Chapter 3

Solving Systems of Linear Equations By Gaussian Elimination

3.1 Mathematical Preliminaries

In this chapter we consider the problem of computing the **solution of a system** of n linear equations in n unknowns. The scalar form of that system is as follows:

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

Written in matrix form, (S) is equivalent to:

$$Ax = b, \tag{3.1}$$

where the coefficient square matrix $A \in \mathbb{R}^{n,n}$, and the column vectors $x, b \in \mathbb{R}^{n,1} \cong \mathbb{R}^n$. Specifically,

$$A = \begin{pmatrix} a_{11} & a_{12} & ... & ... & a_{1n} \\ a_{21} & a_{22} & ... & ... & a_{2n} \\ ... & ... & ... & ... & ... \\ a_{n1} & a_{n2} & ... & ... & a_{nn} \end{pmatrix}$$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{pmatrix} \text{ and } b = \begin{pmatrix} b_1 \\ b_2 \\ ... \\ b_n \end{pmatrix}.$$

We assume that the basic linear algebra property for systems of linear equations like (3.1) are satisfied. Specifically:

Proposition 3.1. *The following statements are equivalent:*

1. *System (3.1) has a unique solution.*
2. $\det(A) \neq 0$.
3. *A is invertible.*

In this chapter, our objective is to present the basic ideas of a **linear system solver**. It consists of two main procedures allowing to solve efficiently (3.1).

1. The first, referred to as **Gauss elimination (or reduction)** reduces (3.1) into an equivalent system of linear equations, which matrix is **upper triangular**. Specifically one shows in section 4 that

   $$Ax = b \Longleftrightarrow Ux = c,$$

   where $c \in \mathbb{R}^n$ and $U \in \mathbb{R}^{n,n}$ is given by:

   $$U = \begin{pmatrix} u_{11} & u_{12} & ... & ... & u_{1n} \\ 0 & u_{22} & ... & ... & u_{2n} \\ ... & ... & ... & ... & ... \\ 0 & 0 & ... & u_{n-1,n-1} & u_{n-1,n} \\ 0 & 0 & ... & 0 & u_{nn} \end{pmatrix}.$$

   Thus, $u_{ij} = 0$ for $i > j$. Consequently, one observes that A is invertible if and only if

   $$\Pi_{i=1}^{n} u_{ii} = u_{11}u_{22}...u_{nn} \neq 0, \text{ i.e. } u_{ii} \neq 0 \ \forall i.$$

2. The second procedure consists in solving by **back substitution** the upper triangular system

   $$Ux = c. \tag{3.2}$$

A picture that describes the two steps of the linear solver is:

Input $A, b \rightarrow$ **Gauss Reduction** $\rightarrow$ Output $U, c \rightarrow$ **Back Substitution** $\rightarrow$ Output x

Our plan in this chapter is as follows. We start in section 2 by discussing issues related to computer storage. It is followed in section 3 by the presentation of the back substitution procedure that solves upper triangular systems, such as (3.2). Finally in section 4 we present various versions of Gauss reduction, the simplest of which is **Naive Gaussian elimination**.

3.2 Computer Storage for matrices. Data Structures

The data storage for A and b is through one data structure: the **augmented matrix** $AG \in \mathbb{R}^{n,n+1}$, given by:

$$AG = \begin{pmatrix} a_{11} & a_{12} & ... & ... & a_{1n} & b_1 \\ a_{21} & a_{22} & ... & ... & a_{2n} & b_2 \\ ... & ... & ... & ... & ... & ... \\ a_{n1} & a_{n2} & ... & ... & a_{nn} & b_n \end{pmatrix}$$

We generally assume that the matrix A is a **full** matrix, that is "most of its elements are non-zero". Storing the augmented matrix AG for a full matrix in its standard form, would then require $N = n \times (n+1)$ words of computer memory. If one uses single precision, $4N$ bytes would be necessary, while using double precision would necessitate $8N$ bytes for that storage.
For instance, when the matrix size is $n = 2^k$, the computer memory for double precision computation should exceed $N = 8 \times 2^k(2^k + 1) \approx O(2^{2k+3})$ bytes.
The following table illustrates some magnitudes of memory requirements.

k	$n = 2^k$	$N = n \times (n+1)$	≈in Megabytes	
			IEEE single precision	IEEE double precision
3	8	72	2.7×10^{-4}	5.5×10^{-4}
6	64	4160	1.6×10^{-2}	3.2×10^{-2}
8	256	65792	0.25	0.5
10	1024	1049600	4	8

Practically, computer storage is usually **one-dimensional**. As a result, matrix elements are either stored **column-wise** (as in MATLAB), or **row-wise**. In the case where the elements of the augmented matrix AG are contiguously stored by columns, this storage would obey the following sequential pattern:

$$|\underbrace{a_{11}\, a_{21} \ldots a_{n1}}_{\text{column 1}}|\underbrace{a_{12} \ldots a_{n2}}_{\text{column 2}}|\ldots|\underbrace{a_{1n} \ldots a_{nn}}_{\text{column n}}|\underbrace{b_1\, b_2 \ldots,\, b_n}_{\text{column n+1}}|$$

while if stored by rows, the storage pattern for the augmented matrix elements becomes:

$$|\underbrace{a_{11}\, a_{12} \ldots a_{1n}\, b_1}_{\text{line 1 1}}|\underbrace{a_{21} \ldots a_{2n}\, b_2}_{\text{line 2}}|\ldots|\underbrace{a_{n1} \ldots a_{nn}\, b_n}_{\text{line n}}|$$

Once Gauss reduction has been applied to the original system $Ax = b$, the resulting upper triangular system $Ux = c$ would necessitate the storage of the upper triangular matrix U and the right hand side vector c. Obviously, the augmented matrix for this system is given by:

$$UG = \begin{pmatrix} u_{11} & u_{12} & \ldots & \ldots & u_{1n} & c_1 \\ 0 & u_{22} & \ldots & \ldots & u_{2n} & c_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & \ldots & 0 & u_{nn} & c_n \end{pmatrix}$$

Since by default, the lower part of the matrix U consists of zeros, this part of the storage shall not be waisted but used for other purposes, particularly that of storing the **multiplying factors**, which are essential parameters to carry out Gauss elimination procedure. Hence, at this stage we may consider the data structure UG whether stored by rows or by columns as consisting of the elements of U and c and unused storage space:

$$UG = \begin{pmatrix} u_{11} & u_{12} & \ldots & \ldots & u_{1n} & c_1 \\ \boxed{\text{unused}} & u_{22} & \ldots & \ldots & u_{2n} & c_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \boxed{\text{unused}} & \ldots & \ldots & \boxed{\text{unused}} & u_{nn} & c_n \end{pmatrix}$$

We turn now to the Back substitution procedure.

3.3 Back Substitution for Upper Triangular Systems

Although this procedure comes after the completion of the Gauss Reduction step, we shall deal with it from the start. It indeed provides the importance of this global approach.
Considering (3.2) in its scalar form, with all diagonal elements $u_{ii} \neq 0$, gives:

$$\begin{pmatrix} u_{11} & u_{12} & ... & ... & u_{1n} \\ 0 & u_{22} & ... & ... & u_{2n} \\ ... & ... & ... & ... & ... \\ 0 & 0 & ... & u_{n-1,n-1} & u_{n-1,n} \\ 0 & 0 & ... & 0 & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ ... \\ c_{n-1} \\ c_n \end{pmatrix}$$

Solving this system by the **back substitution** procedure reduces such procedure to solving n equations, each one in one unknown only.
We give two versions of the back-substitution process: the first one is **column oriented,** while the second one is **row oriented**. We then evaluate and compare the computational complexity of each version.

1. **Column-version:** The two main steps are as follows:

   (a) Starting with $j = n : -1 : 1$, solve the last equation for x_j, where $x_j = c_j/u_{j,j}$.

   (b) In all rows above, that is from row $i = 1 : (j-1)$, compute the new right hand side vector that results by "shifting" the last column of the matrix (terms in x_j) to the right hand side. For example when $j = n$, the new system to solve at this step is as follows:

$$\begin{pmatrix} u_{11} & u_{12} & ... & ... & u_{1,n-1} \\ 0 & u_{22} & ... & ... & u_{2,n-1} \\ ... & ... & ... & ... & ... \\ 0 & 0 & ... & 0 & u_{n-1,n-1} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} c_1 - u_{1n}x_n \\ c_2 - u_{2n}x_n \\ ... \\ c_{n-1} - u_{n-1,n}x_n \end{pmatrix}$$

This process is repeated till the 1^{st} row is reached, where:

$$u_{11}x_1 = c_1 - u_{1,n}x_n - u_{1,n-1}x_{n-1} - ... - u_{12}x_2$$

leading thus to $x_1 = (c_1 - u_{1,n}x_n - u_{1,n-1}x_{n-1} - ... - u_{12}x_2)/u_{11}$.

