INTRODUCTORY NUMERICAL ANALYSIS

Lecture Notes

by

Mircea Andrecut

Universal Publishers/UPUBLISH.COM

2000

Parkland, FL, USA

Introductory Numerical Analysis: Lecture Notes

ISBN: 1-58112-757-X

Universal Publishers/uPUBLISH.com
USA • 2000

www.upublish.com/books/andrecut.htm

To my father and my son

Preface

The aim of this book is to provide a simple and useful introduction for the fresh students into the vast field of numerical analysis. Like any other introductory course on numerical analysis, this book contains the basic theory, which in the present text refers to the following topics: linear equations, nonlinear equations, eigensystems, interpolation, approximation of functions, numerical differentiation and integration, stochastics, ordinary differential equations and partial differential equations. Also, the text is restricted to twelve lectures and one appendix, covering only one semester of teaching (2 hours/lecture and 2 hours/exam).

Because the students need to quickly understand why the numerical methods correctly work, the proofs of theorems were shorted as possible, insisting more on ideas than on a lot of algebra manipulation. The included examples are presented with a minimum of complications, emphasizing the steps of the algorithms.

The numerical methods described in this book are illustrated by computer programs written in C. Our goal was to develop very simple programs which are easily to read and understand by students. Also, the programs should run without modification on any compiler that implements the ANSI C standard.

Because our intention was to easily produce screen input-output (using, `scanf` and `printf`), in case of WINDOWS visual programming environments, like Visual C++ (Microsoft) and Borland C++ Builder, the project should be *console-application*. This will be not a problem for DOS and LINUX compilers.

If this material is used as a teaching aid in a class, I would appreciate if under such circumstances, the instructor of such a class would send me a note at the address below informing me if the material is useful. Also, I would appreciate any suggestions or constructive criticism regarding the content of these lecture notes.

Dr. M. Andrecut
Faculty of Physics,
“Babes-Bolyai” University,
Cluj-Napoca, 3400, Romania
e-mail: andrecut@phys.ubbcluj.ro

TABLE OF CONTENTS

Lecture 1

Exact Methods for Linear Systems

Introduction

Let us consider the problem of solving the *linear system of equations*:

$$A \cdot X = B .$$

Here, the raised dot denotes *matrix product*, $A = [a_{ij}]$ is a $N \times N$ matrix with $\det(A) \neq 0$, $B = [b_i]$ is the right-hand side written as a column vector, and $X = [x_i]$ is the *unknown vector*:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \cdots & \cdots & \cdots & \cdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix}, \quad B = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_N \end{bmatrix}, \quad X = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_N \end{bmatrix}.$$

The formal solution of this linear system can be written as

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

where A^{-1} is the *inverse* of the matrix A. The inverse of a $N \times N$ matrix A is defined to be a matrix A^{-1} such that

$$A \cdot A^{-1} = A^{-1} \cdot A = I ,$$

where $I = [\delta_{ij}]$ is the $N \times N$ identity matrix ($\delta_{ij} = 1$ if $i = j$; $\delta_{ij} = 0$ if $i \neq j$). The square matrix A is invertible iff $\det(A) \neq 0$. The inverse is then unique and to find A^{-1} we can solve

$$A \cdot (A^{-1}) = I .$$

This corresponds to solving a linear system for N right-hand sides.

The *exact methods* for solving linear systems gives the exact solution (in the absence of roundoffs) after a finite number of arithmetic and logical operations.

Gauss Elimination Method

Gauss elimination is the most frequently used exact method for solving systems of linear equations.

Let us start with the simple example of three equations with three unknowns:

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

On the assumption that $a_{11} \neq 0$, the first equation of the system is divided by a_{11}. Then from each remaining equation we subtract the first equation multiplied by the appropriate coefficient a_{i1}. As a result of these operations, the system takes the form:

$$\begin{bmatrix} 1 & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix},$$

where

$$\begin{cases} a_{1j}^{(1)} = a_{1j} / a_{11} & j = 1, 2, 3, \\ b_1^{(1)} = b_1 / a_{11} & \\ a_{ij}^{(1)} = a_{ij} - a_{i1} a_{1j}^{(1)} & i = 2, 3, \quad j = 1, 2, 3 \\ b_i^{(1)} = b_i - a_{i1} b_1^{(1)} & i = 2, 3 \end{cases}.$$

