Chapter 2

Linear Systems

2.1 Introduction

To solve the linear system $Ax = b$ one can try several different algorithms. One is to find the inverse of A and multiply both sides with it; this is computationally very expensive. Another approach is to take a guess at the solution and iterate refining your guess until the error you make is suitably small; we will study iterative methods later on. For now we will use so-called direct methods which decompose A into pieces each of which is easy to invert. The first such method we will study is the LU-Decomposition. The idea is to factor $A = LU$ where L is lower-triangular and U is upper-triangular. Then you solve the pair of equations $Lz = b$ and then $Ux = z$. The difficulty in this arrangement is to perform the factorization: What are L and U in terms if the entries of A?

2.1.1 Addition and multiplication

Take $N \times N$ matrices A, B, C where $A = B + C$. In F77, this is usually written as:

F77

```
      dimension a(n,n),b(n,n),c(n,n)
      do 20 j=1,n
          do 10 i=1,n
              a(i,j)=b(i,j)+c(i,j)
10        continue
20    continue
```

In F90/95 it is simpler:

F90

```
real, dimension(n,n) :: a, b, c
a = b + c
```

although in both cases N^2 operations are performed.

Things are a bit different when performing matrix multiplications. In F77, the $A = BC$ matrix product can be accomplished with the following subroutine.

F77

```
      subroutine matmult(l,u,n,a)
          dimension l(n,n), u(n,n), a(n,n)
          do 100 i=1,n
              do 100 j=1,n
                  a(i,j)=0.
                  do 100 k=1,n
100                   a(i,j) = a(i,j) + l(i,k)*u(k,j)
    return
    end
```

Using this type of loop structure is not a good idea when Fortran 90 is available, because you have a matrix multiplication available that is almost always implemented more efficiently than any compilation of your Fortran code will ever be. F90 has an intrinsic function called '`matmul`', so that the above 3 loops are coded in one single statement:

F90

```
a = matmul(l,u)
```

Time savings associated with things like matrix multiplication can make a big difference in many applications. If you look at the Fortran matrix multiply above you will see that the operation takes N^3 multiplications and the same number of additions. You do not have to get to very large matrices or very frequent use of the matrix multiply before you are looking at significant savings in computer time by using an optimally coded function like '`matmul`'.

2.2 Gaussian elimination and LU decomposition

Whether you know it or not, you have used *Gaussian elimination* to solve systems of linear equations. What you probably never considered is that the method can be approached in a very systematic way, permitting implementation in a computer program. Also of importance is the fact that with very minimal additional effort, the program for *Gaussian*

elimination can be enhanced to perform *Lower-Upper matrix factorization* (write any non-singular matrix as a product of a lower triangular and an upper triangular matrix).

2.2.1 Motivation

To solve the following system equation:

$$\begin{aligned} x + y + z &= 2 && (2.1)\\ 2x + 3y + z &= 1 && (2.2)\\ 2x + y + 2z &= 5 && (2.3)\\ & && (2.4) \end{aligned}$$

we first write in arrays as:

$$\left(\begin{array}{ccc|c} 1 & 1 & 1 & 2 \\ 2 & 3 & 1 & 1 \\ 2 & 1 & 2 & 5 \end{array}\right)$$

multiply row 1 by (-2) and add it to row 2:

$$\left(\begin{array}{ccc|c} 1 & 1 & 1 & 2 \\ 0 & 1 & -1 & -3 \\ 2 & 1 & 2 & 5 \end{array}\right)$$

multiply row1 by (-2) and add it to row 3:

$$\left(\begin{array}{ccc|c} 1 & 1 & 1 & 2 \\ 0 & 1 & -1 & -3 \\ 0 & -1 & 0 & 1 \end{array}\right)$$

multiply row 2 by 1 and add it to row 3:

$$\left(\begin{array}{ccc|c} 1 & 1 & 1 & 2 \\ 0 & 1 & -1 & -3 \\ 0 & 0 & -1 & -2 \end{array}\right)$$

So, we have

$$\begin{aligned} x + y + z &= 2 && (2.5)\\ y - z &= -3 && (2.6)\\ -z &= -2 && (2.7)\\ & && (2.8) \end{aligned}$$

And from the last equation, *back substitution* can be used to find the solution to z, y, and x. The principle of Gauss elimination is to eliminate the coefficients that are below the diagonal of the matrix.

In general, the approach is:

Starting with $Ax = b$ define the matrix (A, b), then

- multiply a row by a non-zero real number d,
- swap two rows,
- add [d times one row] to another one.
- finally, the original equation is transformed to $Ux = c$ with an upper triangular matrix U, from which the unknowns x can be found by back substitution.