The corresponding algorithm is implemented as follows:

```
function x = ColBackSubstitution(U,c)
% Input: U  an upper-triangular invertible matrix, and
%            c a column vector
% Output: solution vector x of system Ux = c
% Storage is column oriented
n=length(c) ;
for j=n:-1:1
       x(j)=c(j)/U(j,j);
       for i=1: j-1
            c(i)=c(i) - U(i,j) * x(j);
       end
end
```

The number of floating point operations used in this algorithm is n^2, and is computed as follows:

- For every $j, (j = 1 : n)$: 1 division is needed to compute $x(j)$ adding up therefore to a total of n flops.
- For every $j, (j = 1 : n)$ and for every $i, (i = 1 : j - 1)$, to compute each modified right hand side term $c(i)$: 1 addition + 1 multiplication are used, that sum up to a total of:

$$\sum_{j=1}^{n}\sum_{i=1}^{j-1} 2 = \sum_{j=1}^{n} 2[(j-1) - 1 + 1] = 2(1+2+...+(n-1)) = n(n-1)$$

As for the $2nd$ version, the rows are successively and completely solved for one unknown, starting with the last one $(i = n)$.

2. **Row-version:**

```
% Input and Output as in "ColBackSubstitution" above
% Storage is row oriented
function x = RowBackSubstitution(U,c)
```

```
n=length(c);
x(n)=c(n)/U(n,n);
for i=n-1:-1:1
    for j=i+1:n
         c(i)=c(i)-U(i,j) * x(j);
    end
x(i)=c(i)/U(i,i);
end
```

It is easy to verify in that case that the total number of flops used remains equal to n^2 .

3.4 Gauss Reduction

Our starting point is to assume "ideal mathematical conditions" allowing to carry the **reduction** without any safeguard. Before setting formally these assumptions, we work out the following example:

Example 3.1. *Consider the reduction of the following system into upper triangular form :*

$$\left\{\begin{array}{rrrrl} x_1 & -x_2 & +2x_3 & +x_4 & =1 \\ 3x_1 & +2x_2 & +x_3 & +4x_4 & =1 \\ 5x_1 & 8x_2 & +6x_3 & +3x_4 & =1 \\ 4x_1 & +2x_2 & +5x_3 & +3x_4 & =-1 \end{array}\right. \tag{3.3}$$

The corresponding augmented matrix being:

$$\left(\begin{array}{rrrrr} 1 & -1 & 2 & 1 & 1 \\ 3 & 2 & 1 & 4 & 1 \\ 5 & 8 & 6 & 3 & 1 \\ 4 & 2 & 5 & 3 & -1 \end{array}\right)$$

We proceed by applying successively 3 Gauss reductions. In each one of these, the following **linear algebra elementary operation** is being used: at the k^{th} reduction, $k = 1, 2, 3$, and for $i = k + 1, ..., 4$

$$\text{(New) Equ } i \leftarrow \text{ (Previous) Equ } i - \text{(multiplier)} \times \text{ Pivot Equ } k \tag{3.4}$$

More explicitly:

1. **Reduction 1**. The **pivot equation** is the 1^{st} equation ($k = 1$), the **pivot element** is $a_{11} = 1$. The respective **multipliers** for i successively $2, 3, 4$ are $\{\frac{a_{1i}}{a_{11}} = 3, 5, 4\}$. Thus, performing (3.4) repeatedly:

$$\text{Equation } 2 \leftarrow \text{ Equation } 2 - 3 \times \text{ Pivot Equation } 1,$$

$$\text{Equation } 3 \leftarrow \text{ Equation } 3 - 5 \times \text{ Pivot Equation } 1,$$

$$\text{Equation } 4 \leftarrow \text{ Equation } 4 - 4 \times \text{ Pivot Equation } 1,$$

At this stage, the modified augmented matrix is:

$$\begin{pmatrix} 1 & -1 & 2 & 1 & 1 \\ 0 & 5 & -5 & 1 & -2 \\ 0 & 13 & -4 & -2 & -4 \\ 0 & 6 & -3 & -1 & -5 \end{pmatrix}.$$

In order not to waste the implicitly zero storage locations, we use them to place the multipliers of the first reduction. Hence, at the accomplishment of reduction 1, the augmented matrix takes the form:

$$\begin{pmatrix} 1 & -1 & 2 & 1 & 1 \\ \boxed{3} & 5 & -5 & 1 & -2 \\ \boxed{5} & 13 & -4 & -2 & -4 \\ \boxed{4} & 6 & -3 & -1 & -5 \end{pmatrix}.$$

with the understanding that "boxed" elements are the corresponding multipliers.

2. **Reduction 2**. Perform repeatedly operation (3.4) with the **second pivot equation** ($k = 2$), the pivot element being here $a_{22} = 5$, and i successively 3,4. The multipliers are respectively $\{\frac{a_{2i}}{a_{22}} = \frac{13}{5}, \frac{6}{5}\}$.

$$\text{Equation } 3 \leftarrow \text{ Equation } 3 - \frac{13}{5} \times \text{ Equation } 2,$$

$$\text{Equation } 4 \leftarrow \text{ Equation } 4 - \frac{6}{5} \times \text{ Equation } 2,$$

The 2^{nd} reduction yields the following augmented matrix:

$$\begin{pmatrix} 1 & -1 & 2 & 1 & 1 \\ 0 & 5 & -5 & 1 & -2 \\ 0 & 0 & 9 & -23/5 & 6/5 \\ 0 & 0 & 3 & -11/5 & -13/5 \end{pmatrix}.$$

Adding the multipliers of the second reduction, the contents of the augmented matrix updated data structure are as follows:

$$\begin{pmatrix} 1 & -1 & 2 & 1 & 1 \\ \boxed{3} & 5 & -5 & 1 & -2 \\ \boxed{5} & \boxed{13/5} & 9 & -23/5 & 6/5 \\ \boxed{4} & \boxed{6/5} & 3 & -11/5 & -13/5 \end{pmatrix}.$$

Finally, we come to the last reduction.

3. **Reduction 3**. Perform operation (3.4) with the **third pivot equation** ($k = 3$), the pivot element being $a_{33} = 9$, and the sole multiplier being $\{\frac{a_{3i}}{a_{33}} = \frac{1}{3}\}$, for $i = 4$. Specifically:

$$\text{Equation } 4 \leftarrow \text{ Equation } 4 - \frac{1}{3} \times \text{ Equation } 3,$$

yields the augmented matrix:

$$\begin{pmatrix} 1 & -1 & 2 & 1 & 1 \\ 0 & 5 & -5 & 1 & -2 \\ 0 & 0 & 9 & -23/5 & 6/5 \\ 0 & 0 & 0 & -2/3 & -3 \end{pmatrix}.$$

Placing the multipliers, the updated augmented matrix is then:

$$\begin{pmatrix} 1 & -1 & 2 & 1 & 1 \\ \boxed{3} & 5 & -5 & 1 & -2 \\ \boxed{5} & \boxed{13/5} & 9 & -23/5 & 6/5 \\ \boxed{4} & \boxed{6/5} & \boxed{1/3} & -2/3 & -3 \end{pmatrix}.$$

The Back Substitution applied on the upper triangular system yields:

$$(x_1 = -217/30, x_2 = 17/15, x_3 = 73/30, x_4 = 9/2)$$

We may now discuss the assumptions leading to the Naive Gauss elimination.

3.4.1 Naive Gauss Elimination

The adjective **Naive** applies because this form is the simplest form of Gaussian elimination. It is not usually suitable for automatic computation unless essential modifications are made. We give first the condition that allows theoretically the procedure to work out successfully.

Definition 3.1. *A square matrix A_n has the* **principal minor property**, *if all its principal sub-matrices A_i, $i = 1, ..., n$ are invertible, where*

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

If a matrix A verifies Definition 3.1, the pivot element at each reduction is well defined and is located on the main diagonal. Thus, $\forall b \in \mathbb{R}^{n,1}$, the following algorithms can be applied on the augmented matrix $[A|b]$. The first one assumes that the matrix A is stored column-wise.

```
% The algorithm is column oriented
% The matrix A is assumed to have the principal minor property
% At reduction k, the kth equation is the pivot equation,  A(k,k)
% is the pivot element, and equations 1,..,k remained unchanged
function[U, c]=NaiveGauss(A,b)
n=length(b) ;
for k=1:n-1
% Get the pivot element and the multipliers proceeding by columns
    piv=A(k,k);
    for i=k+1:n
        A(i,k)=A(i,k)/piv;
    end
% Modify the body of matrix A  proceeding by columns
    for j=k+1:n
        for i=k+1:n
            A(i,j)=A(i,j)-A(i,k)*A(k,j);
        end
    end
```

```
% Modify the right hand side b
    for i=k+1:n
         b(i)=b(i)-A(i,k)*b(k);
     end
end
% Extract c and U proceeding by columns
c=b;
U=triu(A);
```

The flop count for this algorithm can be easily evaluated:

1. To find the multipliers:

$$\sum_{k=1}^{n-1}\sum_{i=k+1}^{n} 1 = \sum_{k=1}^{n-1} n-k = 1+2+...+(n-1) = \frac{n(n-1)}{2} \text{ divisions}$$

2. To modify the body of the matrix:

$$\sum_{k=1}^{n-1}\sum_{j=k+1}^{n}\sum_{i=k+1}^{n} 2 = \sum_{k=1}^{n-1}\sum_{j=k+1}^{n} 2(n-k) = 2\sum_{k=1}^{n-1}(n-k)^2 = 2[1^2+2^2+...+(n-1)^2]$$

$$= 2\left[\frac{n(n-1)(2n-1)}{6}\right] \text{ operations.}$$

3. To modify the right hand side vector:

$$\sum_{k=1}^{n-1}\sum_{i=k+1}^{n} 2 = 2\sum_{k=1}^{n-1} n-k = 2[1+2+...+(n-1)] = 2\left[\frac{n(n-1)}{2}\right] \text{ operations}$$

In term of **flops**, this would total to:

$$\frac{n(n-1)}{2} + \frac{n(n-1)(2n-1)}{3} + n(n-1) = \frac{n(n-1)}{6}(7+4n) = O(\frac{2n^3}{3}).$$

The next version requires the same number of flops but is row oriented

```
% The algorithm is row oriented
% The matrix A is assumed to have the principal minor property
% At reduction k, the kth equation is the pivot equation and A(k,k)
% is the pivot element, and equations 1,..,k remained unchanged
function[c,U]=naiveGauss(A,b)
n=length(b) ;
```

```
for k=1:n-1
% Get the pivot element
    piv=A(k,k);
% Proceed by row: get the multiplier for equation i
    for i=k+1:n
        A(i,k)=A(i,k)/piv;
% and modify its remaining coefficients, then its right hand side
           for j=k+1:n
            A(i,j)=A(i,j)-A(i,k)*A(k,j);
        end
    b(i)=b(i)-A(i,k)*b(k);
    end
end
% Extract c and U
c=b;
U=triu(A);
```

The above 2 versions that are the simplest expressions of Gaussian elimination, do not take into account the eventual sensitivity of the system to **propagate round-off errors**.

3.4.2 Partial Pivoting strategies: Unscaled (Simple) and Scaled Partial Pivoting

When computing in floating point systems $\mathbb{F}$, there are several situations where the application of the Naive Gaussian elimination algorithms fails although the matrix A may verify the principal minor property.
As an illustration consider first the case where the pivot element is relatively small in $\mathbb{F}$. This would lead to large multipliers that worsen the round-off errors, as shown in the following example.

Example 3.2. *Consider the following* 2×2 *system of equations, where* ϵ *is a small non zero number:*

$$\begin{cases} \epsilon x_1 + x_2 = 2 \\ 3x_1 + x_2 = 1 \end{cases} \tag{3.5}$$

The exact solution to this problem in $\mathbb{R}$ is $x_1 \approx \frac{-1}{3}$ and $x_2 \approx 2$.

Naive Gauss elimination where the pivot is ϵ leads to:

$$\begin{cases} \epsilon x_1 + x_2 = 1 \\ (1 - \frac{3}{\epsilon})x_2 = 1 - \frac{6}{\epsilon} \end{cases}$$

and the back substitution procedure would give:

$$\begin{cases} x_2 = \frac{1-6/\epsilon}{1-3/\epsilon} \\ x_1 = \frac{2-x_2}{\epsilon} \end{cases}$$

If these calculations are performed in a floating point system $\mathbb{F}$, as $1/\epsilon$ is large, then

$$\begin{cases} 1 - \frac{6}{\epsilon} \approx -\frac{6}{\epsilon} \\ 1 - \frac{3}{\epsilon} \approx -\frac{3}{\epsilon} \end{cases}$$

The computed solutions in that case are incorrect, with:

$$x_2 \approx 2 \text{ and } x_1 \approx 0.$$

However, if we perform a permutation of the equations before the reduction process, then the equivalent system becomes :

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

Carried out, Naive Gauss reduction would lead to:

$$\begin{cases} 3x_1 + x_2 = 1 \\ (1 - \frac{\epsilon}{3})x_2 = 2 - \frac{\epsilon}{3} \end{cases}$$

Back substitution in this case would clearly give: $x_2 \approx 2$ and $x_1 \approx -1/3$. ■

This example leads us to conclude that some type of strategy is essential for selecting new pivot equations and new pivots at each Gaussian reduction. Theoretically **Complete Pivoting** would be the best approach. This process requires at each stage, first searching over all entries of adequate submatrices - in all rows and all columns - for the largest entry in absolute value and then permuting rows and columns to move that entry into the required pivot position. This would be quite expensive as a great amount of searching and data movement would be involved. However, scanning just the 1^{st} column in the submatrix at each reduction and selecting as pivot the greatest absolute value entry accomplishes our goal, thus avoiding too small

or zero pivots. This is **Unscaled (or Simple) Partial Pivoting**. It would solve the posed problem, but compared to Complete Pivoting strategy, it does not involve an examination of the entries in the rows of the matrix.
Moreover, rather than interchanging rows through the this partial pivoting procedure, that is to avoid the data movement, we use an **indexing array**. Thus, the order in which the equations are used is denoted by the row vector **IV** called the **Index Vector**. At first, IV is set to $[1, 2, ..., n]$, then at each reduction, if there would be a permutation in the rows, it is performed only on IV which acts as a vector of pointers to the memory location of the rows. In fact, at each reduction, IV=$[i_1, i_2, ..., i_n]$ which is a permutation of the initial vector IV. This definitely eliminates the time consuming and unnecessary process of moving around the coefficients of equations in the computer memory.

We formalize now the Unscaled Partial Pivoting procedure.

1. **Gaussian Elimination with Unscaled Partial Pivoting**
   This strategy consists in first finding at reduction k, the "best" pivot equation. This is achieved by identifying the maximum absolute value element in the k^{th} column, located in some row ranging from the k^{th} row to the last. More explicitly:
   - At reduction $k = 1$, seek i_1 in the set $\{1, 2, ..., n\}$ such that:

$$|a_{i_1,1}| = \max_{1 \le i \le n} |a_{i1}| = \max \{|a_{11}|, |a_{21}|, ..., |a_{n1}|\}$$

   then perform a permutation of row 1 and row i_1 in IV only. Row i_1 is the first pivot equation, and $a_{i1,1}$ is the pivot element. We write $IV([1, i_1]) = IV([i_1, 1])$, meaning that at this stage,

$$IV = [i_1, ..., 1, ..., n] = [i_1, i_2, ..., i_n]$$

   - At reduction k, seek i_k in $\{IV(k), ..., IV(n)\}$, such that:

$$|a_{i_k,k}| = \max_{IV(k) \le i \le IV(n)} |a_{ik}| = \max \left\{|a_{IV(k),k}|, |a_{IV(k+1),k}|, ..., |a_{IV(n),k}|\right\}$$

   repositioning i_k in IV will set $i_k = i_k + (k - 1)$, so that row $IV(i_k)$ is the pivot equation and $a_{IV(ik),k}$ is the pivot element. Perform then a permutation of rows $IV(k)$ and $IV(i_k))$ in the last IV. Therefore one writes:

$$IV([k, i_k]) = IV([i_k, k])$$

As such, in case of effective row permutation, the Naive Gauss Elimination algorithm is modified as follows:

```
% The algorithm is column oriented
% At reduction k, a search is made in the kth column (in rows k to n)
% to find the maximum absolute value column element (p=max)
n=length(b);
for k=1:n-1
        [p,ik]=max(abs(A(k:n,k)));
        % Permutation of rows k and ik is then performed
        A([k ik])=A([ik k]);
        piv=A(k,k);
       .....................
```

If an index vector is referred to, the algorithm proceeds as follows.

```
function[U,c]=PartialPivotingGauss(A,b)
% An index vector is used to keep track of the location of the rows
n=length(b);
IV=1:n
%At reduction k, find the absolute value maximum column element and its position in I
for k=1:n-1
     [p, ik]=max(abs(A(IV(k:n),k));
% find the position of ik in last IV
     ik=ik + k - 1 ;
     % Permutation of rows k and ik is then performed through IV
     IV([k ik])=IV([ik k]);
     % Identify the pivots
      piv=A(IV(k),k);
% Find the multipliers
for i=k+1:n
       A(IV(i),k)=A(IV(i),k)/piv;
end
% Modify the body of matrix A and right hand side b
for j=k+1:n
       for i=k+1:n
               A(IV(i),j)=A(IV(i),j)-A(IV(i),k)*A(IV(k),j);
       end
end
for i=k+1:n
```

```
      b(IV(i))=b(IV(i))-A(IV(i),k)*b(IV(k));
end
%Extract U,c
c=b(IV);
U=triu(A(IV,:));
```