The first unknown was eliminated from all equations, except the first. Now, on the assumption that $a_{22}^{(1)} \neq 0$ we divide the second equation by the coefficient $a_{22}^{(1)}$ and eliminate the second unknown x_2 from the third equation, resulting

$$\begin{bmatrix} 1 & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 & 1 & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(2)} \end{bmatrix},$$

where

$$\begin{cases} a_{2j}^{(2)} = a_{2j}^{(1)} \big/ a_{22}^{(1)} & j = 2, 3, \\ b_2^{(2)} = b_2^{(1)} \big/ a_{22}^{(1)} & \\ a_{ij}^{(2)} = a_{ij}^{(1)} - a_{i2}^{(1)} a_{2j}^{(2)} & i = 2, 3, \quad j = 2, 3 \\ b_i^{(2)} = b_i^{(1)} - a_{i2}^{(1)} b_2^{(2)} & i = 3 \end{cases}.$$

Finally, the third equation is divided by $a_{33}^{(2)}$, which for a nonsingular matrix must be nonzero $a_{33}^{(2)} \neq 0$. The resulted equivalent system is

$$\begin{bmatrix} 1 & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 & 1 & a_{23}^{(2)} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(3)} \end{bmatrix},$$

where $b_3^{(3)} = b_3^{(2)} \big/ a_{33}^{(2)}$. The solution is obtained by *backward substitution*

$$\begin{cases} x_3 = b_3^{(3)} \\ x_2 = b_2^{(2)} - a_{23}^{(2)} x_3 \\ x_1 = b_1^{(1)} - (a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3) \end{cases}.$$

One can see that the final matrix $A^{(3)}$ is *upper-triangular* with the elements on the main diagonal equal to 1. As a bonus, the determinant of matrix A is given by

$$\det(A) = a_{11} a_{22}^{(1)} a_{33}^{(2)}.$$

Let us return to the general case of N equations with N unknowns. After k elimination steps, we have to eliminate the unknown x_k. The system is given by

$$\begin{bmatrix} 1 & a_{12}^{(1)} & \dots & a_{1k}^{(1)} & a_{1k+1}^{(1)} & \dots & a_{1N}^{(1)} \\ \vdots & 1 & \dots & a_{2k}^{(2)} & a_{2k+1}^{(2)} & \dots & a_{2N}^{(2)} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 1 & a_{kk+1}^{(k)} & \dots & a_{kN}^{(k)} \\ 0 & 0 & 0 & 0 & a_{k+1k+1}^{(k)} & \dots & a_{k+1N}^{(k)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & a_{Nk+1}^{(k)} & \dots & a_{NN}^{(k)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ x_{k+1} \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_k^{(k)} \\ b_{k+1}^{(k)} \\ \vdots \\ b_N^{(k)} \end{bmatrix},$$

where

$$\begin{cases} a_{kk}^{(k)} = 1 \\ a_{kj}^{(k)} = a_{kj}^{(k-1)} \big/ a_{kk}^{(k-1)}, \quad j = k+1, \dots, N \\ b_k^{(k)} = b_k^{(k-1)} \big/ a_{kk}^{(k-1)} \\ a_{ik}^{(k)} = 0 \\ a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{ik}^{(k-1)} a_{kj}^{(k)}, \quad i, j = k+1, \dots, N \\ b_i^{(k)} = b_i^{(k-1)} - a_{ik}^{(k-1)} b_k^{(k)} \end{cases}.$$

Finally, the equivalent system is reduced to the upper-triangular form

$$\begin{bmatrix} 1 & a_{12}^{(1)} & \dots & a_{1k}^{(1)} & a_{1k+1}^{(1)} & \dots & a_{1N}^{(1)} \\ \vdots & 1 & \dots & a_{2k}^{(2)} & a_{2k+1}^{(2)} & \dots & a_{2N}^{(2)} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & a_{kk+1}^{(k)} & \dots & a_{kN}^{(k)} \\ 0 & 0 & \dots & 0 & 1 & \dots & a_{k+1N}^{(k+1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ x_{k+1} \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_k^{(k)} \\ b_{k+1}^{(k+1)} \\ \vdots \\ b_N^{(N)} \end{bmatrix}.$$

Using the back substitution, the solution vector is obtained as follows

$$\begin{cases} x_N = b_N^{(N)} \\ x_k = b_k^{(k)} - \sum\limits_{i=k+1}^{N} a_{ki}^{(k)} x_i, \quad k = N-1, \dots, 1 \end{cases}.$$

Obviously, the determinant of the matrix A is given by

$$\det(A) = a_{11} a_{22}^{(1)} \dots a_{NN}^{(N-1)}.$$

In order to avoid the effect of computational error corresponding to a division by a possible zero diagonal element $a_{kk}^{(k)} = 0$, we must use the Gaussian method with *pivoting*.