The corresponding algorithm has a general form of a triple-nested loop

```
for i = ...
  for j = ...
    for k = ...
      a_ij = a_ij − (a_ik / a_kk) a_kj
    end
  end
end
```

2.2.2 The L and U matrices

Any non-singular matrix A can be expressed as a product $A = LU$, with uniquely determined lower triangular matrix L and upper triangular matrix U of the form:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{pmatrix} \tag{2.9}$$

(i.e. the diagonal elements of L are all 1). Writing out the equations in the 1st column gives us

$$a_{11} = u_{11} \tag{2.10}$$
$$a_{21} = l_{21}u_{11} \tag{2.11}$$
$$a_{31} = l_{31}u_{11} \tag{2.12}$$
$$a_{41} = l_{41}u_{11}\,, \tag{2.13}$$

which determines the elements u_{11}, l_{21}, l_{31} and l_{41}. The equations in the second column imply

$$a_{12} = u_{12} \tag{2.14}$$
$$a_{22} = l_{21}u_{12} + u_{22} \tag{2.15}$$
$$a_{32} = l_{31}u_{12} + l_{32}u_{22} \tag{2.16}$$
$$a_{42} = l_{41}u_{12} + l_{42}u_{22}\,, \tag{2.17}$$

which gives us u_{12}, etc. This is done using the so-called *Crout's Algorithm* (see § 2.7.3).

2.2.3 Backward and forward substitution

The LU decomposition can be performed in a way similar to Gaussian elimination. It is useful, e.g. for the solution of the exactly determined system of linear equations $Ax = b$, when there is more than one right-hand side b.

With $A = LU$, the system becomes

$$LUx = b \tag{2.18}$$

or

$$Ly = b \qquad \text{and} \qquad Ux = y\ . \tag{2.19}$$

y can be computed by forward substitution and x by backward substitution.

Specifically, $Ly = b$ gives us the y as

$$\begin{aligned} y_1 &= b_1 && (2.20)\\ l_{21}y_1 + y_2 &= b_2 && (2.21)\\ l_{31}y_1 + l_{32}y_2 + y_3 &= b_3 && (2.22)\\ l_{41}y_1 + l_{42}y_2 + l_{43}y_3 + y_4 &= b_4 && (2.23) \end{aligned}$$

and then $Ux = y$ gives us the solution x from

$$\begin{aligned} u_{11}x_1 + u_{12}x_2 + u_{13}x_3 + u_{14}x_4 &= y_1 && (2.24)\\ u_{22}x_2 + u_{23}x_3 + u_{24}x_4 &= y_2 && (2.25)\\ u_{33}x_3 + u_{34}x_4 &= y_3 && (2.26)\\ u_{44}x_4 &= y_4 && (2.27) \end{aligned}$$

Notes:

- The programs which performs the above described LU decomposition are described in the Appendix.
- In general, since the determinant of L is 1 (by construction, all diagonal elements are equal to 1), y can simply be found as $y = L^{-1}b$.
- Sometimes, LU decomposition requires pivoting to avoid division by zero or small numbers (see § 2.3).
- For N equations with N unknowns Gauss elimination, or determining L and U, takes N^3 computer operations (multiplies and adds). However, once L and U are known, solving the system $LUx = b$ only takes of the order of N^2 operations. Now imagine what it means when you are dealing with matrices with $N = 1000$.

2.2.4 Using Gaussian elimination in LU decomposition

Gaussian elimination can be used to decompose A into L and U.

To illustrate this let us perform the LU decomposition based on the Gauss elimination on

$$A = \begin{pmatrix} 3 & -0.1 & -0.2 \\ 0.1 & 7.00 & -0.3 \\ 0.3 & -0.2 & 10 \end{pmatrix} \tag{2.28}$$

Solution:

$$U = \begin{pmatrix} 3 & -0.1 & -0.2 \\ 0 & 7.00333 & -0.29333 \\ 0 & 0 & 10.0120 \end{pmatrix} \tag{2.29}$$

The factors used were:

$$f_{21} = 0.1 \ / \ 3 = 0.03333 \tag{2.30}$$

$$f_{31} = 0.3 \ / \ 3 = 0.1 \tag{2.31}$$

$$f_{32} = -0.19 \ / \ 7.0033 = -0.02713 \tag{2.32}$$

Thus

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.033333 & 1 & 0 \\ 0.1 & -0.02713 & 1 \end{pmatrix} \tag{2.33}$$

$$A = LU = \begin{pmatrix} 1 & 0 & 0 \\ 0.033333 & 1 & 0 \\ 0.1 & -0.02713 & 1 \end{pmatrix} \begin{pmatrix} 3 & -0.1 & -0.2 \\ 0 & 7.00333 & -0.29333 \\ 0 & 0 & 10.0120 \end{pmatrix} \tag{2.34}$$

The result in LU form is then:

$$\begin{pmatrix} 3 & -0.1 & -0.2 \\ 0.099999 & 7 & -0.3 \\ 0.3 & -0.2 & 9.9996 \end{pmatrix} \tag{2.35}$$

The small difference is due to round off error which can be dealt with using *refinement techniques* described in § 2.4.

