\ \lambda_{n1}\, y_1 + \lambda_{n2}\, y_2 + \dots + \lambda_{nn}\, y_n = b_n \end{cases} \tag{1.2.7}$$

$$\begin{cases} \mu_{11}\, x_1 + \mu_{12}\, x_2 + \dots + \mu_{1n}\, x_n = y_1 \\ \qquad\qquad \mu_{22}\, x_2 + \dots + \mu_{2n}\, x_n = y_2 \\ \qquad\qquad\qquad\qquad \dots\dots\dots\dots\dots \\ \qquad\qquad\qquad\qquad\qquad \mu_{nn}\, x_n = y_n. \end{cases} \tag{1.2.8}$$

Proof: immediate.

Both systems are triangular and therefore easy to solve. What we need is a procedure to generate factorization. For this we observe that we obtain L and U by Gaussian elimination. In fact, we have the following theorem:

Theorem 1.2.2. *The solution of the system (1.2.7) is given by:*

$$\begin{cases} y_1 = \dfrac{b_1}{\lambda_{11}} \\ y_i = \left(b_i - \displaystyle\sum_{j=1}^{i-1} \lambda_{ij} y_j \right) \cdot \dfrac{1}{\lambda_{ii}}, \quad i = 2, 3, \dots, n \end{cases} \tag{1.2.9}$$

and the solution of the system (1.2.8) is given by:

$$\begin{cases} x_n = \dfrac{y_n}{\mu_{nn}} \\ x_i = \left(y_i - \displaystyle\sum_{j=i+1}^{n} \mu_{ij} x_j \right) \cdot \dfrac{1}{\mu_{ii}}, \quad i = 2, 3, \ldots, n. \end{cases} \tag{1.2.10}$$

Proof: immediate.

Theorem 1.2.3. *If A is a non-singular matrix and satisfies the conditions:*

$$det(A^{[i]}) \neq 0, \qquad i = 1, 2, \ldots, n-1 \tag{1.2.11}$$

then there exists a lower-triangular matrix L and an upper-triangular matrix U such that the equality (1.2.2) takes place. We denoted by $det(A^{[i]})$ the corner determinants until order $n-1$ of the matrix A.

Proof: the matrices:

$$L = \begin{pmatrix} \lambda_{11} & 0 & 0 & \ldots & 0 \\ \lambda_{21} & \lambda_{22} & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \lambda_{n1} & \lambda_{n2} & \lambda_{n3} & \ldots & \lambda_{nn} \end{pmatrix} \qquad \text{and} \qquad U = \begin{pmatrix} \mu_{11} & \mu_{12} & \mu_{13} & \ldots & \mu_{nn} \\ 0 & \mu_{22} & \mu_{23} & \ldots & \mu_{2n} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & \mu_{nn} \end{pmatrix}$$

are considered having $n(n+1)$ unknowns. Computing the product $L \cdot U$ and equating by A we obtain a system having n^2 equations and n^2+n unknowns. For the uniqueness of the solution, n elements of the matrix L or U should be specified.

Remark 1.2.1. The additional conditions on the unknowns can be specified in the following ways:

- *the diagonal elements of the matrix U are $\mu_{ii} = 1$, known as the* ***Crout method****;*
- *the diagonal elements of the matrix L are $\lambda_{ii} = 1$, known as the* ***Doolittle method****.*

In the case of the Crout factorization we have:

$$a_{ij} = \sum_{k=1}^{\min(i,j)} \lambda_{ik} \cdot \mu_{kj} \qquad \mu_{ii} = 1 \qquad i, j = 1, \ldots, n \tag{1.2.12}$$

For $j = 1$ we obtain:

$$a_{i1} = \lambda_{i1} \cdot \mu_{11} = \lambda_{i1}\,; \qquad i, j = 1, \ldots, n \tag{1.2.13}$$

hence, the first column of the matrix L is equal to the first column of the matrix A.

If in (1.2.12) we take $i = 1$ then we obtain:

$$a_{1j} = \lambda_{11} \cdot \mu_{1j}, \qquad j = 1, 2, \ldots, n \tag{1.2.14}$$

hence,

$$\mu_{1j} = \frac{a_{1j}}{a_{11}}, \qquad j = 1, 2, \ldots, n \tag{1.2.15}$$

which represent the elements of the first row from the matrix U.

Supposing that the first $r-1$ columns of the matrix L and the first $r-1$ rows of the matrix U are determined from (1.2.12) we have:

$$a_{ir} = \lambda_{ir} + \sum_{k=1}^{r-1} \lambda_{ik} \cdot \mu_{kr} \qquad i = r, \ldots, n$$

$$a_{rj} = \lambda_{rr} \cdot \mu_{rj} + \sum_{k=1}^{r-1} \lambda_{rk} \cdot \mu_{kj} \qquad j = r+1, \ldots, n$$

and hence λ_{ir} (column r of L) and μ_{rj} (row r of U), are obtained in the case of the Crout factorization.

Exercises

Solve the following systems using LU factorization:

a) $$\begin{cases} 4x - 2y + z = 16 \\ -3x - y + 3z = 3 \\ x - y + 3z = 6 \end{cases}$$

b) $$\begin{cases} 2x + 3y + 3z = 2 \\ 5y + 7z = 2 \\ 6x + 9y + 8z = 5 \end{cases}$$

The algorithm of implementation of the LU factorization is:

// Determination of x vector representing the solution of the system $Ux = y$:

for $k = 1 \ldots n$

$$u_{kk} = 1$$

$$l_{kk} = a_{kk} - \sum_{s=1}^{k-1} l_{ks} \cdot u_{sk}$$

$$\text{for} j = k+1 \ldots n$$

$$u_{ki} = \left(a_{ki} - \sum_{s=1}^{k-1} l_{ks} \cdot u_{si}\right) \cdot \frac{1}{l_{kk}}$$

$$l_{ik} = \left(a_{ik} - \sum_{s=1}^{k-1} l_{is} \cdot u_{sk}\right) \cdot \frac{1}{u_{kk}}$$

// Determination of y vector representing the solution of the system $Ly = b$:

$$y_1 = \frac{b_1}{l_{11}}$$

for $i = 2 \ldots n$

$$y_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij} \cdot y_j\right) \cdot \frac{1}{l_{ii}}$$

// Determination of x vector representing the solution of the system $Ux = y$:

$$x_n = \frac{y_n}{u_{nn}}$$

for $i = n-1 \ldots 1$

$$x_i = \left(y_i - \sum_{j=i+1}^{n} u_{ij} \cdot x_j\right) \cdot \frac{1}{u_{ii}}$$

Input data:

- n - dimension of the space

- $(a_{ij})_{\substack{i=1,\ldots,n \\ j=1,\ldots,n}}$ - matrix of the system

- $(b_i)_{i=1,\ldots,n}$ - column vector from the right hand side

Output data:

- $(x_i)_{i=1,\ldots,n}$ - the solution of the system $Ax = b$

Implementation of the above algorithm, in Borland C language, is:

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<malloc.h>
float **Matrice(int imin, int imax, int jmin, int jmax);
float *Vector(int n);
void CitireMat(float **a, int n, int m);
void CitireVect(float *a, int n);
```

```
void ScriereMat(float **a, int n, int m);
void ScriereVect(float *a, int n);
void LU(float **a, float *b, int n);
void main(void)
{
    float **a, *b;
    int n;
    a=Matrice(1,n,1,n);
    b=Vector(n);
    printf("n= ");
    scanf("%d", &n);
    CitireMat(a,n,n);
    ScriereMat(a,n,n);
    CitireVect(b,n);
    ScriereVect(b,n);
    printf("——————————SOLUTIE——————————\n");
    LU(a,b,n);
}
void CitireMat(float **a, int n, int m)
{
    int i,j;
    for(i=1; i<=n; i++)
        for(j=1; j<=m; j++)
        {
            printf("[%i][%i]=",i,j);
            scanf("%f",&a[i][j]);
        }
}
void CitireVect(float *a, int n)
{
    int i;
    for(i=1; i<=n; i++)
    {
        printf("[%i]=",i);
        scanf("%f", &a[i]);
    }
}
void ScriereMat(float **a, int n, int m)
{
    int i,j;
    for(i=1; i<=n; i++)
```

```
        {
            for(j=1;j<=m;j++)
            printf("%g ",a[i][j]);
            printf("\n");
        }
    }
    void ScriereVect(float *a, int n)
    {
        int i;
        for(i=1; i<=n; i++)
        {
            printf("%g",a[i]);
            printf("\n");
        }
    }
    float **Matrice(int imin, int imax, int jmin, int jmax)
    {
        int i, ni=imax-imin+1,nj=jmax-jmin+1;
        float **p;
        p=(float **) malloc ((size_t)(ni*sizeof(float *)));
        p-=imin;
        p[imin]=(float *) malloc ((size_t) (ni*nj*sizeof(float)));
        p[imin]-=jmin;
        for(i=imin+1;i<=imax;i++) p[i]=p[i-1]+nj;
        return p;
    }
    float *Vector(int n)
    {
        float *p;
        p=(float *)malloc((size_t) (n*sizeof(float)));
        return p;
    }
    void LU(float **a, float *b, int n)
    {
        float **u, **l, *y, *x, suma;
        int i,k,j;
        u=Matrice(1,n,1,n);
        l=Matrice(1,n,1,n);
        x=Vector(n);
        y=Vector(n);
        for(k=1;k<=n;k++)
```

```
    {
        u[k][k]=1;
        suma=0;
        for(j=1;j<=k-1;j++)
            suma=suma+l[k][j]*u[j][k];
        l[k][k]=a[k][k]-suma;
        suma=0;
        for(i=k+1;i<=n;i++)
        {
            for(j=1;j<=k-1;j++)
            suma=suma+l[k][j]*u[j][i];
            u[k][i]=(a[k][i]-suma)/l[k][k];
            suma=0;
            for(j=1;j<=k-1;j++)
                suma=suma+l[i][j]*u[j][k];
            l[i][k]=a[i][k]-suma;
        }
    }
    y[1]=b[1]/l[1][1];
    for(i=2;i<=n;i++)
    {
        suma=0;
        for(j=1;j<=i-1;j++)
            suma=suma+l[i][j]*y[j];
        y[i]=(b[i]-suma)/l[i][i];
    }
    x[n]=y[n]/u[n][n];
    for(i=n-1;i>=1;i–)
    {
        suma=0;
        for(j=i+1;j<=n;j++)
            suma=suma+u[i][j]*x[j];
        x[i]=(y[i]-suma)/u[i][i];
    }
    ScriereVect(x,n);
}
```

1.3 Tridiagonal Systems

Definition 1.3.1. *A tridiagonal system is a Cramer system*

$$Ax = b \tag{1.3.1}$$

in which the matrix A is tridiagonal (has nonzero elements only on the main diagonal and in the positions adjacent to the diagonal), i.e., A has the form:

$$A = \begin{pmatrix} b_1 & c_2 & 0 & 0 & \dots & \dots & \dots & 0 \\ a_2 & b_2 & c_3 & 0 & \dots & \dots & \dots & 0 \\ 0 & a_3 & b_3 & c_3 & \dots & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & a_{n-1} & b_{n-1} & c_n \\ 0 & \dots & \dots & \dots & \dots & 0 & a_n & b_n \end{pmatrix} \tag{1.3.2}$$

A tridiagonal system can be solved very efficiently by the factorization method. Thus, using the Crout factorization we have the following matrices L and U corresponding to the given tridiagonal matrix:

$$L = \begin{pmatrix} \beta_1 & 0 & 0 & \dots & 0 & 0 \\ a_2 & \beta_2 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \beta_{n-1} & 0 \\ 0 & 0 & 0 & \dots & a_n & \beta_n \end{pmatrix} \tag{1.3.3}$$

$$U = \begin{pmatrix} 1 & \nu_2 & 0 & \dots & 0 & 0 \\ 0 & 1 & \nu_3 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & \nu_n \\ 0 & 0 & 0 & \dots & \dots & 1 \end{pmatrix} \tag{1.3.4}$$

Theorem 1.3.1. *The equality $A = L \cdot U$ is satisfied if and only if the following equalities take place:*

$$\beta_1 = b_1 \tag{1.3.5}$$

$$\beta_i \cdot \nu_{i+1} = c_{i+1} \qquad i = 2, \dots, n-1 \tag{1.3.6}$$

$$a_i \cdot \nu_i + \beta_i = b_i \qquad i = 2, \dots, n \tag{1.3.7}$$

Proof: If $A = L \cdot U$ then:

- multiplying the row 1 of the matrix L with the column 1 of the matrix U we obtain $\beta_1 \cdot 1$; equating this with the element from the first row and the first column of the matrix A, i.e. b_1, we obtain $\beta_1 = b_1$.

- multiplying the row i of the matrix L with the column $i+1$ of the matrix U we obtain $\beta_i \cdot \nu_{i+1}$; equating this with the element from the row i and from the column $i+1$ of the matrix A, i.e. c_{i+1}, we obtain $\beta_i \cdot \nu_{i+1} = c_{i+1}$.

- multiplying the row i of the matrix L with the column i of the matrix U we obtain $a_i \cdot \nu_i + \beta_i$; equating this with the element from the row i and from the column i of the matrix A, i.e. b_i, we obtain $a_i \cdot \nu_i + \beta_i = b_i$.

Reciprocally, if the relations (1.3.5), (1.3.6) and (1.3.7) take place then the equality $A = L \cdot U$ is satisfied.

□

Let us denote by x_{ij} an arbitrary element of the matrix A; we observe that:

- for $i = 1$ we have: $x_{11} = b_1,\ x_{12} = c_1$ and $x_{ij} = 0$ if $j > 2$.
- for $i = 2, \ldots, n-1$ we have: $x_{ij} = 0$ if $j \notin \{i-1, i, i+1\}$ and $x_{i,i-1} = a_i,\ x_{ii} = b_i,\ x_{i,i+1} = c_{i+1}$.
- for $i = n$ we have: $x_{nj} = 0$ if $j < n-1$ and $x_{n,n-1} = a_n,\ x_{nn} = b_n$.

An arbitrary element λ_{ij} of the matrix L has the properties:

- for $i = 1,\ \lambda_{11} = \beta_1$ and $\lambda_{1j} = 0$ for $j \neq 1$
- $\lambda_{ij} = 0$ for $i = 2, \ldots, n$ if $j \notin \{i, i-1\}$ and $\lambda_{i,i-1} = a_i,\ \lambda_{ii} = \beta_i$.

An arbitrary element μ_{ij} of the matrix U is:

- $\mu_{ij} = 0$ for $i = 1, \ldots, n-1$ if $j \notin \{i, i+1\}$ and $\mu_{ii} = 1,\ \mu_{i,i+1} = \nu_{i+1}$
- for $i = n$ we have $\mu_{nj} = 0$ for $j < n$ and $\mu_{nn} = 1$.

An arbitrary element γ_{ij} of the matrices product $L \cdot U$ is given by:

$$\gamma_{ij} = \sum_{k=1}^{n} \lambda_{ik} \cdot \mu_{kj}.$$

For $i = 1$ and $j \leq 2$ we have $\gamma_{11} = b_1 = x_{11}$ and $\gamma_{12} = c_1 = x_{12}$; the element γ_{1j} for $j > 2$ is:

$$\gamma_{1j} = 0 = x_{ij}$$

For $i = 2, \ldots, n-1$, γ_{ij} is:

$$\gamma_{ij} = \lambda_{i,i-1} \cdot \mu_{i-1,j} + \lambda_{ii} \cdot \mu_{ij}$$

- if $j \notin \{i-1, i, i+1\}$ then $\mu_{i-1,j} = \mu_{ij} = \mu_{i+1,j} = 0$ and $\gamma_{ij} = 0 = x_{ij}$
- if $j = i-1$ then $\mu_{i-1,i-1} = 1,\ \mu_{i,i-1} = 0$ and $\gamma_{i,i-1} = a_i = x_{i,i-1}$
- if $j = i$ then $\mu_{i-1,i} = \nu_i,\ \mu_{i,i} = 1$ and $\gamma_{ii} = a_i \cdot \nu_i + \beta_i = b_i = x_{i,i}$
- if $j = i+1$ then $\mu_{i-1,i+1} = 0,\ \mu_{i,i+1} = \nu_{i+1}$ and $\gamma_{i,i+1} = \beta_i \cdot \nu_{i+1} = c_{i+1} = x_{i,i+1}$.

In the same way, it can be proved that $\gamma_{nj} = x_{nj}$.

Theorem 1.3.2. *If the equality $A = L \cdot U$ is satisfied and the numbers β_i verifying (1.3.5), (1.3.6) and (1.3.7) are nonzero, then the solution of the system (1.3.1) is given by the relations:*

$$x_n = y_n \ \text{ and } \ x_i = y_i - \nu_{i+1} \cdot x_{i+1}, \quad i = n-1, n-2, \ldots, 1, \tag{1.3.8}$$

in which $y_1, y_2, \ldots, y_n$ are given by the equalities:

$$y_1 = d_1/\beta_1 \ \text{ and } \ y_i = (d_i - a_i \cdot y_{i-1})/\beta_i, \quad i = 2, \ldots, n. \tag{1.3.9}$$

Proof: If x represents the solution of the equation $Ax = d$ and $A = LU$ then $LUx = d$. Let be $y = Ux$ and $Ly = d$.

From $Ly = d$ and $\beta_i \neq 0$ we have $y_1 = d_1/\beta_1$ and $y_i = (d_i - a_i \cdot y_{i-1})/\beta_i,\ i = 2, \ldots, n$.

From $Ux = y$ we have $x_n = y_n$ and $x_i = y_i - \nu_{i+1} \cdot x_{i+1},\ i = n-1, n-2, \ldots, 1$. □

Propostion 1.3.1. *If $|b_1| > |c_2|,\ |b_i| > |a_i| + |c_{i+1}|$ for $i = 2, \ldots, n-1$ and $|b_n| > |a_n|$ the system:*

$$\begin{cases} \beta_1 = b_1 \\ \beta_i \cdot \nu_{i+1} = c_{i+1}, \quad i = 2, \ldots, n-1 \\ a_i \cdot \nu_i + \beta_i = b_i, \quad i = 2, \ldots, n-1 \end{cases} \tag{1.3.10}$$

in unknowns $\beta_1, \ldots \beta_n$ and $\nu_2, \ldots, \nu_n$ has a unique solution and $\beta_i \neq 0,\ i = 1, \ldots, n$.

Proof: Outline: The numbers $\beta_1, \ldots \beta_n$ and $\nu_2, \ldots, \nu_n$ are obtained in the order $\beta_1, \nu_2,\ \beta_2, \nu_3,\ \beta_3, \nu_4, \ldots, \beta_n, \nu_n$ using relations $\beta_i \cdot \nu_{i+1} = c_{i+1}$ and $a_i \cdot \nu_i + \beta_i = b_i$.

Definition 1.3.2. *A quadratic matrix $A = (a_{ij})$ of order n is strict diagonally dominant if its elements verify the relations:*

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \qquad i = 1, 2, \ldots, n. \tag{1.3.11}$$

Propostion 1.3.2. *If the tridiagonal matrix A is diagonally dominant then the factorization $A = L \cdot U$ with L and U of the forms (1.3.3) and (1.3.4) is possible.*

Proof: Consequence of the Proposition 1.3.1.

Exercises

1. Solve the followings system using LU method:

$$\begin{cases} x + 2y = 3 \\ 3x + 2y + 4z = 1 \\ y + z + 2t = -2 \\ 2z + 3t = 3 \end{cases}$$

The algorithm of implementation of the LU factorization in the case of the tridiagonal matrices is:

// Construction of the matrices L and U:
$\beta_1 = b_1$
$\gamma_2 = \dfrac{c_2}{\beta_1}$
for $i = 2 \dots n-1$
$\beta_i = b_i - a_i \cdot \gamma_i$
$\gamma_{i+1} = \dfrac{c_{i+1}}{\beta_i}$
$\beta_n = b_n - a_n \cdot \gamma_n$
// Determination of the vector y as a solution of the system $Ly = d$:
$y_1 = \dfrac{d_1}{\beta_1}$
for $i = 2 \dots n$
$y_i = (d_i - a_i \cdot y_{i-1})/\beta_i$
// Determination of the vector x as a solution of the system $Ux = y$:
$x_n = y_n$
for $i = n-1 \dots 1$
$x_i = y_i - \gamma_{i+1} \cdot x_{i+1}$

Input data:

- n - dimension of the space

- $(a_i)_{i=2,\dots,n}$ - the adjacent lower diagonal

- $(b_i)_{i=1,\dots,n}$ - the main diagonal

- $(c_i)_{i=2,\ldots,n}$ - the adjacent upper diagonal

- $(d_i)_{i=1,\ldots,n}$ - the column vector from the right hand side

Output data:

- $(x_i)_{i=1,\ldots,n}$ - the solution of the system $Ax = d$

Implementation of the above algorithm, in Borland C language, is:

```
#include<stdio.h>
#include<malloc.h>
float **Matrice(int imin, int imax, int jmin, int jmax);
float *Vector(int n);
void CitireVect(float *a,int init, int n);
void ScriereVect(float *a,int init, int n);
void Tridiag(float *a, float *b, float *c, float *d,int n);
void main(void)
{
    float *a, *b, *c,*d;
    int n;
    a=Vector(n);
    b=Vector(n);
    c=Vector(n);
    d=Vector(n);
    printf("n= ");
    scanf("%d", &n);
    printf("Introduceti elementele de pe diagonala maina:\n");
    CitireVect(b,1,n);
    ScriereVect(b,1,n);
    printf("Introduceti elementele de deasupra diagonalei principale:\n");
    CitireVect(c,2,n);
    ScriereVect(c,2,n);
    printf("Introduceti elementele de sub diagonala principala:\n");
    CitireVect(a,2,n);
    ScriereVect(a,2,n);
    printf("Introduceti termenii liberi:\n");
    CitireVect(d,1,n);
    ScriereVect(d,1,n);
    printf("————————SOLUTIE————————\n");
    Tridiag(a,b,c,d,n);
}
void CitireVect(float *a,int init, int n)
```

```
{
    int i;
    for(i=init; i<=n; i++)
    {
        printf("[%i]=",i);
        scanf("%f", &a[i]);
    }
}
void ScriereVect(float *a,int init, int n)
{
    int i;
    for(i=init; i<=n; i++)
    {
        printf("%g",a[i]);
        printf("\n");
    }
}
void Tridiag(float *a, float *b, float *c, float *d,int n)
{
    int i;
    float *beta,* gamma,*x,*y;
    beta=Vector(n);
    gamma=Vector(n);
    x=Vector(n);
    y=Vector(n);
    beta[1]=b[1];
    gamma[2]=c[2]/beta[1];
    for(i=2;i<=n-1;i++)
    {
        beta[i]=b[i]-a[i]*gamma[i];
        gamma[i+1]=c[i+1]/beta[i];
    }
    beta[n]=b[n]-a[n]*gamma[n];
    y[1]=d[1]/beta[1];
    for(i=2;i<=n;i++)
        y[i]=(d[i]-a[i]*y[i-1])/beta[i];
    x[n]=y[n];
    for(i=n-1;i>=1;i–)
        x[i]=y[i]-gamma[i+1]*x[i+1];
    ScriereVect(x,1,n);
}
```

1.4 Cholesky Factorization

The Cholesky factorization is a method used in computation of a numerical solution of the Cramer systems which are symmetric and have positive definite matrices.

Definition 1.4.1. *The Cramer system*

$$Ax = b \tag{1.4.1}$$

is called ***symmetric*** *if the matrix A of the system is symmetric, i.e. A has the property:*

$$A^T = A. \tag{1.4.2}$$

Definition 1.4.2. *A quadratic matrix A of the order n (symmetric or non-symmetric) is called* ***positive definite****, if for any $x \in \mathbb{R}^n, x \neq 0$, the followings inequality is satisfied:*

$$x^T Ax > 0 \tag{1.4.3}$$

Theorem 1.4.1. (Sylvester)
A matrix A is positive definite if and only if all corner determinants $\det(A^{[k]})$, $k = 1, 2, \ldots, n$ *are strictly positive, where we denoted by $A^{[k]}$ the matrix:*

$$A^{[k]} = (a_{ij})_{\substack{1 \leq i \leq k \\ 1 \leq j \leq k}} \tag{1.4.4}$$

Theorem 1.4.2. (Cholesky factorization)
A matrix A is symmetric and positive definite if and only if there exists a non-singular and lower triangular matrix L having real elements, such that the followings equality is satisfied:

$$A = L \cdot L^T. \tag{1.4.5}$$

Proof: We will prove that, if there exists a non-singular and lower triangular matrix L such that $A = L \cdot L^T$, then the matrix A is symmetric and positive definite.
Symmetry: $A = L \cdot L^T \Rightarrow A^T = (L \cdot L^T)^T = (L^T)^T \cdot L^T = L \cdot L^T = A \Rightarrow A = A^T$.

A is positive definite: let be $x \in \mathbb{R}^n$ and the number $x^T Ax$. We have:

$$x^T Ax =< Ax, x >=< L \cdot L^T x, x >=< L^T x, L^T x >= \|L^T x\|^2 > 0, (\forall) x \neq 0;$$

$$L^T x = 0 \Leftrightarrow x = 0.$$

In the followings, we will show who are the elements of the matrix L. We consider:

$$L = \begin{pmatrix} \lambda_{11} & 0 & 0 & \dots & 0 \\ \lambda_{21} & \lambda_{22} & 0 & \dots & 0 \\ \lambda_{31} & \lambda_{32} & \lambda_{33} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ \lambda_{n1} & \lambda_{n2} & \lambda_{n3} & \dots & \lambda_{nn} \end{pmatrix} \tag{1.4.6}$$

and

$$L^T = \begin{pmatrix} \lambda_{11} & \lambda_{21} & \lambda_{31} & \dots & \lambda_{n1} \\ 0 & \lambda_{22} & \lambda_{32} & \dots & \lambda_{n2} \\ 0 & 0 & \lambda_{33} & \dots & \lambda_{n3} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \lambda_{nn} \end{pmatrix} \tag{1.4.7}$$

The element from row i and column j of the product $L \cdot L^T$ is:

$$p_{ij} = \sum_{k=1}^{min(i,j)} \lambda_{ik} \cdot \lambda_{jk} \tag{1.4.8}$$

and it is equal to a_{ij}

$$p_{ij} = a_{ij}, \quad i, j = 1, 2, \dots, n \tag{1.4.9}$$

- for $i = j = 1$ we find:

  $p_{11} = \lambda_{11}^2 = a_{11}, \quad a_{11} > 0$ from A positive definite, and hence

  $\lambda_{11} = \sqrt{a_{11}}, \quad \lambda_{11} \in \mathbb{R}_+^*$;

- for $j = 1$ and $i = 2, \dots, n$ we find:

  $$p_{i1} = \lambda_{i1} \cdot \lambda_{11} = a_{i1} \quad \text{and hence} \quad \lambda_{i1} = \frac{a_{i1}}{\lambda_{11}};$$

- for fixed j and $i = j$ we find:

  $$p_{ii} = \sum_{k=1}^{i} \lambda_{ik}^2 = a_{ii} \quad \text{and hence} \quad \lambda_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} \lambda_{ik}^2},$$

  it existence being assured on the basis of the Sylvester's Theorem;

- for fixed j and $i = j+1, j+2, \dots, n$ we find:

  $$p_{ij} = \sum_{k=1}^{j} \lambda_{ik} \cdot \lambda_{jk} = a_{ij} \quad \text{and hence} \quad \lambda_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} \lambda_{ik} \cdot \lambda_{jk} \right) \cdot \frac{1}{\lambda_{jj}}.$$

In this way, we can conclude that, if $A = L \cdot L^T$ then the elements λ_{ij} of the matrix L are given by the formulas:

$$\lambda_{11} = \sqrt{a_{11}} \quad \text{and} \quad \lambda_{i1} = \frac{a_{i1}}{\lambda_{11}} \qquad i = 2, \ldots, n; \tag{1.4.10}$$

$$\lambda_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} \lambda_{ik}^2} \qquad i = 2, \ldots, n; \tag{1.4.11}$$

$$\lambda_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} \lambda_{ik} \cdot \lambda_{jk} \right) \cdot \frac{1}{\lambda_{jj}} \qquad i = j+1, j+2, \ldots, n. \tag{1.4.12}$$

Thus, if the matrix A is symmetric and positive definite, then the formulas (1.4.10), (1.4.11) and (1.4.12) define a non-singular and lower triangular matrix L for which $A = L \cdot L^T$. □

Theorem 1.4.3. *If the matrix A is symmetric and positive definite, then the solution of the system*

$$Ax = b \tag{1.4.13}$$

is given by the formulas:

$$x_n = \frac{y_n}{\lambda_{nn}} \quad \textit{and} \quad x_i = \left(y_i - \sum_{k=i+1}^{n} \lambda_{ki} \cdot x_k \right) \cdot \frac{1}{\lambda_{ii}} \qquad i = n-1, n-2, \ldots, 1 \tag{1.4.14}$$

in which $y_1, \ldots, y_n$ are given by:

$$y_1 = \frac{b_1}{\lambda_{11}} \quad \textit{and} \quad y_i = \left(b_i - \sum_{k=1}^{i-1} \lambda_{ik} \cdot y_k \right) \cdot \frac{1}{\lambda_{ii}} \qquad i = 2, 3, \ldots, n. \tag{1.4.15}$$

Exercises

1. Solve the following systems using Cholesky factorization:

a) $\begin{cases} x_1 + x_2 + x_3 = 2 \\ x_1 + 5x_2 + 5x_3 = 4 \\ x_1 + 5x_2 + 14x_3 = -5 \end{cases}$

b) $\begin{cases} x_1 + 2x_2 + 3x_3 = 0 \\ 2x_1 + 5x_2 + x_3 = -12 \\ 3x_1 + x_2 + 35x_3 = 59 \end{cases}$

The algorithm for the Cholesky factorization is:

// Construction of the matrix L
$k = 1 \ldots n$

$$l_{kk} = \sqrt{a_{kk} - \sum_{s=1}^{k-1} l_{ks}^2}$$

$i = k+1 \ldots n$

$$l_{ik} = \left(a_{ik} - \sum_{s=1}^{k-1} l_{is} \cdot l_{ks} \right) \cdot \frac{1}{l_{kk}}$$

// Determination of the vector y as solution of the system Ly = b:

$$y_1 = \frac{b_1}{l_{11}}$$

for $i = 2 \ldots n$

$$y_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij} \cdot y_j \right) \cdot \frac{1}{l_{ii}}$$

// Determination of the vector x as solution of the system $L^T x = y$:

$$x_n = y_n / l_{nn}$$

for $i = n-1 \ldots 1$

$$x_i = \left(y_i - \sum_{j=i+1}^{n} l_{ji} \cdot x_j \right) \cdot \frac{1}{l_{ii}}$$

Input data

- n - dimension of the space

- $(a_{ij})_{\substack{i=1,\ldots,n \\ j=1,\ldots,n}}$ - matrix of the system

- $(b_i)_{i=1,\ldots,n}$ - column vector from the right hand side

Output data:

- $(x_i)_{i=1,\ldots,n}$ - the solution of the system $Ax = b$

Implementation of the above algorithm for the Cholesky factorization, in Borland C language, is:

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
```

```
#include<malloc.h>
void Cholesky(float **a, float *b, int n);
void main(void)
{
    float **a, *b;
    int n;
    a=Matrice(1,n,1,n);
    b=Vector(n);
    printf("n= ");
    scanf("%d", &n);
    CitireMat(a,n,n);
    ScriereMat(a,n,n);
    CitireVect(b,n);
    ScriereVect(b,n);
    printf("————————SOLUTIE————————\n");
    Cholesky(a,b,n);
}
void Cholesky(float **a, float *b, int n)
{
    float **l, *y, *x, suma;
    int i,k,j;
    l=Matrice(1,n,1,n);
    x=Vector(n);
    y=Vector(n);
    for (k = 1; k<=n; k++)
    {
        suma = 0.0;
        for (i = 1; i<=k-1; i++)
            suma = suma + l[k][i] * l[k][i];
        l[k][k] = sqrt(a[k][k] - suma);
        for (i = k+1; i<=n; i++)
        {
            suma = 0.0;
            for (j = 1; j<=k-1; j++)
                suma = suma + l[i][j] * l[k][j];
            l[i][k] = ( a[i][k] - suma ) / l[k][k];
        }
    }
    y[1]=b[1]/l[1][1];
    for(i=2;i<=n;i++)
    {
```

```
            suma=0.0;
            for(j=1;j<=i-1;j++)
                suma=suma+l[i][j]*y[j];
            y[i]=(b[i]-suma)/l[i][i];
        }
        x[n]=y[n]/l[n][n];
        for(i=n-1;i>=1;i–)
        {
            suma=0.0;
            for(j=i+1;j<=n;j++)
                suma=suma+l[j][i]*x[j];
        x[i]=(y[i]-suma)/l[i][i];
        }
        ScriereVect(x,n);
    }
```

1.5 Householder Factorization

Householder Factorization is a numerical method for solving symmetrical systems of Cramer type.

Definition 1.5.1. *The Cramer system:*

$$Ax = b \tag{1.5.1}$$

is called ***symmetric****, if the matrix A is symmetric, i.e. A has the property:*

$$A^T = A. \tag{1.5.2}$$

Definition 1.5.2. *The* ***Householder Factorization*** *of the matrix A means to determine a matrix U having the property that UAU is a symmetric and tridiagonal matrix T, i.e.*

$$UAU = T. \tag{1.5.3}$$

Householder factorization is based on the followings result:

Propostion 1.5.1. *For any quadratic matrix A of order n and symmetric, there exists a vector $v = (v_1, v_2, \ldots, v_n)^T$ such that the column vector $a^1 = Ae_1, e_1 = (1, 0, \ldots, 0)^T$ (a^1 represents the first column of the matrix A) has the property:*

$$a^1 - \frac{2v \cdot < v, a^1 >}{\|v\|^2} = \lambda \cdot e_1. \tag{1.5.4}$$

Proof: Let be v a vector of the form:

$$v = a^1 + \alpha \cdot e_1 \tag{1.5.5}$$

where α is an unknown real constant; this constant will be determined by imposing to the vector $v = a^1 + \alpha \cdot e_1$ to satisfy the condition (1.5.4). Replacing the vector $v = a^1 + \alpha \cdot e_1$ in the equality (1.5.4) we find

$$\frac{a^1 \cdot [\|v\|^2 - 2(\|a^1\|^2 + \alpha \cdot a_{11})] - 2\alpha \cdot e_1 \cdot (\|a^1\|^2 + \alpha \cdot a_{11})}{\|v\|^2} = \lambda \cdot e_1. \tag{1.5.6}$$

and hence we obtain the equality:

$$\|v\|^2 - 2(\|a^1\|^2 + \alpha \cdot a_{11}) = 0. \tag{1.5.7}$$

Taking into account the equality:

$$\|v\|^2 = \|a^1\|^2 + 2\alpha \cdot a_{11} + \alpha^2 \tag{1.5.8}$$

we have:

$$\alpha = \pm\|a^1\|. \tag{1.5.9}$$

Choosing $\alpha = \pm\|a^1\|$ and $v = a^1 + \alpha \cdot e_1$ it is easier to see that the equality (1.5.4) is verified. □

Remark 1.5.1. In the following, we consider the vector v (called Householder vector) given by:

$$v = a^1 + sign(a_{11}) \cdot \|a^1\| \cdot e_1. \tag{1.5.10}$$

Remark 1.5.2. The Proposition 1.5.1 establishes the fact that, for any symmetric matrix A, the components $2, 3, \ldots, n$ of the vector

$$\frac{2v \cdot < v, a^1 >}{\|v\|^2} \tag{1.5.11}$$

coincide with those of the vector a^1 (the first column of the matrix A), and hence the components $2, 3, \ldots, n$ of the vector:

$$a^1 - \frac{2v \cdot < v, a^1 >}{\|v\|^2} \tag{1.5.12}$$

are null.

Propostion 1.5.2. *For any symmetric matrix A, the matrix P defined by*

$$P = I - \frac{2 \cdot v \cdot v^T}{\|v\|^2} \tag{1.5.13}$$

is symmetric and it has the property that the components $2, 3, \ldots, n$ of the first column of the matrix PA are null, where we denoted by v the vector defined by $v = a^1 + sign(a_{11}) \cdot \|a^1\| \cdot e_1$, i.e. the matrix $v \cdot v^T$ is:

$$v \cdot v^T = \begin{pmatrix} v_1^2 & v_1\, v_2 & \ldots & v_1\, v_n \\ v_2\, v_1 & v_2^2 & \ldots & v_2\, v_n \\ \ldots & \ldots & \ldots & \ldots \\ v_n\, v_1 & v_n\, v_2 & \ldots & v_n^2 \end{pmatrix} \tag{1.5.14}$$

Proof: The equality $P^T = P$ is immediate and hence P is a symmetric matrix.
The elements on the first column of the matrix PA are the components of the vector (1.5.12), and hence the components $2, 3, \ldots, n$ are null.

Definition 1.5.3. *We call **Householder matrix** of order $n-1$ associated with the matrix A, denoted by P_{n-1}, a matrix of the form:*

$$P_{n-1} = I_{n-1} - \frac{2 \cdot v \cdot v^T}{\|v\|^2} \tag{1.5.15}$$

where: $v = a^1_{n-1} + sign(a_{21}) \cdot \|a^1_{n-1}\| \cdot e_1$ represents the vector constituted with elements of the vector a^1 (the first column of the matrix A), $e_1 = (\underbrace{1, 0, \ldots, 0}_{n-1})^T$ and I_{n-1} is the unity matrix of order $n-1$.

Propostion 1.5.3. *The Householder matrix P_{n-1}, associated with the symmetric matrix A, is symmetric and it has the property that the matrix U_1 defined by:*

$$U_1 = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & & & \\ & & P_{n-1} & \\ 0 & & & \end{pmatrix} \tag{1.5.16}$$

is symmetric and verifies:

$$U_1\, A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ \alpha^1 & a^1_{22} & a^1_{23} & \ldots & a^1_{2n} \\ 0 & a^1_{32} & a^1_{33} & \ldots & a^1_{3n} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & a^1_{n2} & a^1_{n3} & \ldots & a^1_{nn} \end{pmatrix} \tag{1.5.17}$$

$$A^{(1)} = U_1 A U_1 = \begin{pmatrix} a_{11} & \alpha_1 & 0 & \dots & 0 \\ \alpha^1 & a_{22}^{(1)} & a_{23}^{(1)} & \dots & a_{2n}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & \dots & a_{3n}^{(1)} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & a_{n2}^{(1)} & a_{n3}^{(1)} & \dots & a_{nn}^{(1)} \end{pmatrix} \tag{1.5.18}$$

Proof: The symmetry of the Householder matrix P_{n-1} results from the definition. From the symmetry of P_n we obtain the symmetry of U_1. Relations (1.5.17) and (1.5.18) are verified by simple computations.