Example 3.3. *Solve the following system using Unscaled Partial Pivoting Gaussian reduction.*

$$(3.6) \qquad \begin{cases} 3x_1 & -13x_2 & +9x_3 & +3x_4 & = -19 \\ -6x_1 & +4x_2 & +x_3 & -18x_4 & = -34 \\ 6x_1 & -2x_2 & +2x_3 & +4x_4 & = 16 \\ 12x_1 & -8x_2 & +6x_3 & +10x_4 & = 26 \end{cases}$$

We first initialize the index vector of the system:

IV	1	2	3	4

The augmented matrix for the system above is:

$$\begin{pmatrix} 3 & -13 & 9 & 3 & -19 \\ -6 & 4 & 1 & -18 & -34 \\ 6 & -2 & 2 & 4 & 16 \\ 12 & -8 & 6 & 10 & 26 \end{pmatrix}$$

(a) **Reduction 1** Seek the pivot equation:

$$\max\{3, |-6|, 6, 12\} = 12.$$

First occurrence of the maximum is the 4^{th} one, i.e. at IV(4)=4 (meaning that at this stage, the 4^{th} component of IV is equation 4). So, one needs to perform the permutation of rows 1 and 4 through the index vector IV, the pivot equation becoming effectively equation 4 and the pivot element being 12. Updating the index vector, computing the multipliers $a_{IV(i),1}/12$, $i = 2, 3, 4$ and simultaneouslly modifying the body of matrix and right hand side leads to:

IV	4	2	3	1

$$\begin{pmatrix} \boxed{1/4} & -11 & 15/2 & 1/2 & -51/2 \\ \boxed{\text{-}1/2} & 0 & 4 & -13 & -21 \\ \boxed{1/2} & 2 & -1 & -1 & 3 \\ 12 & -8 & 6 & 10 & 26 \end{pmatrix}$$

(b) **Reduction 2** Similarly, one starts with a search for the pivot equation:

$$\max_{IV(2),IV(3),IV(4)} \{|a_{IV(2),2}|, |a_{IV(3),2}, |a_{IV(4),2}|\}$$
$$= \max\{|-11|, 0, 2\} = 11$$

The maximum 11 occurs at $IV(4) = 1$. Hence we perform the permutation of Equations $IV(2) = 2$ and $IV(4) = 1$. Thus, the pivot equation is row 1 and the pivot element is -11. Computing the multipliers and proceeding into the modifications of the remaining part of the augmented matrix leads to the following profile of the index vector and of the matrix data:

IV	4	1	3	2

$$\begin{pmatrix} \boxed{1/4} & -11 & 15/2 & 1/2 & -51/2 \\ \boxed{\text{-}1/2} & \boxed{0} & 4 & -13 & -21 \\ \boxed{1/2} & \boxed{\text{-}2/11} & 4/11 & -10/11 & -18/11 \\ 12 & -8 & 6 & 10 & 26 \end{pmatrix}$$

(c) **Reduction 3** In this last stage, seek the pivot equation:

$$\max_{IV(3),IV(4)} \{|a_{IV(3),3}|, |a_{IV(4),3}|\} = \max\{4, 4/11\} = 4.$$

The maximum 4 occurs at $IV(4) = 2$. Hence we perform the permutation of Equations $IV(4) = 2$ and $IV(3) = 3$. It is easily verified at the end of the process the contents of the data structure are as follows:

IV	4	1	2	3

$$\begin{pmatrix} \boxed{1/4} & -11 & 15/2 & 1/2 & -51/2 \\ \boxed{-1/2} & \boxed{0} & 4 & -13 & -21 \\ \boxed{1/2} & \boxed{-2/11} & \boxed{1/11} & 3/11 & 3/11 \\ 12 & -8 & 6 & 10 & 26 \end{pmatrix}$$

Obviously, back substitution yields:

$$x_4 = 1, x_3 = -2, x_2 = 1, x_1 = 3$$

Consider now the special case of a system of equations where the coefficients in a same row have a relatively large variation in magnitude. Gaussian elimination with Simple Partial Pivoting is not sufficient and could lead to incorrect solutions as shown in the following example.

Example 3.4. *Consider the following* 2×2 *system of equations, where* C *is a large positive number.*

$$\begin{cases} 3x_1 + Cx_2 = C \\ x_1 + x_2 = 3 \end{cases} \tag{3.7}$$

The exact solution to this problem in $\mathbb{R}$ is $x_1 \approx 2$ and $x_2 \approx 1$. Applying the Simple Partial Pivoting Gauss elimination, and since

$$\max\{3, 1\} = 3$$

the first row is the pivot equation, the pivot is 3 and the sole multiplier is $\frac{1}{3}$. This leads to:

$$\begin{cases} 3x_1 + Cx_2 = C \\ (1 - \frac{1}{3}C)x_2 = 3 - \frac{1}{3}C \end{cases}$$

where the back substitution procedure gives:

$$\begin{cases} x_2 = \frac{3-\frac{1}{3}C}{1-\frac{1}{3}C} \\ x_1 = \frac{C(1-x_2)}{3} \end{cases}$$

If these calculation are performed in a floating point system $\mathbb{F}$ with

finite fixed precision, and since C is large, then

$$\begin{cases} 3 - \frac{1}{3}C \approx -\frac{1}{3}C \\ 1 - \frac{1}{3}C \approx -\frac{1}{3}C \end{cases}$$

Therefore, the computed solutions would be:

$$x_2 \approx 1 \text{ and } x_1 \approx 0.$$

However scaling the rows first then selecting as pivot the scaled absolute value entry, improves the situation. The rowsscales vector being $S = [C, 1]$, to select the pivot equation, one would compute

$$\max\{\frac{3}{C}, \frac{1}{1}\} = 1$$

Consequently, in this example, the second row is selected as pivot equation. Now the pivot is 1 and the multiplier is 3. Carried out, the scaled partial pivoting Gauss reduction would lead to:

$$\begin{cases} (C-3)x_2 = (C-9) \\ x_1 + x_2 = 3 \end{cases}$$

Back substitution in this case would clearly give: $x_2 \approx 1$ and $x_1 \approx 2$. ■

In view of this example, a more elaborated version than the Simple Partial Pivoting would be the **Scaled Partial Pivoting**, where we set up a strategy that simulates a scaling of the row vectors and then selects as a pivot element the relatively largest scaled absolute value entry in a column. This process would in some way, load balance the entries of the matrix.
We formalize now this variation of Simple Pivoting strategies.

2. **Gaussian Elimination with Scaled Partial Pivoting**
In this strategy, scaled values are used to determine the best partial pivoting possible, particularly if there are large variations in magnitude of the elements within a row. Besides the index vector IV that is created to keep track of the equation-permutations of the system, a **scale factor** must be computed for each equation. We define the absolute value maximum element of each row s_i by:

$$s_i = \max_{1 \le j \le n} \{|a_{ij}|\} \; ; 1 \le i \le n$$

The column **scale vector** is therefore: $s = [s_1, s_2, ..., s_n]'$.
For example in starting the forward elimination process, we do not arbitrarily use the first equation as the pivot equation as in the Naive-Gauss elimination, nor do we select the row with maximum absolute value in the entries of the first column, as in the Simple Partial Pivoting strategy. Instead we scan first in column 1 the ratios

$$\left\{ \frac{|a_{i,1}|}{s_i} \ , i = 1, ..., n \right\}$$

and select the equation (or row) for which this ratio is greatest. Let i_1 be the 1^{st} index for which the ratio is greatest, then:

$$\frac{|a_{i_1,1}|}{s_{i_1}} = \max_{1 \le i \le n} \left\{ \frac{|a_{i,1}|}{s_i} \right\}$$

Interchange i_1 and 1 in the index vector only, which is now IV=$[i_1, i_2, ...i_n]$.
In a similar way, proceed next to further reduction steps. Notice that through this procedure, the scale factors are computed once. They are not changed after each pivot step as the additional amount of computations are not worthwhile.
We give now a version of the newly devised algorithm.

```
% Initialize IV and seek the scales
IV=1:n ;
for i=1:n
    s(i)=max(abs(A(i,1:n))
end
% Alternatively: s=(max(abs(A')))'
% At reduction k, find the absolute value of maximum scaled column element
for k=1:n-1
    [p, ik]=max(abs(A(IV(k:n),k) ./ s(IV(k:n)) ) ;
    ik=ik+k-1;
    IV([k  ik])= IV([ik  k]) ;
.........Same as  Partial Pivoting..............
```

As an illustration to the method, let us apply the Scaled Partial Pivoting Gaussian reduction on the system of equations of the preceding example.

Example 3.5.

