Theory and Applications of Numerical Approximation Techniques

Theory and Applications of Numerical Approximation Techniques

By

Amelia Bucur and Adrian Nicolae Branga

Cambridge Scholars Publishing

Theory and Applications of Numerical Approximation Techniques

By Amelia Bucur and Adrian Nicolae Branga

This book first published 2024

Cambridge Scholars Publishing

Lady Stephenson Library, Newcastle upon Tyne, NE6 2PA, UK

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

ISBN (10): 1-5275-9383-5
ISBN (13): 978-1-5275-9383-1

TABLE OF CONTENTS

FOREWORD

This scientific book is written for specialists on the theory and application of numerical approximation techniques. It is designed primarily for mathematicians, scientists, and engineers for their studies, works, scientific papers, etc.

AIMS OF THE BOOK, *are: to be an advanced level work in numerical methods and their applications for systems of linear and nonlinear equations, for interpolation, for numerical integration and for solving problems with differential equations or partial differential equations or fractional differential equations; to use techniques in C++, Maple, and Mathematics; to develop mathematical methods through computation; to develop numerical methods in the context of case studies.*

OBJECTIVES OF THE BOOK*, are: to develop numerical methods for data analysis, optimisation, linear algebra and differential equations; to use C++, Maple, and MATLAB skills in numerical methods, programming and graphics; to apply numerical methods and softwares to mathematical problems and obtain solutions; to present these solutions in a coherent manner for the scientific world.*

USING THE BOOK. *This is a scientific book, but it is flexible and can be used by mathematicians and also by engineers, economists, etc.*

USING THE COMPUTER. *In practice, mathematicians, engineers, economists, etc., use computers to apply numerical methods to solve problems. Therefore, we strongly recommend that the computer be integrated into scientific works. We have chosen C++, Maple and MATLAB for simulations because these have a wide academic distribution.*

Technology and the use of software in the educational process have become more important in recent times. Solving some problems with traditional methods could take a long time, which is why programs such as Maple could help professors, engineers, students, etc., to calculate them faster. There are multiple software types (Axiom, Bertini, C++, CoCoA, Delphi, Derive, eViews, GeoGebra, Geometer's Sketchpad, GraphThing, Graph Interface (GRIN), Java, Macaulay2, MATLAB, MathCad, Mathematica, Maxima, MuPAD, Python, SageMath, Scilab, Simulink, Singular, SPSS, TKSolver, WinQSB, etc.) which are specialized in solving mathematical problems; however, in this book the authors have chosen

mainly C++ and Maple as examples of software with a very wide academic distribution.

Of course, in many cases, numerical methods give approximative solutions when carried out on a computer. The answers have some errors, and are accurate up to at most machine precision or accurate up to software floating point precision.

The book recommends the use of C++, Maple, MATLAB and other related programs for scientific research, for the application of mathematics in different scientific technique fields, for particular cases of numerical methods and graphs respectively; and mentions the need for the education system to integrate blended teaching and learning methods, in which computer programs are used together with traditional methods, in order to improve the teaching process for mathematics.

CONFLICTS OF INTEREST. *Both of the authors contributed equally. The authors declare they have not used artificial intelligence (AI) tools in the creation of this book.*

ACKNOWLEDGEMENT. *Project financed by Lucian Blaga University of Sibiu through research grant LBUS-IRG-2022-08.*

Assoc. Prof. PhD Amelia Bucur[1]

Assoc. Prof. PhD Adrian Nicolae Branga[2]

[1,2]*Lucian Blaga University of Sibiu (LBUS),*
Faculty of Sciences, Department of Mathematics and Informatics,
Dr. I. Rațiu, Street, No. 5-7, 550012, Sibiu, Romania

CHAPTER 1

SOLVING SYSTEMS OF LINEAR EQUATIONS

Linear equations were created in the year 1843 by the Irish mathematician Sir William Rowan Hamilton. He induced relationships between variables to find their values. The concept of the system of equations was used for the first time in Europe by Rene Descartes in the year 1637.

Many studies through time have presented properties of the systems of equations, developed methods to find the solutions, and created examples (i.e., Berezin & Zhidkov 1973, 45-69; Branga 2013, 63-108; Branga & Totoi 2020, 7-82; Bucur 2022, 85-94; Burden & Faires 2010, 357-430; Ciarlet & Lions 2003, 3-1176; Coman 1994, 10-260; Dahlquist & Björck 2007, 26-41; Dinu 2008, 20-41; Gautschi 2012, 1-54; Hoffman 1992, 17-77; Householder 2006, 10-240; Moin 2010, 227-234; Ostrowski 1966, 10-35; Sauer 2011, 71-130; Scott 2011, 10-334; Stoer, Bulirsch, Bartels, Gautschi & Witzgall 2002, 167-259).