Remark 1.5.3. The Householder matrix of order $n-2$, denoted by P_{n-2}, is constructed with help of the column vector a_{n-2}^2 formed with the elements $n-2$ of the column matrix $A^{(1)}$. The corresponding matrix U_2 is defined by:

$$U_2 = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & & & \\ \vdots & \vdots & & P_{n-2} & \\ 0 & 0 & & & \end{pmatrix} \tag{1.5.19}$$

and it has the property:

$$A^{(2)} = U_2 A^{(1)} U_2 = \begin{pmatrix} a_{11} & \alpha_1 & 0 & 0 & \dots & 0 \\ \alpha_1 & a_{22}^{(1)} & \alpha_2 & 0 & \dots & 0 \\ 0 & \alpha_2 & a_{33}^{(2)} & a_{34}^{(2)} & \dots & a_{3n}^{(2)} \\ \dots & \dots & \dots & \dots & \dots & \\ 0 & 0 & a_{n3}^{(2)} & a_{n4}^{(2)} & \dots & a_{nn}^{(2)} \end{pmatrix} \tag{1.5.20}$$

Using the matrix P_{n-2}, we obtained a new row and a new column of the tridiagonal matrix at which we will reduce the matrix A.

Continuing by $n-1$ transformations, we obtain the equality:

$$UAU = T \tag{1.5.21}$$

in which T is a tridiagonal matrix.

Theorem 1.5.1. *If $UAU = T$ then the solution of the system $Ax = b$ is given by:*

$$x = Uy \tag{1.5.22}$$

where y represents the solution of the system

$$Ty = Ub. \tag{1.5.23}$$

Proof: $Ax = b$ and $UAU = T \Rightarrow U^{-1}TU^{-1}x = b \Rightarrow TU^{-1}x = Ub$. Denoting $y = U^{-1}x \Rightarrow Ty = Ub$ we obtain $x = Uy$. □

Exercises

1. Determine the solutions of the following systems, using Householder factorization:

a) $\begin{cases} 2x_1 + 2x_2 + x_3 = 2 \\ 2x_1 - x_2 + x_3 = \sqrt{5} \\ x_1 + x_2 + 2x_3 = 0 \end{cases}$

b) $\begin{cases} x_1 - x_2 + 2x_3 + 2x_4 = 0 \\ -x_1 + x_2 + 3x_3 = -1 \\ 2x_1 + 3x_2 - x_3 + 2x_4 = 2 \\ 2x_1 + 2x_3 = 1 \end{cases}$

The algorithm for the Householder factorization is:

for $i = 1 \ldots n-2$
  for $l = 1 \ldots n$
    for $m = 1 \ldots n$
      if $m = l$ then $u_{ml} = 1$
      if $m \neq l$ then $u_{ml} = 0$
  //We generate the vector v
  norm a $= \sqrt{\sum_{j=i+1}^{n} a_{ij}^2}$
  $e_{i+1} = 1$
  for $j = i+2 \ldots n$
    $e_j = 0$
  for $j = i+1 \ldots n$
    $v_j = a_{ij} + sign(a_{i,i+1}) \cdot$ norm a $\cdot e_j$
  norm v$= \sum_{j=i+1}^{n} v_j^2$
  //We generate the matrix U
  $j = i+1 \ldots n$
    $k = i+1 \ldots n$
      $u_{jk} = u_{jk} - 2 \cdot v_j \cdot v_k /$ norm v
  // D=AU
  $m = 1 \ldots n$
    $l = 1 \ldots n$
      $d_{ml} = \sum_{k=1}^{n} a_{mk} \cdot u_{kl}$
  //A=UD=UAU

$$m = 1 \ldots n$$
$$l = 1 \ldots n$$
$$a_{ml} = \sum_{k=1}^{n} u_{mk} \cdot d_{kl}$$

Using the above algorithm, we determine the solution of the system UAUy=Ub, and after that, we obtain the solution of the system Ax=b:

for i=1...n

$$x_i = \sum_{j=1}^{n} u_{ij} \cdot y_j$$

Input data:

- n - space dimension

- $(a_{ij})_{\substack{i=1,\ldots,n \\ j=1,\ldots,n}}$ - matrix of the system

- $(b_i)_{i=1,\ldots,n}$ - the column vector from right hand side

Output data:

- $(x_i)_{i=1,\ldots,n}$ - solution of the system $Ax = b$

Implementation of the above algorithm for the Householder factorization, in Borland C language, is:

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
float **Matrice(int imin, int imax, int jmin, int jmax);
float *Vector(int n);
void CitireMat(float **a, int n, int m);
void CitireVect(float *a, int n);
void ScriereMat(float **a, int n, int m);
void ScriereVect(float *a, int n);
void Householder(float **a,float *b, int n);
void main(void)
{
    float **a, *b;
    int n;
    a=Matrice(1,n,1,n);
    b=Vector(n);
    printf("n= ");
    scanf("%d", &n);
```

```
    CitireMat(a,n,n);
    ScriereMat(a,n,n);
    CitireVect(b,n);
    ScriereVect(b,n);
    printf("————————————SOLUTIE————————————\n");
    Householder(a,b,n);
}
void Householder(float **a, float *b, int n)
{
    int i,j,k;
    float **u, **c,xnorm, *e, vnorm, *v, suma;
    int sign;
    c=Matrice(1,n,1,n);
    u=Matrice(1,n,1,n);
    e=Vector(n);
    v=Vector(n);
    for(i=1;i<=n-2;i++)
    {
        for(j=1;j<=n;j++)
            for(k=1;k<=n;k++)
            {
                if(k==j) u[j][k]=1;
                else u[j][k]=0;
            }
        xnorm=0;
        for(j=i+1;j<=n;j++)
        {
            e[j]=0;
            xnorm=xnorm+a[i][j]*a[i][j];
        }
        e[i+1]=1;
        xnorm=sqrt(xnorm);
        vnorm=0;
        for(j=i+1;j<=n;j++)
        {
            if(a[i][i+1]>0) sign=1;
            else
            {
                if(a[i][i+1]<0) sign =-1;
                else sign=0;
            }
```

```
                v[j]=a[i][j]+sign*xnorm*e[j];
                vnorm=vnorm+v[j]*v[j];
            }
            for(j=i+1;j<=n;j++)
                for(k=i+1;k<=n;k++)
                    u[j][k]=u[j][k]-2*v[j]*v[k]/vnorm;
        }
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
            {
                c[i][j]=0.0;
                for(k=1;k<=n;k++)
                    c[i][j]+=a[i][k]*u[k][j];
             }
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
            {
                a[i][j]=0.0;
                for(k=1;k<=n;k++)
                    a[i][j]+=u[i][k]*c[k][j];
            }
    }
```

1.6 The Jacobi Method

The Jacobi method is an iterative method by successive approximations (iterations), to solve numerically Cramer systems.
The iteration technique is popular for finding roots of equations. Generalized fixed point iteration is applied to systems of linear equations, in order to produce accurate results.
Let be the Cramer system:

$$Ax = b. \tag{1.6.1}$$

We denote by a_{ij} the elements of the matrix A, $i, j = 1, \ldots, n$, and we consider the matrices D, L, U:

$$D = \begin{pmatrix} a_{11} & 0 & 0 & \ldots & 0 \\ 0 & a_{22} & 0 & \ldots & 0 \\ 0 & 0 & a_{33} & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & a_{nn} \end{pmatrix} \tag{1.6.2}$$

$$L = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ a_{21} & 0 & 0 & \dots & 0 \\ a_{31} & a_{32} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & 0 \end{pmatrix} \tag{1.6.3}$$

$$U = \begin{pmatrix} 0 & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & 0 & a_{23} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a_{n-1,n} \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix} \tag{1.6.4}$$

Remark 1.6.1. The matrix A can be written as:

$$A = D + L + U. \tag{1.6.5}$$

Theorem 1.6.1. *If $a_{ii} \neq 0$ for $i = 1, 2, \dots, n$, then $x^{(*)}$ is solution of the system (1.6.1) if and only if $x^{(*)}$ verifies:*

$$x^{(*)} = D^{-1}[b - (L + U)x^{(*)}]. \tag{1.6.6}$$

Proof: If $x^{(*)}$ is solution of the system (1.6.1), then $Ax^{(*)} = b$. From this, we obtain successively:

$$(L + D + U)x^{(*)} = b \Rightarrow Dx^{(*)} + (L + U)x^{(*)} = b \Rightarrow$$
$$\Rightarrow Dx^{(*)} = b - (L + U)x^{(*)} \Rightarrow x^{(*)} = D^{-1}[b - (L + U)x^{(*)}].$$

Reciprocally: if $x^{(*)}$ verifies (1.6.6) then:

$$x^{(*)} = D^{-1}[b - (L + U)x^{(*)}],$$

and hence:

$$Dx^{(*)} = b - (L + U)x^{(*)} \Rightarrow (D + L + U)x^{(*)} = b \Rightarrow Ax^{(*)} = b.$$

□

Definition 1.6.1. *Let be $x^{(0)} \in \mathbb{R}^n$ a given vector. The sequence of vectors:*

$$x^{(k+1)} = D^{-1}[b - (L + U)x^{(k)}] \qquad k = 0, 1, 2, \dots \tag{1.6.7}$$

*is called **Jacobi trajectory of vector** $\mathbf{x^{(0)}}$.*

Definition 1.6.2. *We say that the **Jacobi trajectory of the vector** $\mathbf{x^{(0)}}$ **converges**, if the sequence $x^{(k+1)}$ defined by (1.6.7) is convergent.*

In the case in which the Jacobi trajectory of the vector $x^{(0)}$ converges, this trajectory will be called Jacobi sequence of the successive approximations.

Theorem 1.6.2. *If the Jacobi trajectory of vector $x^{(0)}$ converges, then the limit of the Jacobi sequence, of the successive approximations, is solution of the system (1.6.1).*

Proof: We denote by $x^{(*)}$ the limit of the Jacobi sequence of the successive approximations. Passing to limit for $k \to \infty$ in the relation (1.6.7), we obtain the equality $x^{(*)} = D^{-1}[b - (L+U)x^{(*)}]$. On the base of the Theorem 1.6.1 we have that $x^{(*)}$ is a solution of the system (1.6.1). □

Theorem 1.6.3. *The Jacobi trajectory of vector $x^{(0)}$ converges, if and only if the sequence $y^{(k)}$ defined by:*

$$\begin{aligned} y^{(k+1)} &= -D^{-1}(L+U)y^{(k)} \qquad k = 1, 2, \ldots \\ y^{(0)} &= x^{(0)} - x^{(*)} \end{aligned} \tag{1.6.8}$$

converges to zero, where $x^{(*)}$ represents the solution of the system (1.6.1). The matrix $-D^{-1}(L+U)$ is called the Jacobi matrix.

Proof: We will prove that, the vector $x^{(k+1)}$ on the Jacobi trajectory of vector $x^{(0)}$ and the vector $y^{(k+1)}$ defined by (1.6.8) verify:

$$y^{(k+1)} = x^{(k+1)} - x^{(*)} \qquad k = 0, 1, 2, \ldots \tag{1.6.9}$$

For $k = 0$ we must to prove the equality $y^{(1)} = x^{(1)} - x^{(*)}$. For this aim, using (1.6.8) we compute $y^{(1)}$, and we obtain:

$$\begin{aligned} y^{(1)} &= -D^{-1}(L+U)y^{(0)} = -D^{-1}(L+U)x^{(0)} + D^{-1}(L+U)x^{(*)} = \\ &= x^{(1)} - D^{-1}b + D^{-1}(L+U)x^{(*)} = x^{(1)} - D^{-1}[b - (L+U)x^{(*)}] = \\ &= x^{(1)} - x^{(*)}. \end{aligned}$$

We suppose now that (1.6.9) is true for $k = l$:

$$y^{(l+1)} = x^{(l+1)} - x^{(*)},$$

and we show that this is true for $k = l + 1$, too, i.e.

$$y^{(l+2)} = x^{(l+2)} - x^{(*)}.$$

Thus, using (1.6.8) we compute $y^{(l+2)}$ and we obtain:

$$\begin{aligned} y^{(l+2)} &= -D^{-1}(L+U)y^{(l+1)} = -D^{-1}(L+U)x^{(l+1)} + D^{-1}(L+U)x^{(*)} = \\ &= x^{(l+2)} - D^{-1}b + D^{-1}(L+U)x^{(*)} = x^{(l+2)} - D^{-1}[b - (L+U)x^{(*)}] = \\ &= x^{(l+2)} - x^{(*)}. \end{aligned}$$

In this way, we proved that the relation (1.6.9) is true for any $k = 0, 1, 2, \ldots$.

From (1.6.9), we obtain that if $x^{(k+1)}$ converges then $y^{(k+1)}$ converges, too. Moreover, according to the Theorem (1.6.2) we have $\lim\limits_{k\to\infty} x^{(k+1)} = x^{(*)}$, and hence $\lim\limits_{k\to\infty} y^{(k+1)} = 0$. If $y^{(k+1)}$ converges to zero, from (1.6.9) we obtain that $x^{(k+1)}$ converges to $x^{(*)}$. □

Theorem 1.6.4. *The sequence of vectors, defined by:*

$$y^{(k+1)} = -D^{-1}(L+U)y^{(k)} \qquad k = 1, 2, \ldots \tag{1.6.10}$$

converges to zero for any $y^{(0)} \in I\!R^n$, if and only if the spectral radius ρ of the matrix $-D^{-1}(L+U)$ is strictly sub-unitary.

A sufficient condition for convergence to zero of a sequence $y^{(k+1)}$, for any $y^{(0)}$, is given by the next theorem in which the matrix norm is defined as follows:

$$\|A\| = \max_{\|x\|\neq 0} \frac{\|Ax\|}{\|x\|},$$

where A is a quadratic matrix of dimension n and $x = (x_1, x_2, ..., x_n)^T$.

Theorem 1.6.5. *If norm of the matrix $-D^{-1}(L+U)$ is strictly sub-unitary, then for any $y^{(0)} \in I\!R^n$, the sequence $y^{(k+1)}$ defined by (1.6.8), converges to zero.*

Proof: By mathematical induction, the following inequality is proved :

$$\|y^{(k+1)}\| \leq \| - D^{-1}(L+U)\|^{k+1} \cdot \|y^{(0)}\| \qquad k = 0, 1, 2, \ldots$$

Remark 1.6.2. If the spectral radius ρ of the matrix $-D^{-1}(L+U)$ is strictly sub-unitary, then for any $x^{(0)} \in I\!R^n$, the Jacobi sequence of the successive approximations converges to the solution of the system (1.6.1).

Remark 1.6.3. If norm of the matrix $-D^{-1}(L+U)$ is strictly sub-unitary, then for any $x^{(0)} \in I\!R^n$, the Jacobi sequence of the successive approximations converges to the solution of the system (1.6.1).

Theorem 1.6.6. *The components $x_1^{(k+1)}, \ldots, x_1^{(k+1)}$ of the vector x^{k+1} situated on the Jacobi trajectory of the vector $x^{(0)}$ are given by relations:*

$$x_i^{(k+1)} = \left(b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij} \cdot x_j^{(k)} \right) \cdot \frac{1}{a_{ii}} \qquad i = 1, 2, \ldots, n \quad k = 0, 1, \ldots \tag{1.6.11}$$

Exercises

1. Decide if the Jacobi method can be applied for solving the followings system:

$$\begin{cases} 5x_1 & - & 2x_2 & + & 3x_3 & = & -1 \\ -3x_1 & + & 9x_2 & + & x_3 & = & 2 \\ 2x_1 & - & x_2 & - & 7x_3 & = & 3 \end{cases}$$

The algorithm of the Jacobi method:

for $i = 1 \ldots n$
  $x_i = 0$
repeat
  for $i = 1 \ldots n$

$$x_i^{k+1} = \left(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} \cdot x_j^k \right) \cdot \frac{1}{a_{ii}}$$

until $\max\limits_{1 \leq i \leq n} |x_i^{k+1} - x_i^k| < \varepsilon$

Input data:

- n - space dimension
- $(a_{ij})_{\substack{i=1,\ldots,n \\ j=1,\ldots,n}}$ - matrix of the system
- $(b_i)_{i=1,\ldots,n}$ - the column vector form the right hand side

Output data:

- $(x_i)_{i=1,\ldots,n}$ - solution of the system $Ax = b$
- k - number of steps

1.7 The Gauss-Seidel Method

The Gauss-Seidel method is an iterative method of successive approximations (iterations), for solving Cramer systems numerically. The advantage of this method, is that it improves substantially the rate of convergence..
Considering the Cramer system:

$$Ax = b, \tag{1.7.1}$$

we decompose the matrix A as

$$A = L + D + U \tag{1.7.2}$$

where: L is a lower triangular matrix, D is a diagonal matrix and U is an upper triangular matrix.

Theorem 1.7.1. *If $a_{ii} \neq 0$ for $i = 1, 2, \ldots, n$ then $x^{(*)}$ is a solution of the system (1.7.1) if and only if $x^{(*)}$ verifies:*

$$x^{(*)} = (L + D)^{-1}(b - Ux^{(*)}). \tag{1.7.3}$$

Proof: If $x^{(*)}$ is a solution of the system (1.7.1) then $Ax^{(*)} = b$. We obtain successively:

$$(L + D)x^{(*)} + Ux^{(*)} = b \Rightarrow (L + D)x^{(*)} = b - Ux^{(*)} \Rightarrow x^{(*)} = (L + D)^{-1}(b - Ux^{(*)}).$$

Reciprocally: if $x^{(*)}$ verifies (1.7.3) then:

$$x^{(*)} = (L + D)^{-1}(b - Ux^{(*)}).$$

From this we obtain:

$$(L + D)x^{(*)} = b - Ux^{(*)} \Rightarrow (L + D)x^{(*)} + Ux^{(*)} = b \Rightarrow Ax^{(*)} = b.$$

□

Definition 1.7.1. *For a vector $x^{(0)} \in \mathbb{R}^n$, the sequence of vectors $x^{(k)}$ defined by:*

$$x^{(k+1)} = (L + D)^{-1}(b - Ux^{(k)}) \tag{1.7.4}$$

is called ***Gauss-Seidel trajectory of the vector*** $\mathbf{x^{(0)}}$.

Definition 1.7.2. *We say that* ***Gauss-Seidel trajectory of the vector*** $\mathbf{x^{(0)}}$ ***converges*** *if sequence $x^{(k+1)}$ defined by (1.7.4) converges.*

In the case in which the Gauss-Seidel trajectory of the vector $x^{(0)}$ converges, it is called Gauss-Seidel sequence of successive approximations.

Theorem 1.7.2. *If the Gauss-Seidel trajectory of the vector $x^{(0)}$ converges, then the limit of the Gauss-Seidel sequence of successive approximations is a solution of the system (1.7.1).*

Proof: We denote by $x^{(*)}$ the limit of the Gauss-Seidel sequence of the successive approximations. Passing to limit for $k \to \infty$ in relation (1.7.4) we obtain the equality $x^{(*)} = (L+D)^{-1}(b-Ux^{(*)})$. On the base of the Theorem 1.7.1, we have that $x^{(*)}$ is solution of the system (1.7.1). □

Theorem 1.7.3. *The Gauss-Seidel trajectory of the vector $x^{(0)}$ converges if and only if the sequence $y^{(k+1)}$ defined by:*

$$\begin{aligned} y^{(k+1)} &= -(L+D)^{-1}Uy^{(k)} \qquad k=1,2,\dots \\ y^{(0)} &= x^{(0)}-x^{(*)} \end{aligned} \tag{1.7.5}$$

converges to zero, where we denoted by $x^{(*)}$ the solution of the system (1.7.1). The matrix $-(L+D)^{-1}U$ is called the Gauss-Seidel matrix.

Proof: We will prove that, the vector $x^{(k+1)}$ on the trajectory of $x^{(0)}$, and the vector $y^{(k+1)}$ given by (1.7.5) verify:

$$y^{(k+1)} = x^{(k+1)} - x^{(*)} \qquad k=0,1,2,\dots \tag{1.7.6}$$

For $k=0$, the equality $y^{(1)} = x^{(1)} - x^{(*)}$ must be shown. Thus, computing $y^{(1)}$ with (1.7.5) we find:

$$\begin{aligned} y^{(1)} &= -(L+D)^{-1}Uy^{(0)} = -(L+D)^{-1}U(x^{(0)}-x^{(*)}) = \\ &= -(L+D)^{-1}Ux^{(0)} + (L+D)^{-1}Ux^{(*)} = \\ &= x^{(1)} - (L+D)^{-1}b + (L+D)^{-1}Ux^{(*)} = x^{(1)} - (L+D)^{-1}(b-Ux^{(*)}) = \\ &= x^{(1)} - x^{(*)}. \end{aligned}$$

Supposing that (1.7.6) is true for $k=l$:

$$y^{(l+1)} = x^{(l+1)} - x^{(*)},$$

we will show that this is true for $k=l+1$, i.e:

$$y^{(l+2)} = x^{(l+2)} - x^{(*)}.$$

We compute $y^{(l+2)}$ using (1.7.5) and we obtain:

$$\begin{aligned} y^{(l+2)} &= -(L+D)^{-1}Uy^{(l+1)} = -(L+D)^{-1}U(x^{(l+1)}-x^{(*)}) = \\ &= -(L+D)^{-1}Ux^{(l+1)} + (L+D)^{-1}Ux^{(*)} = \\ &= x^{(l+2)} - (L+D)^{-1}b + (L+D)^{-1}Ux^{(*)} = x^{(l+2)} - (L+D)^{-1}(b-Ux^{(*)}) = \\ &= x^{(l+2)} - x^{(*)}. \end{aligned}$$

Thus, we proved that (1.7.4) is true for any $k=0,1,2,\dots$.

From (1.7.6) we obtain that, if $x^{(k+1)}$ converges then $y^{(k+1)}$ converges, too. Moreover, from Theorem 1.7.2 we have $\lim\limits_{k\to\infty} x^{(k+1)} = x^{(*)}$, and hence $\lim\limits_{k\to\infty} y^{(k+1)} = 0$. Due to the fact that $y^{(k+1)}$ converges to zero, from (1.7.6) it results that $x^{(k+1)}$ converges to $x^{(*)}$. □

Theorem 1.7.4. *The sequence of vectors:*

$$y^{(k+1)} = -(L+D)^{-1}Uy^{(k)} \qquad k = 1, 2, \ldots \tag{1.7.7}$$

converges to zero for any $y^{(0)} \in I\!R^n$ if and only if the spectral radius ρ of the Gauss-Seidel matrix $-(L+D)^{-1}U$ is strictly sub-unitary.

A sufficient condition for the sequence $y^{(k+1)}$ to converge to zero, for any $y^{(0)}$, is given by the next theorem in which the matrix norm is defined as follows:

$$\|A\| = \max_{\|x\| \neq 0} \frac{\|Ax\|}{\|x\|},$$

where A is a quadratic matrix of dimension n and $x = (x_1, x_2, ..., x_n)^T$.

Theorem 1.7.5. *If norm of the matrix $-(L+D)^{-1}U$ is strictly sub-unitary then for any $y^{(0)} \in I\!R^n$, the sequence $y^{(k+1)}$ defined by (1.7.7) converges to zero.*

Proof: Using mathematical induction, the following inequality is proved:

$$\|y^{(k+1)}\| \leq \| - (L+D)^{-1}U\|^{k+1} \cdot \|y^{(0)}\| \qquad k = 0, 1, 2, \ldots$$

Remark 1.7.1. If the spectral radius ρ of the matrix $-(L+D)^{-1}U$ is strictly sub-unitary, then for any $x^{(0)} \in I\!R^n$, the Gauss-Seidel sequence of the successive approximations converges to the solution of the system (1.7.1).

Remark 1.7.2. If norm of the matrix $-(L+D)^{-1}U$ is strictly sub-unitary, then for any $x^{(0)} \in I\!R^n$, the Gauss-Seidel sequence of successive approximations converges to the solution of the system (1.7.1).

Propostion 1.7.1. *The points on the Gauss-Seidel trajectory of the vector $x^{(0)}$ verify:*

$$x^{(k+1)} = D^{-1}(b - Lx^{(k+1)} - Ux^{(k)}) \qquad k = 0, 1, 2, \ldots \tag{1.7.8}$$

Proof: Considering the equality (1.7.4): $x^{(k+1)} = (L+D)^{-1}(b - Ux^{(k)})$ we obtain successively:

$$\begin{aligned} (L+D)x^{(k+1)} &= b - Ux^{(k)} \Rightarrow \\ \Rightarrow Lx^{(k+1)} + Dx^{(k+1)} &= b - Ux^{(k)} \Rightarrow \\ \Rightarrow x^{(k+1)} &= D^{-1}(b - Lx^{(k+1)} - Ux^{(k)}). \end{aligned}$$

□

Consequence 1.7.1. *The components* $x_1^{(k+1)}, \ldots, x_1^{(k+1)}$ *of the vector* $x^{(k+1)}$ *on the Gauss-Seidel trajectory of the vector* $x^{(0)}$*, are given by the relations:*

$$x_1^{(k+1)} = \left(b_1 - \sum_{j=2}^{n} a_{1j} \cdot x_j^{(k)} \right) \cdot \frac{1}{a_{11}} \tag{1.7.9}$$

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} \cdot x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} \cdot x_j^{(k)} \right) \cdot \frac{1}{a_{ii}} \qquad i = 2, \ldots, n; \quad k = 0, 1, \ldots \tag{1.7.10}$$

Exercises

1. Search if the Jacobi and Gauss-Seidel methods can be applied for solving the followings system:

$$\begin{cases} 4x_1 + x_2 = -1 \\ 4x_1 + 3x_2 = -2 \end{cases}$$

The algorithm of the Gauss-Seidel method:

for $i = 1 \ldots n$
    $x_i = 0$
repeat
    for $i = 1 \ldots n$

$$x_i^{k+1} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} \cdot x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} \cdot x_j^{k} \right) \cdot \frac{1}{a_{ii}}$$

until $\max\limits_{1 \le i \le n} |x_i^{k+1} - x_i^k| < \varepsilon$

Input data:

- n - space dimension

- $(a_{ij})_{\substack{i=1,\ldots,n \\ j=1,\ldots,n}}$ - matrix of the system

- $(b_i)_{i=1,\ldots,n}$ - column vector from the right hand side

Output data:

- $(x_i)_{i=1,\ldots,n}$ - solution of the system $Ax = b$

- k - number of steps

1.8 Successive Over-relaxation (SOR) Method

The successive over-relaxation method is an iterative method for numerically solving systems of Cramer type, by successive approximations. The aim of this method, is to improve substantially the rate of convergence of the Gauss-Seidel method.
Considering the Cramer system:

$$Ax = b, \tag{1.8.1}$$

we decompose the matrix A as:

$$A = L + D + U, \tag{1.8.2}$$

where L is a lower triangular matrix, D is a diagonal matrix, and U is an upper triangular matrix.

If $a_{ii} \neq 0$ for $i = 1, 2, \ldots, n$, then for any $x^{(0)} \in I\!R^n$ we consider the Gauss-Seidel trajectory of the vector $x^{(0)} \in I\!R^n$ defined by:

$$x^{(k+1)} = (L + D)^{-1}(b - Ux^{(k)}) \qquad k = 0, 1, 2, \ldots \tag{1.8.3}$$

The points of this trajectory verify the relations:

$$x^{(k+1)} = D^{-1}(b - Lx^{(k+1)} - Ux^{(k)}) \qquad k = 0, 1, 2, \ldots \tag{1.8.4}$$

Definition 1.8.1. *The vectors $\Delta x^{(k)}$ defined by:*

$$\Delta x^{(k)} = x^{(k+1)} - x^{(k)} \qquad k = 0, 1, 2, \ldots \tag{1.8.5}$$

are called ***corrections.***

In terms of corrections, the relations (1.8.4) are written in the form:

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)} = x^{(k)} + D^{-1}(b - Lx^{(k+1)} - (U + D)x^{(k)}) \qquad k = 0, 1, 2, \ldots \tag{1.8.6}$$

Definition 1.8.2. *Let be $\omega > 0$. The sequence defined by:*

$$x^{(k+1)} = x^{(k)} + \omega\Delta x^{(k)} = x^{(k)} + \omega D^{-1}(b - Lx^{(k+1)} - (U + D)x^{(k)}) \qquad k = 0, 1, 2, \ldots \tag{1.8.7}$$

is called ***trajectory of the vector $\mathbf{x^{(0)}}$ obtained by successive relaxations.***

Remark 1.8.1. The relations (1.8.7) define the **successive relaxation method** and is called the **under-relaxation method** if $\omega < 1$, and the **over-relaxation method** if $\omega > 1$. For $\omega = 1$, the relation (1.8.7) is reduced to the Gauss-Seidel successive approximation (1.7.4).

Theorem 1.8.1. *The vectors $x^{(k)}$ on the trajectory of $x^{(0)}$ obtained by successive over-relaxation, verify:*

$$x^{(k+1)} = \left(L + \frac{1}{\omega} D\right)^{-1} \left\{b - \left[\left(1 - \frac{1}{\omega}\right) D + U\right] x^{(k)}\right\} \qquad k = 0, 1, 2, \ldots \tag{1.8.8}$$

Proof: Considering the equality (1.8.7):

$$x^{(k+1)} = x^{(k)} + \omega D^{-1}(b - Lx^{(k+1)} - (U + D)x^{(k)}) \qquad k = 0, 1, 2, \ldots$$

from which we obtain:

$$\frac{1}{\omega} Dx^{(k+1)} = \frac{1}{\omega} Dx^{(k)} + b - Lx^{(k+1)} - (U + D)x^{(k)},$$

$$\left(L + \frac{1}{\omega} D\right) x^{(k+1)} = b - \left[\left(1 - \frac{1}{\omega}\right) D + U\right] x^{(k)},$$

$$x^{(k+1)} = \left(L + \frac{1}{\omega} D\right)^{-1} \left\{b - \left[\left(1 - \frac{1}{\omega}\right) D + U\right] x^{(k)}\right\}.$$

□

Definition 1.8.3. *We say that the **trajectory** of the vector $x^{(0)}$, obtained by successive over-relaxations **converges**, if the sequence $x^{(k+1)}$ defined by (1.8.8) is convergent.*

In the case in which the trajectory of vector $x^{(0)}$ obtained by successive over-relaxations, converges then it is called sequence of successive approximations.

Theorem 1.8.2. *If the trajectory of the vector $x^{(0)}$, obtained by successive over-relaxations, converges then the limit of the sequence of successive approximations is solution of the system (1.8.1).*

Proof: Suppose that the trajectory of the vector $x^{(0)}$, obtained by successive over-relaxations, converges to the vector $x^{(*)}$: $\lim\limits_{k \to \infty} x^{(k)} = x^{(*)}$. Passing to limit for $k \to \infty$ in (1.8.8) we obtain:

$$x^{(*)} = \left(L + \frac{1}{\omega} D\right)^{-1} \left\{b - \left[\left(1 - \frac{1}{\omega}\right) D + U\right] x^{(*)}\right\} \tag{1.8.9}$$

Form here, it results successively:

$$\left(L+\frac{1}{\omega}\,D\right)x^{(*)}=b-\left(1-\frac{1}{\omega}\right)\,Dx^{(*)}-U\,x^{(*)},$$

$$Lx^{(*)}+\frac{1}{\omega}\,Dx^{(*)}=b-Dx^{(*)}+\frac{1}{\omega}\,Dx^{(*)}-U\,x^{(*)},$$

$$(L+D+U)x^{(*)}=b,$$

$$Ax^{(*)}=b.$$

□

Theorem 1.8.3. *The trajectory of the vector $x^{(0)}$, obtained by successive over-relaxations, converges if and only if the sequence $y^{(k)}$ defined by:*

$$\begin{aligned} y^{(k+1)} &= -\left(L+\frac{1}{\omega}\,D\right)^{-1}\left[\left(1-\frac{1}{\omega}\right)\,D+U\right]\,y^{(k)} \qquad k=1,2,\\ y^{(0)} &= x^{(0)}-x^{(*)} \end{aligned} \tag{1.8.10}$$

converges to zero.

Proof: Using mathematical induction, we obtain that the vector $x^{(k+1)}$ on the trajectory of the vector $x^{(0)}$, and vector $y^{(k+1)}$ defined by (1.8.10) verify:

$$y^{(k+1)}=x^{(k+1)}-x^{(*)} \qquad k=0,1,2,\ldots \tag{1.8.11}$$

Theorem 1.8.4. *The sequence of the vectors $y^{(k+1)}$ defined by:*

$$y^{(k+1)}=-\left(L+\frac{1}{\omega}\,D\right)^{-1}\left[\left(1-\frac{1}{\omega}\right)\,D+U\right]\,y^{(k)} \qquad k=1,2,\ldots \tag{1.8.12}$$

converges to zero for any $y^{(0)}\in\mathbb{R}^n$ if and only if the spectral radius ρ of the matrix
$-\left(L+\frac{1}{\omega}\,D\right)^{-1}\left[\left(1-\frac{1}{\omega}\right)\,D+U\right]$ *is strictly sub-unitary.*

A sufficient condition for the sequence $y^{(k+1)}$ to converge to zero, for any $y^{(0)}$, is given by the next theorem in which the matrix norm is defined as follows:

$$\|A\|=\max_{\|x\|\neq 0}\frac{\|Ax\|}{\|x\|},$$

where A is a quadratic matrix of dimension n and $x=(x_1,x_2,...,x_n)^T$.

Theorem 1.8.5. *If the norm of the matrix* $-\left(L+\frac{1}{\omega}\,D\right)^{-1}\left[\left(1-\frac{1}{\omega}\right)D+U\right]$ *is strictly sub-unitary, then for any* $y^{(0)}\in I\!R^n$ *the sequence* $y^{(k+1)}$ *defined by (1.8.12) tends to zero.*

Remark 1.8.2. If the spectral radius ρ of the matrix $-\left(L+\frac{1}{\omega}\,D\right)^{-1}\left[\left(1-\frac{1}{\omega}\right)\,D+U\right]$ is strictly sub-unitary, then for any $x^{(0)}\in I\!R^n$, the successive approximations sequence, obtained by successive over-relaxations, converges to the solution of the system (1.8.1).

Remark 1.8.3. If the norm of the matrix $-\left(L+\frac{1}{\omega}\,D\right)^{-1}\left[\left(1-\frac{1}{\omega}\right)\,D+U\right]$ is strictly sub-unitary, then for any $x^{(0)}\in I\!R^n$, the successive approximations sequence, obtained by successive over-relaxations, converges to the solution of the system (1.8.1).

Theorem 1.8.6. *(Kahan)*
The successive relaxation method converges only if ω *is chosen in the interval* $0<\omega<2$.

Proof: Outline
Since the spectral radius of the matrix $-\left(L+\frac{1}{\omega}\,D\right)^{-1}\left[\left(1-\frac{1}{\omega}\right)\,D+U\right]$ should be sub-unitary, the definition of the spectral radius, and the properties of the determinants we have: $max|\lambda_i|\geq|1-\omega|$ and $|1-\omega|<1$.
Moreover, the convergence speed is maximized by selecting ω such that the spectral radius to be minimum (i.e. ω should have a maximal value).

Theorem 1.8.7. *The components* $x_1^{(k+1)},\ldots,x_1^{(k+1)}$ *of the vector* x^{k+1} *on the trajectory of the vector* $x^{(0)}$*, obtained by successive over-relaxations, verify the relations:*

$$x_1^{(k+1)}=(1-\omega)\cdot x_1^{(k)}+\frac{\omega}{a_{11}}\left[b_1-\sum_{j=1}^{n}a_{1j}\cdot x_j^{(k)}\right] \tag{1.8.13}$$

$$x_i^{(k+1)}=(1-\omega)\cdot x_i^{(k)}+\frac{\omega}{a_{ii}}\left[b_i-\sum_{j=1}^{i-1}a_{ij}\cdot x_j^{(k+1)}-\sum_{j=i}^{n}a_{ij}\cdot x_j^{(k)}\right] \tag{1.8.14}$$

$$i=2,\ldots,n;\quad k=0,1,\ldots$$

Exercises

Decide if the Jacobi, Gauss-Seidel and successive over-relaxations methods methods can be applied for solving the followings system:

$$\begin{cases} 2x_1 + x_2 = 3 \\ 4x_1 + 3x_2 = -5 \end{cases}$$

The algorithm for implementation of the successive over-relaxation:

for $i = 1 \ldots n$
  $x_i = 0$
repeat
  for $i = 1 \ldots n$

$$x_i^{k+1} = (1-\omega) \cdot x_i^k + \frac{\omega}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} \cdot x_j^{k+1} - \sum_{j=i}^{n} a_{ij} \cdot x_j^k \right]$$

until $\max\limits_{1 \le i \le n} |x_i^{k+1} - x_i^k| < \varepsilon$

Input data:

- n - space dimension

- $(a_{ij})_{\substack{i=1,\ldots,n \\ j=1,\ldots,n}}$ - matrix of the system

- $(b_i)_{i=1,\ldots,n}$ - the column vector from the right hand side

- ω

Output data:

- $(x_i)_{i=1,\ldots,n}$ - solution of the system $Ax = b$

- k - step number

Implementation of the algorithms Jacobi, Gauss-Seidel and successive over-relaxation, in Borland C language:

```
#include<stdio.h>
#include<malloc.h>
#include<conio.h>
#include <math.h>
float **Matrice(int imin, int imax, int jmin, int jmax);
float *Vector(int n);
void CitireVect(float *a, int n);
void ScriereVect(float *a, int n);
void CitireMat(float **a, int n, int m);
```