We first set the index vector and evaluate the scales of the system:

IV	1	2	3	4

Augmented matrix					Scales
3	−13	9	3	−19	13
−6	4	1	−18	−34	18
6	−2	2	4	16	6
12	−8	6	10	26	12

(a) **Reduction 1** Seek the pivot equation:

$$\max\{3/13, 6/18, 1, 1\}.$$

First occurrence of the maximum is the 3rd one, i.e. at IV(3)=3 (meaning that the 3^{rd} component of IV is equation 3). So, one needs to perform the permutation of rows 1 and 3 through the index vector IV, the pivot equation becoming equation 3 and the pivot element being 6. Updating the index vector and computing the multipliers $a_{IV(i),1}/6, i = 2, 3, 4$ would yield:

IV	3	2	1	4

Augmented matrix					Scales
1/2	−13	9	3	−19	13
-1	4	1	−18	−34	18
6	−2	2	4	16	6
2	−8	6	10	26	12

Modifying the body of matrix and right hand side leads to:

Augmented matrix					Scales
1/2	−12	8	1	−27	13
-1	2	3	−14	−18	18
6	−2	2	4	16	6
2	−4	2	2	−6	12

(b) **Reduction 2** Similarly to reduction 1, one starts with a search for the pivot equation:

$$\max_{IV(2),IV(3),IV(4)}\{\frac{|a_{IV(2),2}|}{s_{IV(2)}}, \frac{|a_{IV(3),2}|}{s_{IV(3)}}, \frac{|a_{IV(4),2}|}{s_{IV(4)}}\} = \max\{2/18, 12/13, 4/12\}.$$

The maximum 12/13 occurs at $IV(3) = 1$. Hence we perform the permutation of Equations $IV(2) = 2$ and $IV(3) = 1$. Thus, the pivot equation is row 1 and the pivot element is -12. Computing the multipliers and proceeding into the modifications of the remaining part of the augmented matrix leads to the following profile of the index vector and of the matrix data:

IV	3	1	2	4

Augmented matrix					Scales
1/2	-12	8	1	-27	13
-1	-1/6	13/3	$-83/6$	$-45/2$	18
6	-2	2	4	16	6
2	1/3	$-2/3$	5/3	3	12

(c) **Reduction 3** This last step keeps the index vector unchanged since $\max\left\{|\frac{13}{3\times 18}|;|\frac{2}{3\times 12}|\right\} = \frac{13}{3\times 18}$. It is easily verified at the end of the process the contents of the data structure are as follows:

IV	3	1	2	4

Augmented matrix					Scales
1/2	-12	8	1	-27	13
-1	-1/6	13/3	$-83/6$	$-45/2$	18
6	-2	2	4	16	6
2	1/3	-2/13	$-6/13$	$-6/13$	12

Obviously, back substitution yields:

$$x_4 = 1,\ x_3 = -2,\ x_2 = 1,\ x_1 = 3.$$

3.5 LU Decomposition

A major by-Product of Gauss Elimination is the **decomposition** or **factorization** of a matrix A into the product of a **unit lower triangular matrix L** by an upper triangular one U. We will base our arguments on the systems of equations (3.3) and (3.6).

1. **First case : Naive Gauss**
   Going back to (3.3) and on the basis of the multipliers of Naive Gauss elimination, let L and U be respectively the unit lower and the upper triangular matrices of the process:

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 5 & \frac{13}{5} & 1 & 0 \\ 4 & \frac{6}{5} & \frac{1}{3} & 1 \end{pmatrix} \quad ; \quad U = \begin{pmatrix} 1 & -1 & 2 & 1 \\ 0 & 5 & -5 & 1 \\ 0 & 0 & 9 & -23/5 \\ 0 & 0 & 0 & -2/3 \end{pmatrix}$$

   Note that the product LU verifies:
   (3.8)

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 5 & \frac{13}{5} & 1 & 0 \\ 4 & \frac{6}{5} & \frac{1}{3} & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 & 2 & 1 \\ 0 & 5 & -5 & 1 \\ 0 & 0 & 9 & -\frac{23}{5} \\ 0 & 0 & 0 & -\frac{2}{3} \end{pmatrix} = \begin{pmatrix} 1 & -1 & 2 & 1 \\ 12 & -8 & 6 & 10 \\ 3 & 2 & 1 & 4 \\ 4 & 2 & 5 & 3 \end{pmatrix}$$

   which is precisely:

$$LU = A.$$

   This identity obeys to the following theorem ([6], [15]):

   Theorem 3.1. *Let $A \in \mathbb{R}^{n,n}$ be a square matrix verifying the principal minor property. If A is processed through Naive Gauss reduction, then A is factorized uniquely into the product of a unit lower triangular matrix L and an upper triangular matrix U associated to the reduction process, with*

$$A = LU$$

2. **Second case: Partial Pivoting**
   Consider now the Scaled Partial Pivoting reduction applied on (3.6). Based on the last status of $IV = [3, 1, 2, 4]$, we extract successively the unit lower and the upper triangular matrices of the process:

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1/2 & 1 & 0 & 0 \\ -1 & -1/6 & 1 & 0 \\ 2 & 1/3 & -2/13 & 1 \end{pmatrix} \quad ; \quad U = \begin{pmatrix} 6 & -2 & 2 & 4 \\ 0 & -12 & 8 & 1 \\ 0 & 0 & 13/3 & -83/6 \\ 0 & 0 & 0 & -6/13 \end{pmatrix}$$

Computing the product LU gives:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1/2 & 1 & 0 & 0 \\ -1 & -1/6 & 1 & 0 \\ 2 & 1/3 & -2/13 & 1 \end{pmatrix} \begin{pmatrix} 6 & -2 & 2 & 4 \\ 0 & -12 & 8 & 1 \\ 0 & 0 & 13/3 & -83/6 \\ 0 & 0 & 0 & -6/13 \end{pmatrix} = \begin{pmatrix} 6 & -2 & 2 & 4 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \\ 12 & -8 & 6 & 10 \end{pmatrix}$$

The product matrix is the matrix A up to a **permutation matrix** $P = P(IV)$, associated to the final status of the index vector . We write then

$$LU = P(IV)A$$

where P is defined as follows:

Definition 3.2. *Let $I \in \mathbb{R}^{n,n}$, be the identity matrix defined by its* **rows**, *i.e.*

$$I = \begin{pmatrix} e_1 \\ e_2 \\ ... \\ e_n \end{pmatrix}$$

Let $IV = [i_1, i_2, ..., i_n]$ be the last status of the index vector through the Partial Pivoting procedures. The permutation matrix P associated to IV is a permutation of the identity matrix I, and is given by the **row matrix:**

$$P = P(IV) = \begin{pmatrix} e_{i_1} \\ e_{i_2} \\ ... \\ e_{i_n} \end{pmatrix}$$

In example 3.5, the final status of $IV = [3, 1, 2, 4]$. Thus,

$$P = P(IV) = \begin{pmatrix} e_3 \\ e_1 \\ e_2 \\ e_4 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Note then that the product:

$$PA = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \\ 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \end{pmatrix}$$

is precisely the product LU found above. Hence the LU decomposition theorem which generalizes Theorem 3.1 stands as follows:

Theorem 3.2. *Let a square matrix $A \in \mathbb{R}^{n,n}$ be processed through partial pivoting Gauss reduction. If the unit lower triangular matrix L , the upper triangular matrix U and the index vector IV are extracted from the final status of the process then:*

$$P(IV)A = LU$$

where $P(IV)$ is the permutation matrix associated to the reduction process.

Note also that this decomposition of A is unique.

The LU decomposition or factorization of A is particularly helpful in computing the **determinant** of A, in solving different systems of equations $Ax = b$, where the coefficient matrix A is held constant, or also in computing the **inverse of A**.

3.5.1 Computing the Determinant of a Matrix

Clearly from theorems 3.1 and 3.2, we conclude respectively that in the first case

$$\det(A) = \det(L) \times \det(U)$$

while in the second case

$$\det(A) = (-1)^s \times \det(L) \times \det(U)$$

as $\det(P) = (-1)^s$, s being the number of permutations performed on IV through the Partial Pivoting procedures.
These results are stated hereafter:

Theorem 3.3. *(a) Under the hypothesis of Theorem 3.1,*

$$\det(A) = \prod_{i=1}^{n} u_{ii},$$

(b) Under the hypothesis of Theorem 3.2,

$$\det(A) = (-1)^s \prod_{i=1}^{n} u_{ii},$$

where u_{ii}, $, i = 1, ..., n$ are the diagonal elements of the upper triangular matrix U associated to the reduction process.

One easily verifies that in example 3.4

$$\det(A) = 1 \times 5 \times 9 \times \frac{2}{3} = 30$$

while in example 3.5:

$$det(A) = 6 \times (-12) \times 13/3 \times (-6/13) = 144$$

since $s = 2$.