In this chapter we will present methods for solving Cramer-type systems of linear equations (the number of equations is equal to the number of unknowns and the determinant of the system matrix is zero).

Systems of linear equations have many types of applications in both theoretical and practical sciences (i.e., applications of mathematics to the quantitative study of business and economic problems, financial occupations, the social sciences, applications in engineering, software applications, medical applications, etc.). Also, systems of linear equations can be used in many cases to represent real-world problems, when two or three or n variables (unknowns) exist and you are given two or three or n pieces of information about how those variables are connected and related.

Let the system of n linear equations be

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \tag{1.1}$$

with the coefficients $a_{ij} \in R, \forall\, i,j = \overline{1,n}$, the free terms $b_i \in R, \forall\, i = \overline{1,n}$, and the unknowns $x_1, x_2, \ldots, x_n$. It can also be written as a matrix equation.

The system (1.1) can also be written as a matrix

$$Ax = b, \tag{1.2}$$

if we use the vector of unknowns x, the vector of free terms b and the matrix of coefficients A:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix}, b = \begin{pmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{pmatrix}, A = \left(a_{ij}\right)_{1 \le i,j \le n}.$$

In the case in which matrix A is invertible, the solution for the matrix equation (1.2) is $x = A^{-1}b$.

Cramer's rule is:

$x_i = \frac{\Delta_i}{\Delta} \forall\, i = \overline{1,n}$,

where $\Delta = detA$ and in the determinants Δ_i, the column with the number "i" is replaced with the column of the free terms of the system.

Since in practice, most of the time, matrix A has a large number of lines (and columns), and the calculation of the inverse matrix A^{-1} and of the determinants Δ, Δ_i is difficult, iterative methods are required to solve these systems. These methods can be typed in specific software, such as the program C++.

There are many numerical methods for solving linear systems of equations, such as Gaussian elimination, pivoting strategies, the Cramer method, matrix inversion, matrix factorization, iterative techniques, etc. As examples, we present a method for upper triangular systems, a method for lower triangular systems, the Gauss method, LU factorization – the first variant, LU factorization – the second variant, the Crout method, the Doolittle method, and the Cholesky method.

1.1 Upper Triangular Systems

If we have an upper triangular matrix, then the system is written as:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \\ a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2 \\ a_{33}x_3 + \cdots + a_{3n}x_n = b_3 \\ \ldots\ldots\ldots\ldots \\ a_{nn}x_n = b_n \end{cases}$$

and the matrix of coefficients is:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22} & a_{23} & \dots & a_{2n} \\ 0 & 0 & a_{33} & \dots & a_{3n} \\ & & & \dots\dots\dots & \\ 0 & 0 & 0 & \dots & a_{nn} \end{pmatrix}.$$

When the system is a Cramer system of equations, then $detA \neq 0$. But, for an upper triangular matrix, the determinant can compute as a product:

$$detA = \prod_{i=1}^{n} a_{ii} = a_{11}a_{22} \dots a_{nn},$$

That means that $\prod_{i=1}^{n} a_{ii} \neq 0$ implies $a_{ii} \neq 0$ for all index *i*.

The system in the matrix form is:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22} & a_{23} & \dots & a_{2n} \\ 0 & 0 & a_{33} & \dots & a_{3n} \\ & & & \dots\dots\dots & \\ 0 & 0 & 0 & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \dots \\ b_n \end{pmatrix}.$$

The above system is solved by inverse substitution, which means the *Back Substitution method*. This method first determines x_n, then x_{n-1} and so on until a last unknown, x_1. From the last equation we have $x_n = \frac{b_n}{a_{nn}}$ and replacing it in the penultimate equation of the system will compute x_{n-1}; the process continues until the last unknown has been computed, namely x_1 from the first equation of the system.

ALGORITHM

Input parameters

- The matrix of the system, $A = (a_{ij})_{1 \leq i,j \leq n}$, with $detA \neq 0$ ($a_{ii} \neq 0, \forall\, i = \overline{1,n}$) and $a_{ij} = 0, \forall\, i = \overline{2,n}, \forall\, j = \overline{1,n-1}$;
- The free terms vector $b = (b_i)_{i=\overline{1,n}}$.

Output parameters

- The solution of the upper triangular system $Ax = b$, $x = (x_i)_{i=\overline{1,n}}$.

Pseudocode

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

For $i = n-1, \dots, 1$ do

$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}}$.