```
void ScriereMat(float **a, int n, int m);
int gauss_seidel(float **a, float *b,int n,float *xi);
int jacobi(float **a, float *b,int n, float *xi);
int relaxare_succesiva(float **a, float *b,int n,float *xi);
void main(void)
{
    float **a, *b,*xgs, *xjac, *xrel,omega;
    int i,n, nr_pasi_gs=0,nr_pasi_jac=0,nr_pasi_rel=0;
    printf("n="); scanf("%d",&n);
    a=Matrice(1,n,1,n);
    b=Vector(n);
    xgs=Vector(n);
    xjac=Vector(n);
    xrel=Vector(n);
    CitireMat(a,n,n);
    ScriereMat(a,n,n);
    CitireVect(b,n);
    ScriereVect(b,n);
    nr_pasi_gs = gauss_seidel(a,b,n,xgs);
    nr_pasi_jac = jacobi(a,b,n,xjac);
    nr_pasi_rel = relaxare_succesiva(a,b,n,xrel);
    printf("————————————-SOLUTIE————————————-\n");
    printf("Gauss-Seidel\t\tJacobi\t\t\tRelaxare-succesiva\n");
    for(i=1; i<=n; i++)
    printf("Xgs[%d] = %lf\tXj[%d]= %lf\t\tXgs[%d] = %lf\n", i, xgs[i],
    i, xjac[i], i, xrel[i]);
    printf("Numarul de pasi:\n");
    printf("Gauss-Seidel\tJacobi\tRel-succ\n%d\t\t%d\t%d\n",
       nr_pasi_gs, nr_pasi_jac, nr_pasi_rel);
    printf("Apasati o tasta pt. a termina");
    getch();
}
double epsilon = 0.0000000001;
float max(float zi[], float xi[],int n); /* returneaza maximul dintre |zi-xi|
*/
int gauss_seidel(float **a, float *b,int n, float *xi)
{
    float *z,suma;
    int i,j,k=0;
    z=Vector(n);
    for(i=1; i<=n; i++)
```

```
        xi[i] = 0;
    do
    {
        for(i=1; i<=n; i++)
            z[i] = xi[i];
        for(i=1; i<=n; i++)
        {
            suma=0.0;
            for(j=1; j<=n; j++)
                if (i != j)
                    suma += a[i][j]*xi[j];
            xi[i] = 1.0/a[i][i]*(b[i] - suma);
        }
        k++;
    }while (max(z, xi,n) >= epsilon);
    return k;
}
int jacobi(float **a, float *b,int n,float *xi)
{
    float *z,suma;
    int k=0, i,j;
    z=Vector(n);
    for(i=1; i<=n; i++)
        xi[i] = 0;
    do
    {
        for(i=1; i<=n; i++)
            z[i] = xi[i];
        for(i=1; i<=n; i++)
        {
            suma=0.0;
            for(j=1; j<=n; j++)
                if (i != j)
                    suma += a[i][j]*z[j];
            xi[i] = 1.0/a[i][i]*(b[i] - suma);
         }
        k++;
    }while (max(z, xi,n) >= epsilon);
    return k;
}
int relaxare_succesiva(float **a, float *b,int n,float *xi)
```

```
{
    float *z,suma,omega;
    int k=0, i,j;
    printf("omega="); scanf("%f",&omega);
    for(i=1; i<=n; i++)
        xi[i] = 0;
    do
    {
        for(i=1; i<=n; i++)
            z[i] = xi[i];
        for(i=1; i<=n; i++)
        {
            suma=0.0;
            for(j=1; j<=n; j++)
                if (i != j)
                    suma += a[i][j]*xi[j];
            xi[i] = (1-omega)*z[i] + omega/a[i][i]*(b[i] - suma);
        }
        k++;
    }while (max(z, xi,n) >= epsilon);
return k;
}
float max(float zi[], float xi[], int n)
/* returneaza maximul dintre |zi-xi| */
{
    int i;
    float maxim = fabs(zi[1] - xi[1]);
    for(i=2, maxim = fabs(zi[1] - xi[1]); i<=n; i++)
        if(maxim < fabs(zi[i]-xi[i]))
    maxim = fabs(zi[i]-xi[i]);
return maxim;
}
```

Chapter 2

Numerical Solutions of Equations and Systems of Nonlinear Equations

Definition 2.0.1. *A* ***nonlinear system*** *of n algebraic equations and n unknowns, is a system of the form:*

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ \ldots\ldots\ldots\ldots\ldots & \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{cases} \tag{2.0.1}$$

or

$$F(x) = 0, \tag{2.0.2}$$

where: $F(x)$ represents the vector $(f_1(x), \ldots, f_n(x))^T$ in which the functions $f_1, \ldots, f_n : D \subset I\!R^n \to I\!R^1$ are given, and x is the vector $(x_1, x_2, \ldots, x_n)^T$.

A solution of a system (2.0.1) is an ordered system of numbers $x_1^*, \ldots, x_n^*$ which, replaced in the system, verify the equalities (2.0.1).

2.1 Fixed-Point Iterative Method

The system (2.0.1) is transformed into the system:

$$\begin{cases} g_1(x_1, x_2, \ldots, x_n) = x_1 \\ \quad\ldots\ldots\ldots\ldots\ldots \\ g_n(x_1, x_2, \ldots, x_n) = x_n, \end{cases} \tag{2.1.1}$$

if we choose the functions $g_1, \ldots, g_n : D \subset I\!R^n \to I\!R^1$, for example, of the form:

$$g_i(x_1, \ldots, x_n) = f_i(x_1, \ldots, x_n) + x_i.$$

The ordered system of numbers $x_1^*, \ldots, x_n^*$ is a solution of the system (2.0.1) if and only if it is a solution of the system (2.1.1).

In the following, we consider the nonlinear systems written as (2.1.1) or in the vector form:

$$G(x) = x \tag{2.1.2}$$

where: x is the vector $(x_1, x_2, \ldots, x_n)^T$, and $G(x)$ is the vector $(g_1(x), \ldots, g_n(x))^T$.

The function $G : D \subset I\!R^n \to I\!R^n$ defined by:

$$G(x) = (g_1(x), \ldots, g_n(x))^T \qquad \text{for } x^0 \in D, \tag{2.1.3}$$

is called nonlinear operator.

Remark 2.1.1. The operator $G(x)$ can be expressed using the operator $F(x)$ defined by (2.0.2), as follows:

$$G(x) = x - [F'(x)]^{-1} \cdot F(x),$$

where $F'(x)$ represents the Jacobi matrix which is supposed continuous and invertible.
This formula can be deduced from the equivalences:

$$F(x) = 0 \Leftrightarrow [F'(x)]^{-1} \cdot F(x) = 0 \Leftrightarrow x - [F'(x)]^{-1} \cdot F(x) = x \Leftrightarrow G(x) = x.$$

Definition 2.1.1. *The vector $x^{(*)} \in D$ is called **fixed point** for the operator G if:*

$$G(x^{(*)}) = x^{(*)}. \tag{2.1.4}$$

It is evident that $x^{(*)}$ is a fixed point for G if and only if $x^{(*)}$ is a solution of the system (2.1.1) equivalent with (2.0.1).

In the followings, we suppose that the operator G has a fixed point $x^{(*)}$. To find the fixed point $x^{(*)}$ of the operator G (i.e. the solution $x^{(*)}$ of the system (2.0.1)), we use the algorithm:

$$x^{(k+1)} = G(x^k), \quad k = 0, 1, 2, \ldots \qquad x^{(0)} \in D \tag{2.1.5}$$

with x^0 given.

Definition 2.1.2. *The point $x^{(*)}$ is called **attractor** if there is an open sphere $S(x^{(*)}, r) = \{x \in I\!R^n \mid \|x - x^{(*)}\| < r\}$ having the properties:*

1. *$S(x^{(*)}, r) \subset D$ and $x^{(k)}$ obtained from (2.1.5) is well defined for $\forall\, x^{(0)} \in S(x^{(*)}, r)$;*
2. *$\forall\, x^{(0)} \in S(x^{(*)}, r)$ the sequence $x^{(k)}$ defined of (2.1.5) belongs to D and $x^{(k)} \xrightarrow[k \to \infty]{} x^{(*)}$.*

Remark 2.1.2. If $x^{(*)}$ is an attractor, then $x^{(k)}$ is called the successive approximation sequence of the fixed point $x^{(*)}$.

Theorem 2.1.1. *Let $G : D \subset \mathbb{R}^n \to \mathbb{R}^n$ a nonlinear operator and $x^{(*)} \in D$ a fixed point of G. If there is an open sphere $S(x^{(*)}, r) \subset D$ and a constant $\alpha \in (0, 1)$ such that:*

$$\|G(x) - x^{(*)}\| \leq \alpha \|x - x^{(*)}\|, \qquad \forall\, x \in S(x^{(*)}, r)$$

then $x^{(*)}$ is an attractor and

$$\limsup_{k\to\infty} \frac{\|x^{(k+1)} - x^{(*)}\|}{\|x^{(k)} - x^{(*)}\|} \leq \alpha,$$

$$\limsup_{k\to\infty} \|x^{(k)} - x^{(*)}\|^{1/k} \leq \alpha.$$

Proof: For $x^{(0)} \in S(x^{(*)}, r)$ we consider $x^{(1)} = G(x^{(0)})$. From the inequality

$$\|G(x^{(0)}) - x^{(*)}\| \leq \alpha \|x^{(0)} - x^{(*)}\|$$

it results

$$\|x^{(1)} - x^{(*)}\| \leq \alpha \|x^{(0)} - x^{(*)}\| < \alpha \cdot r < r,$$

from which we obtain:

$$x^{(1)} \in S(x^{(*)}, r) \subset D.$$

After that, we consider $x^{(2)} = G(x^{(1)})$, and from

$$\|G(x^{(1)}) - x^{(*)}\| \leq \alpha \|x^{(1)} - x^{(*)}\| \leq \alpha^2 \|x^{(0)} - x^{(*)}\| < \alpha^2 \cdot r < r$$

we obtain

$$x^{(2)} \in S(x^{(*)}, r) \subset D.$$

In the followings, we consider $x^{(3)} = G(x^{(2)})$, and using a similar evaluation we have:

$$\|x^{(3)} - x^{(*)}\| \leq \alpha^3 \|x^{(0)} - x^{(*)}\| < \alpha^3 \cdot r < r.$$

By mathematical induction it is shown that $x^{(k+1)} = G(x^{(k)})$ is well defined and the terms of this sequence verify:

$$\|x^{(k)} - x^{(*)}\| \leq \alpha^k \|x^{(0)} - x^{(*)}\| \qquad \forall\, k = 1, 2, \ldots$$

Because $\alpha \in (0, 1)$, the sequence α^k converges to 0 for $k \to \infty$, from which we obtain that $x^{(k)}$ converges to $x^{(*)}$ for $k \to \infty$.

If there exists k_0 such that $x^{(k)} = x^{(*)}$ for $k \geq k_0$, then $\limsup_{k\to\infty} \|x^{(k+1)} - x^{(*)}\| = 0 \leq \alpha$.
If $x^{(k)} \neq x^{(*)}$ for $k \geq k_0$, then the inequality $\|x^{(k+1)} - x^{(*)}\| \leq \alpha \|x^{(k)} - x^{(*)}\|$ shows that

$$\frac{\|x^{(k+1)} - x^{(*)}\|}{\|x^{(k)} - x^{(*)}\|} \leq \alpha,$$

and hence

$$\limsup_{k\to\infty} \frac{\|x^{(k+1)} - x^{(*)}\|}{\|x^{(k)} - x^{(*)}\|} \leq \alpha.$$

From $\|x^{(k)} - x^{(*)}\| \leq \alpha^k \|x^{(0)} - x^{(*)}\|$, the following inequality results:

$$\limsup_{k\to\infty} \|x^{(k)} - x^{(*)}\|^{1/k} \leq \alpha.$$

□

Theorem 2.1.2. *Let $G : D \subset \mathbb{R}^n \to \mathbb{R}^n$ be a nonlinear operator and $x^{(*)} \in D$ a fixed point of G. If G is of $\mathcal{C}^1$-class on D, and the spectral radius ρ of the Jacobi matrix in $x^{(*)}$, associated to the operator G, is strictly sub-unitary ($\rho < 1$), then $x^{(*)}$ is an attractor and*

$$\limsup_{k\to\infty} \|x^{(k)} - x^{(*)}\|^{1/k} = \rho.$$

Theorem 2.1.3. *Let $G : D \subset \mathbb{R}^n \to \mathbb{R}^n$ be a nonlinear operator and $x^{(*)} \in D$ a fixed point of G. If G is of $\mathcal{C}^1$-class on D, and the norm μ of the Jacobi matrix in $x^{(*)}$, associated to the operator G, is strictly sub-unitary ($\mu < 1$), then $x^{(*)}$ is an attractor and*

$$\limsup_{k\to\infty} \|x^{(k)} - x^{(*)}\|^{1/k} = \mu.$$

Theorem 2.1.4. *(Fixed-Point)*
Let $G : D \subset \mathbb{R}^n \to \mathbb{R}^n$ a nonlinear operator. If G is of $\mathcal{C}^1$-class on D, and the norm μ of the Jacobi matrix is strictly sub-unitary ($\mu < 1$) for any $x \in D$, then for any x^0 the sequence of iterates

$$x^{(k+1)} = G(x^k), \quad k = 0, 1, 2, \ldots \qquad x^{(0)} \in D$$

converges to a unique fixed point $x^{(*)} \in D$.

Consequence 2.1.1. *If the Jacobi matrix of G is null in $x^{(*)}$, then $x^{(*)}$ is an attractor and*

$$\limsup_{k\to\infty} \|x^{(k)} - x^{(*)}\|^{1/k} = 0 = \limsup_{k\to\infty} \frac{\|x^{(k+1)} - x^{(*)}\|}{\|x^{(k)} - x^{(*)}\|}.$$

2.2 The Newton Method

The Newton method is an iterative method based on successive approximations, for solving systems of n algebraic equations having n unknowns. We consider the system:

$$F(x) = 0 \tag{2.2.1}$$

in which $F : D \subset I\!R^n - I\!R^n$ is a nonlinear operator. We admit that (2.2.1) has a solution $x^{(*)} \in D$, F is of $\mathcal{C}^1$-class on D and $F'(x)$ is invertible for any $x \in D$.

Definition 2.2.1. *If for a vector $x^{(0)} \in D$ the sequence:*

$$x^{(k+1)} = x^{(k)} - [F'(x^{(k)})]^{-1} \cdot F(x^{(k)}), \qquad k = 0, 1, 2, \ldots \tag{2.2.2}$$

is well defined, then it is called ***the classical sequence of successive iterations of Newton****; we denoted by $F'(x^{(k)})$ the Jacobi matrix in the point $x^{(k)}$ which is supposed continuous and invertible.*

Definition 2.2.2. *If for a vector $x^{(0)} \in D$ the sequence:*

$$x^{(k+1)} = x^{(k)} - [F'(x^{(0)})]^{-1} \cdot F(x^{(k)}), \qquad k = 0, 1, 2, \ldots \tag{2.2.3}$$

is well defined, then it is called ***the simplified sequence of successive iterations of Newton****.*

Theorem 2.2.1. *If for a vector $x^{(0)} \in D$, the classical sequence of successive iterations of Newton converges, then the limit of the sequence $x^{(k)}$ is a solution of the equation (2.2.1).*

Proof: We denote by x the limit of the sequence $x^{(k)}$. Passing to limit for $k \to \infty$ in relation (2.2.2) we obtain:

$$x = x - [F'(x)]^{-1} \cdot F(x),$$

and hence $[F'(x)]^{-1} \cdot F(x) = 0$, i.e. $F(x) = 0$.

Theorem 2.2.2. *If for a vector $x^{(0)} \in D$, the simplified sequence of successive iterations of Newton converges, then the limit of the sequence $x^{(k)}$ is a solution of the equation (2.2.1).*

Proof: We denote by x the limit of the sequence $x^{(k)}$ and passing to limit for $k \to \infty$ in the relation (2.2.3) we obtain:

$$x = x - [F'(x^{(0)})]^{-1} \cdot F(x)$$

and hence $[F'(x^{(0)})]^{-1} \cdot F(x) = 0$, i.e. $F(x) = 0$.

□

Theorem 2.2.3. *Let $F : D \subset \mathbb{R}^n \to \mathbb{R}^n$ and the equation $F(x) = 0$ having the solution $x^{(*)} \in D$. If there is an open sphere $S(x^{(*)}, r) = \{x \in \mathbb{R}^n \mid \|x - x^{(*)}\| < r\}$ on which F is of $\mathcal{C}^1$-class and $F'(x^{(*)})$ is non-singular then in the case of the classical Newton method, $x^{(*)}$ is an attractor.*

Proof: The Consequence 2.1.1 from previous section is applied.

Theorem 2.2.4. *Under the hypothesis of the previous theorem, if the spectral radius ρ of the matrix*

$$I - [F'(x^{(0)})]^{-1} \cdot F'(x^{(*)})$$

is strictly sub-unitary then in the case of the simplified Newton method, $x^{(*)}$ is an attractive fixed point.

Proof: The Theorem 2.1.2 is applied.

Exercises

1. Using Fixed-Point and Newton methods, find the solutions of the following systems:

a) $$\begin{cases} \dfrac{x_1^2 + x_2}{6} - x_1 = 0 \\ \dfrac{x_1 + x_2^2}{8} - x_2 = 0 \end{cases} \qquad \text{on the domain} \quad D = [0,1] \times [1,2]$$

b) $$\begin{cases} x^3 - 20x - 1 = 0 \\ x^3 + xy - 10y + 10 = 0 \end{cases} \qquad \text{on the domain} \quad D = [0,1] \times [1,2]$$

The algorithm of the classical Newton method in the case $n = 2$ is the following:

//Computation of the Jacobi matrix
h=0.0001

$$jac_{11} = \frac{f_1(x_1 + h, x_2) - f_1(x_1, x_2)}{h}$$

$$jac_{21} = \frac{f_2(x_1 + h, x_2) - f_2(x_1, x_2)}{h}$$

$$jac_{12} = \frac{f_1(x_1, x_2 + h) - f_1(x_1, x_2)}{h}$$

$$jac_{22} = \frac{f_2(x_1, x_2 + h) - f_2(x_1, x_2)}{h}$$

// Classical Newton method
for $i = 1 \dots n$
$x_i = x_i^0$
repeat
- the Jacobi matrix is computed in x^k
- a method for solving the linear system is used :
$J_F(x^k)z^k = -F(x^k)$
for $i = 1 \dots n$
$x_i^{k+1} = z_i^k + x_i^k$

until $\max\limits_{1 \le i \le n} |f_i(x^{k+1})| < \varepsilon$

Input data:

- $n = 2$ dimension for the presented case

- initial iteration $x^0 = (x_1^0, \dots, x_n^0)$

Output data:

- $(x_i)_{i=1,\dots,n}$ - the solution of the system

- k - number of the steps

2.3 Quasi-non-expansion Operators

Let $D \subset I\!R^n$, D be an open set and $G : D \subset I\!R^n \to I\!R^n$ a nonlinear operator, $x^{(*)} \in D$ a fixed point of G: $G(x^{(*)}) = x^{(*)}$.

Definition 2.3.1. *The operator G is called* ***quasi-non-expansive*** *on $\overline{S}(x^{(*)}, r) \subset D$, if:*

$$\|G(x) - x^{(*)}\| < \|x - x^{(*)}\|, \qquad \forall\, x \in S(x^{(*)}, r), \quad x \neq x^{(*)} \tag{2.3.1}$$

Remark 2.3.1. If G is quasi-non-expansive operator on $S(x^{(*)}, r)$ then $x^{(*)}$ is a unique fixed point in $\overline{S}(x^{(*)}, r)$.

Theorem 2.3.1. *If $G : D \subset I\!R^n \to I\!R^n$ is a continuous operator on D and $x^{(*)} \in D$ is a fixed point of G, G is quasi-non-expansive on $\overline{S}(x^{(*)}, r) \subset D$, then for any $x^{(0)} \in \overline{S}(x^{(*)}, r)$ the sequence $x^{(k+1)} = G(x^{(k)})$ converges to $x^{(*)}$ for $k \to \infty$.*

Proof: The inequality:

$$\|x^{(k+1)} - x^{(*)}\| = \|G(x^{(k)}) - x^{(*)}\| < \|x^{(k)} - x^{(*)}\| \qquad k = 0, 1, 2, \ldots$$

shows that the sequence $\|x^{(k)} - x^{(*)}\|$ decreases.
Denoting $L = \lim\limits_{k\to\infty} \|x^{(k)} - x^{(*)}\|$ and supposing that $L \neq 0$, we consider a subsequence $x^{(k_j)}$ of $x^{(k)}$ which converges to the vector $y^{(*)} \in \overline{S}(x^{(*)}, r)$, for which we have $\|y^{(*)} - x^{(*)}\| = L \neq 0$. Under these hypothesis we obtain:

$$L = \lim_{j\to\infty} \|G(x^{(k_j)}) - x^{(*)}\| = \|G(y^{(*)}) - x^{(*)}\| < \|y^{(*)} - x^{(*)}\| = L$$

which is absurd. It results that $L = 0$, i.e. $x^{(k)} \xrightarrow[k\to\infty]{} x^{(*)}$.

□

In the followings, we will show that the operator from the classical Newton iteration is quasi-non-expansive in a certain sphere, from which we obtain the local convergence of the Newton method.

Theorem 2.3.2. *Let $F : D \subset \mathbb{R}^n \to \mathbb{R}^n$ be a nonlinear operator and $x^{(*)}$ a solution of the equation $F(x) = 0$. We suppose that F is of $\mathcal{C}^1$-class on the sphere $\overline{S}(x^{(*)}, r) \subset D$ and that the followings conditions are satisfied:*

1. *$[F'(x)]^{-1}$ exists and $\|[F'(x)]^{-1}\| \leq \beta$, $\forall\, x \in \overline{S}$;*
2. *$\|F'(x) - F'(y)\| \leq \gamma \|x - y\|$, $\forall\, x, y \in \overline{S}$;*
3. *the constants β, γ, r satisfy the relation $\beta \cdot \gamma \cdot r < 1$.*

In these conditions, for any $x^{(0)} \in \overline{S}$, the sequence $x^{(k)}$ generated by the classical Newton method converges to $x^{(*)}$.

Proof: We consider the operator $G : S \subset D \to \mathbb{R}^n$ defined by

$$G(x) = x - [F'(x)]^{-1} \cdot F(x).$$

We show that G is quasi-non-expansive on S. Let $x \in S$ and $R(x) = F(x) - F'(x)(x - x^{(*)}) = F(x) - F(x^{(*)}) - F'(x)(x - x^{(*)})$. According to the theorem of the means we have:

$$\|R(x)\| = \|(F'(y) - F'(x))(x - x^{(*)})\| \leq \gamma \|y - x\| \cdot \|x - x^{(*)}\| \leq \gamma \cdot r \cdot \|x - x^{(*)}\|.$$

From here we obtain:

$$\|[F'(x)]^{-1} \cdot R(x)\| \leq \|[F'(x)]^{-1}\| \cdot \|R(x)\| \leq \beta \cdot \gamma \cdot r \|x - x^{(*)}\| < \|x - x^{(*)}\|, \quad \forall x \in \overline{S}, x \neq x^{(*)}.$$

We compute $\|G(x) - x^{(*)}\|$ and we find:

$$\begin{aligned}\|G(x) - x^{(*)}\| &= \|x - x^{(*)} - [F'(x)]^{-1} \cdot F(x)\| = \\ &= \|x - x^{(*)} - [F'(x)]^{-1} \cdot (F'(x)(x - x^{(*)}) + R(x))\| = \\ &= \|[F'(x)]^{-1} \cdot R(x)\| < \|x - x^{(*)}\|\end{aligned}$$

for any $x \in \overline{S}, x \neq x^{(*)}$.
It follows that G is quasi-non-expansive on S and according to Theorem 2.3.1 the sequence generated by the Newton method converges to $x^{(*)}$.

□

Theorem 2.3.3. *Under hypothesis of Theorem 2.3.2, the sequence generated by the iterations:*

$$x^{(k+1)} = x^{(k)} - \|F'(x^{(k)})\|^{-2} \cdot [F'(x^{(k)})]^T \cdot F(x^{(k)})$$

converges to $x^{(*)}$ for any $x^{(0)} \in S$; these iterations are called the gradient method.

Proof: As in the previous case we prove that the operator

$$G(x) = x - \|F'(x)\|^{-2} \cdot [F'(x)]^T \cdot F(x)$$

is quasi-non-expansive on $\overline{S}$. From here, on the basis of Theorem 2.3.1, the convergence of the sequence $x^{(k)}$ to $x^{(*)}$ results for any $x^{(0)} \in S$.
In this case, we consider:

$$R(x) = F(x) - F'(x)(x - x^{(*)}) = F(x) - F(x^{(*)}) - F'(x)(x - x^{(*)}).$$

Based on the previous theorem, $R(x)$ verifies

$$\|R(x)\| \leq \gamma \cdot r \cdot \|x - x^{(*)}\| \qquad \forall\, x \in \overline{S}.$$

Because $\beta\gamma r < 1$ we have $\gamma r < \frac{1}{\beta}$ and we can write:

$$\|R(x)\| < \frac{1}{\beta} \cdot \|x - x^{(*)}\|.$$

Taking into account the inequality $\|F'(x)^{-1}\| \leq \beta$ we obtain:

$$\|R(x)\| < \|F'(x)^{-1}\|^{-1} \cdot \|x - x^{(*)}\|$$

Because:

$$\|x - x^{(*)}\| = \|F'(x) \cdot [F'(x)]^{-1} \cdot (x - x^{(*)})\| \leq \|[F'(x)]^{-1}\| \cdot \|F'(x) \cdot (x - x^{(*)})\|$$

it results:

$$\|x - x^{(*)}\| \cdot \|[F'(x)]^{-1}\|^{-1} \leq \|F'(x) \cdot (x - x^{(*)})\|$$

hence:

$$\|R(x)\| < \|F'(x) \cdot (x - x^{(*)})\| \qquad \forall\, x \in \overline{S}.$$

Let's evaluate $\|G(x) - x^{(*)}\|^2$ for $x \in \overline{S}$ and $x \neq x^{(*)}$. We have:

$$\|G(x) - x^{(*)}\|^2 = \|x - \|F'(x)\|^{-2} \cdot [F'(x)]^T \cdot F(x) - x^{(*)}\|^2$$

$$= < x - x^{(*)} - \|F'(x)\|^{-2} \cdot [F'(x)]^T \cdot F(x), x - x^{(*)} - \|F'(x)\|^{-2} \cdot [F'(x)]^T \cdot F(x) >=$$

$$= < x - x^{(*)}, x - x^{(*)} > -2 \cdot < x - x^{(*)}, \|F'(x)\|^{-2} \cdot [F'(x)]^T \cdot F(x) > +$$

$$+ < \|F'(x)\|^{-2} \cdot [F'(x)]^T \cdot F(x), \|F'(x)\|^{-2} \cdot [F'(x)]^T \cdot F(x) > =$$

$$= \|x - x^{(*)}\|^2 - 2 \cdot < x - x^{(*)}, \|F'(x)\|^{-2} \cdot [F'(x)]^T \cdot F(x) > +$$

$$+ \ \|F'(x)\|^{-4} \cdot \|F'(x)^T \cdot F(x)\|^2 \ \leq$$

$$\leq \|x - x^{(*)}\|^2 - 2 \cdot \|F'(x)\|^{-2} \cdot < x - x^{(*)}, [F'(x)]^T \cdot F(x) > +$$

$$+ \ \|F'(x)\|^{-4} \cdot \|F'(x)^T\|^2 \cdot \|F(x)\|^2 \ \leq$$

$$\leq \|x - x^{(*)}\|^2 - \|F'(x)\|^{-2} < 2 \cdot F'(x)(x - x^{(*)}), F(x) >$$

$$+ \|F'(x)\|^{-2} \cdot \|F(x)\|^2.$$

Taking into account the equality:

$$R(x) = F(x) - F'(x)(x - x^{(*)})$$

we have:

$$\|R(x)\|^2 \ = \ \|F(x)\|^2 - 2 \cdot < F(x), F'(x)(x - x^{(*)}) > + \|F'(x)(x - x^{(*)})\|^2 \ .$$

It follows:

$$-2 \cdot < F(x), F'(x)(x - x^{(*)}) >= \|R(x)\|^2 - \|F(x)\|^2 - \|F'(x)(x - x^{(*)})\|^2.$$

We can conclude that for $\|G(x) - x^{(*)}\|^2$ we have:

$$\begin{aligned}\|G(x) - x^{(*)}\|^2 &\leq \|x - x^{(*)}\|^2 + \|F'(x)\|^{-2} \cdot [\|R(x)\|^2 - \|F(x)\|^2 - \|F'(x)(x - x^{(*)})\|^2] + \\ &+ \|F'(x)\|^{-2} \cdot \|F(x)\|^2 \\ &= [\|F'(x)^{-1}\| \cdot \|R(x)\|]^2 < \|x - x^{(*)}\|^2.\end{aligned}$$

□

Chapter 3

Interpolation, Polynomials Approximation, Spline Functions

Suppose that $X = \{x_i \mid x_i \in I\!R^1, i = 0, 1, \ldots, m\}$ is a set of $m+1$ distinct real numbers: $x_0 < x_1 < \ldots < x_{m-1} < x_m$ and a function $f : X \to I\!R^1$ known only by its values in x_i: $y_i = f(x_i), i = 0, 1, \ldots, m$.
Because we do not have an analytic expression of the function f, for calculating its value at an arbitrary point we should determine a function φ, called interpolating function, which takes values y_i in the points x_i: $y_i = \varphi(x_i), i = 0, 1, \ldots, m$.
The form of the approximating function will be chosen such that the conditions $y_i = \varphi(x_i), i = 0, 1, \ldots, m$ conduct to a linear system of algebraic equations. A possible choice is the following: we take a set of $m+1$ simple known functions $\varphi_i(x), i = 0, 1, \ldots, m$ and a set of $m+1$ unknown parameters $a_i, i = 0, 1, \ldots, m$. Using these we can write the approximating function:

$$\varphi(x) = \sum_{i=0}^{m} a_i \varphi_i(x).$$

In order to assure the uniqueness of the solution of the linear system, $m+1$ unknown parameters a_i and $m+1$ unknown distinct functions $\varphi_i(x)$ (linearly independent), were chosen.
In the followings, we will consider functions which have this property. A set of such functions is given by the monomials $x^i, i = 0, 1, \ldots, m$, and in this case the interpolating function is a polynomial of degree m:

$$\varphi(x) = \sum_{i=0}^{m} a_i x^i.$$

In this way, the problem of determining the interpolating function is reduced to the determination of the coefficients a_i from the interpolating condition,

i.e. it is reduced to solving the linear system:

$$\sum_{i=0}^{m} a_i x_j^i = y_j, j = 0, 1, \ldots, m$$

equivalent to:

$$\sum_{i=0}^{m} a_i x_j^i = f(x_j), j = 0, 1, \ldots, m,$$

i.e.,

$$\begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 + \ldots + a_m x_0^m = f(x_0) \\ a_0 + a_1 x_1 + a_2 x_1^2 + \ldots + a_m x_1^m = f(x_1) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_0 + a_1 x_m + a_2 x_m^2 + \ldots + a_m x_m^m = f(x_m). \end{cases}$$

This system has a unique solution because its determinant is different of zero (it's a determinant of Vandermonde type and x_i are distinct).
We can conclude that the interpolating polynomial is unique for a given function f and for given data points $x_0 < x_1 < \ldots < x_{m-1} < x_m$. This approximating polynomial is also called *global interpolation* due to the fact that only one polynomial is used on the interval $[a, b]$.

3.1 The Newton Divided Difference Formulas

An interpolating polynomial is given by the **Newton polynomial with divided differences**.

Definition 3.1.1. *For a given function $f : X \to \mathbb{R}^1$, **the first divided difference** of f at the point x_r is given by the number:*

$$(\mathcal{D}^1 f)(x_r) = \frac{f(x_{r+1}) - f(x_r)}{x_{r+1} - x_r}.$$

The first divided difference of f at the point x_r is denoted by $[x_r, x_{r+1}, f]$ or $f(x_r, x_{r+1})$:

$$[x_r, x_{r+1}, f] = f(x_r, x_{r+1}) = \frac{f(x_{r+1}) - f(x_r)}{x_{r+1} - x_r}.$$

Remark 3.1.1. We denote by $\mathcal{F}_m$ the set of real functions defined on X: $\mathcal{F}_m = \{f \mid f : X \to \mathbb{R}^1\}$. For a function $f \in \mathcal{F}_m$ we consider the divided differences:

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0}, \frac{f(x_2) - f(x_1)}{x_2 - x_1}, \ldots, \frac{f(x_m) - f(x_{m-1})}{x_m - x_{m-1}}$$

denoted by $\dfrac{f(x_{r+1})-f(x_r)}{x_{r+1}-x_r}, \quad r=0,1,\ldots,m-1.$

These divided differences constitute a set of m numbers attached to the points $x_0 < x_1 < \ldots < x_{m-1}$:

$$x_0 \longrightarrow \frac{f(x_1)-f(x_0)}{x_1-x_0} = [x_0, x_1, f]$$

$$x_1 \longrightarrow \frac{f(x_2)-f(x_1)}{x_2-x_1} = [x_1, x_2, f]$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$x_{m-1} \longrightarrow \frac{f(x_m)-f(x_{m-1})}{x_m-x_{m-1}} = [x_{m-1}, x_m, f].$$

In this way, we obtain a function $\mathcal{D}^1 f$ defined on the set $\{x_i \mid i=0,\ldots,m-1\}$. Thus, $\mathcal{D}^1 f$ is a real function defined on the set $\{x_0 < \ldots < x_{m-1}\}$ as follows:

$$(\mathcal{D}^1 f)(x_r) = [x_r, x_{r+1}, f], \qquad r=0,1,\ldots,m-1.$$

We consider the sets of functions $\mathcal{F}_m = \{f \mid f : X \to I\!R^1\}$ and $\mathcal{F}_{m-1} = \{f \mid f : \{x_0, x_1, \ldots, x_{m-1}\} \to I\!R^1\}$, and the operator $\mathcal{D}^1 : \mathcal{F}_m \to \mathcal{F}_{m-1}$ defined by

$$f \longmapsto \mathcal{D}^1 f$$

where $(\mathcal{D}^1 f)(x_r) = [x_r, x_{r+1}, f]$.

Definition 3.1.2. *The operator $\mathcal{D}^1 : \mathcal{F}_m \longrightarrow \mathcal{F}_{m-1}$ defined by $(\mathcal{D}^1 f)(x_r) = [x_r, x_{r+1}, f]$, $r=0,1,\ldots,m-1$ is called* ***the operator of the first divided difference.***

Propostion 3.1.1. *The operator of the first divided difference is a linear operator.*

Proof: Let be $f, g \in \mathcal{F}_m$ and $\alpha, \beta \in I\!R^1$. Calculating $[\mathcal{D}^1(\alpha f + \beta g)](x_r)$ we

find:

$$\mathcal{D}^1(\alpha f+\beta g)(x_r)=[x_r,x_{r+1},\alpha f+\beta g]=$$

$$=\frac{(\alpha f+\beta g)(x_{r+1})-(\alpha f+\beta g)(x_r)}{x_{r+1}-x_r}=$$

$$=\frac{\alpha f(x_{r+1})+\beta g(x_{r+1})-\alpha f(x_r)-\beta g(x_r)}{x_{r+1}-x_r}=$$

$$=\alpha\cdot\frac{f(x_{r+1})-f(x_r)}{x_{r+1}-x_r}+\beta\cdot\frac{g(x_{r+1})-g(x_r)}{x_{r+1}-x_r}=$$

$$=\alpha\cdot(\mathcal{D}^1f)(x_r)+\beta\cdot(\mathcal{D}^1f)(x_r).$$

□

Definition 3.1.3. ***The second divided difference*** *of the function f at the point x_r, $r\leq m-2$, is the number:*

$$(\mathcal{D}^2f)(x_r)=\frac{(\mathcal{D}^1f)(x_{r+1})-(\mathcal{D}^1f)(x_r)}{x_{r+2}-x_r}=\frac{[x_{r+1},x_{r+2},f]-[x_r,x_{r+1},f]}{x_{r+2}-x_r},$$

and it will be denoted by:

$$[x_r,x_{r+1},x_{r+2},f]=\frac{[x_{r+1},x_{r+2},f]-[x_r,x_{r+1},f]}{x_{r+2}-x_r}$$

$$[x_r,x_{r+1},x_{r+2},f]=\frac{(\mathcal{D}^1f)(x_{r+1})-(\mathcal{D}^1f)(x_r)}{x_{r+2}-x_r}.$$

Propostion 3.1.2. *The following equality takes place:*

$$(\mathcal{D}^2f)(x_r)=[x_r,x_{r+1},x_{r+2},f]=\frac{f(x_r)}{(x_r-x_{r+1})(x_r-x_{r+2})}+\frac{f(x_{r+1})}{(x_{r+1}-x_r)(x_{r+1}-x_{r+2})}+$$

$$+\frac{f(x_{r+2})}{(x_{r+2}-x_r)(x_{r+2}-x_{r+1})}$$

Proof: According to the definition we have:

$$[x_r, x_{r+1}, x_{r+2}, f] = \frac{(\mathcal{D}^1 f)(x_{r+1}) - (\mathcal{D}^1 f)(x_r)}{x_{r+2} - x_r}$$

$$= \frac{1}{x_{r+2} - x_r} \cdot \left[\frac{f(x_{r+2}) - f(x_{r+1})}{x_{r+2} - x_{r+1}} - \frac{f(x_{r+1}) - f(x_r)}{x_{r+1} - x_r} \right] =$$

$$= \frac{1}{x_{r+2} - x_r} \cdot \left[\frac{f(x_{r+2})(x_{r+1} - x_r)}{(x_{r+2} - x_{r+1})(x_{r+1} - x_r)} - \right.$$

$$\left. - \frac{f(x_{r+1})(x_{r+1} - x_r + x_{r+2} - x_{r+1})}{(x_{r+2} - x_{r+1})(x_{r+1} - x_r)} + \frac{f(x_r)(x_{r+2} - x_{r+1})}{(x_{r+2} - x_{r+1})(x_{r+1} - x_r)} \right] =$$

$$= \frac{f(x_{r+2})}{(x_{r+2} - x_r)(x_{r+2} - x_{r+1})} - \frac{f(x_{r+1})}{(x_{r+2} - x_{r+1})(x_{r+1} - x_r)} +$$

$$+ \frac{f(x_r)}{(x_{r+2} - x_r)(x_{r+1} - x_r)} =$$

$$= \frac{f(x_r)}{(x_r - x_{r+1})(x_r - x_{r+2})} + \frac{f(x_{r+1})}{(x_{r+1} - x_r)(x_{r+1} - x_{r+2})} +$$

$$+ \frac{f(x_{r+2})}{(x_{r+2} - x_r)(x_{r+2} - x_{r+1})}.$$

□

Remark 3.1.2. For any $k \leq m$ we can define **the k^{th} divided difference** of the function f at the point x_r ($r \leq m - k$):

$$(\mathcal{D}^k f)(x_r) = \frac{(\mathcal{D}^{k-1} f)(x_{r+1}) - (\mathcal{D}^{k-1} f)(x_r)}{x_{r+k} - x_r} = \frac{[x_{r+1}, \ldots, x_{r+k}, f] - [x_r, x_{r+1}, \ldots, x_{r+k-1}, f]}{x_{r+k} - x_r}.$$

The function which associates to the point x_r the k^{th} divided difference of f at x_r, is denoted by $(\mathcal{D}^k f)$.

By mathematical induction it is shown that the equality:

$$(\mathcal{D}^k f)(x_r) = \sum_{i=0}^{k} \frac{f(x_{r+i})}{(x_{r+i} - x_r)(x_{r+i} - x_{r+1}) \ldots (x_{r+i} - x_{r+i-1})(x_{r+i} - x_{r+i+1}) \ldots (x_{r+i} - x_{r+k})}$$

takes place, where it can be observed that the factor $(x_{r+i} - x_{r+i})$ is missing from the denominator.

Remark 3.1.3. Considering the set of functions $\mathcal{F}_{m-k} = \{f : \{x_0, x_1, \ldots, x_{m-k}\} \to \mathbb{R}^1\}$, using the k^{th} divided difference we can associate to every function $f \in \mathcal{F}_m$ the function $\mathcal{F}_{m-k}$:

$$f \longmapsto \mathcal{D}^k f$$

where $\mathcal{D}^k f$ is defined by $(\mathcal{D}^k f)(x_r) = [x_r, x_{r+1}, \ldots, x_{r+k}, f]$ for $r \leq m - k$.

The correspondence $f \longmapsto \mathcal{D}^k f$ will be denoted by $\mathcal{D}^k$ and will be called the operator of the k^{th} divided difference.

Remark 3.1.4. The operator $\mathcal{D}^k : \mathcal{F}_m \to \mathcal{F}_{m-k}$ of the k^{th} divided difference is linear.

Remark 3.1.5. For $k = m$, the m^{th} divided difference is defined only at x_0 and it is given by:

$$(\mathcal{D}^m f)(x_0) = \sum_{i=0}^{m} \frac{f(x_i)}{(x_i - x_0)(x_i - x_1)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_m)}.$$

Thus, the following equality can be proved:

Propostion 3.1.3.

$$(\mathcal{D}^m f)(x_0) = \frac{(Wf)(x_0, x_1, \ldots, x_m)}{V(x_0, x_1, \ldots, x_m)}$$

where:

$$(Wf)(x_0, x_1, \ldots, x_m) = \begin{vmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{m-1} & f(x_0) \\ 1 & x_1 & x_1^2 & \ldots & x_1^{m-1} & f(x_1) \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & x_m & x_m^2 & \ldots & x_m^{m-1} & f(x_m) \end{vmatrix}$$

and

$$V(x_0, x_1, \ldots, x_m) = \begin{vmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{m-1} & x_0^m \\ 1 & x_1 & x_1^2 & \ldots & x_1^{m-1} & x_1^m \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & x_m & x_m^2 & \ldots & x_m^{m-1} & x_m^m \end{vmatrix}$$

Proof: Using the Vandermonde formula we have:

$$V(x_0, x_1, \ldots, x_m) = \begin{vmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{m-1} & x_0^m \\ 1 & x_1 & x_1^2 & \ldots & x_1^{m-1} & x_1^m \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & x_m & x_m^2 & \ldots & x_m^{m-1} & x_m^m \end{vmatrix} = \prod_{i>j} (x_i - x_j).$$

Developing Wf with respect to the last column we obtain:

$$\begin{aligned}(Wf)(x_0,x_1,\ldots,x_m) &= (-1)^{m+2}\cdot f(x_0)\cdot V(x_1,\ldots,x_m)+\\ &+ (-1)^{m+3}\cdot f(x_1)\cdot V(x_0,x_2,\ldots,x_m)+\\ &+ \cdots+(-1)^{2(m+1)}\cdot f(x_m)\cdot V(x_0,\ldots,x_{m-1}).\end{aligned}$$

Hence,

$$\begin{aligned}&\frac{(Wf)(x_0,x_1,\ldots,x_m)}{V(x_0,x_1,\ldots,x_m)} =\\ &= \frac{1}{\prod\limits_{i>j}(x_i-x_j)}\cdot\sum_{k=0}^{m}(-1)^{m+2+k}\cdot f(x_k)\cdot\prod_{\substack{i>j\\ i,j\neq k}}(x_i-x_j) =\\ &= \sum_{k=0}^{m}(-1)^{m+2+k}\cdot f(x_k)\cdot\frac{\prod\limits_{\substack{i>j\\ i,j\neq k}}(x_i-x_j)}{\prod\limits_{i>j}(x_i-x_j)} =\\ &= \sum_{k=0}^{m}(-1)^{m+2+k}\cdot\frac{f(x_k)}{(x_k-x_0)\ldots(x_k-x_{k-1})(x_{k+1}-x_k)\ldots(x_m-x_k)} =\\ &= \sum_{k=0}^{m}(-1)^{m+2+k}\cdot\frac{f(x_k)}{(x_k-x_0)\ldots(x_k-x_{k-1})(-1)^{m-k}(x_k-x_{k+1})\ldots(x_k-x_m)} =\\ &= \sum_{i=0}^{m}\frac{f(x_i)}{(x_i-x_0)(x_i-x_1)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_m)}\\ &= (\mathcal{D}^m f)(x_0).\end{aligned}$$

□

Remark 3.1.6. From the obtained representation of $(\mathcal{D}^m f)(x_0)$ it results that, for any permutation $(i_0,i_1,\ldots,i_m)$ of the numbers $(0,1,\ldots,m)$, we have:

$$[x_{i_0},x_{i_1},\ldots,x_{i_m};f]=[x_0,x_1,\ldots,x_m;f].$$

In other words, the m^{th} divided difference does not depend of the order of knots.

Propostion 3.1.4. *If f is a polynomial of the maximum degree $m-1$, then $(\mathcal{D}^m f)(x_0) = 0,\ \forall\, x_0$.*

Proof: If f is a polynomial of the maximum degree $m-1$, then $f(x) = \sum_{k=0}^{m-1} a_k x^k$. Taking into account that $\mathcal{D}^m$ is a linear operator, we have:

$$(\mathcal{D}^m f)(x_0) = \sum_{k=0}^{m-1} a_k \mathcal{D}^m(x^k)(x_0).$$

On the other hand, we have:

$$\mathcal{D}^m(x^k)(x_0) = \frac{W(x^k)(x_0, x_1, \ldots, x_m)}{V(x_0, x_1, \ldots, x_m)}$$

with $W(x^k)(x_0, x_1, \ldots, x_m) = \begin{vmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{m-1} & x_0^k \\ 1 & x_1 & x_1^2 & \ldots & x_1^{m-1} & x_1^k \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & x_m & x_m^2 & \ldots & x_m^{m-1} & x_m^k \end{vmatrix} = 0.$

In this way, we find the equality from the enunciation.

□

Propostion 3.1.5. *If $f, g : X \to \mathbb{R}^1$ then:*

$$[x_0, x_1, \ldots, x_m, f \cdot g] = \sum_{k=0}^{m} [x_0, x_1, \ldots, x_k, f] \cdot [x_k, \ldots, x_m, g].$$

Proof: Mathematical induction with respect to m will be used.

Thus, for $m = 1$ the left hand side is $[x_0, x_1, f \cdot g]$ which represents the first divided difference of $f \cdot g$. So, according to the definition we have:

$$[x_0, x_1, f \cdot g] = \frac{(f \cdot g)(x_1) - (f \cdot g)(x_0)}{x_1 - x_0} = \frac{f(x_1) \cdot g(x_1) - f(x_0) \cdot g(x_0)}{x_1 - x_0} =$$

$$= \frac{f(x_1) \cdot g(x_1) - f(x_0) \cdot g(x_1) + f(x_0) \cdot g(x_1) - f(x_0) \cdot g(x_0)}{x_1 - x_0} =$$

$$= g(x_1) \cdot [x_0, x_1, f] + f(x_0) \cdot [x_0, x_1, g] =$$

$$= f(x_0) \cdot [x_0, x_1, g] + [x_0, x_1, f] \cdot g(x_1).$$

We suppose that the relation:

$$[x_0, x_1, \ldots, x_m, f \cdot g] = \sum_{k=0}^{m-1} [x_0, x_1, \ldots, x_k, f] \cdot [x_k, \ldots, x_{m-1}, g]$$

is true, and we compute $[x_0, x_1, \ldots, x_m, f \cdot g]$. Thus, we obtain:

$$\begin{aligned}
&[x_0, x_1, \ldots, x_m, f \cdot g] = \\
&= \frac{1}{x_m - x_0} \cdot [[x_1, \ldots, x_m, f \cdot g] - [x_0, \ldots, x_{m-1}, f \cdot g]] = \\
&= \frac{1}{x_m - x_0} \cdot \left[\sum_{k=0}^{m-1} [x_1, \ldots, x_{k+1}, f] \cdot [x_{k+1}, \ldots, x_m, g] - \right. \\
&\quad \left. - [x_0, \ldots, x_k, f] \cdot [x_k, \ldots, x_{m-1}, g] \right] = \\
&= \frac{1}{x_m - x_0} \cdot \sum_{k=0}^{m-1} [x_1, \ldots, x_{k+1}, f] \cdot [x_{k+1}, \ldots, x_m, g] - \\
&\quad - [x_0, \ldots, x_k, f] \cdot [x_k, \ldots, x_{m-1}, g] + \\
&\quad + [x_0, \ldots, x_k, f] \cdot [x_{k+1}, \ldots, x_m, g] - [x_0, \ldots, x_k, f] \cdot [x_{k+1}, \ldots, x_m, g] = \\
&= \frac{1}{x_m - x_0} \cdot \sum_{k=0}^{m-1} [x_0, \ldots, x_k, f] \cdot \{[x_{k+1}, \ldots, x_m, g] - [x_k, \ldots, x_{m-1}, g]\} + \\
&\quad + \frac{1}{x_m - x_0} \cdot \sum_{k=0}^{m-1} [x_{k+1}, \ldots, x_m, g] \cdot \{[x_1, \ldots, x_{k+1}, f] - [x_0, \ldots, x_k, f]\} = \\
&= \frac{1}{x_m - x_0} \cdot \sum_{k=0}^{m-1} [x_0, \ldots, x_k, f] \cdot (x_m - x_k) \cdot [x_k, \ldots, x_m, g] + \\
&\quad + \frac{1}{x_m - x_0} \cdot \sum_{k=1}^{m} [x_k, \ldots, x_m, g] \cdot (x_k - x_0) \cdot [x_0, \ldots, x_k, f] =
\end{aligned}$$

$$= \frac{1}{x_m - x_0} \cdot \{(x_m - x_0) \cdot [x_0, f] \cdot [x_0, \ldots, x_m, g] +$$

$$+ \sum_{k=1}^{m-1} (x_m - x_0) \cdot [x_0, \ldots, x_k, f] \cdot [x_k, \ldots, x_m, g] +$$

$$+ (x_m - x_0) \cdot [x_0, \ldots, x_m, f] \cdot [x_m, g]\} =$$

$$= \sum_{k=0}^{m} [x_0, \ldots, x_k, f] \cdot [x_k, \ldots, x_m, g].$$

□

We return to the objective of this section: **the Newton polynomial with divided differences**. Thus, using the divided differences presented above, we will construct the interpolating polynomial of degree m:

$$\varphi(x) = \sum_{i=0}^{m} a_i x^i,$$

which approximates the function $f : X \to \mathbb{R}^1$ known only by its values in the data points x_i: $y_i = f(x_i), i = 0, 1, \ldots, m$.
The approximating function $\varphi(x)$ is a polynomial function of degree m, denoted by $p_m(x)$, for which the coefficients a_i will be computed using divided differences:

$$a_i = [x_0, x_1, \ldots, x_i, f], i = 0, 1, \ldots, m.$$

Definition 3.1.4. *We call* ***Newton polynomial with divided differences*** *which approximates the function $f : X \to \mathbb{R}^1$ given by: $y_i = f(x_i), i = 0, 1, \ldots, m$, the polynomial function of degree m:*

$$p_m(x) = f(x_0) + [x_0, x_1, f](x - x_0) + [x_0, x_1, x_2, f](x - x_0)(x - x_1) + \ldots$$

$$+ [x_0, x_1, \ldots, x_m, f](x - x_0)(x - x_1) \ldots (x - x_{m-1}).$$

Remark 3.1.7. Newton polynomial with divided differences has the property that its graph passes through the points $(x_i, y_i) = (x_i, f(x_i)), i = 0, 1, \ldots, m$.

Using the Newton polynomial with divided differences, the function $f : X \to \mathbb{R}^1$ given by: $y_i = f(x_i), i = 0, 1, \ldots, m$, is written as:

$$f(x) = f(x_0) + (\mathcal{D}^1 f)(x_0)(x - x_0) + (\mathcal{D}^2 f)(x_0)(x - x_0)(x - x_1) + \ldots$$

$$+ (\mathcal{D}^m f)(x_0)(x - x_0)(x - x_1) \ldots (x - x_{m-1}) + R_m(x),$$

where $R_m(x)$ represents the remainder term (the error of approximation) of the interpolating polynomial. In order to evaluate the approximating error, we need supplementary information about the approximating function $f(x)$ and its derivatives.

Propostion 3.1.6. (Mean Value Theorem)
Let be $a \leq x_0 < x_1 < \ldots < x_m \leq b$. If $f \in \mathcal{C}^{m-1}[a,b]$ and $f^{(m)}$ is derivable on (a,b), then there exists ξ such that:

$$(\mathcal{D}^m f)(x_0) = \frac{1}{m!} \cdot f^{(m)}(\xi) \qquad a < \xi < b.$$

Proof: We will consider the auxiliary function:

$$\varphi(x) = (Wf)(x, x_0, \ldots, x_{m-1}) + (\mathcal{D}^m f)(x_0) \cdot V(x, x_0, \ldots, x_{m-1}).$$

The function φ has the property $\varphi(x_k) = 0,\ k = 0, 1, 2, \ldots, m$. Applying successively the Theorem of Rolle on the subintervals determined by these points, we obtain that $\varphi^{(m)}$ has at least one zero $\xi \in (a,b)$. Because

$$\varphi^{(m)}(x) = [f^{(m)}(x) - m! \cdot \mathcal{D}^m f(x_0)] \cdot V(x_0, \ldots, x_{m-1})$$

it results

$$(\mathcal{D}^m f)(x_0) = \frac{f^{(m)}(\xi)}{m!}.$$

□

From the *Mean Value Theorem*, we obtain the expression of the m^{th} divided difference as function of the m^{th} derivative of the function f. In this way, the approximating error becomes:

$$R_m(x) = \frac{f^{(m+1)}(\xi)}{(m+1)!}(x - x_0)(x - x_1)\ldots(x - x_{m-1})(x - x_m),$$

where $\xi \in (a,b)$.
The interpolating polynomial appears especially as a component of other numerical algorithms (numerical integration or numerical differentiation). In these applications, equal intervals are considered given by equidistant knots (i.e. the distance between two consecutive knots is equal to a constant h called step of the mesh):

$$x_{i+1} - x_i = h,\ \ i = 0, 1, \ldots, m-1.$$

We introduce the forward difference operator $\triangle$ and the backward difference operator $\bigtriangledown$ defined as follows:

$$\triangle f(x) = f(x+h) - f(x),$$

$$\bigtriangledown f(x) = f(x) - f(x-h).$$

The results obtained by applying the operators $\bigtriangleup$ or $\bigtriangledown$ to the function $f(x)$, define the first finite differences.
Thus, divided differences can be expressed using finite differences and step h.
Hence, for the first divided differences we have:

$$[x_0, x_1, f] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{f(x_0+h) - f(x_0)}{h} = \frac{\bigtriangleup f(x_0)}{h},$$

$$[x_{m-1}, x_m, f] = [x_m, x_{m-1}, f] = \frac{f(x_m) - f(x_{m-1})}{x_m - x_{m-1}} = \frac{f(x_m) - f(x_m - h)}{h} = \frac{\bigtriangledown f(x_m)}{h}.$$

For the second divided differences we obtain:

$$[x_0, x_1, x_2, f] = \frac{[x_1, x_2, f] - [x_0, x_1, f]}{x_2 - x_0} = \frac{\frac{\bigtriangleup f(x_1)}{h} - \frac{\bigtriangleup f(x_0)}{h}}{2h} = \frac{\bigtriangleup^2 f(x_0)}{2!h^2},$$

$$[x_{m-2}, x_{m-1}, x_m, f] = \frac{[x_1, x_2, f] - [x_0, x_1, f]}{x_2 - x_0} = \frac{\frac{\bigtriangledown f(x_m)}{h} - \frac{\bigtriangledown f(x_{m-1})}{h}}{2h} = \frac{\bigtriangledown^2 f(x_m)}{2!h^2}.$$

By mathematical induction, the m^{th} divided differences are obtained:

$$[x_0, x_1, x_2, \ldots, x_m, f] = \frac{\bigtriangleup^m f(x_0)}{m!h^m},$$

$$[x_0, x_1, x_2, \ldots, x_m, f] = \frac{\bigtriangledown^m f(x_m)}{m!h^m}.$$

Based on the above formulas, the Newton polynomial with forward finite differences is:

$$p_m(x) = f(x_0) + \frac{\bigtriangleup f(x_0)}{h}(x - x_0) + \frac{\bigtriangleup^2 f(x_0)}{2!h^2}(x - x_0)(x - x_1) + \ldots$$

$$+ \frac{\bigtriangleup^m f(x_0)}{m!h^m}(x - x_0)\ldots(x - x_{m-1}),$$ and the Newton polynomial with backward finite differences is:

$$p_m(x) = f(x_m) + \frac{\bigtriangledown f(x_m)}{h}(x - x_m) + \frac{\bigtriangledown^2 f(x_m)}{2!h^2}(x - x_m)(x - x_{m-1}) + \ldots$$

$$+ \frac{\bigtriangledown^m f(x_m)}{m!h^m}(x - x_m)\ldots(x - x_1).$$

Exercises

1. Approximate numerically the function $f(x) = \sqrt{x}$, knowing its values in the knots: $x_0 = 1$, $x_1 = 1.5$, $x_2 = 2$, $x_3 = 2.5$, $x_4 = 3$, as follows:

- using the Newton polynomial with divided differences;
- using the Newton polynomial with forward finite differences;
- using the Newton polynomial with backward finite differences.

Plot, on the same cartesian coordinates, the obtained interpolating polynomial functions together with the given function $f(x) = \sqrt{x}$.
2. Using the *Mean Value Theorem*, approximate numerically the derivatives $f'(0.1)$, $f''(0.2)$, $f'''(0.1)$, where the function f is given as follows:

x_i	0.1	0.2	0.3	0.4
$f(x_i)$	0.995	0.98007	0.95534	0.92106

In the followings, we will present the algorithm for the implementation of the k^{th} divided difference at the point x_r.

If $r > n - k$, then we cannot compute the derivative of order k at the point x_r.

If $r \leq n - k$, then the k^{th} divided difference is:

$$(\mathcal{D}^k f)(x_r) = \sum_{i=0}^{k} \left[f(x_{r+i}) \cdot \prod_{\substack{j=0 \\ j \neq i}}^{k} \frac{1}{x_{r+i} - x_{r+j}} \right]$$

Input data:

- n
- x_i - knots, $i = 0, \ldots, n$
- $f(x_i)$ - values of the function f at the knots
- r - the order of the knot in which we approximate the derivative
- k - order of the derivative

Output data:

- approximating the derivative of order k at the knot x_r

Implementation in Borland C:

```
#include<stdio.h>
float **Matrice(int imin, int imax, int jmin, int jmax);
float *Vector(int n);
void CitireVect(float *a, int n);
void ScriereVect(float *a, int n);
float dif_divizate(float *x, float *f,int n, int r, int k);
void main()
{
    float *f, *x;
    int n,r,k;
    printf("n= "); scanf("%d",&n);
    x=Vector(n);
    f=Vector(n);
    printf("Introduceti nodurile: \n");
    CitireVect(x,n);
    printf("Introduceti valorile functiei in noduri: \n");
    CitireVect(f,n);
    printf("Introduceti ordinul nodului in care se face derivata: ");
    scanf("%d",&r);
    printf("Introduceti ordinul derivatei: "); scanf("%d",&k);
    printf("Valoarea derivatei de ordinul %d in x[%d] este:
    %g",k,r, dif_divizate(x, f,n, r, k));
}
float dif_divizate(float *x, float *f,int n, int r, int k)
{
    float suma, produs;
    int i,j;
   if(r>n-k){ printf("Nu se poate calcula derivata de ordin %d in punctul
              x[%d] !", k,r); }
    else{
        suma=0.0;
        for(i=0;i<=k;i++)
        {
            produs=f[r+i];
            for(j=0;j<=k;j++)
                if(j!=i) produs=produs/(x[r+i]-x[r+j]);
                    suma=suma+produs;
        }
}
return suma;
```

}

3.2 The Lagrange Interpolating Polynomial

Let $[a,b] \subset I\!R^1$, the knots $x_i \in [a,b]$, $i = 0, 1, \ldots, m$ such that $x_i \neq x_j$ for $i \neq j$, and a function $f : [a,b] \to I\!R^1$.

Definition 3.2.1. *The problem of determining the polynomial P which has minimum degree and satisfies the property:*

$$P(x_i) = f(x_i), \qquad i = 0, 1, \ldots, m,$$

is called ***the Lagrange interpolating polynomial problem****.*

Definition 3.2.2. *A solution of the Lagrange interpolating polynomial problem (if it exists) is called* ***Lagrange interpolating polynomial*** *and it will be denoted by $L_m f$, L_m being the operator which associates f to the polynomial $L_m f$.*

Theorem 3.2.1. *The Lagrange interpolating polynomial problem has a unique solution which is a polynomial of degree m.*

Proof: We consider a polynomial of degree m having undetermined coefficients:

$$P_m(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m.$$

Imposing this polynomial to satisfy:

$$P_m(x_i) = f(x_i) \qquad i = 0, 1, \ldots, m$$

we obtain the following linear system having $m+1$ equations and $m+1$ unknowns $a_0, a_1, \ldots, a_m$:

$$\begin{aligned}
a_0 + a_1 x_0 + a_2 x_0^2 + \ldots + a_m x_0^m &= f(x_0) \\
a_0 + a_1 x_1 + a_2 x_1^2 + \ldots + a_m x_1^m &= f(x_1) \\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\
a_0 + a_1 x_m + a_2 x_m^2 + \ldots + a_m x_m^m &= f(x_m).
\end{aligned}$$

The determinant of the system is

$$V(x_0, x_1, \ldots, x_m) = \begin{vmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^m \\ 1 & x_1 & x_1^2 & \ldots & x_1^m \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & x_m & x_m^2 & \ldots & x_m^m \end{vmatrix} = \prod_{\substack{i,j=0 \\ i>j}}^{m} (x_i - x_j) \neq 0$$

and it follows that, the system has a unique solution. This means that there exists a unique polynomial of degree m which satisfies the conditions $P(x_i) = f(x_i)$. The fact that does not exist polynomial of degree less then m which verifies $P(x_i) = f(x_i)$ does not exist, can be proved supposing the contrary (case in which a system of $m+1$ equations with maximum m unknowns is obtained, which does not have a solution for any f).

□

Definition 3.2.3. *The Lagrange interpolating polynomials corresponding to the functions $l_i : [a,b] \to I\!R^1$ defined by $l_i(x_i) = 1$ and $l_i(x_j) = 0$ for $i \neq j$, are called* ***fundamental Lagrange interpolating polynomials****.*

Theorem 3.2.2. *The fundamental Lagrange interpolating polynomials $l_i(x)$, $i = \overline{0,m}$ are given by the formulas:*

$$l_i(x) = \frac{(x-x_0)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_m)}{(x_i-x_0)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_m)}.$$

Proof: The system from the previous proof is solved for the particularly case of the functions l_i. Thus, we obtain:

$$l_i(x) = \frac{1}{V(x_0,\dots,x_m)} \cdot \begin{vmatrix} 1 & x_0 & \dots & x_0^m \\ \dots & \dots & \dots & \dots \\ 1 & x_{i-1} & \dots & x_{i-1}^m \\ 1 & x & \dots & x^m \\ 1 & x_{i+1} & \dots & x_{i+1}^m \\ \dots & \dots & \dots & \dots \\ 1 & x_m & \dots & x_m^m \end{vmatrix}$$

i.e.,

$$l_i(x) = \frac{V(x_0,\dots,x_{i-1},x,x_{i+1},\dots,x_m)}{V(x_0,\dots,x_{i-1},x_i,x_{i+1},\dots,x_m)} =$$

$$+\frac{(x-x_0)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_m)}{(x_i-x_0)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_m)}.$$

□

Remark 3.2.1. The Lagrange interpolating polynomial which approximates the function f is given by the formula:

$$(L_m f)(x) = \sum_{i=0}^{m} l_i(x) \cdot f(x_i).$$

This equality can be obtained directly by verification.

Theorem 3.2.3. *The Lagrange interpolating operator L_m defined on the set $\mathcal{F} = \{f : [a,b] \to I\!R^1\}$ taking values in the set of the polynomials of degree m, it is linear and idempotent i.e., $(L_m f)(x) = (L_m^2 f)(x)$ (it is a projector).*

Proof: Linearity:

$$L_m(\alpha f + \beta g)(x) = \sum_{i=0}^{m} l_i(x) \cdot (\alpha f(x_i) + \beta g(x_i)) = \alpha(L_m f)(x) + \beta(L_m g)(x).$$

For proving that the operator is idempotent we consider the functions $l_k(x) = x^k$, $k = 0, 1, \ldots, n$. We observe that: $L_m(l_k)(x) = l_k(x) = x^k$, $k = 0, 1, \ldots, m$ and hence we obtain:

$$\begin{aligned} L_m(f)(x) = a_0 + a_1 x + \ldots + a_m x^m \Rightarrow \\ \Rightarrow L_m(L_m f)(x) = a_0 + a_1 x + \ldots + a_m x^m \Rightarrow (L_m^2 f)(x) = (L_m f)(x). \end{aligned}$$

□

Remark 3.2.2. If we consider the operator L_m defined on the set of continuous functions defined on $[a,b]$, then the norm of the operator L_m defined by $\|L_m\| = \sup_{\|f\| \leq 1} \|L_m f\|$, where $\|f\| = \sup_{x \in [a,b]} |f(x)|$, is evaluated as follows:

$$\|L_m\| = \max_{a \leq x \leq b} \sum_{i=0}^{m} |l_i(x)|.$$

Definition 3.2.4. *The difference $f(x) - (L_m f)(x) = (R_m f)(x)$ is called* ***truncation error of order*** **m**, *and the approximating formula $f(x) = (L_m f)(x) + (R_m f)(x)$ is called* ***the Lagrange approximation formula.***

Theorem 3.2.4. *The truncation error $(R_m f)(x)$ from the Lagrange approximation formula is a linear and idempotent operator.*

Proof: Linearity of $(R_m f)(x)$ results from the linearity of $(L_m f)(x)$. For proving that the error is idempotent, we take into account that, if f is a polynomial of degree m then $L_m = f$. It results that, if f is a polynomial of degree m, then $R_m f = f - L_m f = 0$. Thus, $R_m(R_m f) = R_m(f - L_m f) = R_m f$ for any f, i.e., $R_m^2 f = R_m f$.

□

In the followings, we will present some formulas for the representation of the truncation error of the Lagrange interpolating formula.

Theorem 3.2.5. *Let be* $\alpha = \min\{x, x_0, \ldots, x_m\}$ *and* $\beta = \max\{x, x_0, \ldots, x_m\}$. *If* f *is of* $\mathcal{C}^m$*-class on* $[\alpha, \beta]$ *and* $f^{(m)}$ *is derivable on* (α, β) *then there exists* $\xi \in (\alpha, \beta)$ *such that*

$$(R_m f)(x) = \frac{u(x)}{(m+1)!} \cdot f^{(m+1)}(\xi)$$

where $u(x)$ *is the polynomial defined by*

$$u(x) = (x - x_0)(x - x_1) \ldots (x - x_m).$$

Proof: Let be the function F defined by

$$F(z) = \begin{vmatrix} u(z) & (R_m f)(z) \\ u(x) & (R_m f)(x) \end{vmatrix}$$

From the hypothesis on f it results that $F \in \mathcal{C}^m[\alpha, \beta]$ and there exists $F^{(m+1)}$ on (α, β). Moreover, we observe that $F(x) = 0$ and $F(x_i) = 0$, $i = 0, 1, \ldots, m$, hence F has $m + 2$ distinct zeros on $[\alpha, \beta]$. Applying successively the Rolle theorem it results that $F^{(m+1)}$ has a least one zero in this interval. Taking the derivative of order $m+1$ of the function F and imposing the condition $F^{(m+1)}(\xi) = 0$ we obtain

$$F^{(m+1)}(\xi) = \begin{vmatrix} (m+1)! & f^{(m+1)}(\xi) \\ u(x) & (R_m f)(x) \end{vmatrix} = 0,$$

where we used that $(R_m f)^{(m+1)} = f^{(m+1)} - (L_m f)^{(m+1)} = f^{(m+1)}$. Computing this determinant we obtain $(R_m f)(x)$ from the enunciation.

□

Theorem 3.2.6. *For* $f : [a, b] \to \mathbb{R}^1$ *we have*

$$(R_m f)(x) = u(x) \cdot [x, x_0, \ldots, x_m; f], \qquad x \in [a, b].$$

Proof: Supposing that $x \neq x_k$, $k = 0, 1, \ldots, m$ (this is a natural condition because $(R_m f)(x_k) = 0$, $k = 0, 1, \ldots m$), based on the divided difference formula we obtain:

$$[x, x_0, \ldots, x_m; f] = \frac{f(x)}{u(x)} + \sum_{k=0}^{m} \frac{f(x_k)}{(x_k - x) \cdot u'(x_k)}$$

where $u(x) = (x - x_0) \ldots (x - x_m)$, and $u'(x_k) = (x_k - x_0) \ldots (x_k - x_{k-1})(x_k - x_{k+1}) \ldots (x_k - x_m)$. Multiplying every member of this formula with $u(x)$, we get the equality from the enunciation.

□

Remark 3.2.3. If the interpolating knots are equidistant, i.e. $x_i = x_0 + ih$, $i = 0, 1, \ldots, m$, $h > 0$, then the Lagrange interpolating polynomial and the truncation error can be written as follows :

$$(L_m f)(x_0 + th) = \frac{t^{m+1}}{m!} \cdot \sum_{i=0}^{m} (-1)^{m-i} C_m^i \frac{1}{t-i} \cdot f(x_i),$$

$$(R_m f)(x_0 + th) = \frac{h^{m+1} \cdot t^{m+1}}{(m+1)!} \cdot f^{(m+1)}(\xi).$$

Practical points of view for the interpolating polynomial:

1. The Newton and Lagrange interpolating polynomials are different only by the form; the truncation errors are the same, if we consider the same mesh of knots. From numerical calculus point of view, the Newton interpolating is preferred because this request a small number of the arithmetic if we compare with the Lagrange interpolating. Both algorithms use the same memory size.

2. If we denote by α and β the smaller and the larger interpolating knots, respectively, then from computational point of view, the following interpolating polynomials are indicated: for x not far of α, it is indicated to use the Newton polynomial with forward finite differences; for x not far of β, it is indicated to use the Newton polynomial with backward finite differences.

3. Other interpolating polynomial forms correspond to the space of the functions in which interpolator is searched. Thus, if we choose the spline functions as space, then we obtain spline polynomial interpolation; if we choose the trigonometric functions as space, then we obtain spline polynomial trigonometric, etc.

Exercises

1. Using the Lagrange interpolating polynomial, approximate numerically $\cos(0.12)$, in the case in which the followings values are known:

x_i	0.1	0.2	0.3	0.4
$\cos(x_i)$	0.995	0.98007	0.95534	0.92106

2. Using the Lagrange interpolating polynomial, compute the Australian population from the years 1960, 1970 and 1975, if the followings data are given:

an	1954	1961	1971	1976
population	8.99	10.51	12.94	13.92

For implementation of the Lagrange interpolation polynomial the following formula will be used:

$$L(x) = \sum_{i=0}^{n} \left[f(x_i) \cdot \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \right]$$

Input data:

- n
- x_i - knots, $i = 0, \ldots, n$
- $f(x_i)$ - values of the function f in the knots
- x - point in which the value of the function f is approximated

Output data:

- L(x)

Implementation of the above algorithm, using the Borland C language:

```
#include<stdio.h>
float **Matrice(int imin, int imax, int jmin, int jmax);
float *Vector(int n);
void CitireVect(float *a, int n);
void ScriereVect(float *a, int n);
float Lagrange(float *x, float *f,int n, float a);
void main()
{
    float *f, *x,a;
    int n;
    printf("n= "); scanf("%d",&n);
    x=Vector(n);
    f=Vector(n);
    printf("Introduceti nodurile: \n");
    CitireVect(x,n);
    printf("Introduceti valorile functiei in noduri: \n");
    CitireVect(f,n);
    printf("Introduceti abscisa punctului in care se aproximeaza valoarea
    functiei: "); scanf("%f",&a);
```