The *pivoting procedure* consists in interchanging rows (partial pivoting) or rows and columns (full pivoting) so as to put a particularly desirable element (the pivot) in the diagonal position. In practice, the pivot corresponds to the largest (in magnitude) available element.

Example
Apply the Gauss elimination method with partial pivoting to the following system:

$$\begin{bmatrix} 1 & 2 & 1 \\ 1 & 0 & 2 \\ 2 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}.$$

- interchange $(row\,1)$ and $(row\,3)$;

$$\begin{bmatrix} 2 & 1 & 2 \\ 1 & 0 & 2 \\ 1 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$$

- $-(1/2)\times(row\,1)+(row\,2)$;
- $-(1/2)\times(row\,1)+(row\,3)$;

$$\begin{bmatrix} 2 & 1 & 2 \\ 0 & -1/2 & 1 \\ 0 & 3/2 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}$$

- interchange $(row\,2)$ and $(row\,3)$;

$$\begin{bmatrix} 2 & 1 & 2 \\ 0 & 3/2 & 0 \\ 0 & -1/2 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ 0 \end{bmatrix}$$

- $(1/3)\times(row\,2)+(row\,3)$;

$$\begin{bmatrix} 2 & 1 & 2 \\ 0 & 3/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ -1/3 \end{bmatrix}$$

- Backward substitution gives

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5/3 \\ -2/3 \\ -1/3 \end{bmatrix}.$$

C program

```
/*--------------------------------------------------------------------
Gaussian Elimination and Pivoting
(Upper-Triangularization Followed by Back Substitution)

To find the solution of AX=B, by reducing the matrix A to
upper-triangular form and performing the back substitution.
--------------------------------------------------------------------*/
       #include<stdio.h>
       #include<stdlib.h>
       #include<math.h>
/*------------------------------------------------------------------*/
       #define Nmax  20     // The maximum number of equations
/*------------------------------------------------------------------*/
void Gauss(int N, double a[][Nmax+1])
{
int j, k, p, q, t;         // Loop counters
int row[Nmax];             // Vector with row-numbers
double x[Nmax], sum, m;    // Solution-vector

// Initialize the vector with row-numbers
for(j=1; j<=N; j++) row[j-1]=j-1;
// Start upper-triangularization routine
for(p=1; p<=N-1; p++)
       {
       for(k=p+1; k<=N; k++)
              {
              if(fabs(a[row[k-1]][p-1])>fabs(a[row[p-1]][p-1]))
                     {
                     t=row[p-1];
                     row[p-1]=row[k-1]; row[k-1]=t;
                     }
              }
       if(a[row[p-1]][p-1]==0)
              {
              printf("\n ERROR: the matrix is singular !");
              exit(1);
              }
       for(k=p+1; k<=N; k++)
              {
              m=a[row[k-1]][p-1]/a[row[p-1]][p-1];
              for(q=p+1; q<=N+1; q++)
                     {
                     a[row[k-1]][q-1]-=m*a[row[p-1]][q-1];
                     }
              }
       }
// End of upper-triangularization routine
// Start the backward substitution routine
if(a[row[N-1]][N-1]==0)
       {
       printf("\n Error: the matrix is singular !");
       exit(1);
       }
x[N-1]=a[row[N-1]][N]/a[row[N-1]][N-1];
for(k=N-1; k>=1; k--)
       {
       sum=0; for(q=k+1; q<=N; q++) sum+=a[row[k-1]][q-1]*x[q-1];
       x[k-1]=(a[row[k-1]][N]-sum)/a[row[k-1]][k-1];
       }
// End of the backward substitution routine
// Solution vector
for(k=1; k<=N; k++) a[k-1][N]=x[k-1];
}
/*-------------------------------------------------------------*/
void main(void)
{
int j, k;                   // Loop counters
double a[Nmax][Nmax+1];     // a=[A, B]  in  AX=B
int N;                      // Number of equations

// Start the main routine
// Initialize the augmented matrix a=[A, B]
printf("\n Input the number of equations (N<%d) = ", Nmax);
scanf("%d", &N);
printf(" Enter elements of a=[A, B] row by row:\n");
for(k=1; k<=N; k++)
       {
       for(j=1; j<=N+1; j++)
              {
              printf(" Input a[%d][%d]=", k, j);
              scanf("%lf", &a[k-1][j-1]);
              }
       }
Gauss(N, a);
printf("\n Solution:");
for(k=1; k<=N; k++) printf("\n x[%d]=%lf", k, a[k-1][N]);
}
/*-------------------------------------------------------------*/
```