3.5.2 Computing the Inverse of A

The LU decomposition of a matrix A is also useful in computing its inverse denoted by A^{-1} and verifying the property

$$AA^{-1} = I$$

where I is the identity matrix. Let c_j and e_j represent respectively the j^{th} column of A^{-1} and that of I, then one writes:

$$A[c_1\, c_2 \ldots c_n] = [e_1\, e_2 \ldots e_n] \tag{3.9}$$

1. **First case : Naive Gauss**
   Under the hypothesis of Theorem 1 and since $LU = A$, then (3.9) is equivalent to:
   $$LU[c_1\, c_2 \ldots c_n] = [e_1\, e_2 \ldots e_n]$$
   To obtain A^{-1} it is therefore enough to solve for c_j, in turn:
   $$LUc_j = e_j, \text{ for } j = 1, ..., n$$
   By letting $Uc_j = y$, one has then to solve successively the following 2 triangular systems:
   (i) The Lower triangular system $Ly = e_j$, and get the vector y by **Forward substitution**.
   (ii) The Upper triangular system $Uc_j = y$, and get thej^{th} column c_j by **Backward substitution**.

   Example 3.6. *Use the LU decomposition of A based on the Naive Gauss reduction applied to (3.3), to find the first column of A^{-1}*

Referring to Example 3.1, solving:
(i) The Lower triangular system $Ly = e_1$, gives $y = [1, -3, 14/5, 4/3]'$ by Forward substitution
(ii) The Upper triangular system $Uc_1 = y$, gives $c_1 = [158/45, -41/45, -32/45, -2]'$ by Backward substitution

2. **Second case : Partial Pivoting**
Under the hypothesis of Theorem 2 and since $LU = PA$, then (3.9) is equivalent to:

$$PAA^{-1} = P$$

or equivalently:

$$LU[c_1\, c_2 \ldots c_n] = [p_1\, p_2 \ldots p_n]$$

where p_j is the j^{th} column of P.
To obtain A^{-1} it is therefore enough to solve for c_j, in turn:

$$LUc_j = p_j, \text{ for } j = 1, ..., n$$

using the same 2 steps as in the first case above.

Remark 3.1. *Note that in Definition 2, the Permutation matrix P is defined in terms of its rows, while in the process of computing A^{-1}, one has first to identify the columns of P.*

Example 3.7. *Use the LU decomposition of A based on the Scaled Partial Pivoting reduction applied to (3.6), to find the last column of A^{-1}*

Referring to Example 3.3, solving:
(i) The Lower triangular system $Ly = p_4$, gives $y = [0, 0, 0, 1]'$ by Forward substitution
(ii) The Upper triangular system $Uc_1 = y$, gives $c_4 = [155/72, -115/24, -83/12, -13/6]'$ by Backward substitution

3.5.3 Solving Linear Systems using LU Factorization

Generalizing the method above, if the LU factorization of A is available, one can as well solve systems $Ax = v$ involving the same coefficient matrix A and varying the right hand side vector v. That is, one solves 2 triangular systems:

3.6 Exercises

1. Solve each of the following systems using Naive Gaussian elimination and Back substitution. Showing the multipliers at each stage. Carry four significant figures and round to the closest.

(a) $\begin{cases} 3x_1 + 4x_2 + 3x_3 = 10 \\ x_1 + 5x_2 - x_3 = 7 \\ 6x_1 + 3x_3 + 7x_3 = 15 \end{cases}$

(b) $\begin{cases} 3x_1 + 2x_2 - 5x_3 = 0 \\ 2x_1 - 3x_2 + x_3 = 0 \\ x_1 + 4x_2 - x_3 = 4 \end{cases}$

(c) $\begin{cases} 3x_1 + 2x_2 - x_3 = 7 \\ 5x_1 + 3x_2 + 2x_3 = 4 \\ -x_1 + x_2 - 3x_3 = -1 \end{cases}$

2. Apply the Naive Gauss elimination on the following matrices, showing the multipliers at each stage. Carry four significant figures and round to the closest.

(a) $\begin{bmatrix} 1 & 3 & 2 & 1 \\ 4 & 2 & 1 & 2 \\ 2 & 1 & 2 & 3 \\ 1 & 2 & 4 & 1 \end{bmatrix}$

(b) $\begin{bmatrix} 1 & -1 & 2 & 1 \\ 3 & 2 & 1 & 4 \\ 5 & 8 & 6 & 3 \\ 4 & 2 & 5 & 3 \end{bmatrix}$

3. Solve each of the following systems using Gaussian elimination with Unscaled Partial Pivoting and Back substitution. Write the index array and the multipliers at each step. Carry four significant figures and round to the closest.

(a) $\begin{cases} 3x_1 + 4x_2 + 3x_3 = 10 \\ x_1 + 5x_2 - x_3 = 7 \\ 6x_1 + 3x_3 + 7x_3 = 15 \end{cases}$

(b) $\begin{cases} 3x_1 + 2x_2 - 5x_3 = 0 \\ 2x_1 - 3x_2 + x_3 = 0 \\ x_1 + 4x_2 - x_3 = 4 \end{cases}$

(c) $\begin{cases} 3x_1 + 2x_2 - x_3 = 7 \\ 5x_1 + 3x_2 + 2x_3 = 4 \\ -x_1 + x_2 - 3x_3 = -1 \end{cases}$

4. Apply the Unscaled Partial Pivoting Gauss elimination on the following matrices, showing the multipliers and the Index vector at each stage. Carry four significant figures and round to the closest.

(a) $\begin{bmatrix} 1 & 3 & 2 & 1 \\ 4 & 2 & 1 & 2 \\ 2 & 1 & 2 & 3 \\ 1 & 2 & 4 & 1 \end{bmatrix}$

(b) $\begin{bmatrix} 1 & -1 & 2 & 1 \\ 3 & 2 & 1 & 4 \\ 5 & 8 & 6 & 3 \\ 4 & 2 & 5 & 3 \end{bmatrix}$

5. Solve each of the following systems using Gaussian Scaled Partial Pivoting and Back substitution. Write the index array, thescales vector and the multipliers at each step.Carry four significant figures and round to the closest.

(a) $\begin{cases} 2x_1 & -x_2 & +3x_3 & +7x_4 & = 15 \\ 4x_1 & +4x_2 & & +7x_4 & = 11 \\ 2x_1 & +x_2 & +x_3 & +3x_4 & = 7 \\ 6x_1 & +5x_2 & +4x_3 & +17x_4 & = 31 \end{cases}$

(b) $\begin{cases} 2x_1 + 4x_2 - 2x_3 = 6 \\ x_1 + 3x_2 + 4x_3 = -1 \\ 5x_1 + 2x_2 = 2 \end{cases}$

(c) $\begin{cases} -x_1 + x_2 - 3x_4 = 4 \\ x_1 + 3x_3 + x_4 = 0 \\ x_2 - x_3 - x_4 = 3 \\ 3x_1 + x_3 + 2x_4 = 1 \end{cases}$

6. Apply the Scaled Partial Pivoting Gauss elimination on the following matrices, showing the Index vector, thescales vector and the multipliers at each stage. Carry four significant figures and round to the closest.

(a) $\begin{bmatrix} 2 & 3 & -4 & 1 \\ 1 & -1 & 0 & -2 \\ 3 & 3 & 4 & 3 \\ 4 & 1 & 0 & 4 \end{bmatrix}$

(b) $\begin{bmatrix} 1 & 0 & 3 & 0 \\ 0 & 1 & 3 & -1 \\ 3 & -3 & 0 & 6 \\ 0 & 2 & 4 & -6 \end{bmatrix}$

(c) $\begin{bmatrix} 4 & 7 & 3 \\ 1 & 3 & 2 \\ 2 & -4 & -1 \end{bmatrix}$

(d) $\begin{bmatrix} 8 & -1 & 4 & 9 & 2 \\ 1 & 0 & 3 & 9 & 7 \\ -5 & 0 & 1 & 3 & 5 \\ 4 & 3 & 2 & 2 & 7 \\ 3 & 0 & 0 & 0 & 9 \end{bmatrix}$

7. Consider the following system of 2 equations in 2 unknowns:

$$(S) \begin{cases} 10^{-5}x + y = 7 \\ x + y = 1 \end{cases}$$

   (a) Find the exact solution of (S) in $\mathbb{R}$.

   (b) Use the Naive Gauss reduction to solve (S) in $F(10, 4, -25, +26)$ and compare the result with the exact solution.

   (c) Use the Partial Pivoting Gauss reduction to solve (S) in $F(10, 4, -25, +26)$ and compare the result with the exact solution.

8. Consider the following system of 2 equations in 2 unknowns:

$$(S) \begin{cases} 2x + 10^5 y = 10^5 \\ x + y = 3 \end{cases}$$

   (a) Find the exact solution of (S) in $\mathbb{R}$.

   (b) Use the simple Partial Pivoting Gauss reduction to solve (S) in $F(10, 4, -25, +26)$ and compare the result with the exact solution.

   (c) Use the ScaledPartial Pivoting Gauss reduction to solve (S) in $F(10, 4, -25, +26)$ and compare the result with the exact solution.

9. Based on the Naive Gauss reduction applied to each coefficient matrix A of Exercise 2:

   (a) Determine the Lower Unit triangular matrix L and the Upper unit triangular matrix U, then verify that A=LU.