PROCEDURE IN C++

```
#include <iostream>
using namespace std;
void InputParameters(int& n, double a[100][100], double b[100])
{
	int i, j;
	cout << "n= ";
	cin >> n;
	for (i = 1; i <= n; i++)
	{
		for (j = i; j <= n; j++)
		{
		cout << "a[" << i << "][" << j << "]=";
		cin >> a[i][j];
		}
	}
	for (i = 1; i <= n; i++)
	{
		cout << "b[" << i << "]=";
		cin >> b[i];
	}
}
void UpperTriangularSystem(int n, double a[100][100], double b[100],
double x[100])
{
	int i, j;
	double S;
	x[n] = b[n] / a[n][n];
	for (i = n - 1; i >= 1; i--)
	{
		S = 0;
		for (j = i + 1; j < = n; j++)
		{
			S = S + a[i][j] * x[j];
		}
		x[i] = (b[i] - S) / a[i][i];
	}
}
void OutputParameters(int n, double x[100])
{
	int i;
```

```
        for (i = 1; i <= n; i++)
        {
                cout << "x[" << i << "]=" << x[i] << " ";
        }
}
int main()
{
        int n;
        double a[100][100], b[100], x[100];
        InputParameters(n, a, b);
        UpperTriangularSystem(n, a, b, x);
        OutputParameters(n, x);
}
```

1.2 Lower Triangular Systems

In the case of a system with a lower triangular matrix of coefficients, matrix *A* has the following form:

$$A = \begin{pmatrix} a_{11} & 0 & 0 & \dots & 0 & 0 \\ a_{21} & a_{22} & 0 & \dots & 0 & 0 \\ a_{31} & a_{32} & a_{33} & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{n,n-1} & a_{nn} \end{pmatrix}.$$

When the system is a Cramer system of equations, then $detA \neq 0$. But, for a lower triangular matrix, the determinant can compute as a product:

$$detA = \prod_{i=1}^{n} a_{ii} = a_{11}a_{22} \dots a_{nn},$$

which means that $\prod_{i=1}^{n} a_{ii} \neq 0$ implies $a_{ii} \neq 0$ for all index *i*.

The system in the matrix form is:

$$\begin{pmatrix} a_{11} & 0 & 0 & \dots & 0 & 0 \\ a_{21} & a_{22} & 0 & \dots & 0 & 0 \\ a_{31} & a_{32} & a_{33} & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{n,n-1} & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \dots \\ b_n \end{pmatrix}.$$

That implies:

$$\begin{cases} a_{11}x_1 = b_1 \\ a_{21}x_1 + a_{22}x_2 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \dots + a_{nn}x_n = b_n \end{cases}.$$

The above system is solved by direct substitution, which means the *Forward Substitution method.* This method first determines x_1, then x_2 and so on until a last unknown, x_n. From the first equation we have $x_1 = \frac{b_1}{a_{11}}$ and replacing it in the second equation of the system will compute x_2; the process continues until the last unknown has been computed, namely x_n from the last equation of the system.

ALGORITHM

Input parameters

- The matrix of the system $A = (a_{ij})_{1 \le i,j \le n}$ with $detA \neq 0$ ($a_{ii} \neq 0, \forall\, i = \overline{1,n}$) and $a_{ij} = 0, \forall\, i = \overline{1,n-1}, \forall\, j = \overline{i+1,n}$;
- The free terms vector $b = (b_i)_{i=\overline{1,n}}$.

Output parameters

- The solution of the lower triangular system $Ax = b$, $x = (x_i)_{i=\overline{1,n}}$.

Pseudocode

$x_1 = \frac{b_1}{a_{11}}$

For $i = 2, \dots, n$ do

$x_i = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j}{a_{ii}}$.

PROCEDURE IN C++

```
#include <iostream>
using namespace std;
void InputParameters(int& n, double a[100][100], double b[100])
{
	int i, j;
	cout << "n= ";
	cin >> n;
	for (i = 1; i <= n; i++)
	{
		for (j = 1; j <= i; j++)
		{
			cout << "a[" << i << "][" << j << "]=";
			cin >> a[i][j];
		}
	}
	for (i = 1; i <= n; i++)
	{
```

```
                cout << "b[" << i << "]=";
                cin >> b[i];
        }
}
void LowerTriangularSystem(int n, double a[100][100], double b[100],
double x[100])
{
        int i, j;
        double S;
        x[1] = b[1] / a[1][1];
        for (i = 2; i <= n; i++)
        {
                S = 0;
                for (j = 1; j <= i - 1; j++)
                {
                        S = S + a[i][j] * x[j];
                }
                x[i] = (b[i] - S) / a[i][i];
        }
}
void OutputParameters(int n, double x[100])
{
        int i;
        for (i = 1; i <= n; i++)
        {
                cout << "x[" << i << "]=" << x[i] << " ";
        }
}
int main()
{
        int n;
        double a[100][100], b[100], x[100];
        InputParameters (n, a, b);
        LowerTriangularSystem (n, a, b, x);
        OutputParameters (n, x);
}
```

1.3 The Gauss Method

Let the system $Ax = b$, with $A \in M_n(R), detA \neq 0$.