Remark

By convention, the first index of an element a_{ij} denotes its row, the second its column. In mathematical notation indices take the values $i, j = 1, 2, \ldots, N$, in a C notation the indices take values $i, j = 0, 1, \ldots, N-1$.

Example
Solve the linear system $A \cdot X = B$, where

$$a = [A, B] = \begin{bmatrix} 1 & -2 & -1 & 2 & -2 \\ 2 & 0 & 1 & 2 & 5 \\ 2 & 0 & 4 & 1 & 7 \\ 1 & 6 & 1 & 2 & 16 \end{bmatrix}$$

```
Solution:
x[1] = 1.000000
x[2] = 2.000000
x[3] = 1.000000
x[4] = 1.000000
```

LU Decomposition

A more efficient way for solving linear systems is the *LU decomposition method* which implies a *lower-upper factorization* of the given matrix A. In other words, we seek to represent the matrix A as a product of two triangular matrices, such that

$$A = L \cdot U$$

with

$$L = \begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ l_{N1} & l_{N2} & \dots & l_{NN} \end{bmatrix}, \quad U = \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1N} \\ 0 & u_{22} & \dots & u_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & u_{NN} \end{bmatrix}.$$

The LU decomposition can be used to solve the linear system

$$A \cdot X = B = (L \cdot U) \cdot X = L \cdot (U \cdot X) = B$$

by first solving for the vector Y the linear system

$$L \cdot Y = B$$

and then solving

$$U \cdot X = Y .$$

The auxiliary vector Y is computed by *forward substitution*:

$$y_1 = b_1 / l_{11} , \quad y_i = \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij} y_j \right), \quad i = 2, \ldots, N .$$

The solution vector X is then obtained by *backward substitution* in the same manner as in the Gauss elimination method

$$x_N = y_N / u_{NN} , \quad x_i = \frac{1}{u_{ii}} \left(y_i - \sum_{j=i+1}^{N} u_{ij} x_j \right), \quad i = N-1, \ldots, 1 .$$

The system $L \cdot U = A$ is equivalent to the following N^2 equations

$$\sum_{k=1}^{N} l_{ik} u_{kj} = a_{ij} , \quad i, j = 1, \ldots, N ,$$

with $N^2 + N$ unknowns l_{ik}, u_{kj} (the diagonal being represented twice). In fact, it is always possible to put $l_{ii} = 1$, $i = 1, \ldots, N$. Also, due to the triangular structure of L and U, the summation index k will not run over the whole interval $[1, \ldots, N]$. We have, in fact, the following equations:

$$\sum_{k=1}^{i} l_{ik} u_{kj} = a_{ij} \quad \text{for} \quad i \le j ,$$

$$\sum_{k=1}^{j} l_{ik} u_{kj} = a_{ij} \quad \text{for} \quad i > j ,$$

This leads to the following recursive procedure for computing l_{ik}, u_{kj} :

$$u_{1j} = a_{1j} , \quad j = 1, \ldots, N$$

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} , \quad i = 2, \ldots, N$$

$$l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right), \quad i = j+1, \ldots, N .$$

As a consequence of the LU decomposition *the determinant* is calculated very easily as following

$$\det(A) = \det(L\cdot U) = \det(L)\cdot\det(U) = u_{11}\cdot u_{22}\cdot \ldots \cdot u_{NN}\,.$$