   (b) Use the LU decomposition of A to compute the Determinant of A

   (c) Use the LU decomposition of A to determine the inverse of A

10. Based on the Unscaled Partial Pivoting Gauss reduction applied to each coefficient matrix A of Exercise 4:

    (a) Determine L: the Lower Unit triangular matrix, U: the Upper unit triangular matrix and P: the Permutation matrix, then verify that PA=LU.

    (b) Use the LU decomposition of A to compute the Determinant of A

    (c) Use the LU decomposition of A to determine the inverse of A

11. Based on the Scaled Partial Pivoting Gauss reduction applied to each coefficient matrix A of Exercise 6:

    (a) Determine L: the Lower Unit triangular matrix, U: the Upper unit triangular matrix and P: the Permutation matrix, then verify that PA=LU.

    (b) Use the LU decomposition of A to compute the Determinant of A

    (c) Use the LU decomposition of A to determine the inverse of A

12. Apply the Naive Gauss reduction on the coefficient matrix of exercise 2(b), then find the last row of the inverse of A.

13. Apply the Unscaled Partial Pivoting Gauss reduction on the coefficient matrix of exercise 4(b), then find the $2nd$ and $4th$ columns of the inverse of A.

14. Apply the Scaled Partial Pivoting Gauss reduction on the coefficient matrix of exercise 6(c), then find the last column of the inverse of A.

15. Apply the Naive Gauss reduction on the following strictly diagonally dominant band matrices. Determine at each reduction, the multipliers

and the elements of the matrix that are modified. Extract the Upper triangular matrix U and the Lower unit triangular matrix L obtained at the end of this process.

Definition 3.3. *A square matrix A of size $n \times n$ is* **strictly diagonally dominant** *if for every row, the magnitude of the diagonal entry is larger then the sum of the magnitude of all the other non diagonal entries in that row. i.e.*

$$|A(i,i)| > \sum_{j=1}^{n} |A(i,j)| \ \ ;\forall\ i = 1, 2, ..., n$$

As such, the Naive Gauss reduction is successfully applicable on the matrix.

(a) Let T_n be a triangular matrix, with

$$T_n = \begin{bmatrix} a1 & b1 & 0 & 0 & ... & 0 \\ c1 & a2 & b2 & 0 & ... & 0 \\ 0 & c2 & a3 & b3 & ... & 0 \\ .... & ... & ... & ... & ... & ... \\ .... & ... & ... & ... & ... & ... \\ .... & ... & ... & c_{n-2} & a_{n-1} & b_{n-1} \\ 0 & ... & 0 & 0 & c_{n-1} & a_n \end{bmatrix}$$

(b) Let Q_n be an upper quadridiagonal matrix, with

$$Q_n = \begin{bmatrix} a1 & b1 & d1 & 0 & ... & 0 \\ c1 & a2 & b2 & d2 & ... & 0 \\ 0 & c2 & a3 & b3 & ... & 0 \\ .... & ... & ... & ... & ... & ... \\ .... & ... & c_{n-3} & a_{n-2} & b_{n-2} & d_{n-2} \\ .... & ... & ... & c_{n-2} & a_{n-1} & b_{n-1} \\ 0 & ... & 0 & 0 & c_{n-1} & a_n \end{bmatrix}$$

(c) Let q_n be a lower quadridiagonal matrix, with

$$q_n = \begin{bmatrix} a1 & b1 & 0 & 0 & ... & 0 \\ c1 & a2 & b2 & 0 & ... & 0 \\ d1 & c2 & a3 & b3 & ... & 0 \\ .... & ... & ... & ... & ... & ... \\ .... & ... & ... & ... & ... & ... \\ .... & ... & d_{n-3} & c_{n-2} & a_{n-1} & b_{n-1} \\ 0 & ... & 0 & d_{n-2} & c_{n-1} & a_n \end{bmatrix}$$

16. Consider the following 5×5 strictly diagonally dominant lower Hessenberg matrix

$$A = \begin{pmatrix} 4 & 1 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 \\ 1 & 1 & 4 & 1 & 0 \\ 1 & 1 & 1 & 4 & 1 \\ 1 & 1 & 1 & 1 & 4 \end{pmatrix}$$

1- Apply the Naive Gauss reduction on the matrix A showing the status of that matrix after each elimination, then extract out of this process, the Upper triangular matrix U and the Unit Lower triangular matrix P.
2- Check that at each reduction, the multipliers reduce to one value, and at each reduction except the last, the modified elements reduce to two values, in addition to the diagonal element at last reduction. Compute the total number of flops needed for the LU-decomposition of the matrix A.
3- Deduce the total number of flops needed for the LU-decomposition of the $(n \times n)$ diagonally dominant lower Hessenberg matrix B where c is a constant and

$$B = \begin{pmatrix} c & 1 & 0 & 0 & . & . & . & 0 \\ 1 & c & 1 & 0 & . & . & . & 0 \\ 1 & 1 & c & 1 & 0 & . & . & 0 \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ 1 & 1 & 1 & . & . & 1 & c & 1 \\ 1 & 1 & 1 & 1 & . & . & 1 & c \end{pmatrix}$$

Express your answer in terms of n.

3.7 Computer Projects

Exercise 1: Naive Gauss for Special Pentadiagonal Matrices

Definition 3.4. *A* **PentaDiagonal matrix** *A is a square matrix with 5 non zero diagonals: the main diagonal* **d**, *2 Upper subdiagonals* **u** *and* **v**, *and 2 Lower subdiagonals* **l** *and* **m**.

$$A = \begin{bmatrix} d(1) & u(1) & v(1) & 0 & 0 & . & . & 0 \\ l(1) & d(2) & u(2) & v(2) & 0 & . & . & 0 \\ m(1) & l(2) & d(3) & u(3) & v(3) & . & . & 0 \\ 0 & m(2) & l(3) & d(4) & u(4) & . & . & 0 \\ 0 & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ . & . & 0 & m(n-4) & l(n-3) & d(n-2) & u(n-2) & v(n-2) \\ 0 & . & . & 0 & m(n-3) & l(n-2) & d(n-1) & u(n-1) \\ 0 & 0 & . & . & 0 & m(n-2) & l(n-1) & d(n) \end{bmatrix}$$

Definition 3.5. *A PentaDiagonal matrix A is* **strictly diagonally dominant** *if for every row, the magnitude of the diagonal entry is larger then the sum of the magnitude of all the other non diagonal entries in that row.*

$$|d(i)| > |u(i)| + |v(i)| + |l(i-1)| + |m(i-2)| \ ; \forall \ i = 1, 2, ..., n$$

(As such, the matrix A will satisfy the Principal minor property, and the NAIVE GAUSS reduction is successfully applicable on A.)

Let A be a **strictly diagonally dominant pentadiagonal** matrix..

1. Write a MATLAB function

   function [m1,l1,d1,u1,v1]=NaiveGaussPenta(m,l,d,u,v, tol)

   which takes as input 5 column vectors m, l, d, u and v representing the 5 diagonals of A, and some tolerance tol. At each reduction, if the absolute value of the pivot element is less then tol an error message should be displayed, otherwise this function performs Naive Gauss reduction on the matrix A and returns through the process, the 5 modified diagonals m1, l1, d1, u1 and v1.

Your function should neither use the built in MATLAB function that factorizes A into L and U nor use the general code for Naive Gauss reduction. Your code should be designed for **pentadiagonal matrices only** and should use the least number of flops.

2. Write a MATLAB **function x = RowForwardPenta(d, l, m, c)** which takes as input 3 column vectors representing the main diagonal d and 2 lower diagonals l and m of an invertible Lower triangular matrix L and a column vector c. This function performs row-oriented Forward substitution to solve the system Lx=c, using the least number of flops. Your code should be designed for **pentadiagonal matrices only**.

3. Write a MATLAB **function x = RowBackwardPenta(d, u, v, c)** which takes as input 3 vectors column representing the main diagonal d and 2 upper diagonals u and v of an invertible Upper triangular matrix U and a column vector c. This function performs row-oriented Backward substitution to solve the system Ux=c, using the least number of flops. Your code should be designed for **pentadiagonal matrices only**.

4. Write a MATLAB **function B = InversePenta(m, l, d, u, v, tol)** which takes as input the 5 diagonals of the **pentadiagonal** matrix A and outputs B, the inverse of the matrix A.
   Your function should call for the previous functions programmed in parts 1,2 and 3.

5. Write a MATLAB **function T =InverseTransposePenta(m, l, d, u, v, tol)** which takes as input the 5 diagonals of the **pentadiagonal** matrix A and outputs $T = (A^t)^{-1}$, the inverse of the transpose of A.
   Your function should be based on the LU-decomposition of A , and should call for the functions programmed in parts 1, 2 and 3.

   Hint: If $A = LU$, then:

   - $A^t = (LU)^t = U^t L^t$
   - Since $A^t T = I$, then

$$A^t[c_1, c_2, ..., c_n] = U^t L^t[c_1, c_2, ..., c_n] = [e_1, e_2, ..., e_n]$$

$$\Leftrightarrow U^t L^t[c_i] = [e_i], \text{ for } i = 1, 2, ..., n$$

where c_i is the i^{th} column of T and e_i is the i^{th} column of the Identity matrix I.