Let $\tilde{A} = (A|b)$ the extended matrix a of the system.

The Gauss method consists in applying elementary transformations on the matrix $\tilde{A}$, for obtaining an upper triangular matrix, equivalent to the matrix A.

Through elementary transformations applied to the matrix $\tilde{A}$, we understand:

- the change in the two lines between them;
- the multiplication of a line with a non-null real number and its addition to another line.

Steps of the algorithm:

1. In the first column we look for the smallest index *i*, which we will then note with i_1, so that we have $|a_{i_1 1}|$ the largest element of all elements $|a_{i1}|$, $i = \overline{1,n}$. That means:
$$|a_{i_1 1}| = \max_{i=\overline{1,n}} |a_{i1}| .$$

It is preferable to choose the element of the maximum mode, to ensure that this is a non-null number.

Since $detA \neq 0$, we will have $\max_{i=\overline{1,n}} |a_{i1}| > 0$. The case $\max_{i=\overline{1,n}} |a_{i1}| = 0$ means that all the elements in the first column are 0, and this contradicts the relation $detA \neq 0$.

2. The lines L_1 and Li_1 change between them;

3. Any line *Li* is replaced by $L_i - \frac{a_{i1}}{a_{11}} L_1, i = \overline{2,n}$ (we will form zeros under the new element a_{11}).

The steps of the algorithm above are carried out until we get an upper triangular matrix, and then the novel upper triangular system will be solved, using the *Back Substitution method*.

ALGORITHM

Input parameters

- n – the order of the system;
- The matrix *A* of the system $A = (a_{ij})_{1 \leq i,j \leq n}$, with $detA \neq 0$;
- The free terms vector $b = (b_i)_{i=\overline{1,n}}$.

Output parameters

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

Pseudocode

For *i = 1, 2, ..., n*, do

$a_{i,n+1} = b_i$

End For
For *k = 1, 2, ..., n-1*, do
$|a_{p,k}| = \max_{i=\overline{k,n}} |a_{i,k}|$
If $a_{p,k} \neq 0$ then
If $p \neq k$ then
swap lines p and k between them
End If
For *i = k + 1, k + 2, ..., n*, do
For *j = k + 1, k + 2, ..., n+1*, do
$a_{ij} = a_{ij} - \frac{a_{ik}a_{kj}}{a_{kk}}$
End For
$a_{ik} = 0$
End For
Else:
The system *Ax = b* cannot be solved by this algorithm (the matix A is singular)
STOP
End For
$x_n = \frac{a_{n,n+1}}{a_{nn}}$
For *i = n-1, ..., 1*, do
$x_i = \frac{1}{a_{ii}}\left(a_{i,n+1} - \sum_{j=i+1}^{n} a_{ij}x_j\right)$
End For

PROCEDURE IN C++

```
#include <iostream>
#include <fstream>
#include <math.h>
using namespace std;
int n, i, j, k, p;
double a[100][100], b[100], x[100], S, m, aux;
void InputParameters(int& n, double a[100][100], double b[100])
{
        ifstream f("intrare.txt");
        /*cout << "n=";
        cin >> n;*/
        f >> n;
        for (i = 1; i <= n; i++)
                for (j = 1; j <= n; j++)
                {
```

```
                    /*cout << "a[" << i << "][" << j << "]=";
                     cin >> a[i][j];*/
                    f >> a[i][j];
                }
        for (i = 1; i <= n; i++)
        {
                /*cout << "b[" << i << "]=";
                cin >> b[i];*/
                f >> b[i];
        }
}
void GaussMethod(int n, double a[100][100], double b[100], double
x[100])
{
        for (i = 1; i <= n; i++)
                a[i][n + 1] = b[i];
        for (k = 1; k <= n-1; k++)
        {
                m = fabs(a[k][k]);
                p = k;
                for (i = k+1; i <= n; i++)
                        if (fabs(a[i][k]) > m)
                        {
                                m = fabs(a[i][k]);
                                p = i;
                        }
                if (a[p][k] != 0)
                {
                        if (p != k)
                                for (j = 1; j <= n+1; j++)
                                {
                                        aux = a[p][j];
                                        a[p][j] = a[k][j];
                                        a[k][j] = aux;
                                }
                        for (i = k+1; i <=n; i++)
                        {
                                for (j = k+1; j <= n+1; j++)
                                        a[i][j] = a[i][j] - ((a[i][k] *
                                        a[k][j]) / a[k][k]);
                                a[i][k]=0;
```

```
                        }
                }
                else
                {
                        cout << "The system cannot be solved by this
                                algorithm (the matix A is singular)";
                        break;
                }
        }
        x[n] = a[n][n+1] / a[n][n];
        for (i = n-1; i > =1; i--)
        {
                S=0;
                for (j = i+1; j <= n; j++)
                        S = S + a[i][j] * x[j];
                x[i] = (a[i][n+1] - S) / a[i][i];
        }
}
void OutputParameters(int n, double x[100])
{
        for (i = 1; i <= n; i++)
        {
                cout << "x[" << i << "]=";
                cout << x[i] << endl;
        }
}
int main()
{
        InputParameters(n, a, b);
        GaussMethod(n, a, b, x);
        OutputParameters(n, x);
        return 0;
}
```