Pivoting is also an important aspect of the LU algorithm because of the division operation used in the above equations. Here, only a partial pivoting (interchange of rows) can be implemented. It follows that practically we decompose a rowwise permutation of A. Including *the permutation matrix*, the original system is transformed like that

$$(P\cdot A)\cdot X = (L\cdot U)\cdot X = P\cdot B\,.$$

Example

Apply the LU decomposition with pivoting to the following matrix:

$$A = A^{(0)} = \begin{bmatrix} -3 & 2 & 3 & -1 \\ 6 & -2 & -6 & 0 \\ -9 & 4 & 10 & 3 \\ 12 & -4 & -13 & -5 \end{bmatrix}.$$

- $row = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$, $Pivot = A_{41}^{(0)}$,

- $A^{(0\,new)} = \begin{bmatrix} 12 & -4 & -13 & -5 \\ 6 & -2 & -6 & 0 \\ -9 & 4 & 10 & 3 \\ -3 & 2 & 3 & -1 \end{bmatrix}$, $row = \begin{bmatrix} 4 \\ 2 \\ 3 \\ 1 \end{bmatrix}$, $l^{(1)} = \begin{bmatrix} 1 \\ 1/2 \\ -3/4 \\ -1/4 \end{bmatrix}$, $u^{(1)} = \begin{bmatrix} 12 \\ -4 \\ -13 \\ -5 \end{bmatrix}$,

- $A^{(1)} = A^{(0\,new)} - l^{(1)}\cdot u^{(1)T} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 5/2 \\ 0 & 1 & 1/4 & -3/4 \\ 0 & 1 & -1/4 & -9/4 \end{bmatrix}$, $row = \begin{bmatrix} 4 \\ 2 \\ 3 \\ 1 \end{bmatrix}$, $Pivot = A_{32}^{(1)}$,

- $A^{(1\,new)} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1/4 & -3/4 \\ 0 & 0 & 1/2 & 5/2 \\ 0 & 1 & -1/4 & -9/4 \end{bmatrix}$, $row = \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \end{bmatrix}$, $l^{(2)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$, $u^{(2)} = \begin{bmatrix} 0 \\ 1 \\ 1/4 \\ -3/4 \end{bmatrix}$,
- $A^{(2)} = A^{(1\,new)} - l^{(2)} \cdot u^{(2)T} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 5/2 \\ 0 & 0 & -1/2 & -3/2 \end{bmatrix}$, $row = \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \end{bmatrix}$, $Pivot = A_{33}^{(2)}$,
- $A^{(2\,new)} = A^{(2)}$, $row = \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \end{bmatrix}$, $l^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \end{bmatrix}$, $u^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 1/2 \\ 5/2 \end{bmatrix}$,
- $A^{(3)} = A^{(2\,new)} - l^{(3)} \cdot u^{(3)T} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, $row = \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \end{bmatrix}$,
- $l^{(4)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$, $u^{(4)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$.

Hence

$$P \cdot A = L \cdot U,$$

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \; L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3/4 & 1 & 0 & 0 \\ 1/2 & 0 & 1 & 0 \\ -1/4 & 1 & -1 & 1 \end{bmatrix}, \; U = \begin{bmatrix} 12 & -4 & -13 & -5 \\ 0 & 1 & 1/4 & -3/4 \\ 0 & 0 & 1/2 & 5/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