2.3 Pivoting

You should be aware that there are linear systems where growth of roundoff error can not be avoided with Gaussian Elimination or LU decomposition.

If row exchanges (partial pivoting) are necessary, we have to introduce a permutation matrix P . Instead of A one then decomposes PA:

Example1:

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \quad \rightarrow \text{no LU decomposition exists} \tag{2.36}$$

We can decompose this matrix with pivoting through permutation matrix (see § 2.7.1).

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad \rightarrow \text{This matrix has LU decomposition} \tag{2.37}$$

The first matrix (to the left) is a *permutation matrix* (see § 2.7.1).

Example2: Factorizing the following matrix without pivoting:

$$A = \begin{pmatrix} \epsilon/4 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 4/\epsilon & 1 \end{pmatrix}\begin{pmatrix} \epsilon/4 & 1 \\ 0 & -4/\epsilon + 1 \end{pmatrix} \approx \begin{pmatrix} 1 & 0 \\ 4/\epsilon & 1 \end{pmatrix}\begin{pmatrix} \epsilon/4 & 1 \\ 0 & -4/\epsilon \end{pmatrix} \tag{2.38}$$

would end up with very large residual, $r = b - Ax$ if ϵ is small. With pivoting however,

$$PA = \begin{pmatrix} 1 & 1 \\ \epsilon/4 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \epsilon/4 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 0 & 1 - \epsilon/4 \end{pmatrix} \approx \begin{pmatrix} 1 & 0 \\ \epsilon/4 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \tag{2.39}$$

and the residual would be small

In summary, pivoting is achieved by ensuring that the diagonal element of the row under consideration is the largest available. This means that before each row operation, the column below is searched and the row containing the largest element is swapped with the current row. A record must be kept of the new permutation so that the right-hand side elements can be related to the correct rows.

2.4 Iterative refinement

When using the LU solver, it is inevitable that the "answer" will not be exact, only approximate, i.e. LU will not be identical to A. There is a method to refine the answer (track and minimize the errors), based on the so-called "backwards error analysis".

Assume x is the exact solution of $Ax = b$, and (due to round-off error) we have only computed an approximation $x + \delta x$. Then

$$A(x + \delta x) = b + A\delta x \tag{2.40}$$

$$A\,\delta x = \delta b = A(x + \delta x) - b\,. \tag{2.41}$$

We know the right-hand side of Eq. (2.41); it is the residual based on the approximate solution. Notice, however, that one needs both the original matrix A and the factored form. That is, to refine, we must keep both A and LU. This done, the procedure is straightforward.

2.4.1 Example

Take a 3×3 matrix A with vector b which lead to the solution

$$a_{11}\tilde{x}_1 + a_{12}\tilde{x}_2 + a_{13}\tilde{x}_3 = \tilde{b}_1 \,, \tag{2.42}$$
$$a_{21}\tilde{x}_1 + a_{22}\tilde{x}_2 + a_{23}\tilde{x}_3 = \tilde{b}_2 \,, \tag{2.43}$$
$$a_{31}\tilde{x}_1 + a_{32}\tilde{x}_2 + a_{33}\tilde{x}_3 = \tilde{b}_3 \,, \tag{2.44}$$

where $\tilde{x}$ is the approximate solution which gives $A\tilde{x} = \tilde{b}$. The error δx can be written as

$$x_1 = \tilde{x}_1 + \delta x_1 \,, \tag{2.45}$$
$$x_2 = \tilde{x}_2 + \delta x_2 \,, \tag{2.46}$$
$$x_3 = \tilde{x}_3 + \delta x_3 \,, \tag{2.47}$$

which implies

$$a_{11}\delta x_1 + a_{12}\delta x_2 + a_{13}\delta x_3 = b_1 - \tilde{b}_1 = E_1 \tag{2.48}$$
$$a_{21}\delta x_1 + a_{22}\delta x_2 + a_{23}\delta x_3 = b_2 - \tilde{b}_2 = E_2 \tag{2.49}$$
$$a_{31}\delta x_1 + a_{32}\delta x_2 + a_{33}\delta x_3 = b_3 - \tilde{b}_3 = E_3 \,. \tag{2.50}$$

Refinement consists in minimizing the error E by multiple iterations. That is,

1. solve the equations for the correction δx_i;
2. add the correction to the approximate solution;
3. stop when the desired precision is reached (see the algorithm in § 2.7.2).