6. Test each of your functions on 3 different **strictly diagonally dominant pentadiagonal matrices** with $n \geq 5$. (In one of the test cases, choose one of the Pivot elements smaller than tol). Save your inputs and outputs in a word document.

Exercise 2: Naive Gauss Reduction on Upper Hessenberg matrices. A Hessenberg matrix is a special kind of square matrix, one that is "almost" triangular. To be exact, an Upper Hessenberg matrix has zero entries below the first sub-diagonal.

$$H = \begin{bmatrix} H(1,1) & H(1,2) & H(1,3) & . & . & H(1,n) \\ H(2,1) & H(2,2) & H(2,3) & H(2,4) & . & H(2,n) \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ 0 & . & 0 & H(n-1,n-2) & H(n-1,n-1) & H(n-1,n) \\ 0 & . & . & 0 & H(n,n-1) & H(n,n) \end{bmatrix}$$

Definition: An upper Hessenberg matrix H is strictly diagonally dominant if for every row, the magnitude of the diagonal entryis larger than the sum of the magnitude of all the other non diagonal entries in that row.

$$|H(i,i) > |H(i,i-1)| + |H(i,i+1)| + ... + |H(i,n)| \, \forall \, i = 1, 2, ..., n$$

(As such, the matrix H will satisfy the Principal minor property, and the Naive Gauss reduction is successfully applicable on H.)

Le H be a strictly diagonally dominant Upper Hessenberg matrix.

1. Write a MATLAB **function [L, U] = NaiveGaussUHessenberg(H)** that takes as input an $n \times n$ strictly diagonally dominant upper Hessenberg matrix H. This function performs Naive Gauss reduction on the matrix H and returns at the end of the process, the Upper and Unit Lower triangular matrices U and L.
   Your function should neither use the built in MATLAB function that factorizes A into L and U, nor use the general code for Naive Gauss reduction. Your code should be designed for upper Hessenberg matrices only, and should use the least number of flops.

2. Write a MATLAB **function [x] = RowForwardUHessenberg(L, c)** that takes as input an invertible bi-diagonal lower triangular square matrix L of size n (displayed below) and a column vector c of length n. This function performs row-oriented Forward substitution to solve the system Lx=c, using the least number of flops. Your code should be designed for bi-diagonal lower triangular matrices only and should use the least number of flops.

$$L = \begin{bmatrix} L(1,1) & 0 & . & . & . & . & . & 0 \\ L(2,1) & L(2,2) & 0 & 0 & . & . & . & 0 \\ 0 & L(3,2) & L(3,3) & 0 & .. & . & . & 0 \\ 0 & 0 & L(4,3) & L(4,4) & 0 & .) & . & 0 \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ 0 & . & . & . & 0 & L(n-1,n-2) & L(n-1,n-1) & 0 \\ 0 & 0 & . & . & . & 0 & L(n,n-1) & L(n,n) \end{bmatrix}$$

3. Write a MATLAB **function [B] = InverseUHessenberg(H)** that takes as input an invertible upper Hessenberg matrix H, and outputs B, the inverse of H, using the LU-decomposition of H. Your function should call for the previous functions programmed in parts 1 and 2.

4. Test each of your functions above for 2 different upper Hessenberg strictly diagonally dominant matrices, with $n \geq 5$, and save the results in a word document.

Call for previous functions when needed.
Do not check validity of the inputs.

Hint: To construct an $n \times n$ upper Hessenberg strictly diagonally dominant matrix H, proceed as follows. Let:

(a) A = rand(n)
(b) m=max(sum(A))
(c) m1=max(sum(A'))
(d) s=max(m, m1)
(e) B=A + s*eye(n)
(f) H=triu(B, -1)

Exercise 3: Naive Gauss on Arrow Matrices
An Arrow matrix is a special kind of square sparse matrix, in which there is a tridiagonal banded portion, with a column at one side and a row at the bottom.

$$A = \begin{bmatrix} d(1) & u(1) & 0 & . & . & . & . & 0 & c(1) \\ l(1) & d(2) & u(2) & 0 & . & . & . & 0 & c(2) \\ 0 & l(2) & d(3) & u(3) & 0. & . & . & 0 & c(3) \\ 0 & 0 & l(3) & d(4) & u(4) & 0 & . & 0 & c(4) \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ 0 & . & . & 0 & l(n-3) & d(n-2) & u(n-2) & & c(n-2) \\ 0 & . & . & 0 & 0 & l(n-2) & d(n-1) & & c(n-1) \\ r(1) & r(2) & r(3) & r(4) & . & . & r(n-1) & & d(n) \end{bmatrix}$$

Definition: An Arrow matrix A is strictly diagonally dominant if for every row, the magnitude of the diagonal entry is larger than the sum of the magnitude of all the other non diagonal entries in that row. i.e.

$$|d(n-1) > |l(i-1)| + |u(i+1)| + |c(i)| \,\forall\, i = 1, 2, ..., n-2$$

$|d(n-1)| > |l(n-2)| + |c(n-1)|$ and $|d(n)| > |r(1)| + |r(2)| + ... + |r(n-1)|$

(As such, the matrix A will satisfy the Principal minor property, and the Naive Gauss reduction is successfully applicable on A, without need for pivoting.)

Le A be a strictly diagonally dominant Arrow matrix where:
- $d = [d(1), ..., d(n)]$ is a vector of length (n) representing the main diagonal of A.
- $u = [u(1), ..., u(n-2)]$ is a vector of length (n-2), and $[u(1), ..., u(n-2), c(n-1)]$ represents the first Upper diagonal of A.
- $l = [l(1), ..., l(n-2)]$ is a vector of length (n-2), and $[l(1), ..., l(n-2), r(n-1)]$ represents the first Lower diagonal of A.
- $c = [c(1), ..., c(n-1)]$ is a vector of length (n-1), and $c = [c(1), ..., c(n-1), d(n)]$ represents the last column of A.
- $r = [r(1), ..., r(n-1)]$ is a vector of length (n-1), and $r = [r(1), ..., r(n-1), d(n)]$ represents the last row of A.

1. Write a MATLAB **function [d1,u1,l1,c1,r1]=NaiveGaussArrow(d,u,l,c,r)**

that takes as input the 5 vectors defined above representing A. This function performs Naive Gauss reduction on the matrix A and returns at the end of the process, the modified vectors : d1, u1, l1, c1, r1 (including the multipliers) .
Your function should neither use the built in MATLAB function that factorizes A into L and U, nor use the general code for Naive Gauss reduction. Your code should be designed for Arrow matrices only, and should use the least number of flops.

2. Write a MATLAB **function[x]=RowBackwardArrow(d,u,c,b)** that takes as input 3 vectors as defined above, representing an invertible nearly bi-diagonal upper triangular square matrix U of size n (displayed below) and a column vector b of length n. This function performs row-oriented Backrward substitution to solve the system Ux=b, using the least number of flops. Your code should be designed for nearly bi-diagonal upper triangular matrices only and should use the least number of flops.

$$U = \begin{bmatrix} d(1) & u(1) & 0 & . & . & . & 0 & c(1) \\ 0 & d(2) & u(2) & 0 & . & . & 0 & c(2) \\ 0 & 0 & d(3) & u(3) & 0 & . & 0 & c(3) \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ 0 & . & . & 0 & d(n-2) & & u(n-2) & c(n-2) \\ 0 & . & . & . & . & 0 & d(n-1) & c(n-1) \\ 0 & . & . & . & . & . & 0 & d(n) \end{bmatrix}$$

3. Write a MATLAB **function [x] = RowForwardArrow(d, l, r, b)** that takes as input 3 vectors as defined above, representing an invertible nearly bi-diagonal lower triangular square matrix L of size n (displayed below) and a column vector b of length n. This function performs row-oriented Forward substitution to solve the system Lx=b, using the least number of flops. Your code should be designed for nearly bi-diagonal lower triangular matrices only and should use the least number of flops.

$$L = \begin{bmatrix} d(1) & 0 & . & . & . & . & . & 0 \\ l(1) & d(2) & 0 & 0 & . & . & . & 0 \\ 0 & l(2) & d(3) & 0 & . & . & . & 0 \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ 0 & . & . & . & 0 & l(n-2) & d(n-1) & 0 \\ r(1) & r(2) & . & . & . & . & r(n-1) & d(n) \end{bmatrix}$$

4. Write a MATLAB **function [B] = InverseArrow((d, u, l, c, r)** that takes as input the 5 vectors defined above representing an invertible Arrow matrix A, and outputs B, the inverse of A, using the LU-decomposition of A. Your function should call for the previous functions programmed in parts 1, 2 and 3.

5. Test each of your functions above for 2 different Arrow strictly diagonally dominant matrices A, with $n \geq 6$, and save the results in a word document.

   Call for previous functions when needed.
   Do not check validity of the inputs.