C program

```
/*---------------------------------------------------------------------
LU Factorization with Pivoting

To find the solution of the linear system AX=B
by using the following steps:

- Find the matrices P, L, U that satisfy PA=LU, where
             P is the permutation matrix
             L is the lower-triangular matrix
             U is the upper-triangular matrix
- Find the solution Y of the lower-triangular system LY=PB.
- Find the solution X of the upper-triangular system UX=Y.

The diagonal elements of L are all 1 (these values are not stored).
The coefficients of L and U overwrite the matrix A.
The solution is returned in the last column of augmented matrix
a=[A,B].
The determinant of matrix A is given in det variable.
--------------------------------------------------------------------*/
       #include<stdio.h>
       #include<stdlib.h>
       #include<math.h>
/*-------------------------------------------------------------------*/
       #define Nmax  20     // The maximum number of equations
/*------------------------------------------------------------------*/
double LU_factorization(int N, double a[][Nmax+1])
{
int k, p, q, j, t;                // Loop counters
double y[Nmax], x[Nmax];          // Solution-vectors
int row[Nmax];                    // Vector with row-numbers
double sum, d=1.0;

// Initialize the vector with row-numbers
for(j=1; j<=N; j++) row[j-1]=j-1;
// Start the LU factorization routine
for(p=1; p<=N-1; p++)
       {
       for(k=p+1; k<=N; k++)
              {
              if(fabs(a[row[k-1]][p-1])>fabs(a[row[p-1]][p-1]))
                     {
                     d=-d;
                     t=row[p-1];
                     row[p-1]=row[k-1];
                     row[k-1]=t;
                     }
              }
```

```
        if(a[row[p-1]][p-1]==0)
                {
                printf("ERROR: the matrix is singular !\n");
                exit(1);
                }
        d=d*a[row[p-1]][p-1];
        for(k=p+1; k<=N; k++)
                {
                a[row[k-1]][p-1]=a[row[k-1]][p-1]/a[row[p-1]][p-1];
                for(q=p+1; q<=N; q++)
                        {
                a[row[k-1]][q-1]-=a[row[k-1]][p-1]*a[row[p-1]][q-1];
                        }
                }
        }
d=d*a[row[N-1]][N-1];
// Start the forward substitution routine
y[0]=a[row[0]][N];
for(k=2;k<=N;k++)
        {
        sum=0;
        for(q=1; q<=k-1; q++) sum+=a[row[k-1]][q-1]*y[q-1];
        y[k-1]=a[row[k-1]][N]-sum;
        }
if(a[row[N-1]][N-1]==0)
        {
        printf("ERROR: the matrix is singular !\n");
        exit(1);
        }
// Start the back substitution routine
x[N-1]=y[N-1]/a[row[N-1]][N-1];
for(k=N-1; k>=1; k--)
        {
        sum=0;
        for(q=k+1; q<=N; q++) sum+=a[row[k-1]][q-1]*x[q-1];
        x[k-1]=(y[k-1]-sum)/a[row[k-1]][k-1];
        }
// Solution vector
for(k=0; k<N; k++) a[k][N]=x[k];
return d;
}
/*--------------------------------------------------------------------*/
void main(void)
{
int N, k, j;                      // N=number of equations
double a[Nmax][Nmax+1];           // a=[A,B] in AX=B
double det;                       // det=det(A)

// Initialize the augmented matrix a=[A,B]
```

```
printf("\n Input the number of equations (N<%d)=", Nmax);
scanf("%d", &N);
printf(" Enter elements of a=[A,B] row by row:\n");
for(k=1; k<=N; k++)
      {
      for(j=1; j<=N+1; j++)
            {
            printf(" Input a[%d][%d] = ", k, j);
            scanf("%lf", &a[k-1][j-1]);
            }
      }
det=LU_factorization(N, a);
printf("\n Solution:");
for(k=1; k<=N; k++) printf("\n x[%d] = %lf", k, a[k-1][N]);
printf("\n det(A)=%lf", det);
}
/*------------------------------------------------------------*/
```

Example

Solve the linear system $A \cdot X = B$, where

$$a = [A, B] = \begin{bmatrix} 1 & -2 & -1 & 2 & -2 \\ 2 & 0 & 1 & 2 & 5 \\ 2 & 0 & 4 & 1 & 7 \\ 1 & 6 & 1 & 2 & 16 \end{bmatrix}.$$

```
Solution:
x[1] = 1.000000
x[2] = 2.000000
x[3] = 1.000000
x[4] = 1.000000
det(A) = 32.000000
```

Lecture 2

Iterative Methods for Linear Systems

The Method of Simple Iteration

The methods described so far are called *exact methods*. However, in the *exact methods* the roundoff errors accumulate, and if the matrix A is near singular the errors are amplified in an inconvenient way. In such cases one can use the *iterative methods* to find an improved solution.

Let X be the exact solution of the linear system

$$A \cdot X = B ,$$

and let X' be some slightly wrong solution, such that

$$X = X' + \delta X .$$

Inserting this wrong solution in the linear system we find

$$A \cdot \delta X = A \cdot (X - X') = B - A \cdot X' .$$

The right-hand side of this equation is known, since X' is the wrong solution, and we can use it to compute δX and therefore X. Now, we can interpret this equation as an iterative formula

$$A \cdot (X^{(k+1)} - X^{(k)}) = B - A \cdot X^{(k)} , \;\; k = 0, 1, 2, \dots , \;\; X^{(0)} = X' ,$$

used to improve the solution of the linear system.