```
    printf("f(%g)= %g",a, Lagrange(x, f,n, a));
}
float Lagrange(float *x, float *f,int n, float a)
{
    float suma, produs;
    int i,j;
    suma=0.0;
    for(i=0;i<=n;i++)
    {
        produs=f[i];
        for(j=0;j<=n;j++)
            if(j!=i) produs=produs*(a-x[j])/(x[i]-x[j]);
                suma=suma+produs;
    }
return suma;
}
```

3.3 Piecewise Polynomial Approximations: Spline Functions. Introduction

In some cases, the global interpolation presented in the previous sections (on the entire interval $[a, b]$), does not converge. Indeed, even if the truncation errors $R_m(x)$ suggest that the precision increases when the knots number increases, the interpolation polynomial may not converge; an example has been given in 1901 by Runge: $f(x) = 1/(1 + x^2), x \in [-5, 5]$ (truncation error tends to infinity when $m \longrightarrow \infty$). More precisely, the *Faber Theorem* asses that for any set of data points in the interval $[a, b]$ there exists a continuous function for which the digression of the interpolating polynomial increases however much for $m \longrightarrow \infty$. The fact that there exists at least one function for which the interpolating polynomial does not converge, reduces the applicability of the global interpolation, and it is used only as a component of numerical algorithms for small values of m.

Thus, the idea of *piecewise* polynomial approximations comes up, i.e., for every subinterval $[x_{i-1}, x_i]$ of $[a, b]$ another interpolating polynomial is defined. The polygonal function represents the simplest example of piecewise interpolating polynomial (called *spline interpolation*).

The natural *spline* is the curve (called the elastic) obtained by forcing a flexible elastic rod through the n points P_i, $i = \overline{1, n}$ but letting the slope

at the ends be free to equilibrate the position that minimizes the oscillatory behavior of the curve.

Iacob Bernoulli (1705) gives the idea that "the elastic" can be obtained by minimization of the integral from the squared curvature, in a class of admissible functions. Thus, the theory of Euler-Bernoulli concerning to deformation of the thin beams is formulated (1742).

The expression which is minimized in this theory is:

$$E_p = \int_0^l \mu(s) \cdot K^2(s) ds$$

in which: E_p is the potential energy; l is length of the rod; μ is the rod density; s - arc length; K - curvature as function of the arc length.
Choosing the cartesian reference xOy and denoting by f the function whose graphic is "the elastic" we have:

$$K(x) = \frac{f''(x)}{\{1 + [f'(x)]^2\}^{3/2}} \quad and \quad ds = \{1 + [f'(x)]^2\}^{1/2} dx.$$

Supposing the rod homogeneity $\mu(s) = \mu$ (i.e. constant density) and denoting by $(a, f(a))$, $(b, f(b))$ the ends of the curve, the expression which should be minimized becomes:

$$E_p = \mu \cdot \int_a^b \frac{[f''(x)]^2}{\{1 + [f'(x)]^2\}^{5/2}} \, dx.$$

Admitting that $f'(x) \approx c, x \in [a, b]$ we obtain:

$$E_p \approx c_1 \cdot \int_a^b [f''(x)]^2 \, dx,$$

where $c_1 = \dfrac{\mu}{(1 + c^2)^{5/2}}$.

If $P_i = (x_i, y_i)$, $i = 1, 2, \ldots, n$ are n knots which define the division $\Delta : a \leq x_1 < x_2 < \ldots < x_n \leq b$ of the interval $[a, b]$, then the considered problem is reduced to the minimization of the integral:

$$\int_a^b [f''(x)]^2 \, dx$$

on a set of smooth functions for which we impose to pass through P_i:

$$f(x_i) = y_i, \qquad i = 1, 2, \ldots, n.$$

Due to physical reasons it results that f' is continuous and the potential energy $\int_a^b [f''(x)]^2\,dx$ is finite.
In this way, we are led to the set:

$$H^{2,2}_{[a,b]} = \{f \in \mathcal{C}^1_{[a,b]} \mid f' \text{ absolutely continuous on } [a,b] \text{ and } f'' \in L^2_{[a,b]}\}.$$

From this set we will choose those functions which pass through P_i:

$$U(y) = \{f \in H^{2,2}_{[a,b]} \mid f(x_i) = y_i,\ i = 1, 2, \ldots, n\}$$

where y is the vector $y = (y_1, \ldots, y_n)$.

The Euler-Bernoulli problem becomes the following minimum problem with interpolating restrictions:
Determine the element $u^* \in U(y)$ such that:

$$\|u^{*\,\prime\prime}\|_2 = \inf_{u \in U(y)} \|u''\|_2,$$

where $\|\cdot\|_2$ is the norm of the space $L^2_{[a,b]}$:

$$\|u\|_2^2 = \int_a^b u^2(x)\,dx.$$

According to the given model, the spline interpolation problem can be enounced in the following manner:

Definition 3.3.1. *Let X be a linear space, $(Y, \|\cdot\|)$ a normed linear space, $U = \{f \in X \mid f(x_i) = y_i,\ i = 1, \ldots, n\} \subset X$ and $T : X \to Y$. The problem of the determination of the function $u^* \in U$ having the property*

$$\|Tu^*\| = \inf_{u \in U} \|Tu\|$$

is called ***spline interpolation problem****.*

A solution of the spline interpolation problem is called *interpolating spline function*, and the set U is called *interpolating set*.
The elements from the formulation of the spline interpolation problem are: spaces X and Y, interpolating set U and application T. By particularization these elements we can give different types of *spline functions* (polynomial, exponential, trigonometric) which are solutions of the problems of this type. In the case of the Euler-Bernoulli problem presented above, the elements from the spline interpolation problem are:

$$X = H^{2,2}_{[a,b]},$$

$$(Y, \|\cdot\|) = (L^2_{[a,b]}, \|f\|_2^2 = \int_a^b f^2),$$

$$U = U(y) = \{f \in H^{2,2}_{[a,b]} \mid f(x_i) = y_i,\ i = 1, \ldots, n\} \subset H^{2,2}_{[a,b]},$$

$$Tf = f''.$$

3.4 The Spline Polynomial Interpolation

For defining the spaces X, Y and the interpolating set U we consider:

- the set of the functions $f \in \mathcal{C}^{m-1}_{[a,b]}$, $m \in \{1, 2, \ldots\}$, having the derivative $f^{(m-1)}$ absolutely continuous on $[a, b]$:

  $$H^m_{[a,b]} = \{f \in \mathcal{C}^{m-1}_{[a,b]} \mid f^{(m-1)} \text{ absolutely continuous on } [a, b]\},$$

  which admit the following Taylor representation:

  $$f(x) = \sum_{k=0}^{m-1} \frac{(x-a)^k}{k!} \cdot f^{(k)}(a) + \int_a^x \frac{(x-t)^{m-1}}{(m-1)!} \cdot f^{(m)}(t)dt,$$

  hence

  $$X = H^{m,2}_{[a,b]} = \{f \in \mathcal{C}^{m-1}_{[a,b]} \mid f^{(m-1)} \text{ absolutely continuous on } [a, b] \text{ and } f^{(m)} \in L^2_{[a,b]}\}.$$

  The space $X = H^{m,2}_{[a,b]}$ is a subspace of the vectorial space $H^m_{[a,b]}$ and can be organized as a Hilbert space with the scalar product defined by:

  $$< f, g >_{m,2} = \int_a^b f^{(m)}(x) \cdot g^{(m)}(x)\ dx + \sum_{k=0}^{m-1} f^{(k)}(a) \cdot g^{(k)}(a),$$

  which generates the norm:

  $$\|f\|^2_{m,2} = \|f^{(m)}\|_2^2 + \sum_{k=0}^{m-1} \left[f^{(k)}(a)\right]^2.$$

- the linear space with norm:

  $$(Y, \|\cdot\|_2) = (L^2_{[a,b]}, \|f\|_2^2 = \int_a^b f^2(x)dx.$$

- a set of n linear independent functionals $\Phi = \{\varphi_i \mid i = 1, 2, \ldots, n\}$, $m \in I\!N$, defined on $H^{m,2}_{[a,b]}$, for defining the interpolating set:

$$U = U(y) = \{f \in H^{m,2}_{[a,b]} \mid \varphi_i(f) = y_i,\ i = 1, 2, \ldots, n\},$$

$y \in I\!R^n$.

Definition 3.4.1. *The problem of finding those elements $u^* \in U$ which satisfy the property:*

$$\|u^{*(m)}\|_2 = \inf_{u \in U(y)} \|u^{(m)}\|_2$$

is called ***polynomial spline interpolation problem****.*

Remark 3.4.1. The polynomial spline interpolation problem consists of the determination of those function, from the set $U(y)$, which are more appropriate for the polynomial p of maximum degree $m - 1$.

Definition 3.4.2. *A solution u^* of the polynomial spline interpolation problem is called* ***polynomial spline function****.*

The theorem of existence of the solution of the polynomial spline interpolation problem is based on the following lemma.

Lema 3.4.1. *Let be $\{v_1, v_2, \ldots, v_m\}$ a base for the vectorial space of the polynomials of maximum degree $m - 1$. For any number $k \in \{1, 2, \ldots, m\}$, there exists a linear functional φ_k defined on the space $H^{m,2}_{[a,b]}$ having the property:*

$$\varphi_k(v_i) = \delta_{ki}.$$

Proof: Any function $f \in H^{m,2}_{[a,b]}$ can be represented by Taylor formula as follows:

$$f(x) = f(a) + \frac{x-a}{1!} \cdot f'(a) + \ldots + \frac{(x-a)^{m-1}}{(m-1)!} \cdot f^{(m-1)}(a) + \int_a^x \frac{(x-t)^{m-1}}{(m-1)!} \cdot f^{(m)}(t)\, dt.$$

We consider the Taylor polynomial:

$$p_f(x) = f(a) + \frac{x-a}{1!} \cdot f'(a) + \ldots + \frac{(x-a)^{m-1}}{(m-1)!} \cdot f^{(m-1)}(a)$$

which we write in the base $v_i(x)$ as follows:

$$p_f(x) = \sum_{i=1}^{m} c_i(f) \cdot v_i(x).$$

The linear functionals defined on $H^{m,2}_{[a,b]}$ by $\varphi_i(f) = c_i(f)$, $i = \overline{1, n}$ verify $\varphi_k(v_i) = \delta_{ki}$.

□

Theorem 3.4.1. *(existence)*
If the functionals $\varphi_i \in \Phi$, $i = \overline{1,n}$ are bounded and the set:

$$U(y) = \{f \in H^{m,2}_{[a,b]} \mid \varphi_i(f) = y_i,\ i = \overline{1,n}\}$$

is non empty, then the polynomial spline interpolation problem hast at least one a solution.

Proof: We consider the set $U^{(m)} = \{v \mid v = u^{(m)}, u \in U\} \subset L^2_{[a,b]}$ and the problem which consists of the determination of $v^* \in U^{(m)}$ having the property

$$\|v^*\|_2 = \inf_{v \in U^{(m)}} \|v\|_2.$$

If the polynomial spline interpolation problem has a solution, then the considered problem has a solution. If the considered problem has a solution then the polynomial spline interpolation problem has at least one solution. It follows that, if we show that the problem:

$$\|v^*\|_2 = \inf_{v \in U^{(m)}} \|v\|_2$$

has a solution, it results that the polynomial spline interpolation problem has a solution. In the followings we will show that the problem:

$$\|v^*\|_2 = \inf_{v \in U^{(m)}} \|v\|_2$$

has a solution.
We remark that the set $U^{(m)}$ is non empty (because U is non empty). From the linearity of the functionals φ_i for any $u_1, u_2 \in U$ and $\alpha \in [0,1]$ we have:

$$\varphi_i(\alpha u_1 + (1-\alpha)u_2) = \alpha\varphi_i(u_1) + (1-\alpha)\varphi_i(u_2) = \alpha y_i + (1-\alpha)y_i = y_i,$$

$i = \overline{1,n}$. This means that $\alpha u_1 + (1-\alpha)u_2 \in U$ for any $u_1, u_2 \in U$ and $\alpha \in [0,1]$ (U is convex set). Because the derivative $D^{(m)}$ of order m is linear it results that the set $U^{(m)} = D^{(m)}U$ is convex.
We will show that the set $U^{(m)}$ is closed in $L^2_{[a,b]}$, i.e., if (g_k) is a sequence (from $U^{(m)}$) convergent to $g \in L^2_{[a,b]}$ in $L^2_{[a,b]}$ sense, then $g \in U^{(m)}$. For this we will show that there exists a polynomial p of maximum degree $m-1$ such that the function:

$$f(x) = p(x) + \int_a^x \frac{(x-t)^{m-1}}{(m-1)!} \cdot g(t)\ dt$$

belong to U (then, evident $f^{(m)} = g \in U^{(m)}$).

Because $g_k \in U^{(m)}$, there exists $f_k \in U$ such that $g_k = f_k^{(m)}$. It follows:

$$f_k(x) = p_k(x) + \int_a^x \frac{(x-t)^{m-1}}{(m-1)!} \cdot g_k(t)\, dt$$

where p_k are Taylor polynomials of the maximum degree $m-1$.
We suppose that from the n linear functionals $\varphi_1, \varphi_2, \ldots, \varphi_n$, the first m $(m \leq n)$ are linear independent on the set of polynomial of maximum degree $m-1$. In particular, we admit that $\varphi_1, \varphi_2, \ldots, \varphi_m$ are such that the matrix:

$$A = (\varphi_i(v_j))_{\substack{i=\overline{1,m} \\ j=\overline{1,m}}}$$

with $v_j(x) = \dfrac{(x-a)^{j-1}}{(j-1)!}$, is nonsingular.
In this case we have:

$$A \cdot \begin{bmatrix} p_k(a) \\ \vdots \\ p_k^{m-1}(a) \end{bmatrix} = \begin{bmatrix} \varphi_1(p_k) \\ \vdots \\ \varphi_m(p_k) \end{bmatrix}$$

$\varphi_i(f_k) = y_i$ being bounded, and $\varphi_i\left(\displaystyle\int_a^x \frac{(x-t)^{m-1}}{(m-1)!} \cdot g_k(t)\, dt\right)$ convergent to $\varphi_i\left(\displaystyle\int_a^x \frac{(x-t)^{m-1}}{(m-1)!} \cdot g(t)\, dt\right)$, $i = 1, 2, \ldots, n$, from which we have that the sequences $\varphi_i(p_k)$, $k = \overline{1,m}$ are bounded. Hence, every sequence $p_k^{(j)}(a)$, $j = 0, 1, \ldots, m-1$ is bounded and so contains a convergent subsequence $(p_{k_l}^{(j)}(a))$.
Let be $p^{(j)}(a) = \lim\limits_{k_l \to \infty} p_{k_l}^{(j)}(a)$, $j = 0, 1, \ldots, m-1$. Using these values we define a polynomial p of maximum degree $m-1$: $p(x) = p(a) + \dfrac{p'(a)}{1!} \cdot (x-a) + \ldots + \dfrac{p^{(m-1)}(a)}{(m-1)!} \cdot (x-a)^{(m-1)}$. The sequence f_k given by

$$f_k(x) = p_k(x) + \int_a^x \frac{(x-t)^m}{(m-1)!} \cdot g(t)\, dt$$

converges to:

$$f(x) = p(x) + \int_a^x \frac{(x-t)^m}{(m-1)!} \cdot g(t)\, dt.$$

Because the set U is closed we obtain that $f \in U$.

The case $d < m$ can be reduced to the previous case.

Because $U^{(m)}$ is non empty, convex and closed in $L^2_{[a,b]}$, based on a theorem from functional analysis (theorem for the best approximation), it results that the problem $\|v^*\|_2 = \inf_{v \in U^{(m)}} \|v\|_2$ has at least one solution.

□

Theorem 3.4.2. *(uniqueness)*

1. *For any solutions u^*, u^{**} of the polynomial spline interpolation problem, the difference $u^* - u^{**}$ is a polynomial of maximum degree $m-1$.*

2. *The polynomial spline interpolation problem has an unique solution if and only if the set $U_0 = U(0) = \{f \in H^{m,2}_{[a,b]} \mid \varphi_i(f) = 0, i = 1, 2, \ldots, n\}$ does not contain no null polynomials.*

Proof:
1. The first affirmation is a consequence of the uniqueness of the solution of the best approximation problem. Indeed, $(u^*)^{(m)}$ and $(u^{**})^{(m)}$ are solutions of the best approximation problem and hence $(u^*)^{(m)} = (u^{**})^{(m)}$. It results that the difference $u^* - u^{**}$ is a polynomial of maximum degree $m-1$.

2. For proving the second affirmation, first we will show that if the polynomial spline interpolation problem has only one solution then $U_0 = U(0)$ does not contain null polynomials. For this we consider the solution u^* of the polynomial spline interpolation problem and we suppose the contrary, i.e., that the set U_0 contains a polynomial p of degree less than or equal to $m-1$. Considering $u^{**} = p + u^*$, because $\varphi_i(p) = 0$, $i = \overline{1,n}$ results that $\varphi_i(u^{**}) = \varphi_i(u^*)$, $i = \overline{1,n}$. From here we obtain $u^{**} \in U(y)$, which together with the equality $u^{**\,(m)} = u^{*\,(m)}$ proves that u^* and u^{**} are two solutions of the best approximation problem, which is impossible.

Similarly it can be shown that if the set $U_0 = U(0)$ does not contain null polynomials then the polynomial spline interpolation problem has a unique solution.

□

The following theorem establishes an orthogonality property for a spline.

Theorem 3.4.3. *The function $u^* \in U$ is a solution of the polynomial spline interpolation problem if and only if*

$$\int_a^b u^{*\,(m)} \cdot g^{(m)} = 0, \qquad \forall g \in U_0.$$

Proof: $u^* \in U$ is solution of the polynomial spline interpolation problem if and only if $u^{*\,(m)} \in U^{(m)}$ is a solution of the best approximation problem,

and $u^{*(m)} \in U^{(m)}$ is a solution of the best approximation problem if and only if it is orthogonal on $U_0^{(m)}$.

□

Consequence 3.4.1. *If u^* is solution of the polynomial spline interpolation problem then:*

$$\|u^{*(m)}\|_2^2 = \|u^{(m)} - u^{*(m)}\|_2^2 + \|u^{*(m)}\|_2^2, \qquad \forall\, u \in U.$$

Proof: For $u \in U$ we have

$$\|u^{(m)}\|_2^2 = \|u^{(m)} - u^{*(m)}\|_2^2 + \|u^{*(m)}\|_2^2 + 2\int_a^b (u-u^*)^{(m)} \cdot u^{*(m)}.$$

Because $u - u^* \in U_0$, according to Theorem 3.4.3, it results that $\int_a^b (u - u^*)^{(m)} \cdot u^{*(m)} = 0$. □

We consider the set $\mathcal{S}$ defined by:

$$\mathcal{S} = \Big\{ f \in H_{[a,b]}^{m,2} \mid \int_a^b f^{(m)} \cdot g^{(m)} = 0, \;\; \forall g \in U_0 \Big\}.$$

Theorem 3.4.4. *$\mathcal{S}$ is a closed linear subspace of $H_{[a,b]}^{m,2}$.*

Proof: Let be $f_1, f_2 \in \mathcal{S}$ and $\alpha, \beta \in I\!R^1$. We have:

$$\begin{aligned}\int_a^b (\alpha f_1 + \beta f_2)^{(m)} \cdot g^{(m)} &= \int_a^b \alpha f_1^{(m)} \cdot g^{(m)} + \beta f_2^{(m)} \cdot g^{(m)} = \\ &= \alpha \int_a^b f_1^{(m)} \cdot g^{(m)} + \beta \int_a^b f_2^{(m)} \cdot g^{(m)} = 0,\end{aligned}$$

and hence $\mathcal{S}$ is a linear space.

For proving that $\mathcal{S}$ is closed we consider a sequence of functions (f_k) from $\mathcal{S}$ convergent to $f \in H_{[a,b]}^{m,2}$. We should prove that $f \in \mathcal{S}$. From $f_k \in \mathcal{S}$ we have $\int_a^b f_k^{(m)} \cdot g^{(m)} = 0$, and from the convergence condition $f_k \to f$ in $H_{[a,b]}^{m,2}$ results $< f_k - f, g >_{m,2} \xrightarrow[k\to\infty]{} 0$, hence $< (f_k - f)^{(m)}, g^{(m)} >_{L^2} \xrightarrow[k\to\infty]{} 0$.
In this way we have:

$$\int_a^b f^{(m)} \cdot g^{(m)} = \lim_{k\to\infty} \int_a^b (f^{(m)} - f_k^{(m)}) \cdot g^{(m)} + f_k^{(m)} \cdot g^{(m)} = 0$$

for any $y \in U_0$, hence $f \in \mathcal{S}$.

□

Theorem 3.4.5. *$\mathcal{S}$ is the set of all solutions of the polynomial spline interpolation problem if $f \in I\!R^n$ and contains the set of polynomials of maximum degree $m-1$.*

Proof: Let u^* a solution of the polynomial spline interpolation problem. It results that $\int_a^b u^{*\,(m)} \cdot g^{(m)} = 0$ for any $g \in U_0$ and hence $u^* \in \mathcal{S}$. If $f \in \mathcal{S}$ then $\int_a^b f^{(m)} \cdot g^{(m)} = 0$, for any $g \in U_0$, so f is solution of the polynomial spline interpolation problem corresponding to $y_1, y_2, \ldots, y_n$, where $y_i = \varphi_i(f)$, $i = 1, 2, \ldots, n$. The fact that the polynomial p of maximum degree $m-1$ belongs to $\mathcal{S}$ results from the equality $\int_a^b p^{(m)} \cdot g^{(m)} = 0$.

□

Theorem 3.4.6. *Let be $\{v_1, \ldots, v_d\}$ a base of the space $P_{m-1} \cap U_0$ and u_i^* a solution of the polynomial spline interpolation problem on the set $U_i = \{f \in H^{m,2}_{[a,b]} \mid \varphi_i(f) = \delta_{ij},\ j = 1, 2, \ldots, n\}$. The set $\{u_1^*, u_2^*, \ldots, u_n^*\} \cup \{v_1, v_2, \ldots, v_d\}$ is a base for $\mathcal{S}$.*

Proof: Let be $f \in \mathcal{S}$ and $h = f - \sum_{i=1}^{n} u_i^* \cdot \varphi_i(f)$. The function h belongs to the set $\mathcal{S}$ and to U_0; moreover, function h verifies $\int_a^b [h^{(m)}]^2 = 0$. From here $h^{(m)} = 0$, i.e., $h \in P_{m-1}$ and hence $h \in P_{m-1} \cap U_0$. Because the system of vectors $\{v_1, \ldots, v_d\}$ ia a base in $P_{m-1} \cap U_0$ we have:

$$h = \sum_{j=1}^{d} c_j \cdot v_j$$

and thus we obtain:

$$f = \sum_{i=1}^{n} u_i^* \cdot \varphi_i(f) + \sum_{j=1}^{d} c_j \cdot v_j.$$

For showing that the system of functions $\{u_1^*, u_2^*, \ldots, u_n^*, v_1, v_2, \ldots, v_d\}$ is linearly independent, we will consider the relation of linear dependence

$$\sum_{i=1}^{n} a_i \cdot u_i^* + \sum_{j=1}^{d} b_j \cdot v_j = 0.$$

Applying the functional φ_k results

$$\varphi_k\left[\sum_{i=1}^{n} a_i \cdot u_i^* + \sum_{j=1}^{d} b_j \cdot v_j\right] = a_k = 0, \qquad k = 1, 2, \ldots, n,$$

because $\varphi_k(u_i^*) = \delta_{ki}$ and $\varphi_k(v_j) = 0$, $j = 1, \ldots, d$, $(v_j \in U_0)$. Replacing a_k in the relation of linear dependence we obtain

$$\sum_{j=1}^{d} b_j \cdot v_j = 0.$$

Using the linear independence of the vectors $v_1, \ldots, v_d$ we obtain that $b_j = 0$, $j = 1, \ldots, d$.

If $P_{m-1} \cap U_0 = \{0\}$ (the polynomial spline interpolation problem has a unique solution) then the space $\mathcal{S}$ has dimension n and $u_1^*, \ldots, u_n^*$ is a base in $\mathcal{S}$.

□

Due to the properties:

$$\varphi_k(u_i^*) = \delta_{ki}, \qquad k, i = \overline{1, n}$$

functions u_i^* are called **fundamental interpolating spline functions**.

Theorem 3.4.7. *Let be $y = (y_1, \ldots, y_n) \in \mathbb{R}^n$ and $u_1^*, \ldots, u_n^*$ fundamental interpolating spline functions. The function u_y^* defined by*

$$u_y^* = \sum_{i=1}^{n} u_i^* \cdot y_i$$

is solution of the polynomial spline interpolation problem related to the set $U = U(y)$.

Proof: We have $\varphi_k(u_y^*) = y_k$, $k = \overline{1, n}$, hence $u_y \in U$. Because u_i^* are solutions of the polynomial spline interpolation problem which interpolate the sets $U_i = \{f \in H_{[a,b]}^{m,2} \mid \varphi_i(f) = \delta_{ij},\ j = 1, 2, \ldots, n\}$, results that $\int_a^b u_i^{*(m)} \cdot g^{(m)} = 0$, $\forall g \in U_0$, $i = \overline{1, n}$. It follows

$$\int_a^b u_y^{*(m)} \cdot g^{(m)} + \sum_{i=1}^{n} y_i \int_a^b u_i^{*(m)} \cdot g^{(m)} = 0 = 0, \qquad \forall g \in U_0,$$

and hence $u_y^* \in \mathcal{S}$.

□

Remark 3.4.2. The previous theorem shows that if $f \in H^{m,2}_{[a,b]}$ then the function:

$$Sf = \sum_{i=1}^{n} u_i^* \cdot \varphi_i(f)$$

is a spline function which interpolates the function f, i.e. $\varphi_i(Sf) = \varphi_i(f)$. The application $S\colon H^{m,2}_{[a,b]} \to \mathcal{S}$ defined above is a linear operator, and if the polynomial spline interpolation problem has a unique solution then S is idempotent.

Definition 3.4.3. *The operator S defined above is called* ***polynomial interpolating spline operator****.*

The structure of the solution of the polynomial spline interpolation problem depends on the nature of the functionals which defines the interpolating set U. In the followings we will formulate a theorem for the structure of the solution in the case of Birkhoff functionals.
For defining Birkhoff functionals we will consider a division $\Delta : a \leq x_1 < \ldots < x_k \leq b$ of $[a,b]$. For any point of the division, we consider a natural number r_i, $i = \overline{1,k}$ having the property $r_i \leq m-1$, and a set of subscripts $I_i \subset \{0,1,\ldots,r_i\}$, its elements being denoted by $j \in I_i$.

Definition 3.4.4. *The functionals φ_i defined as:*

$$\varphi_{ij}(f) = f^{(j)}(x_i) \qquad i = \overline{1,k},\ j \in I_i$$

are called ***Birkhoff functionals****.*

Theorem 3.4.8. *(structural characterization)*
Let be Φ a set of Birkhoff functionals and U the corresponding interpolating set. The function $u \in U$ is a solution of the polynomial spline interpolation problem if and only if the following properties take place:

i. $u^{(2m)}(x) = 0,\ x \in [x_1, x_k] \backslash \{x_1, \ldots, x_k\}$;

ii. $u^{(m)}(x) = 0,\ x \in (a, x_1) \cup (x_k, b)$;

iii. $u^{(2m-1-\mu)}(x_i - 0) = u^{(2m-1-\mu)}(x_i + 0),\ \mu \in \{0,1,\ldots,m-1\} \backslash I_i$ *for* $i = 1, \ldots, k$.

Remark 3.4.3. The theorem of characterization expresses the fact that a solution u of the polynomial spline interpolation problem , in the case in which Φ is a set of Birkhoff functionals, is a polynomial of degree $2m-i$ on every interval $(x_i, x_{i+1}) \subset (a,b)$, a polynomial of degree $m-1$ on the

extreme intervals $[a, x_1)$ and $(x_k, b]$, and in the points x_i the derivative of order $2m-1-\mu$ is continuous if the value of the μ^{th} derivative in x_i does not belong to Φ. The solution u is called spline of degree $2m-1$ or natural spline of degree $2m-1$.

According to established results we can enounce the following theorem.

Theorem 3.4.9. *If Φ is a set of Birkhoff functionals, U is the corresponding interpolating set and $\mathcal{S}$ is the set of the spline functions which interpolate on U, then the followings assertions are equivalent:*

1. $u \in \mathcal{S} \Leftrightarrow \|u^{(m)}\|_2 = \inf\limits_{u \in U} \|u\|_2$;

2. $u \in \mathcal{S} \Leftrightarrow \int_a^b u^{(m)} \cdot g^{(m)} = 0,\ \forall g \in U_0$;

3. $u \in \mathcal{S} \Leftrightarrow$

$$\begin{cases} u^{(2m)}(x) = 0, & x \in [x_1, x_k] \backslash \{x_i\}_{i=1,\ldots,k} \\ u^{(m)}(x) = 0, & x \in [a, x_1) \cup (x_k, b] \\ u^{(2m-1-\mu)}(x_i + 0) - u^{(2m-1-\mu)}(x_i - 0) = 0, & \mu \in \{0, 1, \ldots, m-1\} \backslash I_i, i = 1, \ldots, k. \end{cases}$$

In particular, if $\varphi_i(f) = f(x_i)$, $i = 1, \ldots, n$, then the condition

$$u^{(2m-1-\mu)}(x_i+0) - u^{(2m-1-\mu)}(x_i-0) = 0,\ \ \mu \in \{0, 1, \ldots, m-1\} \backslash I_i, i = 1, \ldots, k$$

becomes

$$u \in \mathcal{C}^{2m-2}_{[a,b]},$$

and Theorem 3.4.8 permits to write $u(x)$ as follows:

$$u(x) = \sum_{i=0}^{m-1} a_i x^i + \sum_{k=1}^{n} b_k (x - x_k)^{2m-1},$$

in which those $m+n$ parameters a_i and b_k, $i = \overline{0, m-1}$, $k = \overline{1, n}$, are determined by the interpolating conditions $u(x_j) = f(x_j)$, $j = \overline{1, n}$ and from the condition that $u \in P_{m-1}$ on the extreme intervals.

Examples of polynomial spline functions

Let be the interval $[a, b]$ and n points $\{(x_i, y_i)\}_{i=0}^{n-1}$ which determine the knots x_i of the division $\Delta : a = x_0 < x_1 < \ldots < x_{n-1} = b$.

- *The polynomial spline function of first order* $S(x)$ (polygonal line) is the polynomial function determined of $n-1$ polynomials $S_i(x)$ of the first degree (segments of straight lines):

$$S(x) = S_i(x) = s_{i,0} + s_{i,1}(x - x_i) \tag{3.4.1}$$

for $x \in [x_{i-1}, x_i]$, $i = \overline{1, n-1}$, with coefficients $s_{i,0}$, $s_{i,1}$ satisfying the properties:

(i) the spline function passes through every point $\{(x_i, y_i)\}_{i=0}^{n-1}$, i.e. $S(x_i) = y_i$, $i = \overline{0, n-1}$;

(ii) the spline function is continuous on the interval $[a, b]$, i.e. $S_i(x_i) = S_{i+1}(x_i)$, $i = \overline{1, n-2}$.

Imposing for the function $S(x)$ to satisfy the conditions $(i) - (ii)$, the coefficients $s_{i,0}$ and $s_{i,1}$ are obtained, and the following formula for the polynomial spline function of first order is found:

$$S(x) = S_i(x) = y_i + \frac{y_i - y_{i-1}}{x_i - x_{i-1}}(x - x_i),$$

with $x \in [x_{i-1}, x_i]$, $i = \overline{1, n-1}$.

- *polynomial spline function of second order (quadratic spline)* $S(x)$ is the polynomial function determined by $n-1$ polynomials $S_i(x)$ of second degree (segments of parabolas):

$$S(x) = S_i(x) = s_{i,0} + s_{i,1}(x - x_i) + s_{i,2}(x - x_i)^2 \tag{3.4.2}$$

for $x \in [x_{i-1}, x_i]$, $i = \overline{1, n-1}$, with coefficients $s_{i,0}$, $s_{i,1}$ and $s_{i,2}$ satisfying the properties:

(i) the spline function passes through every point $\{(x_i, y_i)\}_{i=0}^{n-1}$, i.e. $S(x_i) = y_i$, $i = \overline{0, n-1}$;

(ii) the spline function is continuous on the interval $[a, b]$, i.e. $S_i(x_i) = S_{i+1}(x_i)$, $i = \overline{1, n-2}$;

(iii) the spline function is smooth on the interval $[a, b]$, i.e. $S_i'(x_i) = S_{i+1}'(x_i)$, $i = \overline{1, n-2}$.

- *polynomial spline function of third order (cubic spline)* $S(x)$ is the polynomial function determined by $n-1$ polynomials $S_i(x)$ of third degree (cubic segments):

$$S(x) = S_i(x) = s_{i,0} + s_{i,1}(x - x_i) + s_{i,2}(x - x_i)^2 + s_{i,3}(x - x_i)^3 \tag{3.4.3}$$

for $x \in [x_{i-1}, x_i]$, $i = \overline{1, n-1}$, with coefficients $s_{i,0}$, $s_{i,1}$, $s_{i,2}$ and $s_{i,3}$ satisfying the properties:

(i) the spline function passes through every point $\{(x_i, y_i)\}_{i=0}^{n-1}$, i.e. $S(x_i) = y_i$, $i = \overline{0, n-1}$;

(ii) the spline function is continuous on the interval $[a, b]$, i.e. $S_i(x_i) = S_{i+1}(x_i)$, $i = \overline{1, n-2}$;

(iii) the spline function is smooth on the interval $[a, b]$, i.e. $S_i'(x_i) = S_{i+1}'(x_i)$, $i = \overline{1, n-2}$;

(iv) the second derivative of the spline function is continuous on the interval $[a, b]$, i.e. $S_i''(x_i) = S_{i+1}''(x_i)$, $i = \overline{1, n-2}$.

Note that each cubic polynomial S_i has $s_{i,0}$, $s_{i,1}$, $s_{i,2}$ and $s_{i,3}$ as unknowns; therefore there are $4n-4$ unknowns corresponding to n knots and hence $n-1$ cubic polynomials, and $4n-6$ equations given by $(i)-(iv)$ (in the case in which we have $n+1$ knots and hence n cubic polynomials, then number of unknowns is $4n$ and number equations is $4n-2$). We still need two more equations that can obtained by imposing boundary conditions at the endpoints x_0 and x_{n-1}. The most commonly boundary conditions are

$$S_1''(x_0) = S_{n-1}''(x_{n-1}) = 0 \tag{3.4.4}$$

or

$$S_1'(x_0) = y_0' \qquad \textit{and} \qquad S_{n-1}'(x_{n-1}) = y_{n-1}'. \tag{3.4.5}$$

The boundary conditions given by Eq. (3.4.4) are called **free** or **natural boundary conditions** and the corresponding cubic spline is called *natural cubic spline*. The boundary conditions given by Eq. (3.4.5) are called **clamped boundary conditions**.

For computing the coefficients $s_{i,0}$, $s_{i,1}$, $s_{i,2}$ and $s_{i,3}$ of the *cubic spline polynomial* we impose for $S(x)$ to satisfy the conditions $(i)-(iv)$. Thus, from Eq. (3.4.3), we have:

$$S_i'(x) = s_{i,1} + 2s_{i,2}(x - x_i) + 3s_{i,3}(x - x_i)^2 \tag{3.4.6}$$

and

$$S_i''(x) = 2s_{i,2} + 6s_{i,3}(x - x_i). \tag{3.4.7}$$

From S_i, we get:

$$S_i(x_i) = s_{i,0} = y_i \tag{3.4.8}$$

and

$$S_{i+1}(x_i) = s_{i+1,0} + s_{i+1,1}h_i + s_{i+1,2}h_i^2 + s_{i+1,3}h_i^3 \tag{3.4.9}$$

for $i = \overline{0, n-2}$, where

$$h_i = x_i - x_{i+1}. \tag{3.4.10}$$

From Eq. (3.4.8), Eq. (3.4.9) and continuity (ii) , we get

$$s_{i+1,1}h_i + s_{i+1,2}h_i^2 + s_{i+1,3}h_i^3 = y_i - y_{i+1}. \tag{3.4.11}$$

Since $S_i'(x_i) = s_{i,1} = S_{i+1}'(x_i)$ and Eq. (3.4.6) we obtain:

$$s_{i,1} = s_{i+1,1} + 2s_{i+1,2}h_i + 3s_{i+1,3}h_i^2.$$

This can be rewritten as

$$2s_{i+1,2}h_i + 3s_{i+1,3}h_i^2 = s_{i,1} - s_{i+1,1} \tag{3.4.12}$$

for $i = \overline{0, n-2}$.
Since $S_i''(x_i) = 2s_{i,2} = S_{i+1}''(x_i)$ and Eq. (3.4.7) we obtain:

$$2s_{i,2} = 2s_{i+1,2} + 6s_{i+1,3}h_i. \tag{3.4.13}$$

Solving this equation for $s_{i+1,3}$ we have:

$$s_{i+1,3} = \frac{s_{i,2} - s_{i+1,2}}{3h_i}. \tag{3.4.14}$$

Substituting for $s_{i+1,3}$ from Eq. (3.4.14) in Eq. (3.4.11) and then solving for $s_{i+1,1}$ gives

$$s_{i+1,1} = \frac{y_i - y_{i+1}}{h_i} - \frac{h_i}{3}(s_{i,2} + 2s_{i+1,2}). \tag{3.4.15}$$

Substituting $s_{i+1,1}$ from Eq. (3.4.15) and $s_{i+1,3}$ from Eq. (3.4.14) in Eq.(3.4.12) and then simplifying yields:

$$h_{i-1}s_{i-1,2}+2(h_i+h_{i-1})s_{i,2}+h_i s_{i+1,2} = 3\frac{y_{i+1}-y_i}{h_i}-3\frac{y_i-y_{i-1}}{h_{i-1}}, \tag{3.4.16}$$

for $i = \overline{1, n-2}$.
For natural or free boundary conditions, from Eq. (3.4.7), we have

$$s_{0,2} = 0, \qquad s_{n-1,2} = 0. \tag{3.4.17}$$

Equations (3.4.16)-(3.4.17) can be written as a tridiagonal system in the unknowns $s_{0,2}, s_{01,2}, \ldots, s_{n-1,2}$ which can be solved by the LU decomposition method developed in the first chapter.
Remark: If conditions $(ii) - (iv)$ are satisfied, functions $S_{i+1}(x)$, $S_{i+1}'(x)$ and $S_{i+1}''(x)$ are replaced with similar conditions written with

$S_{i-1}(x)$, $S'_{i-1}(x)$ and $S''_{i-1}(x)$, then the coefficients $s_{i,2}$, $s_{i,1}$, $s_{i,3}$ and $s_{i,0}$ (in this order) of the *cubic spline polynomial* are given by:

$$h_{i-1}s_{i-1,2}+2(h_i+h_{i-1})s_{i,2}+h_i s_{i+1,2} = 3\frac{y_{i+1}-y_i}{h_i}-3\frac{y_i-y_{i-1}}{h_{i-1}}, \quad (3.4.18)$$

$$s_{i,1} = \frac{y_i-y_{i-1}}{h_{i-1}} - \frac{h_{i-1}}{3}(2s_{i-1,2}+s_{i,2}), \quad (3.4.19)$$

$$s_{i,3} = \frac{s_{i,2}-s_{i-1,2}}{3h_{i-1}}, \quad (3.4.20)$$

$$s_{i,0} = y_i, \quad (3.4.21)$$

where $h_i = x_{i+1} - x_i$.

Exercises

1. Find the natural cubic spline function which interpolates the data:

x_i	27.7	28	29	30
y_i	4.1	4.3	4.1	3.0

Compute S(28.5).

The algorithm for implementation of cubic spline interpolation:

//Construction of tridiagonal system
for $i = 0 \ldots n-1$
  $h_i = x_{i+1} - x_i$
for $i = 2 \ldots n-1$
  $subdiag_i = h_{i-1}$
  $supradiag_i = h_i$
for $i = 1 \ldots n-1$
  $diag_i = 2 \cdot (h_{i-1} + h_i)$
  $term\_liberi_i = 3 \cdot \left(\frac{y_{i+1}-y_i}{h_i} - \frac{y_i-y_{i-1}}{h_{i-1}} \right)$

The LU factorization method is applied for solving the tridiagonal system (constructed above) of $n-1$ equations and $n-1$ unknowns. Solution of this system is the vector $(c_i)_{i=1\ldots n-1}$.

$c_0 = c_n = 0$
for $i = 0 \ldots n-1$
  $a_i = y_i$
  $b_i = \frac{y_{i+1}-y_i}{h_i} - \frac{(2 \cdot c_i + c_{i+1})}{3}$

$$d_i = \frac{c_{i+1} - c_i}{3 \cdot h_i}$$

Input data:

- n

- x_i - knots, $i = 0, \ldots, n$

- y_i - values of f in knots

Output data:

- for $i = 0 \ldots n-1$

$$P_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

Implementation of the above algorithm made in Borland C language:

```
#include<stdio.h>
#include<malloc.h>
float **Matrice(int imin, int imax, int jmin, int jmax);
float *Vector(int n);
void CitireVect(float *a,int init, int n);
void ScriereVect(float *a,int init, int n);
void Tridiag(float *a, float *b, float *c, float *d,float *x,int n);
void spline(float *x, float *y, int n);
void main()
{
    float *y, *x;
    int n;
    printf("Introduceti numarul nodurilor: "); scanf("%d",&n);
    x=Vector(n);
    y=Vector(n);
    printf("Introduceti nodurile: \n");
    CitireVect(x,0,n-1);
    printf("Introduceti valorile functiei in noduri: \n");
    CitireVect(y,0,n-1);
    printf("----------SOLUTIE----------\n");
    spline(x,y,n-1);
}
void spline(float *x, float *y, int n)
{
    float *h, *a, *b, *c, *d, *subD, *diag, *supraD, *lib;
    int i;
    h=Vector(n);
```