1.4 LU Factorization – the First Variant

Factorization methods (factorization = writing a number or another mathematical term as a product of several factors).

These methods consist of decomposing the matrix of coefficients of the system $A= (a_{ij})$ into a product of two matrices, one lower triangular (L), and one upper triangular (U): $A = LU$.

Thus, if the matrix system is written as $Ax = b$, it results that $LUx = b \xrightarrow{Ux=y} Ly = b \xrightarrow{\substack{Forward\ Substitution \\ method}} y$ and then, $Ux = y \xrightarrow{\substack{Back\ Substitution \\ method}} x$.

There are three factorization methods.

In the following, we will use the notation

$$L = \begin{pmatrix} l_{11} & 0 & 0 & \dots & 0 & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 & 0 \\ l_{31} & l_{32} & l_{33} & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{n,n-1} & l_{nn} \end{pmatrix},$$

for a lower matrix,
and the notation

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & u_{22} & u_{23} & \dots & u_{2n} \\ 0 & 0 & u_{33} & \dots & u_{3n} \\ & & & \dots \dots \dots \dots & \\ 0 & 0 & 0 & \dots & u_{nn} \end{pmatrix},$$

for an upper matrix.

Let the matrix $A \in M_n(R)$, $A = \left(a_{ij}\right)_{1 \le i,j \le n}$. For $i \in \{2, \dots n\}$ we will note

$$A_i = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1i} \\ a_{21} & a_{22} & \dots & a_{2i} \\ & & \dots & \\ & & \dots & \\ a_{i1} & a_{i2} & \dots & a_{ii} \end{pmatrix}.$$

If $detA_i \neq 0 \ \forall\, i = \overline{1,n}$ then there exists a factorization for the matrix *A* equal to *LU*. That means that there exist a lower triangular matrix (*L*) and an upper triangular matrix (*U*), and the following equality takes place *A* = *LU*, but this factorization isn't unique.

The number of knowns will be equal to the number of the matrix *L* added to the number of the matrix *U*:

$$2(1 + 2 + \dots + n) = 2\frac{n(n+1)}{2} = n(n+1) = n^2 + n.$$

The number of equations will be equal to the number of elements from the *A* matrix, which means n^2.

To simplify the calculus, we have supposed that n elements are fixed, and the principal diagonal of the matrix *U* is known.

Hence, for this method, the elements $u_{11}, u_{22}, \dots, u_{nn}$ are the known real non-null numbers.

The system *Ax* = *b* will be equivalent to the relation *LUx* = *b*.

First, we will note $y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = Ux$ and we will solve the system *Ly = b*. By using the *Back Substitution method*, we will obtain the numbers $y_1, y_2, \dots, y_n$. Afterwards, we will solve the system *Ux = y*, by using the *Forward Substitution method* and we will obtain $x_1, x_2, \dots, x_n$. These are the elements of the vector *x*.

ALGORITHM

Input parameters

- n – the order of the system;
- The matrix *A* of the system $A = (a_{ij})_{1 \le i,j \le n}$, with $detA_i \neq 0$;
- The free terms vector $b = (b_i)_{i=\overline{1,n}}$;
- $u_{ii} \neq 0$, i = 1,…, n, the diagonal of the matrix U.