A different approach is based on the replacement of the inverse of the matrix A by an approximation $Q^{(0)} \approx A^{-1}$ and defining the residual matrix Θ as

$$\Theta = I - Q^{(0)} \cdot A .$$

Here, I is the identity matrix. It follows that

$$A^{-1} = A^{-1} \cdot ((Q^{(0)})^{-1} \cdot Q^{(0)}) = (A^{-1} \cdot (Q^{(0)})^{-1}) \cdot Q_0 = (Q^{(0)} \cdot A)^{-1} \cdot Q^{(0)} =$$

$$= (I - \Theta)^{-1} \cdot Q^{(0)} = (I + \Theta + \Theta^2 + \dots) \cdot Q^{(0)} .$$

Therefore, the inverse A^{-1} is given by

$$\lim_{n\to\infty} Q^{(n)} = A^{-1}, \quad Q^{(n)} = (I + \Theta + \ldots + \Theta^n) \cdot Q^{(0)} .$$

The solution vector is then given by the following iterative formulas

$$X^{(n)} = Q^{(n)} \cdot B ,$$

$$X^{(n+1)} = Q^{(0)} \cdot B + (I - Q^{(0)} \cdot A) \cdot X^{(n)} = Q^{(0)} \cdot B + \Theta \cdot X^{(n)} ,$$

with $Q^{(0)} \approx A^{-1}$. Here, the most important problem is to establish the conditions for the convergence of the limit $\lim_{n\to\infty} Q^{(n)} = A^{-1}$. In order to do such analysis we need to introduce few concepts regarding the *norm* of vectors and matrices.

Let $X = [x_1, x_2, \ldots, x_N]^T \in \mathbf{R}^N$, then some examples of vector norms are

$$\|X\|_1 = \sum_{k=1}^{N} |x_k|$$ (the *one norm*),

$$\|X\|_2 = \sqrt{\sum_{k=1}^{N} x_k^2}$$ (the *two norm*, or *Euclidean length*),

$$\|X\|_\infty = \max_{1\le k\le N} |x_k|$$ (the *infinity norm*, or *max norm*).

The above norms are particular cases of

$$\|X\|_p = \left(\sum_{k=1}^{N} |x_k|^p \right)^{\frac{1}{p}} .$$

Vector norms are required to satisfy:

$$\|X\| \ge 0 ,\ \forall\, X \in \mathbf{R}^N \text{ and } \|X\| = 0 \Leftrightarrow X = 0 ;$$

$$\|\lambda X\| = |\lambda| \|X\| ,\ \forall\, X \in \mathbf{R}^N \text{ and } \forall \lambda \in \mathbf{R} ;$$

$$\|X + Y\| \le \|X\| + \|Y\| ,\ \forall\, X, Y \in \mathbf{R}^N .$$

One can easily prove that all of the examples given above satisfy these requirements.

The natural norm of a square matrix A is defined in terms of a given vector norm as following

$$\|A\| = \max_{X \neq 0} \frac{\|A \cdot X\|}{\|X\|}.$$

Thus $\|A\|$ measures the maximum relative stretching in a given vector norm that occurs when multiplying all non-zero vectors $X \in \mathbf{R}^N$ by A. From the above definition it immediately results that

$$\|A \cdot X\| \leq \|A\| \cdot \|X\|, \ \forall X \in R^N.$$

Theorem
All matrix norm defined in terms of a given vector norm satisfy

(i) $\|A\| \geq 0$, and $\|A\| = 0 \Leftrightarrow A = O$ (zero matrix);

(ii) $\|\lambda A\| = |\lambda| \|A\|$, $\forall \lambda \in \mathbf{R}$;

(iii) $\|A + B\| \leq \|A\| + \|B\|$.

(iv) $\|A \cdot B\| \leq \|A\| \cdot \|B\|$.