```
        a=Vector(n);
        b=Vector(n);
        c=Vector(n);
        d=Vector(n);
        subD=Vector(n);
        diag=Vector(n);
        supraD=Vector(n);
        lib=Vector(n);
        for(i=0;i<n;i++)
        {
            h[i]=x[i+1]-x[i];
        }
        c[0]=0;
        c[n]=0;
        for(i=2;i<n;i++)
        {
            subD[i]=h[i];
            supraD[i]=h[i];
        }
        for(i=1;i<n;i++)
        {
            diag[i]=2*(h[i-1]+h[i]);
            lib[i]=3.0*((y[i+1]-y[i])/h[i]-(y[i]-y[i-1])/h[i-1]);
        }
        Tridiag(subD,diag,supraD,lib,c,n-1);
        for(i=0;i<n;i++)
        {
            a[i]=y[i];
            b[i]=(y[i+1]-y[i])/h[i]-(2.0*c[i]+c[i+1])*h[i]/3.0;
            d[i]=(c[i+1]-c[i])/(3.0*h[i]);
        }
        for(i=0;i<n;i++)
        {
            printf("P%d (x)=%g +%g(x-%g)+ %g (x-%g)^2+ %g (x-%g)^3
\n",
                i,a[i],b[i],x[i],c[i],x[i],d[i],x[i]);
        }
    }
```

3.5 The Bernstein Polynomial

Definition 3.5.1. *Let $f : [a, b] \to \mathbb{R}^1$. The polynomial defined by*

$$(B_m f)(x) = \sum_{k=0}^{m} C_m^k \cdot x^k \cdot (1-x)^{m-k} \cdot f\left(\frac{k}{m}\right), \qquad x \in [0,1],$$

is called ***Bernstein polynomial*** *(of degree m) for the approximation the function f on $[0, 1]$.*

Theorem 3.5.1. *For any $f, g : [0,1] \to \mathbb{R}^1$ and $\alpha, \beta \in \mathbb{R}^1$ we have:*

$$[B_m(\alpha f + \beta g)](x) = \alpha(B_m f)(x) + \beta(B_m g)(x), \qquad \forall x \in [0,1].$$

Proof:

$$[B_m(\alpha f + \beta g)](x) = \sum_{k=0}^{m} C_m^k \cdot x^k \cdot (1-x)^{m-k} \cdot (\alpha f + \beta g)\left(\frac{k}{m}\right) =$$

$$= \sum_{k=0}^{m} C_m^k \cdot x^k \cdot (1-x)^{m-k} \cdot \left(\alpha f\left(\frac{k}{m}\right) + \beta g\left(\frac{k}{m}\right)\right) =$$

$$= \sum_{k=0}^{m} C_m^k \cdot x^k \cdot (1-x)^{m-k} \cdot \alpha f\left(\frac{k}{m}\right) +$$

$$+ \sum_{k=0}^{m} C_m^k \cdot x^k \cdot (1-x)^{m-k} \cdot \beta g\left(\frac{k}{m}\right) =$$

$$= \alpha \sum_{k=0}^{m} C_m^k \cdot x^k \cdot (1-x)^{m-k} \cdot f\left(\frac{k}{m}\right) +$$

$$+ \beta \sum_{k=0}^{m} C_m^k \cdot x^k \cdot (1-x)^{m-k} \cdot g\left(\frac{k}{m}\right) =$$

$$= \alpha(B_m f)(x) + \beta(B_m f)(x).$$

□

Theorem 3.5.2. *Let be* $f : [0,1] \to I\!R^1$. *If* $f(x) \geq 0$ *for any* $x \in [0,1]$ *then:*

$$(B_m f)(x) \geq 0, \qquad \forall x \in [0,1].$$

Proof: Because $x \in [0,1]$, it results that $x^k \geq 0$ and $(1-x)^{m-k} \geq 0$. From $f(x) \geq 0$ for any $x \in [0,1]$, we have:

$$(B_m f)(x) \geq 0, \qquad \forall\, x \in [0,1].$$

□

Theorem 3.5.3. *If* $f(x) \equiv 1$ *then* $(B_m f)(x) \equiv 1$; *if* $f(x) \equiv x$, *then* $(B_m f)(x) \equiv x$ *and if* $f(x) \equiv x^2$ *then* $(B_m f)(x) \equiv x^2 + \dfrac{x(1-x)}{m}$

Theorem 3.5.4. *If* $f : [a,b] \to I\!R^1$ *verifies* $m \leq f(x) \leq M$, $\forall x \in [0,1]$ *then* $(B_m f)(x)$ *verifies* $m \leq (B_m f)(x) \leq M$, $\forall x \in [0,1]$.

Proof: We consider the function $g(x) = f(x) - m \geq 0$, $\forall x \in [0,1]$. Applying Theorem 3.5.2 we obtain $(B_m g)(x) \geq 0$. On the basis of Theorem 3.5.1 we have $(B_m g)(x) = (B_m f)(x) - m$ and hence $(B_m f)(x) - m \geq 0$, from where we obtain $(B_m f)(x) \geq m$. Analogously the inequality $(B_m f)(x) \leq M$ is obtained. □

Theorem 3.5.5. *(Weierstrass approximation)*
If $f \in \mathcal{C}_{[0,1]}$ *then* $B_m f$ *converges uniformly to* f *on* $[0,1]$ *for* $m \to \infty$.

Proof: If f is continuous on $[0,1]$ then it is uniformly continuous and hence $\forall \varepsilon > 0$, $\exists \delta(\varepsilon) > 0$ such that $\forall x', x'' \in [0,1]$ with $|x' - x''| < \delta(\varepsilon)$ we have $|f(x') - f(x'')| < \frac{\varepsilon}{2}$.
We consider the difference $f(x) - (B_m f)(x)$ written in the form:

$$f(x) - (B_m f)(x) = \sum_{k=0}^{m} \left[f(x) - f\left(\frac{k}{m}\right) \right] \cdot C_m^k \cdot x^k \cdot (1-x)^{m-k}.$$

For $x \in [0,1]$ we denote:

$$I_m = \left\{ k \;\middle|\; \left| x - \tfrac{k}{m} \right| < \delta(\varepsilon) \right\}$$

$$J_m = \left\{ k \;\middle|\; \left| x - \tfrac{k}{m} \right| \geq \delta(\varepsilon) \right\}$$

and we consider $M = \max\{|f(x)|: x \in [0,1]\}$. We evaluate the difference $f(x) - (B_m f)(x)$:

$$|f(x) - (B_m f)(x)| \leq \frac{\varepsilon}{2} \sum_{k \in I_m} C_m^k x^k (1-x)^{m-k} + 2M \sum_{k \in J_m} C_m^k x^k (1-x)^{m-k} \leq$$

$$\leq \frac{\varepsilon}{2} + 2M \sum_{k \in J_m} C_m^k x^k (1-x)^{m-k}.$$

From the inequality $|\frac{k}{m} - x| \geq \delta$ it results that $1 \leq (\frac{k}{m} - x)^2/\delta^2$ and hence

$$\sum_{k \in J_m} C_m^k x^k (1-x)^{m-k} \leq \frac{1}{\delta^2} \sum_{k \in J_m} \left(\frac{k}{m} - x\right)^2 C_m^k x^k (1-x)^{m-k} \leq$$

$$\leq \frac{1}{\delta^2} \sum_{k=0}^{m} \left(\frac{k}{m} - x\right)^2 C_m^k x^k (1-x)^{m-k} \leq$$

$$\leq \frac{1}{\delta^2} \cdot \frac{x(1-x)}{m} \leq \frac{1}{4m\delta^2}.$$

It follows:

$$|f(x) - (B_m f)(x)| \leq \frac{\varepsilon}{2} + \frac{M}{2m\delta^2},$$

from where we have:

$$|f(x) - (B_m f)(x)| \leq \varepsilon \quad \text{if} \quad m > \frac{M}{\varepsilon \delta^2}, \qquad x \in [0,1].$$

□

Definition 3.5.2. *The equality:*

$$f = B_m f + R_m f$$

in which $R_m f(x) = f(x) - B_m f(x)$ *se* ***is called Bernstein formula for approximation****; the term* $R_m f(x)$ *is called error term.*

Theorem 3.5.6. *If* $f \in \mathcal{C}^2_{[0,1]}$ *then:*

$$(R_m f)(x) = -\frac{x(1-x)}{2m} \cdot f''(\xi), \qquad 0 \leq \xi \leq 1,$$

and

$$|(R_m f)(x)| \leq \frac{1}{8m} \cdot \sup\{|f''(\xi)| \mid \xi \in [0,1]\}.$$

Proof: We have:

$$(R_m f)(x) = \int_0^1 \varphi(x,t) f''(t) dt$$

cu $\varphi(x,t) = (x-t) - \sum_{k=0}^{m} C_m^k x^k (1-x)^{m-k} \cdot \left[\frac{k}{m} - t\right]$.

From here we obtain:

$$(R_m f)(x) = -\frac{x(1-x)}{2m} \cdot f''(\xi), \qquad 0 \leq \xi \leq 1.$$

The last inequality from the enunciation results from the inequality $x(1-x) \leq \frac{1}{4}$, $\forall x \in [0,1]$. □

Theorem 3.5.7. *If $f : [a,b] \to I\!R^1$, then the Bernstein polynomial of degree m which approximates the function f pe $[a,b]$ is*

$$(B_m f)(x) = \sum_{k=0}^{m} C_m^k \cdot \left(\frac{y-a}{b-a}\right)^k \cdot \left(\frac{b-y}{b-a}\right)^{m-k} \cdot f\left(a + (b-a) \cdot \frac{k}{m}\right) =$$

$$= \frac{1}{(b-a)^m} \cdot \sum_{k=0}^{m} C_m^k \cdot (y-a)^k \cdot (b-y)^{m-k} \cdot f\left(a + (b-a) \cdot \frac{k}{m}\right).$$

Proof: The Bernstein polynomial of degree m is written for the function $g(x) = f(a + (b-a)x)$, $x \in [0,1]$.

□

Exercises

1. Using the Bernstein polynomial formula determine the Bezier curve associated to the points A(1,1), B(2,-1), C(3,2) and D(4,-1).
Remark: The coordinate functions $x(t)$ and $y(t)$ for the Bezier curve can be written as linear combinations of the Bernstein polynomials:

$$x(t) = \sum_{i=0}^{n} C_n^i \cdot t^i (1-t)^{n-i} x_i,$$

$$y(t) = \sum_{i=0}^{n} C_n^i \cdot t^i (1-t)^{n-i} y_i.$$

The algorithm for determination of the Bezier curve using Bernstein polynomial:

$$x(t) = \sum_{i=0}^{n} \left(\begin{array}{c} n \\ i \end{array} \right) t^i (1-t)^{n-i} x_i$$

$$y(t) = \sum_{i=0}^{n} \left(\begin{array}{c} n \\ i \end{array} \right) t^i (1-t)^{n-i} y_i$$

Input data:

- n

- x_i - knots, $i = 0, \ldots, n$

- y_i - values of f in knots

Output data:

- expressions of $x(t)$ and $y(t)$

Implementation of this algorithm made in Borland C language:

```
#include<stdio.h>
#include<math.h>
#include<conio.h>
#include<malloc.h>
float **Matrice(int imin, int imax, int jmin, int jmax);
float *Vector(int imin, int imax);
void CitireVect(float *a,int init, int n);
void ScriereVect(float *a,int init, int n);
int fact(int n);
float comb(int n, int k);
void main()
{
    float *x, *y;
    int n,i,k;
    printf("Introduceti numarul de noduri: "); scanf("%d", &n);
    x=Vector(0,n-1);
    y=Vector(0,n-1);
    printf("Introduceti abscisele punctelor \n");
    CitireVect(x,0,n-1);
    printf("Introduceti ordonatele punctelor \n");
    CitireVect(y,0,n-1);
    printf("x(t)= %g (1-t)^%d+",x[0],n-1);
```

```
        for(k=1;k<n-1;k++)
            printf("%g t^%d (1-t)^%d+",comb(n,k)*x[k],k,n-1-k);
        printf("%g t^%d\n",x[n-1], n-1);
        printf("y(t)= %g (1-t)^%d+",y[0],n-1);
        for(k=1;k<n-1;k++)
            printf("%g t^%d (1-t)^%d+",comb(n-1,k)*y[k],k,n-1-k);
        printf("%g t^%d\n",y[n-1], n-1);
    }
    int fact(int n)
    {
        int i,prod=1;
        for(i=2;i<=n;i++)
            prod*=i;
    return prod;
    }
    float comb(int n, int k)
    {
    return (fact(n)/(fact(k)*fact(n-k)));
    }
```

Chapter 4

Numerical Differentiation

In the followings, we present two methods for approximating the derivatives of the function f in an arbitrary point:

- the approximation of the derivatives by finite differences used in the case in which the function is known, but computing its derivatives is too difficult;
- the approximation of the derivatives using derivatives of the interpolating polynomials (Newton or Lagrange) used in the case when the function is known only by its values in given points.

Let $X = \{x_i \mid x_i \in I\!R^1, i = 0, 1, \ldots, m\}$ be a set of $m+1$ distinct real numbers $x_0 < x_1 < \ldots < x_{m-1} < x_m$ and the function $f : X \to I\!R^1$ known by its values in the points x_i: $y_i = f(x_i), i = 0, 1, \ldots, m$.

4.1 The Approximation of the Derivatives by Finite Differences

In the previous chapter, forward $\triangle$ and backward $\bigtriangledown$ finite difference operators were defined:

$$\triangle f(x) = f(x+h) - f(x),$$

$$\bigtriangledown f(x) = f(x) - f(x-h).$$

Moreover, the Newton polynomial with forward finite differences:

$$p_m(x) = f(x_0) + \frac{\triangle f(x_0)}{h}(x - x_0) + \frac{\triangle^2 f(x_0)}{2!h^2}(x - x_0)(x - x_1) + \ldots$$

$$+\frac{\triangle^m f(x_0)}{m!h^m}(x-x_0)\dots(x-x_{m-1}),$$

the Newton polynomial with backward finite differences:

$$p_m(x) = f(x_m) + \frac{\nabla f(x_m)}{h}(x-x_m) + \frac{\nabla^2 f(x_m)}{2!h^2}(x-x_m)(x-x_{m-1}) + \dots$$

$$+\frac{\nabla^m f(x_m)}{m!h^m}(x-x_m)\dots(x-x_1).$$

were given.

Hence, for a function f with derivatives up to a sufficiently high order, the following approximation was obtained:

$$f(x) = p_m(x) + \frac{f^{(m+1)}(\xi)}{(m+1)!}(x-x_0)(x-x_1)\dots(x-x_{m-1})(x-x_m).$$

Using these results, the approximations of f by finite differences can be obtained.

In order to express the first derivative by forward finite differences, we make $m = 1$ in the corresponding approximation of f and we compute the first derivative with respect to x:

$$f'(x) = \frac{\triangle f(x_0)}{h} + \frac{f''(\xi)}{2}[(x-x_0)+(x-x_1)] + \frac{f^{(3)}(\eta(x))}{3!}(x-x_0)(x-x_1),$$

where $\xi \in (x_0, x_1)$, $\eta \in (x_0, x_1)$.

Substituting $x = x_0$, $x + h = x_1$ we obtain the following approximation for the first order derivative, written in forward finite differences:

$$f'(x) = \frac{\triangle f(x)}{h} - \frac{f''(\xi)}{2}h, \qquad \xi \in (x, x+h),$$

which is equivalent to

$$f'(x) = \frac{f(x+h)-f(x)}{h} - \frac{f''(\xi)}{2}h, \qquad \xi \in (x, x+h).$$

For obtaining approximation of the first order derivative by backward finite differences we make $m = 1$ in the corresponding approximation of f, we compute the first order derivative with respect to x and we substitute $x = x_1$, $x - h = x_0$:

$$f'(x) = \frac{\nabla f(x)}{h} + \frac{f''(\xi)}{2}h, \qquad \xi \in (x-h, x),$$

which is equivalent to

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \frac{f''(\xi)}{2}h, \qquad \xi \in (x-h, x).$$

In similar ways the higher order derivatives can be obtained.
For example, we compute the second derivative by forward finite differences.
Thus, for $m = 2$ the approximation of the function is:

$$f(x) = f(x_0) + \frac{\triangle f(x_0)}{1!h}(x-x_0) + \frac{\triangle^2 f(x_0)}{2!h^2}(x-x_0)(x-x_1) + \frac{f^{(3)}(\xi)}{3!}(x-x_0)(x-x_1)(x-x_2).$$

Computing the first two derivatives with respect to x we obtain:

$$f'(x) = \frac{\triangle f(x_0)}{1!h} + \frac{\triangle^2 f(x_0)}{2!h^2}[(x-x_0) + (x-x_1)] +$$

$$+ \frac{f^{(3)}(\xi)}{3!}[(x-x_1)(x-x_2) + (x-x_0)(x-x_2) + (x-x_0)(x-x_1)] +$$

$$+ \frac{f^{(4)}(\eta(x))}{4!}(x-x_0)(x-x_1)(x-x_2),$$

$$f''(x) = \frac{\triangle^2 f(x_0)}{2!h^2} + \frac{f^{(3)}(\xi)}{3}[(x-x_0) + (x-x_1) + (x-x_2)] +$$

$$+ \frac{[f^{(4)}(\eta) + f^{(4)}(\xi_1)]}{4!}[(x-x_1)(x-x_2) + (x-x_0)(x-x_2) + (x-x_0)(x-x_1)] +$$

$$+ \frac{f^{(5)}(\eta_1(x))}{5!}(x-x_0)(x-x_1)(x-x_2),$$

where $\xi, \eta, \xi_1, \eta_1 \in (x_0, x_2)$.
Substituting $x = x_1$, $x - h = x_0$, $x + h = x_2$ we obtain:

$$f''(x_1) = \frac{\triangle^2 f(x_0)}{2h^2} - \frac{h^2}{12}f^{(4)}(\xi),$$

which is equivalent to

$$f''(x_1) = \frac{f(x_0+2h) - 2f(x_0+h) + f(x_0)}{h^2} - \frac{h^2}{12}f^{(4)}(\xi), \qquad \xi \in (x_0, x_0+2h).$$

Hence, the approximation of the second order derivative written with forward finite difference is:

$$f''(x_0) = \frac{f(x_0+h) - 2f(x_0) + f(x_0-h)}{h^2} - \frac{h^2}{12}f^{(4)}(\xi), \qquad \xi \in (x_0-h, x_0+h).$$

4.2 The Approximation of the Derivatives Using Derivatives of the Interpolating Polynomials

The first derivative of the function f known by its values in the knots $x_0 < x_1 < \ldots < x_{m-1} < x_m$ of $[a, b]$ can be expressed by the first derivative of the function of Newton interpolating polynomial:

$$f'(x) = \frac{d}{dx}\left(f(x_0) + (\mathcal{D}^1 f)(x_0)(x - x_0) + (\mathcal{D}^2 f)(x_0)(x - x_0)(x - x_1) + \ldots\right.$$
$$\left. + (\mathcal{D}^m f)(x_0)(x - x_0)(x - x_1)\ldots(x - x_{m-1}) + R_m(x)\right),$$

or Lagrange interpolating polynomial:

$$f'(x) = \frac{d}{dx}\left(\sum_{i=0}^{k} \frac{(x - x_0)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_m)}{(x_i - x_0)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_m)} \cdot f(x_i) + (R_m f)(x)\right).$$

The derivative of higher order $f^{(n)}(x)$ of the function f is obtained by deriving n times the interpolating polynomial function.

Exercises

1. Approximate the first order derivative of the function $f(x) = \sqrt{x}$, using its values at the knots: $x_0 = 1$, $x_1 = 1.5$, $x_2 = 2$, $x_3 = 2.5$, $x_4 = 3$.
2. Approximate the derivatives $f'(0.1)$, $f''(0.2)$ if the values of the function f at the following knots are given:

x_i	0.1	0.2	0.3	0.4
$f(x_i)$	0.995	0.98007	0.95534	0.92106

Chapter 5

Numerical Integration

Sometimes it is not possible to evaluate a definite integral exactly by well-known techniques and hence numerical methods are required. Thus, we seek numerical procedures to evaluate the definite integral given by

$$\int_a^b f(x)dx$$

where f is continuous or Riemann-Darboux integrable on $[a, b]$. To approximate this integral we look for $\sum_{j=0}^{N} f(x_j) \cdot c_j$, where x_j are $N+1$ distinct points $a = x_0 < x_1 < \ldots < x_{N-1} < x_N = b$ called the quadrature points or nodes (knots), and the quantities c_j are coefficients called weights. The basic problem is the selection of the nodes and coefficients so that

$$\left| \int_a^b f(x)dx - \sum_{j=0}^{N} f(x_j) \cdot c_j \right|$$

is minimum for a large class of functions.
The classical method is to replace the function f by a polynomial that can be easily integrated. The use of the interpolating polynomial as replacement function leads to a variety of integration formulas, called the **Newton-Cotes formulas**. The interpolating polynomial can be obtained using a Newton divided difference formula for the interpolating polynomial or Lagrange interpolating polynomial.

5.1 The Newton-Cotes Formula, Trapezoidal Rule, Simpson Formula

Let $f : [a,b] \to I\!R^1$ a Riemann-Darboux integrable function and the equidistant knots $x_i = a + ih$, $i = 0, 1, 2, \ldots, N$ of $[a,b]$, ($h = \dfrac{b-a}{N}$ and $x_N = b$).
We consider the Lagrange interpolating polynomial for f:

$$P_N(x) = \sum_{j=0}^{N} f(x_j) \cdot L_j(x)$$

where

$$L_j(x) = \prod_{\substack{i=0 \\ i \neq j}}^{N} \frac{x - x_i}{x_j - x_i},$$

and the integral of the interpolating polynomial:

$$\int_a^b P_N(x)dx = \sum_{j=0}^{N} f(x_j) \cdot c_j \qquad \text{where} \qquad c_j = \int_a^b \prod_{\substack{i=0 \\ i \neq j}}^{N} \frac{x - x_i}{x_j - x_i}\, dx.$$

The **quadrature Newton-Cotes formulas** approximate the value of the integral $\int_a^b f(x)dx$ with the value of the integral $\int_a^b P_N(x)dx$ of the interpolating polynomial:

$$\int_a^b f(x)dx \approx \int_a^b P_N(x)dx = \sum_{j=0}^{N} f(x_j) \cdot c_j \qquad \text{where } \; c_j = \int_a^b \prod_{\substack{i=0 \\ i \neq j}}^{N} \frac{x - x_i}{x_j - x_i}\, dx.$$

The **Trapezoidal rule** and the **Simpson formula** represent particular cases of the quadrature Newton-Cotes formulas. Thus, for $N = 1$ we have:

$$\int_a^b f(x)dx \approx \frac{b-a}{2} \cdot [f(b) + f(a)] = \frac{h}{2} \cdot [f(b) + f(a)]$$

called the **trapezoidal rule** (without error term).
If we apply the trapezoidal rule on each subinterval $[x_{i-1}, x_i]$ of the interval $[a,b]$ (x_i are equidistant knots, $x_i = a + ih$, $i = 0, 1, 2, \ldots, N$), we obtain:

$$\int_a^b f(x)dx = \int_{x_0}^{x_1} f(x)dx + \int_{x_1}^{x_2} f(x)dx + \ldots + \int_{x_{N-1}}^{x_N} f(x)dx \approx$$

$$\approx \frac{h}{2} \cdot \left[f(a) + 2 \cdot \sum_{i=1}^{N-1} f(x_i) + f(b) \right]$$

called **composite trapezoidal rule** (without error term).
For $N = 2$ in the quadrature Newton-Cotes formulas we obtain:

$$\int_a^b f(x)dx \approx \frac{h}{3} \cdot [f(a) + 4f(a+h) + f(b)]$$

called the **Simpson formula**.
In the followings we will present some quadrature Newton-Cotes formulas:
For $N = 3$:

$$\int_a^b f(x)dx \approx \frac{3h}{8} \cdot [f(a) + 3f(a+h) + 3f(a+2h) + f(b)].$$

For $N = 4$:

$$\int_a^b f(x)dx \approx \frac{2h}{45} \cdot [7f(a) + 32f(a+h) + 12f(a+2h) + 32f(a+3h) + 7f(b)].$$

For $N = 5$:

$$\int_a^b f(x)dx \approx \frac{5h}{288} \cdot [19f(a) + 75f(a+h) + 50f(a+2h) + 50f(a+3h) + 75f(a+4h) + 19f(b)].$$

For $N = 6$:

$$\int_a^b f(x)dx \approx \frac{h}{140} \cdot [41f(a) + 216f(a+h) + 27f(a+2h) + 272f(a+3h) +$$

$$+ 27f(a+4h) + 216f(a+5h) + 41f(b)].$$

For $N = 7$:

$$\int_a^b f(x)dx \approx \frac{7h}{17280} \cdot [751f(a) + 3577f(a+h) + 1323f(a+2h) + 2989f(a+3h) +$$

$$+ 2989f(a+4h) + 1323f(a+5h) + 3577f(a+6h) + 751f(b)].$$

In these cases, the truncation errors are established by calculus on the base of the *Mean Value Theorem for Definite Integrals*:

Theorem 5.1.1. *(Mean Value Theorem for Definite Integrals)*
Let f be continuous on a closed interval [a,b] and w be an integrable function which does not change sign on [a,b]. Then there is at least one number $c \in [a,b]$ such that

$$\int_a^b f(x)w(x)dx = f(c)\int_a^b w(x)dx.$$

Thus, for $N = 1$ and $x_0 = a$, $x_1 = b$ we have:

$$f(x) = f(a)\frac{x-b}{a-b} + f(b)\frac{x-a}{b-a} + \frac{f''(\xi)}{2!}(x-a)(x-b),$$

where $a < \xi(x) < b$. Integrating from a to b we obtain

$$\int_a^b f(x)dx = \frac{(b-a)}{2}[f(a)+f(b)] + \frac{1}{2}\int_a^b f''(\xi)(x-a)(x-b)dx.$$

On the base of the *Mean Value Theorem*, the above error term becomes:

$$\frac{1}{2}\int_a^b f''(\xi)(x-a)(x-b)dx = \frac{1}{2}f''(c)\int_a^b (x-a)(x-b)dx,$$

where $a < c < b$. Let $u = x - a$ and $h = b - a$. Then error term is

$$\frac{1}{2}\int_a^b f''(\xi)(x-a)(x-b)dx = \frac{1}{2}f''(c)\cdot\frac{-h^3}{6},$$

and hence the truncation error for $N = 1$ is given by

$$E_1 = -\frac{h^3}{12}\cdot f^{(2)}(c).$$

Analogues, we obtain the followings truncation errors:

$$N = 2 \qquad E_2 = -\frac{h^5}{90} \cdot f^{(4)}(c);$$

$$N = 3 \qquad E_3 = -\frac{3h^5}{80} \cdot f^{(4)}(c);$$

$$N = 4 \qquad E_4 = -\frac{8h^7}{975} \cdot f^{(6)}(c);$$

$$N = 5 \qquad E_5 = -\frac{275h^7}{12096} \cdot f^{(6)}(c);$$

$$N = 6 \qquad E_6 = -\frac{9h^9}{1400} \cdot f^{(8)}(c);$$

$$N = 7 \qquad E_7 = -\frac{8183h^9}{518400} \cdot f^{(8)}(c).$$

Exercises

1. Use the trapezoidal rule, the composite trapezoidal rule and the Simpson formula, respectively, to approximate the definite integral:

$$\int_0^1 \frac{1}{1+x^2}\, dx$$

The algorithms for the trapezoidal rule, composite trapezoidal rule and for the Simpson formula are:

// trapezoidal rule

$h = b - a$

$$\int_a^b f(x)dx \approx \frac{h}{2} \cdot [f(a) + f(b)]$$

// composite trapezoidal rule

$$h = \frac{b-a}{n}$$

$$\int_a^b f(x)dx \approx \frac{h}{2} \cdot \left[f(a) + 2 \cdot \sum_{i=1}^{n-1} f(x_i) + f(b) \right]$$

where $x_i = a + i \cdot h$

// the Simpson formula

$$h = \frac{b-a}{2}$$

$$\int_a^b f(x)dx \approx \frac{h}{3} \cdot [f(a) + 4 \cdot f(a+h) + f(b)]$$

Input data:

- a and b - the ends of the interval on which we approximate definite integral

- n - number of the subintervals

Output data

- value of the integral computed using those 3 methods

Implementation in Borland C:

```
#include<stdio.h>
#include<ctype.h>
#include<math.h>
float f(float x)
{
    return x;
}
float trapezgen(float a, float b, int n)
{
    int i;
    float h,s=0.0;
    h=(b-a)/(n);
    for(i=1;i<n;i++)
        s+=f(a+i*h);
return (h*0.5*(f(a)+f(b))+h*s);
}
float simpson(float a, float b)
{
    int i;
    float h;
    h=(b-a)/2;
return ((h/3)*(f(a)+4*f(a+h)+f(b)));
}
float trapez(float a, float b)
```

```
{
    int i;
    float h;
    h=b-a;
return ((h/2)*(f(a)+f(b)));
}
void main(void){
    int a,b,n;
    char op;
    printf("\n\nIntroduceti a: ");scanf("%d",&a);
    printf("\nIntroduceti b: ");scanf("%d",&b);
    printf("\nIntroduceti n: ");scanf("%d",&n);
    printf("\nMeniu\n");
   printf("Alegeti una dintre optiuni: \n\n metoda (T)rapezului simpla\n
        metoda (G)enerala a trapezelor \n metoda (S)impson \n e(X)it
\n");
    fflush(stdin);
    scanf("%c",&op);
    op=toupper(op);
    while (op !='X')
    {
        switch (op)
            {
                case 'T' : printf("Aproximarea integralei prin metoda
trapezului:%g ", trapez(a,b));
                break;
            case 'G' : printf("Aproximarea integralei prin metoda generala
a trapezelor:%g ",
                        trapezgen(a,b,n));
                break;
                case 'S' : printf("Aproximarea integralei prin metoda lui
Simpson %g ",simpson(a,b));
                break;
            }
        printf("\nMeniu\n");
        printf("\n Alegeti una dintre optiuni: \n\n metoda (T)rapezului
simpla\n metoda (G)enerala a
            trapezelor \n metoda (S)impson \n e(X)it \n");
        fflush(stdin);
        scanf("%c",&op);
        op=toupper(op);
```

```
        }
    }
```

5.2 Gaussian Integration Formulas

The Newton-Cotes formulas are of the form:

$$\int_a^b f(x)\,dx \approx a_0 f(x_0) + a_1 f(x_1) + \ldots + a_N f(x_N), \tag{5.2.1}$$

where the knots $x_0, x_1, \ldots, x_N$ of the interval $[a, b]$ are equidistant.

These methods are clearly preferable for integrating a function that is in equally-spaced tabulated form. However, if a function f is known analytically, there is no need to require equally-spaced (equidistant) nodes for the integrating formulas.

If the knots $x_0, x_1, \ldots, x_N$ are not fixed in advance and if there are no other restrictions on them, then in Eq. (5.2.1) there are $2N + 2$ unknowns or parameters $a_0, a_1, \ldots, a_N$ and $x_0, x_1, \ldots, x_N$, which should satisfy $2N + 2$ equations.

We can obtain the $2N+2$ equations imposing the formula (5.2.1) to be more accurate for polynomials $1, x, x^2, \ldots, x^{2N+1}$. Gauss showed that by selecting $x_0, x_1, \ldots, x_N$ properly it is possible to construct formulas far more accurate than the corresponding Newton-Cotes formulas. The formulas based on this principle are called **Gaussian integration formulas**.

In the followings, we will determine the parameters in the case of two points. More precisely, we will determine the four parameters a_0, a_1, x_0, x_1, (2N+2=4), if the integral involved is of the form $\int_{-1}^{1} f(x)\,dx$.

Hence, the parameters a_0, a_1, x_0, x_1 will be computed such that:

$$\int_{-1}^{1} f(x)\,dx \approx a_0 f(x_0) + a_1 f(x_1)$$

and to obtain an equality for the polynomials $1, x, x^2, x^3$. Imposing these conditions, the following system is obtained:

$$\begin{cases} a_0 + a_1 & = 2 \\ a_0 x_0 + a_1 x_1 & = 0 \\ a_0 x_0^2 + a_1 x_1^2 & = \frac{2}{3} \\ a_0 x_0^3 + a_1 x_1^3 & = 0. \end{cases} \tag{5.2.2}$$

The solution of this nonlinear system is:

$$a_0 = a_1 = 1$$
$$-x_0 = x_1 = \tfrac{\sqrt{3}}{3}$$

and the integration formula becomes:

$$\int_{-1}^{1} f(x)\,dx \approx f\left(-\frac{\sqrt{3}}{3}\right) + f\left(\frac{\sqrt{3}}{3}\right). \tag{5.2.3}$$

This integration formula is called the **two-points Gaussian integration formula**.

It is remarkable that by adding $-\frac{\sqrt{3}}{3}$ and $\frac{\sqrt{3}}{3}$, we get the exact value of an integral of any polynomial degree three or less.

In order to use Eq. (5.2.1) for computing integrals $\int_a^b f(x)\,dx$ on an arbitrary segment $[a,b]$ we make a change of variable:

$$x = \frac{(b-a)t + b + a}{2}$$

and we obtain:

$$\begin{aligned}\int_a^b f(x)\,dx &= \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{(b-a)t+a+b}{2}\right)dt \\ &\approx \frac{b-a}{2}\left[f\left(\frac{(a-b)\sqrt{3}+3(a+b)}{6}\right) + f\left(\frac{(b-a)\sqrt{3}+3(a+b)}{6}\right)\right].\end{aligned}$$

Using the same technique as the one presented for obtaining formula (5.2.3), we can determine formulas for a bigger number of terms x_i and a_i. The inconvenience consists of the difficulty for obtaining solutions of these nonlinear systems, so we present an alternative derivation of these formulas.
Since x_i are unknowns, we use the Lagrange interpolating polynomial which allows arbitrarily-spaced base points :

$$f(x) = \sum_{j=0}^{N} f(x_j)\cdot L_j(x) + \frac{f^{(N+1)}(\xi(x))}{(N+1)!}\prod_{j=0}^{N}(x - x_j) \tag{5.2.4}$$

where $L_j(x)$ is

$$L_j(x) = \prod_{\substack{i=0\\ i\neq j}}^{N}\frac{x - x_i}{x_j - x_i} \qquad \text{and} \qquad -1 < \xi(x) < 1.$$

If f is a polynomial of degree $2N+1$ or less, then the equality (5.2.1) should give the exact value of the integral. In this case, the term $\frac{f^{(N+1)}(\xi(x))}{(N+1)!}$ is a polynomial of degree N or less and will be denoted by $q_N(x)$:

$$q_N(x) = \frac{f^{(N+1)}(\xi(x))}{(N+1)!}. \tag{5.2.5}$$

Replacing (5.2.5) in (5.2.4) and integrating between -1 and 1 we obtain

$$\int_{-1}^{1} f(x)\,dx = \sum_{j=0}^{N} f(x_j) \int_{-1}^{1} L_j(x)\,dx + \int_{-1}^{1} q_N(x) \cdot \prod_{j=0}^{N}(x - x_j)\,dx. \tag{5.2.6}$$

We want to select x_j in such a way that the error term in Eq. (5.2.6) vanishes since $f(x)$ is a polynomial of degree $2N+1$ or less. It follows that we want:

$$\int_{-1}^{1} q_N(x) \cdot \prod_{j=0}^{N}(x - x_j)\,dx = 0. \tag{5.2.7}$$

Because $\prod_{j=0}^{N}(x-x_j)$ is a polynomial of degree $N+1$ and $q_N(x)$ is a polynomial of degree N or less, the equality (5.2.7) is verified if the polynomial $\prod_{j=0}^{N}(x-x_j)$ of degree $N+1$ is orthogonal on all polynomials of degree N or less, on the interval $[-1,1]$.
The **Legendre polynomials** defined by

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ P_i(x) &= \frac{1}{i} \cdot [(2i-1) \cdot x \cdot P_{i-1}(x) - (i-1) \cdot P_{i-2}(x)] \qquad i = 2, 3, \ldots \end{aligned}$$

are orthogonal polynomials over $[-1,1]$ with respect to the weight function $w(x) = 1$:

$$\int_{-1}^{1} P_i(x) \cdot P_j(x) = 0, \qquad i \neq j.$$

The Legendre polynomials are also linearly independent and, therefore, $q_N(x)$ can be written as a linear combination of Legendre polynomials $P_i(x)$, $i = 0, 1, 2, \ldots, N$. If in $\prod_{j=0}^{N}(x - x_j)$ we choose x_j, $j = 0, 1, \ldots, N$ as the zeros

of the Legendre polynomials $P_{N+1}(x)$, then the equality (5.2.7) is satisfied because $\prod_{j=0}^{N}(x - x_j)$ will be collinear with $P_{N+1}(x)$ (this fact it is possible because the zeros of Legendre polynomials $P_{N+1}(x)$ are real and distinct). By selecting the zeros of Legendre polynomial $P_{N+1}(x)$ as knots for (5.2.1), the equality (5.2.6) is reduced to the equality

$$\int_{-1}^{1} f(x)\,dx = \sum_{j=0}^{N} f(x_j) \int_{-1}^{1} L_j(x)\,dx$$

for any polynomial f of degree $2N + 1$ or less. Therefore,

$$\int_{-1}^{1} f(x)\,dx = \sum_{j=0}^{N} f(x_j) \int_{-1}^{1} L_j(x)\,dx + \frac{f^{(2N+2)}(\eta)}{(2N+2)!} \int_{-1}^{1} \sum_{i=0}^{N} (x - x_i)^2\,dx$$

and thus

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{j=0}^{N} a_j f(x_j) \tag{5.2.8}$$

where $a_j = \int_{-1}^{1} L_j(x)\,dx$ and x_j, $j = 0, 1, \ldots, N$ are the zeros of Legendre polynomial $P_{N+1}(x)$.

Exercises

1. Using the Gaussian formulas for $N = 2$ and $N = 3$ approximate the definite integral:

$$\int_{0}^{1} e^{2x}\,dx.$$

Chapter 6

Differential Equations, Initial-Value Problems

Differential equations are divided into two classes, **ordinary** and **partial**, according to the number of independent variables present in the differential equations; one for ordinary and more than one for partial. The **order** of the differential equation s the order of its highest derivative. The general solution of N^{th}-order ordinary differential equation contains N independent arbitrary constants. To determine these N arbitrary constants, we need N conditions. If these conditions are prescribed at one point, then these conditions are called **initial conditions**. A differential equation with initial conditions is called an **initial-value problem** (IVP). If these N conditions are prescribed at different points, then these conditions are called **boundary conditions**. A differential equation with boundary conditions is called a **boundary-value problem** (BVP).

These two types of problems have different properties. An initial-value problem can be identified with a time-dependent problem. In this chapter we develop numerical methods (single-step and multi-step methods) to solve first order differential equations with given initial conditions, and numerical methods to solve second order differential equations with given boundary conditions .

6.1 Finite Difference Method for a Numerical Solution of Initial-Value Problems (IVP)

Let be the initial-value problem

$$\begin{cases} y' & = & f(x,y) \\ y(x_0) & = & y_0 \end{cases} \tag{6.1.1}$$

where $f : (\alpha,\beta) \times (\gamma,\delta) \to I\!R^1$ is a function of $\mathcal{C}^1$-class and $x_0 \in (\alpha,\beta)$, $y_0 \in (\gamma,\delta)$.
We consider the points $x_{i+1} = x_i + h = x_0 + (i+1)h$ for $i = 0, 1, \ldots, N-1$, $h > 0$, and we admit that $x_i \in (\alpha,\beta)$ for $i = \overline{0, N-1}$; $a = x_0$ and $b = x_N = x_0 + Nh$.
If the maximal domain of the solution of (IVP) (6.1.1) contains the points x_i which are referred to as mesh points, $i = \overline{0, N-1}$, then in these points the solution $y = y(x)$ of (IVP) verifies:

$$y'(x_i) = f(x_i, y(x_i)), \qquad i = \overline{0, N-1}. \tag{6.1.2}$$

The simplest way to approximate Eq. (6.1.2) is to replace the derivative $y'(x_i)$ by a divided difference of first order:

$$y'(x_i) \approx \frac{y(x_i + h) - y(x_i)}{h}, \qquad i = \overline{0, N-1}.$$

In this way, the following equation with differences results:

$$y(x_{i+1}) = y(x_i) + hf(x_i, y(x_i)), \qquad i = \overline{0, N-1}, \tag{6.1.3}$$

called **Euler's formula** or **Euler's forward formula**.
Starting with $y(x_0) = y_0$ for $i = 0$ and using the Euler's formula (6.1.3), we first compute $y(x_1)$, then $y(x_2)$, $y(x_3)$, and so on. If h is sufficiently small, then values $y(x_1), y(x_2), \ldots, y(x_N)$ approximate numerical values of the solution of (IVP) (6.1.1) at the points $x_1, x_2, \ldots, x_N$.
The accuracy of a numerical solution is given by the error. More precisely, three central concepts should be followed into a numerical method:

1. convergence;

2. consistency;

3. stability.

A numerical method is said *convergent* if the numerical solution approaches the exact solution as the step size h goes to zero.
A numerical method is said *consistent*if the ration between the local error (difference between the result given by the method and the exact solution) and the step size h, goes to zero.
A numerical method is said*stable* if the errors which appear in a certain step are not amplified in the next steps (i.e. small and controllable errors for producing stable solutions).

If in equation (6.1.2), the derivative $y'(x_i)$ is replaced by

$$y'(x_i) \approx \frac{y(x_i) - y(x_i - h)}{h}, \qquad i = \overline{1, N},$$

then the following equation with differences is obtained:

$$\begin{array}{l} y(x_i) - y(x_{i-1}) - h \cdot f(x_i, y(x_i)) = 0, \qquad i = \overline{1, N}, \\ \text{or} \\ y(x_{i+1}) = y(x_i) + h \cdot f(x_{i+1}, y(x_{i+1})), \qquad i = \overline{0, N-1}, \end{array} \tag{6.1.4}$$

called **implicit Euler formula** or **backward Euler formula**.
The equation with differences (6.1.4) should be solved for obtaining numerically the approximative values of the solution of the (IVP)(6.1.1).
If in equation (6.1.2), the derivative $y'(x_i)$ is replaced by

$$y'(x_i) \approx \frac{y(x_i + h) - y(x_i - h)}{2h}, \qquad i = \overline{1, N},$$

then the following equation with differences is obtained:

$$y(x_{i+1}) = y(x_{i-1}) + 2h \cdot f(x_i, y(x_i)), \qquad i = \overline{1, N-1}, \tag{6.1.5}$$

called **midpoint formula**. For solving (6.1.5), the value $y(x_i)$ should be founded using other method.

Exercises

1. Solve the following (IVP):

$$\begin{cases} y'(x) &= x - 5y(x) \\ y(0) &= 1 \end{cases} \qquad x \in [0, 1]$$

using Euler formula for $n = 4$.

The algorithm for solving an (IVP) using Euler method:

$h = \dfrac{b - x_0}{n}$

for $i = \overline{1 \ldots n}$

$y_i = y_{i-1} + h \cdot f(x_{i-1}, y_{i-1})$

$x_i = x_{i-1} + h$

Input data:

- x_0, b - the ends of the interval for x

- $y_0 = y(x_0)$

- n - the points number

Output data:

- couples (x_i, y_i), $i = \overline{0, n}$

Implementation of the above algorithm in Borland C language:

```
#include<stdio.h>
#include<malloc.h>
float *Vector(int imin, int imax);
float f(float x, float y);
void Euler(float x0, float y0, float b, int n);
void main()
{
    float x0, y0,b;
    int n;
    printf("n= "); scanf("%d", &n);
    printf("b= "); scanf("%f", &b);
    printf("x0= "); scanf("%f", &x0);
    printf("y0= "); scanf("%f", &y0);
    Euler( x0, y0, b, n);
}
void Euler(float x0, float y0, float b, int n)
{
    float h, *x, *y;
    int i;
    x=Vector(0,n);
    y=Vector(0,n);
    x[0]=x0;
    y[0]=y0;
    h=(b-x[0])/n;
```

```
    for(i=1;i<=n;i++)
    {
        y[i]=y[i-1]+h*f(x[i-1],y[i-1]);
        x[i]=x[i-1]+h;
    }
    for(i=0;i<=n;i++)
    {
        printf("x[%d]=%f ", i,x[i]);
        printf("y[%d]=%f\n", i,y[i]);
    }
}
float f(float x, float y)
{
return (x-5*y);
}
float *Vector(int imin, int imax)
{
    float *p;
    p=(float *)malloc((size_t)((imax-imin+1) * sizeof(float)));
return (p-imin);
}
```

6.2 The Taylor Method for a Numerical Solution of IVP

Let be the (IVP)

$$\begin{cases} y' & = & f(x,y) \\ y(x_0) & = & y_0 \end{cases} \tag{6.2.1}$$

where $f : (\alpha, \beta) \times (\gamma, \delta) \to I\!R^1$ is an indefinite derivable function and $x_0 \in (\alpha, \beta)$, $y_0 \in (\gamma, \delta)$.
The solution $y = y(x)$ of (IVP) (6.2.1) is an indefinite derivable and verifies:

$$y(x) = y(x_0) + \frac{y'(x_0)}{1!}(x - x_0) + \frac{y''(x_0)}{2!}(x - x_0)^2 + \ldots + \tag{6.2.2}$$

$$+ \frac{y^{(n)}(x_0)}{n!}(x - x_0)^n + \frac{y^{n+1}(\xi)}{(n+1)!}(x - x_0)^{n+1},$$

where $|x_0 - \xi| < |x_0 - x|$.

If we denote by h the difference $x - x_0$, then (6.2.2) can be written as follows:

$$y(x_0 + h) = y(x_0) + \frac{y'(x_0)}{1!}\, h + \frac{y''(x_0)}{2!}\, h^2 + \ldots + \frac{y^{(n)}(x_0)}{n!}\, h^n + \frac{y^{n+1}(\xi)}{(n+1)!}\, h^{n+1}. \tag{6.2.3}$$

We know $y(x_0)$ from the right-hand side term of this equality, but we do not know the derivatives $y'(x_0)$, $y''(x_0)$, $\ldots, y^{(n)}(x_0)$ and $y^{(n+1)}(\xi)$. In order to find them, let us first find $y'(x)$, $y''(x)$, $\ldots, y^{(n)}(x)$ and $y^{(n+1)}(\xi)$. By successive derivation we have

$$y'(x) = f(x, y)$$

$$y''(x) = \frac{d}{dx}[f(x, y(x))] = f'(x, y) = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} \cdot y'(x) = f_x + f_y \cdot f$$

$$y^{(3)}(x) = f''(x, y) = f_{xx}(x, y(x)) + 2f(x, y(x)) \cdot f_{xy}(x, y(x)) + f^2(x, y(x)) \cdot f_{yy}(x, y(x)) +$$

$$+ f_x(x, y(x)) \cdot f_y(x, y(x)) + f(x, y(x)) \cdot f_y^2(x, y(x))$$

. . . . . .

$$y^{(n)}(x) = f^{(n-1)}(x, y).$$

Using these, the equality (6.2.3) becomes:

$$\begin{aligned} y(x_0 + h) \;\; = \;\; & y(x_0) + \frac{f(x_0, y_0)}{1!}\, h + \frac{f'(x_0, y_0)}{2!}\, h^2 + \frac{f''(x_0, y_0)}{3!}\, h^3 + \ldots + \\ & + \frac{f^{(n-1)}(x_0, y_0)}{n!}\, h^n + \frac{f^{(n)}(\xi, y(\xi))}{(n+1)!}\, h^{n+1}. \end{aligned} \tag{6.2.4}$$

Since $f^{(k)}(x_0, y_0)$ is complicated (otherwise, we would solve the equation analytically), evaluation of higher derivatives is time consuming. Therefore instead of using a high degree Taylor series over a relatively large distance, we divide the interval $[x_0, b]$ into small subintervals and use a lower-degree Taylor series over each subinterval. Let be $a = x_0 < x_1 < x_2 < \ldots < x_N = b$ a partition of the interval $[a = x_0, b]$ into N equally-spaced subintervals of length $h = \frac{b-a}{N}$; $x_i = x_0 + ih$, $i = \overline{0, N}$.

From (6.2.4) we have:

$$\begin{aligned} y(x_1) &= y(x_0) + \frac{f(x_0,y_0)}{1!}\,h + \frac{f'(x_0,y_0)}{2!}\,h^2 + \frac{f''(x_0,y_0)}{3!}\,h^3 + \ldots + \\ &+ \frac{f^{(n-1)}(x_0,y_0)}{n!}\,h^n + \frac{f^{(n)}(\xi,y(\xi))}{(n+1)!}\,h^{n+1}. \end{aligned} \tag{6.2.5}$$

For convenience we denote by $T_n(x,y,h)$ the expression

$$T_n(x,y,h) = f(x,y) + \frac{f'(x,y)}{2!}\,h + \ldots + \frac{f^{(n-1)}(x,y)}{n!}\,h^{n-1} \tag{6.2.6}$$

and we write (6.2.5) in the form

$$y(x_1) = y(x_0) + h \cdot T_n(x_0,y_0,h) + \frac{f^{(n)}(\xi,y(\xi))}{(n+1)!}\,h^{n+1}. \tag{6.2.7}$$

Since we do not know ξ, we can not compute the last term; however, since h is small, we may ignore the last term and we obtain:

$$y(x_1) \approx y_0 + h \cdot T_n(x_0,y_0,h). \tag{6.2.8}$$

The approximate value $y(x_1)$ is now known, so we can find the approximate value of y at x_2:

$$y(x_2) = Y_1 + h \cdot T_n(x_1,Y_1,h),$$

where

$$Y_1 = y_0 + h \cdot T_n(x_0,y_0,h).$$

Similarly, we find the approximate values of y at $x_3, x_4, \ldots, x_N$:

$$\left\{ \begin{array}{lcl} Y_0 & = & y(x_0) \\ Y_{i+1} & = & Y_i + h \cdot T_n(x_i,Y_i,h) \quad \text{for} \quad i = 0,1,\ldots,N-1, \end{array} \right. \tag{6.2.9}$$

where

$$T_n(x_i,y_i,h) = f(x_i,y_i) + \frac{h}{2!}\,f'(x_i,y_i) + \ldots + \frac{h^{n-1}}{n!}\,f^{(n-1)}(x_i,y_i).$$

This method for solving numerically an (IVP) is called **Taylor Series method of order n**. The Taylor Series method of first order is also called Euler's method.
In the case of Euler method we have:

$$\left\{ \begin{array}{lcl} Y_0 & = & y(x_0) \\ Y_{i+1} & = & Y_i + h \cdot f(x_i,Y_i) \quad \text{for} \quad i = 0,1,\ldots,N-1. \end{array} \right. \tag{6.2.10}$$

Exercises

1. Solve the following (IVP):

$$\begin{cases} y'(x) &= y(x) - x \\ y(0) &= 0 \end{cases} \qquad x \in [0,1]$$

using Taylor Series method of order 1, 2 and 3, respectively, for $h = 0.25$.

6.3 The Runge-Kutta Method of the Second Order

In order to solve an initial-value problem:

$$\begin{cases} y' &= f(x,y) \\ y(x_0) &= y_0 \end{cases} \tag{6.3.1}$$

we can use a Taylor Series method of higher order or better accuracy; however, in order to use high orders, we need to evaluate high order derivatives of $f(x,y)$. The Runge-Kutta methods do not require the evaluation of the derivatives of $f(x,y)$ and at the same time, they keep the desirable property of higher-order local truncation error.
Let us start with the Taylor Series method of order two:

$$y(x_{i+1}) = y(x_i) + h \cdot f(x_i, y(x_i)) + \frac{h^2}{2!}\, f'(x_i, y(x_i)) + \frac{h^3}{3!}\, f''(\xi_i, y(\xi_i)). \tag{6.3.2}$$

The idea is to avoid $f'(x_i, y(x_i))$ approximating by forward difference:

$$f'(x_i, y(x_i)) = \frac{f(x_i + h, y(x_i + h)) - f(x_i, y(x_i))}{h} - \frac{h}{2}\, f''(\xi_i, y(\xi_i)),$$

and hence Eq. (6.3.2) becomes:

$$y(x_{i+1}) = y(x_i) + \frac{h}{2}\,[f(x_i, y(x_i)) + f(x_i + h, y(x_i + h))] - \frac{h^3}{12}\, f''(\eta_i, y(\eta_i)), \tag{6.3.3}$$

which conducts to the equation with differences:

$$Y_{i+1} = Y_i + \frac{h}{2}\,[f(x_i, Y_i) + f(x_{i+1}, Y_{i+1})]. \tag{6.3.4}$$

In this equation, the unknown Y_{i+1} from the right-hand side is replaced using Euler's formula

$$Y_{i+1} = Y_i + hf(x_i, Y_i)$$

yielding

$$Y_{i+1} = Y_i + \frac{h}{2}\,[f(x_i, Y_i) + f(x_{i+1}, Y_i + hf(x_i, Y_i))]. \tag{6.3.5}$$

This formula is called the **Runge-Kutta formula of second order**.
Comparing it with the Taylor Series method of second order

$$Y_{i+1} = Y_i + h \cdot f(x_i, Y_i) + \frac{h^2}{2!}\, f'(x_i, Y_i) \tag{6.3.6}$$

the difference is clear. In equation (6.3.5) we do not need $f'(x_i, Y_i)$ and yet it has the same truncation order as equation (6.3.6).

Our objective is to develop a general procedure to derive Eq. (6.3.5) from Eq. (6.3.6) and other similar higher-order methods.
We are looking at the formula:

$$Y_{i+1} = Y_i + w_1 k_1 + w_2 k_2, \tag{6.3.7}$$

where

$$k_1 = h \cdot f(x_i, Y_i)$$
$$k_2 = h \cdot f(x_i + \alpha_2 \cdot h, Y_i + \beta_{21} \cdot k_1)$$

and the constants w_1, w_2, α_2, and β_{21} should be determined such that the Eqs. (6.3.7) and (6.3.6) represent the same equation with differences.
For this aim, we write the left-hand side term of the equation (6.3.7) as follows:

$$Y_{i+1} = Y_i + h \cdot f(x_i, Y_i) + \frac{h^2}{2!} \cdot f'(x_i, Y_i) + \frac{h^3}{3!} \cdot f''(x_i, Y_i) + \text{h.o.t (higher-order terms)}$$

in which replacing

$$f = f(x_i, Y_i) \qquad f_x = f_x(x_i, Y_i) \qquad f_y = f_y(x_i, Y_i)$$
$$f_{xx} = f_{xx}(x_i, Y_i) \qquad f_{xy} = f_{xy}(x_i, Y_i) \qquad f_{yy} = f_{yy}(x_i, Y_i)$$

we obtain

$$\begin{aligned} Y_{i+1} \;&=\; Y_i + h \cdot f + \frac{h^2}{2!} \cdot [f_x + f \cdot f_y] + \\ &+ \frac{h^3}{3!} \cdot [f_{xx} + 2f \cdot f_{xy} + f^2 \cdot f_{yy} + f_y(f_x + f \cdot f_y)] + \text{h.o.t} \end{aligned} \tag{6.3.8}$$

For k_2 from the right-hand side of the equation with differences (6.3.7), we use the Taylor Series formula of second order of a function of two variables,

which gives

$$k_2 = h \cdot f(x_i + \alpha_2 \cdot h, Y_i + \beta_{21} \cdot k_1) =$$

$$= h \cdot \left\{ f(x_i, Y_i) + \left(\alpha_2 \cdot h \cdot \frac{\partial}{\partial x} + \beta_{21} \cdot k_1 \cdot \frac{\partial}{\partial y} \right) f(x_i, Y_i) + \right.$$

$$\left. + \frac{1}{2!} \left(\alpha_2 \cdot h \cdot \frac{\partial}{\partial x} + \beta_{21} \cdot k_1 \cdot \frac{\partial}{\partial y} \right)^2 f(x_i, Y_i) \right\} + \text{t.o.s.}$$

$$= h \left\{ f + \alpha_2 h f_x + \beta_{21} h f \cdot f_y + \frac{h^2}{2} (\alpha_2^2 f_{xx} + 2\alpha_2 \beta_{21} f \cdot f_{xy} + \beta_{21}^2 f^2 f_{yy}) \right\} + \text{h.o.t}$$

Replacing k_1 and k_2 in the right-hand side of the equation (6.3.7) we obtain

$$\begin{aligned} Y_{i+1} &= Y_i + h(w_1 + w_2) \cdot f + w_2 h^2 \cdot [\alpha_2 f_x + \beta_{21} f \cdot f_x] + \\ &\quad + \frac{w_2 h^3}{2} \cdot [\alpha_2^2 f_{xx} + 2\alpha_2 \beta_{21} f \cdot f_{xy} + \beta_{21}^2 f^2 f_{xy}] + \text{h.o.t} \end{aligned} \tag{6.3.9}$$

Comparing the coefficients of h and h^2 on the right-hand of Eqs. (6.3.8) and (6.3.9), by identification we get

$$\begin{cases} w_1 + w_2 = 1 \\ \alpha_2 \cdot w_2 = \frac{1}{2} \\ \beta_{21} \cdot w_2 = \frac{1}{2} \end{cases} \tag{6.3.10}$$

We have four unknowns and only three equations, therefore, we have one degree of freedom in the solution of the Eq. (6.3.10).
Solving Eq. (6.3.10) in terms of α_2 we get

$$\begin{cases} w_2 = \dfrac{1}{2\alpha_2} \\ w_1 = 1 - w_2 = 1 - \dfrac{1}{2\alpha_2} = \dfrac{2\alpha_2 - 1}{2\alpha_2} \\ \beta_{21} = \dfrac{1}{2w_2} = \alpha_2 \end{cases} \tag{6.3.11}$$

We can use one degree of freedom to select the truncation error as small as possible that produced h^3 terms in the expansions of the Eqs. (6.3.8) and (6.3.9).

The asymptotic form of the error term is found by taking the difference between the h^3 terms in Eqs. (6.3.8) and (6.3.9), and is given by:

$$error = h^3 \cdot \left\{ \left(\frac{1}{6} - \frac{\alpha_2}{4} \right) [f_{xx} + 2f \cdot f_{xy} + f^2 \cdot f_{yy}] + \frac{1}{6}\, f_y \cdot [f_x + f \cdot f_y] \right\}. \tag{6.3.12}$$

If M and L are such that

$$|f(x,y)| < M \quad \text{and} \quad \left| \frac{\partial^{i+j} f}{\partial x^i \partial y^j} \right| \leq \frac{L^{i+j}}{M^{i+j}} \quad \text{for} \quad i+j \leq n$$

then error is bounded by:

$$|error| \leq h^3 \cdot M\, L^2 \cdot \left[4 \cdot \left(\frac{1}{6} - \frac{\alpha_2}{4} \right) + \frac{1}{3} \right] \tag{6.3.13}$$

and it is minimum for $\alpha_2 = \frac{2}{3}$:

$$|error| \leq \frac{M \cdot L^2}{3} \cdot h^3. \tag{6.3.14}$$

For $\alpha_2 = \frac{2}{3}$ we have $\beta_{21} = \frac{2}{3}$, $w_1 = \frac{1}{4}$, $w_2 = \frac{3}{4}$ and the equation with differences (6.3.7) becomes:

$$Y_{i+1} = Y_i + \frac{h}{4} \left[f(x_i, Y_i) + 3\, f\Big(x_i + \frac{2}{3}\, h, Y_i + \frac{2}{3}\, h \cdot f(x_i, Y_i) \Big) \right] \tag{6.3.15}$$

which can be written as follows:

$$\begin{cases} \hat{Y}_{i+2/3} = Y_i + \dfrac{2}{3}\, h \cdot f(x_i, Y_i) \\ \\ Y_{i+1} = Y_i + \dfrac{h}{4} \left[f(x_i, Y_i) + 3\, f\Big(x_i + \dfrac{2}{3}\, h, \hat{Y}_{i+2/3} \Big) \right] \end{cases} \tag{6.3.16}$$

This form shows that the method can be viewed as a predictor-corrector method. The first equation predicts $\hat{Y}_{i+2/3}$, a preliminary value, while the second equations gives the correct value Y_{i+1} by using the preliminary value. Other solutions of the system (6.3.10) which are used in literature are:

a) $\alpha_2 = \beta_{21} = 1, w_1 = w_2 = \frac{1}{2}$.
In this case the equation with differences is:

$$Y_{i+1} = Y_i + \frac{h}{2} [f(x_i, Y_i) + f(x_i + h, Y_i + h \cdot f(x_i, Y_i))] \tag{6.3.17}$$

which is known as the **Improved Euler method** or **Heun method**. Eq. (6.3.17) can be written as follows:

$$\begin{cases} \hat{Y}_{i+1} = Y_i + h \cdot f(x_i, Y_i) \\ Y_{i+1} = Y_i + \dfrac{h}{2}\left[f(x_i, Y_i) + f(x_i + h, \hat{Y}_{i+1})\right] \end{cases} \tag{6.3.18}$$

The first equation predicts the preliminary value $\hat{Y}_{i+1}$, while the second equation gives the corrected value Y_{i+1} by using the preliminary value $\hat{Y}_{i+1}$. This cab be viewed as a predictor-corrector method, too.

b) $\alpha_2 = \beta_{21} = \frac{1}{2}, w_1 = 0, w_2 = 1$.
In this case the equation with differences is:

$$Y_{i+1} = Y_i + h \cdot f\left(x_i + \frac{h}{2}, Y_i + \frac{h}{2} \cdot f(x_i, Y_i)\right) \tag{6.3.19}$$

which is known as the **Modified Euler method** or **Improved Polygon method**.
Writing the Eq. (6.3.19) as:

$$\begin{cases} \hat{Y}_{i+1/2} = Y_i + \dfrac{h}{2} \cdot f(x_i, Y_i) \\ Y_{i+1} = Y_i + h \cdot f(x_i + \dfrac{h}{2}, \hat{Y}_{i+1/2}) \end{cases} \tag{6.3.20}$$

this can be viewed as a predictor-corrector method, too.

Exercises

1. Solve (IVP):

$$\begin{cases} y'(x) &= x^3 + x \cdot y^2(x) \\ y(0) &= 1 \end{cases} \qquad x \in [0, 1]$$

using Runge-Kutta method of second order for $n = 10$.

The algorithm for solving numerically an (IVP) using Runge-Kutta method of second order:

$h = \dfrac{b - x_0}{n}$

for $i = 1 \ldots n$

$k_1 = h \cdot f(x_{i-1}, y_{i-1})$
$k_2 = h \cdot f(x_{i-1} + \alpha_2 \cdot h, y_{i-1} + \beta_{21} \cdot k_1)$
$y_i = y_{i-1} + w_1 \cdot k_1 + w_2 \cdot k_2$
$x_i = x_{i-1} + h$

- For Improved Euler method

$\alpha_2 = \beta_{21} = 1$
$w_1 = w_2 = \frac{1}{2}$

- for Modified Euler method

$\alpha_2 = \beta_{21} = \frac{1}{2}$
$w_1 = 0$
$w_2 = 1$

- for Runge-Kutta with the smallest error

$\alpha_2 = \beta_{21} = \frac{2}{3}$
$w_1 = \frac{1}{4}$
$w_2 = \frac{3}{4}$

Input data:

- x_0, b - the ends of the interval for x

- $y_0 = y(x_0)$

- n - number of points

Output data:

- couples (x_i, y_i), $i = \overline{0, n}$

Implementation of this algorithm in Borland C language:

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
float f(float x, float y)
{
    return -1.0*x*y*y;
}
void rk2(float a2, float b21, float w1, float w2, float h, float x0, float y0,
int n){
    int i;
    float h1,h2,x,y;
    for(i = 1; i<=n; i++)
```

```
        {
            h1=h*f(x0,y0);
            h2=h*f(x0+a2*h,y0+b21*h1);
            y=y0+w1*h1+w2*h2;
            x=x0+h;
            printf("(%g,%g)\n",x,y);
            x0=x;
            y0=y;
        }
    }
    void main()
    {
        int n,i;
        float b,h,x0,y0,a2,b21,w1,w2;
        char c;
        printf("Introduceti n: "); scanf("%d",&n);
        printf("Introduceti b: "); scanf("%f",&b);
        printf("Introduceti x0: "); scanf("%f",&x0);
        printf("Introduceti y0: "); scanf("%f",&y0);
        h=(b-x0)/(float)n;
        do
        {
            printf("\n");
            printf(" Metode Runge-Kutta \n");
            printf("1.Metoda Euler imbunatatita\n");
            printf("2.Metoda Euler modificata\n");
            printf("3.Metoda Runge-Kutta cu cea mai mica eroare\n");
            printf("4.Iesire\n");
            printf("Introduceti optiunea: "); scanf("%s", &c);
            printf("\n");
            switch(c)
            {
                case '1': { // Metoda Euler imbunatatita
                          a2=1.0;
                          b21=1.0;
                          w1=0.5;
                          w2=0.5;
                          rk2(a2,b21,w1,w2,h,x0,y0,n);
                        break;
                        }
                case '2': { // Metoda Euler modificata
```

```
                    a2=0.5;
                    b21=0.5;
                    w1=0;
                    w2=1;
                    rk2(a2,b21,w1,w2,h,x0,y0,n);
                  break;
                  }
            case '3': { // Metoda Runge-Kutta cu cea mai mica eroare
                    a2=2.0/3.0;
                    b21=2.0/3.0;
                    w1=0.25;
                    w2=0.75;
                    rk2(a2,b21,w1,w2,h,x0,y0,n);
                  break;
                  }
            case '4': exit(0);
            default: {printf("Optiune gresita!"); break;}
        }
    }while (c != '4');
}
```

6.4 The Runge-Kutta Method of the Third Order and Fourth Order

The Runge-Kutta method of second order were derived from $Y_{i+1} = Y_i + w_1k_1 + w_2k_2$. In order to derive the **Runge-Kutta method of the third-order**, we add one term or stage w_3k_3:

$$Y_{i+1} = Y_i + w_1k_1 + w_2k_2 + w_3k_3 \tag{6.4.1}$$

where

$$\begin{aligned} k_1 &= h \cdot f(x_i, Y_i) \\ k_2 &= h \cdot f(x_i + \alpha_2 h, Y_i + \beta_{21}k_1) \\ k_3 &= h \cdot f(x_i + \alpha_3 h, Y_i + \beta_{31}k_1 + \beta_{32}k_2) \end{aligned}$$

and k_1, k_2, k_3, α_2, β_{21}, α_2, β_{21} α_3, β_{31}, β_{32} are eight unknown constants which should be determined by expanding Eq. (6.4.1) in a Taylor series around (x_i, Y_i) and comparing it with the corresponding terms of the Taylor Series method of order three:

$$Y_{i+1} = Y_i + h \cdot f(x_i, Y_i) + \frac{h^2}{2!} \cdot f'(x_i, Y_i) + \frac{h^3}{3!} \cdot f''(x_i, Y_i) + \text{higher-order terms} \tag{6.4.2}$$

Expanding k_2 and k_3 in Taylor series around (x_i, Y_i) and keeping terms up to h^3, we get:

$$k_2 = h \cdot \{f + h[\alpha_2 f_x + \beta_{21} f \cdot f_y] +$$

$$+ \frac{h^2}{2} \cdot [\alpha_2^2 f_{xx} + 2\alpha_2\beta_{21} f \cdot f_{xy} + \beta_{21}^2 f^2 \cdot f_{yy}] + \text{higher-order terms}\}$$

$$k_3 = h \cdot \{f + h \cdot [\alpha_3 f_x + (\beta_{31} + \beta_{32}) f \cdot f_y] +$$

$$+ \frac{h^2}{2} \cdot [2\beta_{32}(\alpha_2 f_x + \beta_{21} f \cdot f_y) \cdot f_y + \alpha_3^2 f_{xx} +$$

$$+ 2\alpha_3(\beta_{31} + \beta_{32}) f \cdot f_{xy} + (\beta_{31} + \beta_{32})^2 f^2 \cdot f_{xy}] + \text{higher-order terms}\}.$$

Replacing k_1, k_2 and k_3 in Eq. (6.4.1) we obtain

$$Y_{i+1} = Y_i + h \cdot (w_1 + w_2 + w_3) \cdot f + h^2 \cdot \Big\{(w_2\alpha_2 + w_3\alpha_3) \cdot f_x +$$

$$+ f \cdot f_y[w_2\beta_{21} + w_3(\beta_{31} + \beta_{32})]\Big\} + h^3\Big\{\tfrac{1}{2} \cdot (w_2\alpha_2^2 + w_3\alpha_3^2) \cdot f_{xx} +$$

$$+ [w_2\alpha_2\beta_{21} + w_3\alpha_3(\beta_{31} + \beta_{32})] \cdot f \cdot f_{xy} +$$

$$+ \tfrac{1}{2} \cdot [w_2\beta_{21}^2 + w_3(\beta_{31} + \beta_{32})^2] \cdot f^2 \cdot f_{yy} +$$

$$+ w_3\alpha_2\beta_{32} \cdot f_x f_y + w_3\beta_{21}\beta_{32} f \cdot f_y^2\Big\} + \text{higher-order terms}$$

Substituting for $f'(x_i, Y_i)$ and $f''(x_i, Y_i)$ in equation (6.4.2) we have:

$$Y_{i+1} = Y_i + h \cdot f + \frac{h^2}{2} \cdot (f_x + f \cdot f_y) +$$
$$+ \frac{h^3}{3!} \cdot (f_{xx} + 2f \cdot f_{xy} + f^2 \cdot f_{yy} + f_x \cdot f_y + f \cdot f_y^2) + \text{higher-order terms.}$$

Comparing the coefficients of h, h^2, and h^3, by identification we get:

$$\begin{cases} w_1 + w_2 + w_3 = 1 \\ w_2\alpha_2 + w_3\alpha_3 = \frac{1}{2} \\ w_2\beta_{21} + w_2(\beta_{31} + \beta_{32}) = \frac{1}{2} \\ w_2\alpha_2^2 + w_3\alpha_3^2 = \frac{1}{3} \\ w_2\alpha_2\beta_{21} + w_2\alpha_3(\beta_{31} + \beta_{32}) = \frac{1}{3} \\ w_2\beta_{21}^2 + w_2(\beta_{31} + \beta_{32})^2 = \frac{1}{3} \\ w_3\alpha_2\beta_{32} = \frac{1}{6} \\ w_3\beta_{21}\beta_{32} = \frac{1}{6} \end{cases} \tag{6.4.3}$$

Since $w_3\alpha_2\beta_{32} = \dfrac{1}{6} = w_3\beta_{21}\beta_{32}$ it results that $\alpha_2 = \beta_{21}$.
Further,

$$w_2\alpha_2^2 + w_3\alpha_3^2 = \tfrac{1}{3} = w_2\alpha_2\beta_{21} + w_2\alpha_3(\beta_{31} + \beta_{32})$$

and

$$\alpha_2 = \beta_{21}$$

implies $\alpha_3 = \beta_{31} + \beta_{32}$. Thus, the system (6.4.3) is reduced to the system:

$$\begin{cases} w_1 + w_2 + w_3 = 1 \\ w_2\beta_{21} + w_2(\beta_{31} + \beta_{32}) = \frac{1}{2} \\ w_2\beta_{21}^2 + w_2(\beta_{31} + \beta_{32})^2 = \frac{1}{3} \\ w_3\beta_{21}\beta_{32} = \frac{1}{6} \end{cases} \tag{6.4.4}$$

which has 4 equations and 6 unknowns. Thus we obtain a two parameter family of such methods.
The classical method of order three is given by

$$w_1 = w_3 = \frac{1}{6}; \quad w_2 = \frac{2}{3}; \quad \alpha_2 = \frac{1}{2}; \quad \alpha_3 = 1; \quad \beta_{21} = \frac{1}{2}; \quad \beta_{31} = -1; \quad \beta_{32} = 2$$

and it follows that

$$Y_{i+1} = Y_i + \frac{1}{6} \cdot (k_1 + 4k_2 + k_3) \tag{6.4.5}$$

$$\text{where} \qquad \begin{aligned} k_1 &= h \cdot f(x_i, Y_i) \\ k_2 &= h \cdot f\left(x_i + \tfrac{h}{2}, Y_i + \tfrac{k_1}{2}\right) \\ k_3 &= h \cdot f(x_i + h, Y_i - k_1 + 2k_2). \end{aligned}$$

The higher-order Runge-Kutta formulas can be derived in the same way; however, as the order increases the complexity increases very rapidly. The best known **Runge-Kutta method of fourth stage and fourth order** is given by:

$$Y_{i+1} = Y_i + \frac{1}{6} \cdot (k_1 + 2k_2 + 2k_3 + k_4) \tag{6.4.6}$$

$$\text{where} \qquad \begin{aligned} k_1 &= h \cdot f(x_i, Y_i) \\ k_2 &= h \cdot f\left(x_i + \tfrac{h}{2}, Y_i + \tfrac{k_1}{2}\right) \\ k_3 &= h \cdot f\left(x_i + \tfrac{h}{2}, Y_i + \tfrac{k_2}{2}\right) \\ k_4 &= h \cdot f(x_i + h, Y_i + k_3). \end{aligned}$$

At first sight, these formulas seem to be complicated, but they are easy to program and they get a very good speed of convergence.

6.5 The Adams-Bashforth and Adams-Moulton Methods

The methods discussed in the previous subsections depend only on Y_i to obtain the value of Y_{i+1}; therefore those methods are called **single-step methods**. As we move away from x_0, the error $|y(x_{i+1}) - Y_{i+1}|$ increases. Since the approximate values at x_0, $x_1, \ldots, x_i$ are available, it seems reasonable to use those values to approximate Y_{i+1} accurately.

A method that uses k approximate values of y to compute Y_{i+1} is called a **k-step method** or **multi-step method**.
In the followings we present multi-step methods for solving the (IVP):

$$\begin{cases} y' &= f(x, y) \\ y(x_0) &= y_0 \end{cases}$$

where $f : (a, b) \times (c, d) \to \mathbb{R}^1$ is a function of $\mathcal{C}^1$-class.
The solution $y = y(x)$ of this (IVP) verifies:

$$y'(x) = f(x, y(x)), \qquad \forall\, x \in (\alpha, \beta)\ \ a \le \alpha < x_0 < \beta \le b,$$

where (α, β) is the maximum interval on which the solution is defined.

For any knots $x_0 < x_1 < x_2 < \ldots < x_i < x_{i+1} < \ldots < x_N$ from the maximum interval (α, β) on which the solution is defined, we have:

$$\int_{x_i}^{x_{i+1}} y'(x)\,dx = \int_{x_i}^{x_{i+1}} f(x, y(x))\,dx \qquad \text{or} \qquad y(x_{i+1}) = y_i + \int_{x_i}^{x_{i+1}} f(x, y(x))\,dx.$$

Since $y(x)$ is not known and $f(x, y(x))$ cannot be integrated exactly, we approximate $f(x, y(x))$ by an interpolating polynomial that uses the previously obtained data points $(x_i, f(x_i, y(x_i))), (x_{i-1}, f(x_{i-1}, y(x_{i-1}))), \ldots, (x_{i-k}, f(x_{i-k}, y(x_{i-k})))$.