Output parameters

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

Pseudocode

For *r = 1, 2, …, n* do
  For *i = r, r + 1, … , n* do
    $l_{ir} = \frac{(a_{ir} - \sum_{k=1}^{r-1} l_{ik} u_{kr})}{u_{rr}}$
  End For
  For *j = r + 1, …, n* do
    $u_{rj} = \frac{(a_{rj} - \sum_{k=1}^{r-1} l_{rk} u_{kj})}{l_{rr}}$
  End For
End For
For *i = 1, 2, …, n* do
  $y_i = \frac{\left(b_i - \sum_{j=1}^{i-1} l_{ij} y_j\right)}{l_{ii}}$
End For
For *i = n, n-1, …, 1* do
  $x_i = \frac{\left(y_i - \sum_{j=i+1}^{n} u_{ij} x_j\right)}{u_{ii}}$
End For

PROCEDURE IN C++

```
#include <iostream>
using namespace std;
int n, i, j, r, k;
double a[100][100], b[100], x[100], y[100], l[100][100], u[100][100], S;
```

```
void InputParameters(int& n, double a[100][100], double b[100],
double u[100][100])
{
	cout << "n=";
	cin >> n;
	for (i = 1; i <= n; i++)
		for (j = 1; j <= n; j++)
		{
			cout << "a[" << i << "][" << j << "]=";
			cin >> a[i][j];
		}
	for (i = 1; i <= n; i++)
	{
		cout << "b[" << i << "]=";
		cin >> b[i];
	}
	for (i = 1; i <= n; i++)
	{
		u[i][i] = rand() % 101 +10;
		cout << u[i][i] << ' ';
	}
}
void LUFactorizationFirstVariant(int n, double a[100][100], double b[100],
double x[100], double y[100], double l[100][100], double u[100][100])
{
	for (i = 1; i <= n; i++)
		l[i][1] = a[i][1] / u[1][1];
	for (j = 2; j <= n; j++)
		u[1][j] = a[1][j] / l[1][1];
	for (r = 2; r <= n; r++)
	{
		for (i = r; i <= n; i++)
		{
			S = 0;
			for (k = 1; k <= r - 1; k++)
				S = S + l[i][k] * u[k][r];
			l[i][r] = (a[i][r] - S) / u[r][r];
		}
		for (j = r + 1; j <= n; j++)
		{
			S = 0;
```

```
                        for (k = 1; k <= r - 1; k++)
                                S = S + l[r][k] * u[k][j];
                        u[r][j] = (a[r][j] - S) / l[r][r];
                }
        }
        y[1] = b[1] / l[1][1];
        for (i = 2; i <= n; i++)
        {
                S = 0;
                for (j = 1; j <= i - 1; j++)
                        S = S + l[i][j] * y[j];
                y[i] = (b[i] - S) / l[i][i];
        }
        x[n] = y[n] / u[n][n];
        for (i = n - 1; i >= 1; i--)
        {
                S = 0;
                for (j = i+1; j <=n; j++)
                        S = S + u[i][j] * x[j];
                x[i] = (y[i] - S) / u[i][i];
        }
}
void OutputParameters(int n, double x[100])
{
        for (i = 1; i <= n; i++)
        {
                cout << "x[" << i << "]=";
                cout << x[i] << endl;
        }
}
int main()
{
        InputParameters(n, a, b, u);
        LUFactorizationFirstVariant (n, a, b, x, y, l, u);
        OutputParameters(n, x);
}
```

1.5 LU Factorization – the Second Variant

Let the matrix $A \in M_n(R)$, $A = \left(a_{ij}\right)_{1\le i,j\le n}$. For $i \in \{2, \dots n\}$ we will note

$$A_i = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1i} \\ a_{21} & a_{22} & \dots & a_{2i} \\ & \dots & & \\ & \dots & & \\ a_{i1} & a_{i2} & \dots & a_{ii} \end{pmatrix}.$$

If $detA_i \neq 0 \ \forall i = \overline{1,n}$ then there exists a factorization for the matrix *A* equal to *LU*. That means that there exist a lower triangular matrix (*L*) and an upper triangular matrix (*U*), and the following equality takes place *A = LU*, but this factorization isn't unique.

The number of knowns will be equal to the number of the matrix *L* added to the number of the matrix *U*:

$$2(1 + 2 + \dots + n) = 2\frac{n(n+1)}{2} = n(n+1) = n^2 + n.$$

To simplify the calculus, we have supposed that n elements are fixed, and the principal diagonal of the matrix *L* is known.

Hence, for this method, the elements $l_{11}, l_{22}, \dots, l_{nn}$ are known real non-null numbers.

The system *Ax = b* will be equivalent to the relation *LUx = b*.

First, we will note $y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = Ux$ and we will solve the system *Ly = b*. By using the *Forward Substitution method*, we will obtain the numbers $y_1, y_2, \dots, y_n$. Afterwards, we will solve the system *Ux = y*, by using the *Back Substitution method* and we will obtain $x_n, x_{n-1}, \dots, x_1$. These are the elements of the vector *x*.