Proof:

(iii) $$\|A + B\| = \max_{X \neq 0} \frac{\|(A+B) \cdot X\|}{\|X\|} = \max_{X \neq 0} \frac{\|A \cdot X + B \cdot X\|}{\|X\|} \leq$$

$$\leq \max_{X \neq 0} \frac{\|A \cdot X\| + \|B \cdot X\|}{\|X\|} \leq \max_{X \neq 0} \frac{\|A \cdot X\|}{\|X\|} + \max_{X \neq 0} \frac{\|B \cdot X\|}{\|X\|} = \|A\| + \|B\|.$$

(iv) $$\|A \cdot B\| = \max_{X \neq 0} \frac{\|A \cdot (B \cdot X)\|}{\|X\|} \leq \max_{X \neq 0} \frac{\|A\| \|B \cdot X\|}{\|X\|} \leq \max_{X \neq 0} \frac{\|A\| \|B\| \|X\|}{\|X\|} = \|A\| \|B\|.$$

For specific choices of vector norm it is convenient to express the corresponding induced matrix norm in terms of the elements of the matrix. For example, in the case of the *infinity norm* we have

$$\frac{\|A\cdot X\|_\infty}{\|X\|_\infty}=\frac{\max\limits_{1\le i\le N}\left|\sum\limits_{j=1}^{N}a_{ij}x_j\right|}{\|X\|_\infty}\le\frac{\max\limits_{1\le i\le N}\sum\limits_{j=1}^{N}|a_{ij}||x_j|}{\|X\|_\infty}\le$$

$$\le\frac{\max\limits_{1\le i\le N}\sum\limits_{j=1}^{N}|a_{ij}|\|X\|_\infty}{\|X\|_\infty}=\max_{1\le i\le N}\sum_{j=1}^{N}|a_{ij}|.$$

Also, for any square matrix A there is always a vector Y for which

$$\frac{\|A\cdot Y\|_\infty}{\|Y\|_\infty}=\max_{1\le i\le N}\sum_{j=1}^{N}|a_{ij}|.$$

To show this let k be the row of A for which $\sum|a_{kj}|$ is maximum and take the vector

$$Y=[y_1,y_2,\dots,y_N]^T\in R^N\ ,\quad y_j=\begin{cases}+1 & if \quad a_{kj}\ge 0\\ -1 & if \quad a_{kj}<0\end{cases},\quad j=1,\dots,N.$$

It follows that

$$\frac{\|A\cdot Y\|_\infty}{\|Y\|_\infty}=\max_{1\le i\le N}\sum_{j=1}^{N}|a_{ij}y_j|=\sum_{j=1}^{N}|a_{kj}|.$$

From the above considerations, clearly results that

$$\|A\|_\infty=\max_{1\le i\le N}\sum_{j=1}^{N}|a_{ij}| \quad \text{(maximum absolute row sum).}$$

Similarly one can show that

$$\|A\|_1=\max_{1\le j\le N}\sum_{j=1}^{N}|a_{ij}| \quad \text{(maximum absolute column sum).}$$

Example
Let us consider

$$A = \begin{bmatrix} 1 & 2 & -2 \\ 1 & 0 & 3 \\ -1 & 3 & 1 \end{bmatrix},$$

then

$$\|A\|_\infty = \max\{5,4,5\} = 5 \quad \text{and} \quad \|A\|_1 = \max\{3,5,6\} = 6 .$$

Now, we need to introduce few concepts regarding the *eigenvalues* and the *eigenvectors* of matrices. We shall consider how to compute the eigenvalues of matrices in other lecture. For the present it is sufficient to know that if there are non-zero vectors X_i and scalars λ_i which together satisfy the equations

$$A \cdot X_i = \lambda_i X_i , \quad i = 1, \ldots, N ,$$

where A is a square $N \times N$ matrix, then the constants λ_i are called eigenvalues of the matrix A, and X_i are the corresponding eigenvectors. The eigenvalues are solutions of the *characteristic equation*

$$\det(A - \lambda I) = 0 .$$

One can show that

$$\|A\|_2 = \max_{X \neq 0} \frac{\|A \cdot X\|_2}{\|X\|_2} = \max_{1 \le i \le N} \sqrt{\lambda_i} ,$$

where λ_i, $i = 1, \ldots, N$, are the eigenvalues of the matrix $A^T \cdot A$ (these eigenvalues are indeed nonnegative because the matrix $A^T \cdot A$ is real and symmetric).

Theorem
For any matrix norm we have

$$\rho(A) \le \|A\| ,$$

where $\rho(A)$ is the modulus of the eigenvalue of greatest magnitude of a matrix A (the *spectral radius* of matrix A).