Let $k = 0$. Then the equality:

$$y(x_{i+1}) = y(x_i) + \int_{x_i}^{x_{i+1}} f(x, y(x))\,dx$$

becomes:

$$y(x_{i+1}) = y(x_i) + \int_{x_i}^{x_{i+1}} [f(x_i, y(x_i)) + (x - x_i) \cdot f'(\eta_i(x), y(\eta_i(x)))]\,dx$$

$$= y(x_i) + h \cdot f(x_i, y(x_i)) + \frac{h^2}{2} \cdot f'(\xi_i, y(\xi_i)),$$

where: $h = x_{i+1} - x_i$, $x_i < \xi_i < x_{i+1}$ and $f' = \dfrac{\partial f}{\partial x} + \dfrac{\partial f}{\partial y} \cdot y'$;. This gives the one-step Euler method

$$Y_{i+1} = Y_i + hf(x_i, Y_i).$$

Let $k = 1$. Although any interpolating polynomial through $(x_i, f(x_i, y(x_i)))$ and $(x_{i-1}, f(x_{i-1}, y(x_{i-1})))$ can be used, it is very convenient to use the Newton backward difference formula. Let $h = x_{i+1} - x_i = x_i - x_{i-1}$. Then the equality

$$y(x_{i+1}) = y(x_i) + \int_{x_i}^{x_{i+1}} f(x, y(x))\,dx$$

becomes

$$\begin{aligned} y(x_{i+1}) &= y(x_i) + \int_{x_i}^{x_{i+1}} \left[f(x_i, y(x_i)) + (x - x_i) \cdot \frac{\nabla f(x_i, y(x_i))}{h} + \right. \\ &\quad \left. + \frac{(x - x_i)(x - x_{i-1})}{2!} \cdot f''(\eta_i, y(\eta_i)) \right] dx \\ &= y(x_i) + h \cdot f(x_i, y(x_i)) + \frac{h}{2} \cdot \nabla f(x_i, y(x_i)) + \\ &\quad + \frac{f''(\xi_i, y(\xi_i))}{2!} \int_{x_i}^{x_{i+1}} (x - x_i)(x - x_{i-1})\, dx \\ &= y(x_i) + \frac{h}{2} \cdot \{3\, f(x_i, y(x_i)) - f(x_{i-1}, y(x_{i-1}))\} + \frac{5}{12} \cdot h^3 \cdot f''(\xi_i, y(\xi_i)) \end{aligned}$$

where $x_i < \eta_i$ and $\xi_i < x_{i+1}$. This two-step method that uses the information at the points x_i and x_{i-1} is called the **second-order Adams-Bashforth method** and is given by

$$Y_{i+1} = Y_i + \frac{h}{2} \cdot [3\, f(x_i, Y_i) - f(x_{i-1}, Y_{i-1})].$$

Similarly for $k = 2$, using three points $(x_i, f(x_i, y(x_i)))$, $(x_{i-1}, f(x_{i-1}, y(x_{i-1})))$ and
$(x_{i-2}, f(x_{i-2}, y(x_{i-2})))$, we get

$$\begin{aligned} y(x_{i+1}) = y(x_i) + \frac{h}{12} \cdot \{&23\, f(x_i, y(x_i)) - 16\, f(x_{i-1}, y(x_{i-1})) + \\ &+ 5\, f(x_{i-2}, y(x_{i-2}))\} + \frac{3}{8} \cdot h^4 \cdot f^{(3)}(\xi_i, y(\xi_i)), \end{aligned}$$

and the corresponding equation with differences is

$$Y_{i+1} = Y_i + \frac{h}{12} \cdot \{23\, f(x_i, Y_i) - 16\, f(x_{i-1}, Y_{i-1}) + 5\, f(x_{i-2}, Y_{i-2})\}.$$

For $k = 3$ we have:

$$\begin{aligned} y(x_{i+1}) = y(x_i) + \frac{h}{24} \cdot \{&55\, f(x_i, y(x_i)) - 59\, f(x_{i-1}, y(x_{i-1})) + \\ &+ 37\, f(x_{i-2}, y(x_{i-2})) - 9\, f(x_{i-3}, y(x_{i-3}))\} + \frac{251}{720} \cdot h^5 \cdot f^{(4)}(\xi_i, y(\xi_i)) \end{aligned}$$

and the corresponding equation with differences:

$$Y_{i+1} = Y_i + \frac{h}{24} \cdot \{55\, f(x_i, Y_i) - 59\, f(x_{i-1}, Y_{i-1}) + 37\, f(x_{i-2}, Y_{i-2}) - 9\, f(x_{i-3}, Y_{i-3})\}.$$

For $k = 4$ we have:

$$y(x_{i+1}) = y(x_i) + \frac{h}{720} \cdot \{1901\, f(x_i, y(x_i)) - 2774\, f(x_{i-1}, y(x_{i-1})) + 2616\, f(x_{i-2}, y(x_{i-2}))$$
$$- 1274\, f(x_{i-3}, y(x_{i-3})) + 251\, f(x_{i-4}, y(x_{i-4}))\} + \frac{95}{288} \cdot h^6 \cdot f^{(5)}(\xi_i, y(\xi_i))$$

and the corresponding equation with differences:

$$Y_{i+1} = Y_i + \frac{h}{720} \cdot \{1901\, f(x_i, Y_i) - 2774\, f(x_{i-1}, Y_{i-1}) + 2616\, f(x_{i-2}, Y_{i-2})$$
$$- 1274\, f(x_{i-3}, Y_{i-3}) + 251\, f(x_{i-4}, Y_{i-4})\}.$$

In principle, the preceding procedure can be continued to obtain higher-order Adams-Bashforth formulas, but if k increases then the formulas become complex.

Multi-step methods need help getting started. Generally, a k-step method must have starting values $Y_0, Y_1, \ldots, Y_{k-1}$. These starting values must be computed by other methods. However, keep in mind that the obtained starting values must be as accurate as those produced by the final method. If a starting method is of lower order, then use a smaller step size to generate accurate starting values.

By using $(x_i, f(x_i, y(x_i)))$, $(x_{i-1}, f(x_{i-1}, y(x_{i-1})))$, $\ldots$, $(x_{i-k}, f(x_{i-k}, y(x_{i-k})))$, we derived the Adams-Bashforth methods. We can also use $(x_{i+1}, f(x_{i+1}, y(x_{i+1})))$, $(x_{i+2}, f(x_{i+2}, y(x_{i+2})))$, $\ldots$ to form an interpolating polynomial. An interpolating polynomial through $(x_{i+1}, f(x_{i+1}, y(x_{i+1})))$, $(x_i, f(x_i, y(x_i)))$, $\ldots$, $(x_{i-k}, f(x_{i-k}, y(x_{i-k})))$ that satisfies $P(x_j) = f(x_j, y(x_j))$ for $j = i+1, i, \ldots, i-k$ generates a class of methods known as the **Adams-Moulton methods**.

Let $k = 0$. Replacing $f(x, y(x))$ by the interpolating polynomial through $(x_{i+1}, f(x_{i+1}, y(x_{i+1})))$ and $(x_i, f(x_i, y(x_i)))$ in the formula

$$y(x_{i+1}) = y(x_i) + \int_{x_i}^{x_{i+1}} f(x, y(x))\, dx$$

we get

$$y(x_{i+1}) = y(x_i) + \int_{x_i}^{x_{i+1}} \left[f(x_i, y(x_i)) \cdot \frac{x - x_{i+1}}{x_i - x_{i+1}} + f(x_{i+1}, y(x_{i+1})) \cdot \frac{x - x_i}{x_{i+1} - x_i} \right.$$
$$\left. + \frac{(x - x_i)(x - x_{i+1})}{2!} \cdot f''(\xi_i(x), y(\xi_i(x))) \right] dx =$$

$$= y(x_i) + \frac{h}{2} \left[f(x_i, y(x_i)) + f(x_{i+1}, y(x_{i+1})) \right] - \frac{h^3}{12} \cdot f''(\eta_i, y(\eta_i)).$$

We obtain in this way the **second-order Adams-Moulton formula** which is also known as the **Trapezoidal method**:

$$Y_{i+1} = Y_i + \frac{h}{2}\,[f(x_i, Y_i) + f(x_{i+1}, Y_{i+1})].$$

For $k = 1$, using the cubic interpolating polynomial through $(x_{i+1}, f(x_{i+1}, y(x_{i+1})))$, $(x_i, f(x_i, y(x_i)))$ and $(x_{i-1}, f(x_{i-1}, y(x_{i-1})))$ we find

$$\begin{aligned} y(x_{i+1}) = y(x_i) + \frac{h}{12}\,[5\,f(x_{i+1}, y(x_{i+1})) + 8\,f(x_i, y(x_i)) - \\ - f(x_{i-1}, y(x_{i-1}))] - \frac{h^4}{24} \cdot f^{(3)}(\zeta_i, y(\zeta_i)) \end{aligned}$$

and

$$Y_{i+1} = Y_i + \frac{h}{12}\,[5\,f(x_{i+1}, Y_{i+1}) + 8\,f(x_i, Y_i) - f(x_{i-1}, Y_{i-1})].$$

For $k = 2$ we obtain:

$$\begin{aligned} y(x_{i+1}) = y(x_i) + \frac{h}{24}\,[9\,f(x_{i+1}, y(x_{i+1})) + 19\,f(x_i, y(x_i)) - 5\,f(x_{i-1}, y(x_{i-1})) + \\ + f(x_{i-2}, y(x_{i-2}))] - \frac{19}{720} \cdot h^5 \cdot f^{(4)}(\xi_i, y(\xi_i)), \end{aligned}$$

and

$$Y_{i+1} = Y_i + \frac{h}{24}\,[9\,f(x_{i+1}, Y_{i+1}) + 19\,f(x_i, Y_i) - 5\,f(x_{i-1}, Y_{i-1}) + f(x_{i-2}, Y_{i-2})].$$

For $k = 3$ we have:

$$\begin{aligned} y(x_{i+1}) = y(x_i) + \frac{h}{720}\,[251\,f(x_{i+1}, y(x_{i+1})) + 646\,f(x_i, y(x_i)) - 264\,f(x_{i-1}, y(x_{i-1})) + \\ + 106\,f(x_{i-2}, y(x_{i-2})) - 19\,f(x_{i-3}, y(x_{i-3}))] - \frac{3}{160} \cdot h^6 \cdot f^{(5)}(\alpha_i, y(\alpha_i)), \end{aligned}$$

and

$$\begin{aligned} Y_{i+1} = Y_i + \frac{h}{720}\,[251\,f(x_{i+1}, Y_{i+1}) + 646\,f(x_i, Y_i) - 264\,f(x_{i-1}, Y_{i-1}) + \\ + 106\,f(x_{i-2}, Y_{i-2}) - 19\,f(x_{i-3}, Y_{i-3})] \end{aligned}.$$

Note that these equations implicitly define Y_{i+1} reason for which the Adams-Moulton formulas are called **implicit methods**, while Adams-Bashforth methods define Y_{i+1} explicitly.

6.6 The Predictor-corrector Method

In previous section we derived multi-step methods.
We note that in the case of Adams-Moulton method of fourth order:

$$Y_{i+1} = Y_i + \frac{h}{24}\left[9\,f(x_{i+1}, Y_{i+1}) + 19\,f(x_i, Y_i) - 5\,f(x_{i-1}, Y_{i-1}) + f(x_{i-2}, Y_{i-2})\right] \tag{6.6.1}$$

the absolute truncation error is less than that of the fourth-order Adams-Bashforth method:

$$\begin{aligned} Y_{i+1} &= Y_i + \frac{h}{24}\left[55\,f(x_i, Y_i) - 59\,f(x_{i-1}, Y_{i-1}) + \right. \\ &\quad \left. + 37\,f(x_{i-2}, Y_{i-2}) - 9\,f(x_{i-3}, Y_{i-3})\right]. \end{aligned} \tag{6.6.2}$$

Hence, among Eqs. (6.6.1) and (6.6.2), Eq. (6.6.1) is preferable, but it is an implicit formula. If $f(x, y)$ is nonlinear, then generally it is difficult to solve Eq. (6.6.1) explicitly for Y_{i+1}.
However, Eq. (6.6.1) is a nonlinear equation with root Y_{i+1} and can be solved by a successive approximation method. For fixed i, Y_{i+1} is the solution of:

$$y = g(y) \tag{6.6.3}$$

where

$$g(y) = Y_i + \frac{h}{24}\left[9\,f(x_{i+1}, y) + 19\,f(x_i, Y_i) - 5\,f(x_{i-1}, Y_{i-1}) + f(x_{i-2}, Y_{i-2})\right].$$

To solve Eq. (6.6.3), it is very convenient to use the fixed-point iteration method

$$y^{(k+1)} = g(y^{(k)}) \qquad k = 0, 1, 2, \ldots \tag{6.6.4}$$

because Y_{i+1} is a fixed point of g.
If $|g'(y)| < 1$ for all y with $|y - y_{i+1}| < |y^{(0)} - y_{i+1}|$, then the sequence of iterations (6.6.4) converges. Since $g'(y) = \frac{9h}{24} \cdot \frac{\partial f}{\partial y}$, the sequence of iterations (6.6.4) converges if $h < 1/\left(\frac{9h}{24} \cdot \left|\frac{\partial f}{\partial y}\right|\right)$ and $y^{(0)}$ is sufficiently close to Y_{i+1}.
Thus by properly selecting $y^{(0)}$ (sufficiently close to Y_{i+1}), the sequence of iterations (6.6.4) converges without using many iterations.
For calculating $y^{(0)}$ we use the Eq. (6.6.2):

$$\begin{aligned} Y_{i+1}^{(0)} &= Y_i + \frac{h}{24}\left[55\,f(x_i, Y_i) - 59\,f(x_{i-1}, Y_{i-1}) + \right. \\ &\quad \left. + 37\,f(x_{i-2}, Y_{i-2}) - 9\,f(x_{i-3}, Y_{i-3})\right]. \end{aligned} \tag{6.6.5}$$

This approximation is improved using Eq. (6.6.1):

$$\begin{aligned} Y_{i+1}^{(1)} &= Y_i + \frac{h}{24}\left[9\, f(x_{i+1}, Y_{i+1}^{(0)}) + 19\, f(x_i, Y_i)\right. \\ &\qquad \left. -5\, f(x_{i-1}, Y_{i-1}) + f(x_{i-2}, Y_{i-2})\right]. \end{aligned} \tag{6.6.6}$$

We use Eq. (6.6.5) to predict a value of Y_{i+1} and therefore this equation is known as a **predictor**. The value $Y_{i+1}^{(0)}$ given by prediction is replaced in (6.6.6), and in this way a corrected value $Y_{i+1}^{(1)}$ is obtained. Due to this reason, the Eq. (6.6.6) is known as a **corrector**.
A combination of an explicit method to predict and an implicit method to correct is known as a ***predictor-corrector method****.*
It has been shown (Henrici 1962) that if the predictor method has at least the order of the corrector method, then one iteration is sufficient to preserve the asymptotic accuracy of the corrector method.

A commonly used predictor-corrector method is the combination of the fourth-order Adams-Bashforth formula as a predictor and the fourth-order Adams-Moulton formula as a corrector. Thus

$$\begin{cases} Y_{i+1}^{(p)} &= Y_i + \dfrac{h}{24}\left[55\, f(x_i, Y_i) - 59\, f(x_{i-1}, Y_{i-1}) + 37\, f(x_{i-2}, Y_{i-2}) - 9\, f(x_{i-3}, Y_{i-3})\right] \\ Y_{i+1} &= Y_i + \dfrac{h}{24}\left[9\, f(x_{i+1}, Y_{i+1}^{(p)}) + 19\, f(x_i, Y_i) - 5\, f(x_{i-1}, Y_{i-1}) + f(x_{i-2}, Y_{i-2})\right] \end{cases} \tag{6.6.7}$$

The system (6.6.7) is widely used in combination with the fourth-order Runge-Kutta method as starter. Like the fourth-order Runge-Kutta method, the predictor-corrector system (6.6.7) is one of the most reliable and widely used methods for the numerical solution of an initial-value problem.

Exercises

1. Approximate the solution of the following (IVP)

$$\begin{cases} y'(x) &= y^2(x) - x \\ y(0) &= 0 \qquad\qquad\qquad x \in [0, 1] \end{cases}$$

for $n = 10$, using:

a) fourth-order Adams-Bashforth method,

b) fourth-order Adams-Moulton method,

c) fourth-order predictor-corrector method.

Algorithms for implementation:

// fourth-order Adams-Bashforth method

The fourth-order Runge-Kutta method is applied for determining Y_1, Y_2 and Y_3.

for $i = 4 \ldots n$

$x_i = x_{i-1} + h$

$y_i = y_{i-1} + \frac{h}{24}\ (55f(x_{i-1}, y_{i-1}) - 59f(x_{i-2}, y_{i-2}) + 37f(x_{i-3}, y_{i-3}) - 9f(x_{i-4}, y_{i-4}))$

// fourth-order predictor-corrector method

The fourth-order Runge-Kutta method is applied for determining Y_1, Y_2 and Y_3.

for $i = 4 \ldots n$

$x_i = x_{i-1} + h$

$P = y_{i-1} + \frac{h}{24}\ (55f(x_{i-1}, y_{i-1}) - 59f(x_{i-2}, y_{i-2}) + 37f(x_{i-3}, y_{i-3}) - 9f(x_{i-4}, y_{i-4}))$

$y_i = y_{i-1} + \frac{h}{24}\ (9f(x_i + h, P) + 19f(x_{i-1}, y_{i-1}) - 5f(x_{i-2}, y_{i-2}) + f(x_{i-3}, y_{i-3}))$

Input data:

- x_0, b - the ends of the interval of x

- $y_0 = y(x_0)$

- n

Output data:

- the computed couples (x_i, y_i), $i = \overline{0, n}$

Implementation in Borland C language:

```
#include<stdio.h>
#include<malloc.h>
float x0, y0,b;
int n;
float *Vector(int imin, int imax);
float f(float a, float b);
void rk4(int m, float *x, float *y);
void predcor(void);
void main()
```

```
{
    printf("b= "); scanf("%f", &b);
    printf("x0= "); scanf("%f", &x0);
    printf("y0= "); scanf("%f", &y0);
    printf("n= "); scanf("%d", &n);
    predcor();
}
float f(float a, float b)
{
return (b-a);
}
void rk4(int m, float *x, float *y)
{
    float h, k1, k2, k3, k4;
    int i;
    x[0]=x0;
    y[0]=y0;
    h=(b-x[0])/n;
    for(i=1;i<=m;i++)
    {
        k1=h*f(x[i-1],y[i-1]);
        k2=h*f(x[i-1]+h*0.5,y[i-1]+0.5*k1);
        k3=h*f(x[i-1]+h*0.5,y[i-1]+0.5*k2);
        k4=h*f(x[i-1]+h,y[i-1]+k3);
        y[i]=y[i-1]+(k1+2*k2+2*k3+k4)/6;
        x[i]=x[i-1]+h;
    }
}
void predcor(void)
{
    float *x, *y, h, P;
    int i;
    x=Vector(0,n);
    y=Vector(0,n);
    rk4(3,x,y);
    h=(b-x0)/n;
    for(i=4;i<=n;i++)
    {
        x[i]=x[i-1]+h;
          P=y[i-1]+(h/24)*(55*f(x[i-1],y[i-1])-59*f(x[i-2],y[i-2])+37*f(x[i-
3],y[i-3]-9*f(x[i-4],y[i-4])));
```

```
            y[i]=y[i-1]+(h/24)*(9*f(x[i]+h,P)+19*f(x[i-1],y[i-1])-5*f(x[i-2],y[i-
2]+f(x[i-3],y[i-3])));
        }
        for(i=0;i<=n;i++)
        {
            printf("x[%d]=%g\t", i,x[i]);
            printf("y[%d]=%g\n", i,y[i]);
        }
    }
```

6.7 The Finite Differences Method for a Numerical Solution of a Limit Linear Problem

In this section, we consider the most common numerical method for solving a boundary value problem (BVP), the **finite difference method**.
The basic idea underlying the finite difference method is to replace the derivatives in a differential equation by suitable difference quotients and then to solve the resulting system of equations.
We illustrate this method with the following linear second-order ordinary differential equation with **boundary conditions**:

$$y'' = p(x) \cdot y' + q(x) \cdot y + r(x) \qquad x \in [a, b] \tag{6.7.1}$$

$$\begin{cases} y(a) = \alpha \\ y(b) = \beta. \end{cases} \tag{6.7.2}$$

Let us assume that (BVP) (6.7.1)-(6.7.2) has a unique solution.
In order to approximate the solution, we replace the derivatives in Eq. (6.7.1) by finite differences. This reduces Eq. (6.7.1) to a system of linear equations. In order to accomplish this, we divide the interval $[a, b]$ into $N + 1$ equal intervals of length $h = \frac{b-a}{N+1}$:

$$a = x_0 < x_1 < x_2 < \ldots < x_{i-1} < x_i < \ldots < x_N < x_{N+1} = b.$$

The points (knots) x_i represent the mesh points.
At the interior mesh point x_i, $i \neq 0$ and $N + 1$, Eq. (6.7.1) becomes

$$y''(x_i) = p(x_i) \cdot y'(x_i) + q(x_i) \cdot y(x_i) + r(x_i). \tag{6.7.3}$$

The simplest way to approximate Eq. (6.7.3) is to replace the derivatives $y'(x_i)$ and $y''(x_i)$ with numerical derivative by central differences:

$$y'(x_i) = \frac{y(x_i + h) - y(x_i - h)}{2h} - \frac{h^2}{6} \cdot y^{(3)}(\xi_i) \tag{6.7.4}$$

and

$$y''(x_i) = \frac{y(x_i - h) - 2\,y(x_i) + y(x_i + h)}{h^2} - \frac{h^2}{12} \cdot y^{(4)}(\eta_i). \tag{6.7.5}$$

Substituting Eqs. (6.7.4) and (6.7.5) in (6.7.3), we get

$$\begin{aligned} \frac{y(x_i - h) - 2\,y(x_i) + y(x_i + h)}{h^2} &= p\,(x_i) \cdot \frac{y(x_i + h) - y(x_i - h)}{2h} + \\ &+ q(x_i) \cdot y(x_i) + r(x_i) + \frac{h^2}{12} \cdot y^{(4)}(\eta_i) - \frac{h^2}{6} \cdot y^{(3)}(\xi_i). \end{aligned} \tag{6.7.6}$$

Since ξ_i and η_i are not known and h^2 is small, we ignore the last two terms. Denoting the approximate value of y in x_i by Y_i (i.e., $Y_i \approx y(x_i)$), the approximate value of y in $x_i + h$ by Y_{i+1} ($Y_{i+1} \approx y(x_i + h)$), and the approximate of y in $x_i - h$ by Y_{i-1} ($Y_{i-1} \approx y(x_i - h)$), we get from Eq (6.7.6) the following:

$$\frac{Y_{i-1} - 2\,Y_i + Y_{i+1}}{h^2} = p(x_i) \cdot \frac{Y_{i+1} - Y_{i-1}}{2h} + q(x_i) \cdot Y_i + r(x_i), \quad i = 1, 2, \ldots, N, \tag{6.7.7}$$

which is an algebraic linear system of N equations in N unknowns.
Collecting similar terms, Eq. (6.7.7) is rewritten as

$$Y_{i-1}\left(1 + \frac{h}{2}\,p(x_i)\right) - Y_i(2 + h^2 q(x_i)) + Y_{i+1}\left(1 - \frac{h}{2}\,p(x_i)\right) = h^2 r(x_i), \quad i = 1, 2, \ldots, N. \tag{6.7.8}$$

Denoting:

$$a_i = 1 + \frac{h}{2}\,p(x_i)$$

$$b_i = -(2 + h^2 q(x_i))$$

$$c_i = 1 - \frac{h}{2}\,p(x_i)$$

the equality (6.7.8) becomes

$$a_i Y_{i-1} + b_i Y_i + c_i Y_{i+1} = h^2\,r(x_i), \quad i = 1, 2, \ldots, N, \tag{6.7.9}$$

and the boundary conditions become

$$\alpha = y(a) = y(x_0) = Y_0 \qquad \text{and} \qquad \beta = y(b) = y(x_{N+1}) = Y_{N+1}. \tag{6.7.10}$$

Thus, (6.7.9) and (6.7.10) are written in the matrix form

$$A\,y = s \tag{6.7.11}$$

where

$$A = \begin{bmatrix} b_1 & c_1 & 0 & \dots & 0 & 0 \\ a_2 & b_2 & c_2 & \dots & 0 & 0 \\ 0 & a_3 & b_3 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & b_{N-1} & c_{N-1} \\ 0 & 0 & 0 & \dots & a_N & b_N \end{bmatrix} \qquad Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_{N-1} \\ Y_N \end{bmatrix} \qquad s = \begin{bmatrix} h^2\, r(x_1) - a_1\, \alpha \\ h^2\, r(x_2) \\ \vdots \\ h^2\, r(x_{N-1}) \\ h^2\, r(x_N) - c_N\, \beta \end{bmatrix}$$

This tridiagonal system can be solved very efficiently by the factorization method.

Let us consider the second-order ordinary differential equation with **mixed boundary conditions**:

$$y'' = p(x) \cdot y' + q(x) \cdot y + r(x), \qquad x \in [a, b], \tag{6.7.12}$$

$$\begin{cases} \gamma_1 \cdot y(a) + \gamma_2 \cdot y'(a) = \alpha \\ \gamma_3 \cdot y(b) + \gamma_4 \cdot y'(b) = \beta, \end{cases} \tag{6.7.13}$$

The Eq. (6.7.12) is reduced to the system:

$$a_i \cdot Y_{i-1} + b_i \cdot Y_i + c_i \cdot Y_{i+1} = h^2 \cdot r(x_i) \tag{6.7.14}$$

in which we have:

$$a_i = 1 + \frac{h}{2} \cdot p(x_i)$$

$$b_i = -(2 + h^2 \cdot q(x_i)) \tag{6.7.15}$$

$$c_i = 1 - \frac{h}{2} \cdot p(x_i).$$

If we replace $y'(a)$ by the forward finite difference formula and $y'(b)$ by the backward finite difference formula, the conditions (6.7.13) become

$$\begin{gathered} \gamma_1\, Y_0 + \gamma_2\, \frac{Y_1 - Y_0}{h} = \alpha \\ \gamma_3\, Y_{N+1} + \gamma_4\, \frac{Y_{N+1} - Y_N}{h} = \beta. \end{gathered} \tag{6.7.16}$$

Solving the first equation for Y_0 as function of Y_1, and solving the second equation for Y_{N+1} as function of Y_N, the system (6.7.14) reduces to a tridiagonal system. Unfortunately, the first derivative approximation is of the first

order only. To overcome this drawback, we can use higher-order approximations for $y'(a)$ and $y'(b)$. Using the asymmetrical formulas, we get

$$\begin{gathered}\gamma_1 Y_0 + \gamma_2 \frac{-3Y_0 + 4Y_1 - Y_2}{2h} = \alpha \\ \gamma_3 Y_{N+1} + \gamma_4 \frac{3Y_{N+1} - 4Y_N + Y_{N-1}}{2h} = \beta.\end{gathered} \tag{6.7.17}$$

Solving the first equation for Y_0 and the second for Y_{N+1} yields

$$\begin{aligned}Y_0 &= \frac{2h\alpha - \gamma_2(4Y_1 - Y_2)}{2h\gamma_1 - 3\gamma_2} \\ Y_{N+1} &= \frac{2h\beta - \gamma_4(Y_{N-1} - 4Y_N)}{2h\gamma_3 + 3\gamma_4}.\end{aligned} \tag{6.7.18}$$

For $i = 1$, Eq. (6.7.14) reduces to

$$a_1 Y_0 + b_1 Y_1 + c_1 Y_2 = h^2 \cdot r(x_1).$$

Replacing Y_0, we have

$$\left(b_1 - \frac{4a_1\gamma_2}{2h\gamma_1 - 3\gamma_2}\right) Y_1 + \left(c_1 + \frac{a_1\gamma_2}{2h\gamma_1 - 3\gamma_2}\right) Y_2 = h^2 r(x_1) - \frac{2h\alpha a_1}{2h\gamma_1 - 3\gamma_2}. \tag{6.7.19}$$

For $i = N$, Eq. (6.7.14) becomes

$$a_N Y_{N-1} + b_N Y_N + c_N Y_{N+1} = h^2 \cdot r(x_N),$$

and replacing Y_{N+1} from (6.7.18), we get

$$\left(a_N - \frac{c_N\gamma_4}{2h\gamma_3 + 3\gamma_4}\right) Y_{N-1} + \left(b_N + \frac{4c_N\gamma_4}{2h\gamma_3 + 3\gamma_4}\right) Y_N = h^2 r(x_N) - \frac{2h\beta c_N}{2h\gamma_3 + 3\gamma_4}. \tag{6.7.20}$$

In this way we obtain the system of equations

$$A_1 \cdot y = s_1 \tag{6.7.21}$$

in which

$$A = \begin{bmatrix} b_1 - \dfrac{4a_1\gamma_2}{2h\gamma_1 - 3\gamma_2} & c_1 + \dfrac{a_1\gamma_2}{2h\gamma_1 - 3\gamma_2} & 0 & \dots & 0 & 0 \\ a_2 & b_2 & c_2 & \dots & 0 & 0 \\ 0 & a_3 & b_3 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & b_{N-1} & c_{N-1} \\ 0 & 0 & 0 & \dots & a_N - \dfrac{c_N\gamma_4}{2h\gamma_3 + 3\gamma_4} & b_N + \dfrac{4c_N\gamma_4}{2h\gamma_3 + 3\gamma_4} \end{bmatrix}$$

$$y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_{N-1} \\ Y_N \end{bmatrix} \qquad s = \begin{bmatrix} h^2 r(x_1) - \dfrac{2h\alpha a_1}{2h\gamma_1 - 3\gamma_2} \\ h^2 r(x_2) \\ \vdots \\ h^2 r(x_{N-1}) \\ h^2 r(x_N) - \dfrac{2h\beta c_N}{2h\gamma_3 + 3\gamma_4} \end{bmatrix}$$

Exercises

1. Approximate the following (BVP)

$$\begin{cases} y'' &= -y \\ y(0) &= 1 \\ y(\pi/2) &= 0 \end{cases}$$

using the finite difference method for $n = 4$.
The algorithm for implementation is:

$h = \dfrac{b-a}{n+1}$
$x_i = a + ih \qquad \text{for} \quad i = \overline{0,n}$
$a_i = 1 + \frac{h}{2} \cdot p(x_i) \qquad \text{for} \quad i = \overline{1,n}$
$b_i = -(2 + h^2 q(x_i)) \qquad \text{for} \quad i = \overline{1,n}$
$c_i = 1 - \frac{h}{2} \cdot p(x_{i-1}) \qquad \text{for} \quad i = \overline{2,n+1}$
$s_i = h^2 r(x_i) \qquad \text{for} \quad i = \overline{2,n-1}$
$s_1 = h^2 r(x_1) - a_i \cdot \alpha$
$s_n = h^2 r(x_n) - c_{n+1} \cdot \beta$
We solve the tridiagonal system $A \cdot y = s$ where

$$A = \begin{pmatrix} b_1 & c_2 & 0 & \dots & 0 \\ a_2 & b_2 & c_3 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & b_{n-1} & c_n \\ 0 & 0 & \dots & a_n & b_n \end{pmatrix} \qquad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \qquad s = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix}$$

Input data:

- a, b - the ends of the interval of x

- $\alpha = y(a)$

- $\beta = y(b)$

- n

Output data:

- computed couples (x_i, y_i), $i = \overline{1,n}$

Implementation of this algorithm made in Borland C language:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
float *Vector(int imin, int imax);
void tridiag(int n, float *a, float *b, float *c, float *d, float *x);
float p(float x);
float q(float x);
float r(float x);
void diffinite(float lmin, float lmax, float alpha, float beta, int n);
void main()
{
    float lmin,lmax,alpha, beta;
    int n;
    printf("a= "); scanf("%f", &lmin);
    printf("b= "); scanf("%f", &lmax);
    printf("y[%g]= ",lmin); scanf("%f", &alpha);
    printf("y[%g]= ",lmax); scanf("%f", &beta);
    printf("n= "); scanf("%d", &n);
    diffinite(lmin, lmax, alpha, beta, n);
}
void diffinite(float lmin, float lmax, float alpha, float beta, int n)
{
    float *x, *z,h,*a,*b,*c,*s;
    int i;
    x=Vector(0,n);
    z=Vector(0,n);
    a=Vector(0,n);
    b=Vector(0,n);
    c=Vector(0,n);
    s=Vector(0,n);
    h=(lmin-lmax)/(n+1);
    for(i=0;i<=n;i++)
        x[i]=lmin+i*h;
```

```
    for(i=1;i<=n;i++)
    {
        b[i]=-(2+h*h*q(x[i]));
        s[i]=h*h*r(x[i]);
    }
    s[1]=s[1]-alpha*(1+h*0.5*p(x[1]));
    s[n]=s[n]+beta*(1-h*0.5*p(x[n]));
    for(i=2;i<=n;i++)
    {
        a[i]=1+h*0.5*p(x[i]);
        c[i]=1-h*0.5*p(x[i-1]);
    }
    tridiag(n, a, b, c, s, z);
    for(i=1;i<=n;i++)
    {
        printf("x[%d]=%f\t", i,x[i]);
        printf("y[%d]=%f\n", i,z[i]);
    }
}
float p(float x)
{
return 0;
}
float q(float x)
{
return (-1);
}
float r(float x)
{
return 0;
}
```

6.8 The Collocation Method and the Least Squares Method

Let us consider the second-order ordinary differential equation with **mixed boundary conditions**:

$$y'' + p(x) \cdot y' + q(x) \cdot y = f(x), \qquad x \in [a, b], \tag{6.8.1}$$

$$\begin{cases} \gamma_1 \cdot y(a) + \gamma_2 \cdot y'(a) = \alpha \\ \gamma_3 \cdot y(b) + \gamma_4 \cdot y'(b) = \beta. \end{cases} \tag{6.8.2}$$

For approximating a solution of the (BVP) (6.8.1)-(6.8.2) (supposing that it exists and is unique), we consider a set $\{\Phi_0, \Phi_1, \ldots, \Phi_N\}$ of linearly independent functions of $\mathcal{C}^2$-class, which verify:

$$\gamma_1 \cdot \Phi_0(a) + \gamma_2 \cdot \Phi_0'(a) = \alpha \quad \text{and} \quad \gamma_3 \cdot \Phi_0(b) + \gamma_4 \cdot \Phi_0'(b) = \beta \tag{6.8.3}$$

$$\gamma_1 \cdot \Phi_i(a) + \gamma_2 \cdot \Phi_i'(a) = 0 \quad \text{and} \quad \gamma_3 \cdot \Phi_i(b) + \gamma_4 \cdot \Phi_i'(b) = 0, \tag{6.8.4}$$

$i = \overline{1, N}$.
We search an approximative solution, of Eq. (6.8.1), of the form:

$$Y_N(x) = \Phi_0(x) + \sum_{i=1}^{N} c_i \cdot \Phi_i(x) \tag{6.8.5}$$

in which c_i are constants which should be determined. There are many techniques for determining c_i. Since Φ_0 verifies (6.8.3) and Φ_i verifies (6.8.4), for $i = 1, 2, \ldots, N$, the function $Y_N(x)$ verifies the mixed boundary conditions (6.8.2). Replacing function $Y_N(x)$ in the left-hand side of the Eq. (6.8.1) we find the function

$$\Phi_0'' + \sum_{i=1}^{N} c_i \Phi_i'' + p(x) \left[\Phi_0' + \sum_{i=1}^{N} c_i \Phi_i'\right] + q(x) \left[\Phi_0 + \sum_{i=1}^{N} c_i \Phi_i\right] = h.$$

The function $R = h - f$ is called residual function and indicates the measure in which Y_N verifies Eq. (6.8.1):

$$\begin{aligned} R(x; c_1, \ldots, c_N) = \Phi_0''(x) + p(x) \cdot \Phi_0'(x) + q(x) \cdot \Phi_0(x) + \\ + \sum_{i=1}^{N} c_i \left[\Phi_i''(x) + p(x) \cdot \Phi_i'(x) + q(x) \cdot \Phi_i(x)\right] - f(x) \end{aligned} \tag{6.8.6}$$

We say that $Y_N(x)$ is an exact solution if and only if $R(x; c_1, \ldots, c_N) \equiv 0$.

Generally, the solution is not exact, but if the number of functions Φ_i increases then R becomes small. We try to make $R(x; c_1, \ldots, c_N)$ smaller choosing $c_1, \ldots, c_N$.

The collocation method requests that $R(x; c_1, \ldots, c_N)$ is zero in the given points $x_1, x_2, \ldots, x_N$ of $[a, b]$. Taking into account (6.8.6) it results

that:

$$\sum_{i=1}^{N} c_i[\Phi_i''(x_k) + p(x_k)\cdot\Phi_i'(x_k) + q(x_k)\cdot\Phi_i(x_k)] = \\ = f(x_k) - \Phi_0''(x_k) - p(x_k)\cdot\Phi_0'(x_k) - q(x_k)\cdot\Phi_0(x_k) \tag{6.8.7}$$

for $k = 1, 2, \ldots, N$.
The system of relations (6.8.7) is a system of N linear algebraic equations in N unknowns $c_1, \ldots, c_N$ and can be written int the matrix form as follows

$$A \cdot c = b. \tag{6.8.8}$$

We solve this equation and the obtained solution $(c_1, \ldots, c_N)$ will be used for the construction of $Y_N(x)$ of the form (6.8.5) in order to obtain an approximative solution of the (BVP) (6.8.1)-(6.8.2).

The least squares method requests that the integral:

$$I = \int_a^b R^2(x; c_1, \ldots, c_N)dx \tag{6.8.9}$$

has a minimum value.
In the minimum point we have:

$$\frac{\partial I}{\partial c_i} = 2\int_a^b R\cdot\frac{\partial R}{\partial c_i}\,dx = 0, \qquad i = 1, 2, \ldots, N. \tag{6.8.10}$$

Since:

$$\frac{\partial R}{\partial c_i} = \Phi_i''(x) + p(x)\cdot\Phi_i'(x)q(x)\cdot\Phi_i(x,) \qquad i = 1, 2, \ldots N, \tag{6.8.11}$$

replacing R and $\frac{\partial R}{\partial c_i}$ in (6.8.9), we get

$$\sum_{i=1}^{N} c_j \int_a^b [\Phi_j''(x) + p(x)\cdot\Phi_j'(x) + q(x)\cdot\Phi_j(x)]\cdot[\Phi_i''(x) + p(x)\cdot\Phi_i'(x)q(x)\cdot\Phi_i(x)]dx = \\ = -\int_a^b [\Phi_0''(x) + p(x)\cdot\Phi_0'(x) + q(x)\cdot\Phi_0(x) - f(x)]\cdot[\Phi_i''(x) + p(x)\cdot\Phi_i'(x)q(x)\cdot\Phi_i(x)]dx. \tag{6.8.12}$$

The system (6.8) has N linear algebraic equations in N unknowns and can be written in the matrix form

$$A \cdot c = b. \tag{6.8.13}$$

From this system, c is found and it used for the construction of the approximative solution Y_N.

Exercises

1. Approximate the solution of the following (BVP)

$$\begin{cases} y'' + y &= x \\ y(0) &= 0 \\ y(1) &= 0 \end{cases}$$

using:

a) the collocation method for $n = 4$

b) the least squares method $n = 3$.

Bibliography

[1] Berbente Corneliu, Mitran Sorin, Zancu Silviu, *Metode Numerice*, Ed. Tehnica, Bucuresti, 1998.

[2] Beu Titus A., *Calcul numeric in C*, Editia a 2-a, Ed. Alabastra, Cluj-Napoca, 2000.

[3] Coman Gheorghe, *Analiza Numerica*, Ed. Libris, Cluj-Napoca, 1995.

[4] Dinu Mariana, Linca Gheorghe, *Algoritmi si teme speciale de analiza numerica* Ed. Matrix Rom, Bucuresti, 1999.

[5] O. Dogaru, Gh. Bocsan, I. Despi, A. Ionica, V. Iordan, L. Luca, D. Petcu, P. Popovici *Informatica pentru definitivare si grad*, Ed. de Vest, Timisoara, 1998.

[6] Kelley W., Peterson A., *Difference equation, An Introduction with Applications*, Academic Press, Elsevier, 2000.

[7] Maruster Stefan, *Metode numerice in rezolvarea ecuatiilor neliniare*,Ed. Tehnica, Bucuresti, 1981.

[8] P. Naslau, R. Negrea, L. Cadariu, B. Caruntu, D. Popescu, M. Balmez, C. Dumitrascu, *Matematici asistate pe calculator*, Ed. Politehnica, Timisoara, 2005.

[9] Vithal A. Patel, *Numerical Analysis*, Humboldt State University, USA, 1994.