ALGORITHM

Input parameters

- n – the order of the system;
- The matrix *A* of the system $A = \left(a_{ij}\right)_{1\le i,j\le n}$, with $detA_i \neq 0$;
- The free terms vector $b = (b_i)_{i=\overline{1,n}}$;
- $l_{ii} \neq 0$, i = 1,…, n, the diagonal the the matrix L.

Output parameters

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

Pseudocode

For *r = 1, 2, ..., n* do

For $j = r, r + 1, \ldots, n$ do

$$u_{rj} = \frac{(a_{rj} - \sum_{k=1}^{r-1} l_{rk} u_{kj})}{l_{rr}}$$

End For

For $i = r, \ldots, n$ do

$$l_{ir} = \frac{(a_{ir} - \sum_{k=1}^{r-1} l_{ik} u_{kr})}{u_{rr}}$$

End For

End For

For $i = 1, 2, \ldots, n$ do

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

End For

For $i = n, n\text{-}1, \ldots, 1$ do

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

End For

PROCEDURE IN C++

```
#include <iostream>
using namespace std;
int n, i, j, r, k;
double a[100][100], b[100], x[100], y[100], l[100][100], u[100][100], S;
void InputParameters(int& n, double a[100][100], double b[100],
double l[100][100])
{
        cout << "n=";
        cin >> n;
        for (i = 1; i <= n; i++)
                for (j = 1; j <= n; j++)
                {
                        cout << "a[" << i << "][" << j << "]=";
                        cin >> a[i][j];
                }
        for (i = 1; i <= n; i++)
        {
                cout << "b[" << i << "]=";
                cin >> b[i];
        }
        for (i = 1; i <= n; i++)
        {
                l[i][i] = rand() % 4 + 1;
```

```
	}
}
void LUFactorizationSecondVariant(int n, double a[100][100],
double b[100], double x[100], double y[100], double l[100][100],
double u[100][100])
{
	for (j = 1; j <= n; j++)
		u[1][j] = a[1][j] / l[1][1];
	for (i = 2; i <= n; i++)
		l[i][1] = a[i][1] / u[1][1];
	for (r = 2; r <= n; r++)
	{
		for (j = r; j <= n; j++)
		{
			S = 0;
			for (k = 1; k <= r - 1; k++)
				S = S + l[r][k] * u[k][j];
			u[r][j] = (a[r][j] - S) / l[r][r];
		}
		for (i = r; i <= n; i++)
		{
			S = 0;
			for (k = 1; k <= r - 1; k++)
				S = S + l[i][k] * u[k][r];
			l[i][r] = (a[i][r] - S) / u[r][r];
		}
	}
	y[1] = b[1] / l[1][1];
	for (i = 2; i <= n; i++)
	{
		S = 0;
		for (j = 1; j <= i - 1; j++)
			S = S + l[i][j] * y[j];
		y[i] = (b[i] - S) / l[i][i];
	}
	x[n] = y[n] / u[n][n];
	for (i = n - 1; i >= 1; i--)
	{
		S = 0;
		for (j = i + 1; j <= n; j++)
			S = S + u[i][j] * x[j];
```

```
                x[i] = (y[i] - S) / u[i][i];
        }
}
void OutputParameters(int n, double x[100])
{
        for (i = 1; i <= n; i++)
        {
                cout << "x[" << i << "]=";
                cout << x[i] << endl;
        }
}
int main()
{
        InputParameters(n, a, b, l);
        LUFactorizationSecondVariant(n, a, b, x, y, l, u);
        OutputParameters(n, x);
        return 0;
}
```

1.6 The Crout Method

We will also use this method to solve systems of equations of the linear Cramer type. Crout's method is a particular case of LU factorization – the first variant.

Let the matrix $A \in M_n(R)$, $A = \left(a_{ij}\right)_{1 \le i,j \le n}$. For $i \in \{2, \dots n\}$ we will note

$$A_i = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1i} \\ a_{21} & a_{22} & \dots & a_{2i} \\ & \dots & & \\ & \dots & & \\ a_{i1} & a_{i2} & \dots & a_{ii} \end{pmatrix}.$$

If $detA_i \neq 0 \ \forall \ i = \overline{1,n}$ then there exists a factorization for the matrix *A* equal to *LU*. That means that there exist a lower triangular matrix (*L*) and an upper triangular matrix (*U*), and the following equality takes place *A* = *LU*, but this factorization isn't unique.

To simplify the calculus, we have supposed that n elements are fixed, and the principal diagonal of the matrix *U* is known.

Hence, for this method, the elements $u_{11}, u_{22}, \dots, u_{nn}$ are equal to 1.

The system *Ax* = *b* will be equivalent to the relation *LUx* = *b*.

First, we will note $y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = Ux$ and we will solve the system *Ly* = *b*. By using the *Forward Substitution method*, we will obtain the numbers $y_1, y_2, \dots, y_n$. Afterwards, we will solve the system *Ux* = *y*, by using the *Back Substitution method* and we will obtain $x_n, x_{n-1}, \dots, x_1$. These are the elements of the vector *x*.

ALGORITHM

Input parameters

- n – the order of the system;
- The matrix *A* of the system $A = (a_{ij})_{1 \le i,j \le n}$, with $detA_i \neq 0$;
- The free terms vector $b = (b_i)_{i=\overline{1,n}}$.

Output parameters

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

Pseudocode

For *r = 1, 2, …, n* do
  For *i = r, r + 1, … , n* do
    $l_{ir} = a_{ir} - \sum_{k=1}^{r-1} l_{ik}u_{kr}$
  End For
  For *j = r+1, …, n* do
    $u_{rj} = \frac{(a_{rj} - \sum_{k=1}^{r-1} l_{rk}u_{kj})}{l_{rr}}$
  End For
End For
For *i = 1, 2, …, n* do
  $y_i = \frac{\left(b_i - \sum_{j=1}^{i-1} l_{ij}y_j\right)}{l_{ii}}$
End For
For *i = n, n-1, …, 1* do
  $x_i = y_i - \sum_{j=i+1}^{n} u_{ij}x_j$
End For

PROCEDURE IN C++

```
#include <iostream>
using namespace std;
int n, i, j, r, k;
double a[100][100], b[100], x[100], y[100], l[100][100], u[100][100], S;
void InputParameters(int& n, double a[100][100], double b[100])
{
```

```
	cout << "n=";
	cin >> n;
	for (i = 1; i <= n; i++)
		for (j = 1; j <= n; j++)
		{
			cout << "a[" << i << "][" << j << "]=";
			cin >> a[i][j];
		}
	for (i = 1; i <= n; i++)
	{
		cout << "b[" << i << "]=";
		cin >> b[i];
	}
}
void CroutMethod(int n, double a[100][100], double b[100], double x[100],
double y[100], double l[100][100], double u[100][100])
{
	for (i = 1; i <= n; i++)
		l[i][1] = a[i][1];
	for (j = 2; j <= n; j++)
		u[1][j] = a[1][j] / l[1][1];
	for (r = 2; r <= n; r++)
	{
		for (i = r; i <= n; i++)
		{
			S = 0;
			for (k = 1; k <= r - 1; k++)
				S = S + l[i][k] * u[k][r];
			l[i][r] = a[i][r] - S;
		}
		for (j = r + 1; j <= n; j++)
		{
			S = 0;
			for (k = 1; k <= r - 1; k++)
				S = S + l[r][k] * u[k][j];
			u[r][j] = (a[r][j] - S) / l[r][r];
		}
	}
	y[1] = b[1] / l[1][1];
	for (i = 2; i <= n; i++)
	{
```

```
                S = 0;
                for (j = 1; j <= i - 1; j++)
                        S = S + l[i][j] * y[j];
                y[i] = (b[i] - S) / l[i][i];
        }
        x[n] = y[n];
        for (i = n - 1; i >= 1; i--)
        {
                S = 0;
                for (j = i + 1; j <= n; j++)
                        S = S + u[i][j] * x[j];
                x[i] = y[i] - S;
        }
}
void OutputParameters(int n, double x[100])
{
        for (i = 1; i <= n; i++)
        {
                cout << "x[" << i << "]=";
                cout << x[i] << endl;
        }
}
int main()
{
        InputParameters(n, a, b);
        CroutMethod(n, a, b, x, y, l, u);
        OutputParameters(n, x);
        return 0;
}
```

1.7 The Doolittle method

With this method, $L = (l_{ij}), U = (u_{ij})$:

$$l_{ij} = \begin{cases} l_{ij}, & i > j \\ 1, & i = j \\ 0, & i < j \end{cases}, \; u_{ij} = \begin{cases} u_{ij}, & i \le j \\ 0, & i > j \end{cases}.$$

Thus,

$a_{ij} = \sum_{k=1}^{i} l_{ik}u_{kj} = \sum_{k=1}^{i-1} l_{ik}u_{kj} + l_{ii}u_{ij} \underset{l_{ii}=1}{\Longrightarrow} u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}, \;\; i,j = 1,2,\dots,n,$