Notes:

- In general, one or two calls to the refinement code are enough to provide a good solution.
- Iterative refinement doubles the required storage, since both, the original matrix, and the LU factorization are required.
- A small relative residual does not necessarily imply that the computed solution is close to the "true" solution unless the system is well-conditioned.[1]

2.4.2 Example

Consider the linear system

$$\begin{pmatrix} 2.2 & 1.4 \\ 0.8 & 1.0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.4 \\ -0.1 \end{pmatrix} \,, \tag{2.51}$$

[1] A *well-conditioned* matrix is a matrix whose determinant is not close to zero [quantifying what "close to zero" means in this context is very technical]. Such matrices can be stably inverted. An *ill-conditioned* matrix, on the other hand, has a determinant close to zero and creates all sorts of problem when you try to invert it or solve a system involving it.

the exact solution of which is [0.5, −0.5]. Under 2-digit truncating arithmetic, use Gaussian Elimination and LU decomposition to solve the system (your answer should be $[x_1, x_2] = [0.45, -0.48]$). Write a simple program to perform iterative refinement (still using 2-digit truncating arithmetic). How many iterations are required before reaching the desired precision?

2.4.3 Refinement using singular matrices

Take the following system (using 3-digit decimal arithmetic),

$$\begin{pmatrix} 0.641 & 0.242 \\ 0.321 & 0.121 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.883 \\ 0.442 \end{pmatrix} \tag{2.52}$$

Gauss elimination yields the triangular system

$$\begin{pmatrix} 0.641 & 0.242 \\ 0.0 & 0.000242 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.883 \\ -0.000383 \end{pmatrix} \tag{2.53}$$

and back-substitution then gives the solution

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.782 \\ 1.58 \end{pmatrix} \tag{2.54}$$

So the exact residual for this solution is

$$r = b - Ax = \begin{pmatrix} -0.000622 \\ -0.000202 \end{pmatrix}, \tag{2.55}$$

which is as small as can one would expect using 3-digit arithmetic. But the exact solution is

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \tag{2.56}$$

so the error is almost as large as the solution.

How can we explain this phenomenon?

The cause is due to the fact that matrix A is *nearly singular*. The division that determines x_2 is between two quantities that are both on the order of the rounding error, and hence the result is essentially arbitrary. When this arbitrary value of x_2 is substituted into the first equation, the value for x_1 is computed so that the first equation is satisfied, yielding a small residual, but poor solution!

Is there a way out? Not really. In the given example, we can do the calculation in higher precision, and sometimes you will need quadruple precision in order to get a meaningful solution. But if we only know the system (2.52) up to 3 decimals, the original problem is ill-defined, because even if we can use arbitrary precision, we will get completely different

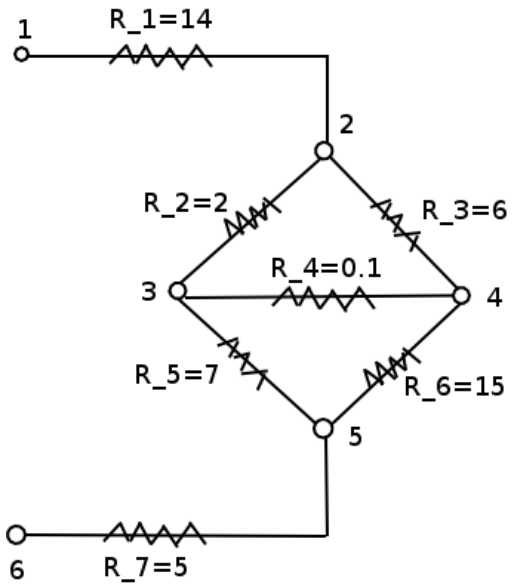

Figure 2.1: LU decomposition in electric circuits

results, depending on whether e.g. the first matrix element is 0.6406 or 0.6414, both of which will be represented as 0.641 in 3 decimal arithmetic.[2]

2.5 Application to electric circuits

For the electric circuit shown in Fig. 2.1, suppose we want to know the current that will flow between points 3 and 4 if a voltage of 5 V is applied between 1 and 6. To do so we use existing laws such as Kirchhoff's law ("The sum of all currents flowing into a node is zero") and Ohm's law ("The current through a resistor equals the voltage across it divided by its resistance") to find the answer. We have 11 unknowns, namely, $i_{12}, i_{23}, i_{24}, i_{34}, i_{45}, i_{56}, V_2, V_3, V_4, V_5$ and 11 equations.

Kirchhoff's law gives us

$$\text{Node 2:} \qquad i_{12} - i_{23} - i_{24} = 0\,, \tag{2.57}$$
$$\text{Node 3:} \qquad i_{23} - i_{34} - i_{35} = 0\,, \tag{2.58}$$
$$\text{Node 4:} \qquad i_{24} + i_{34} - i_{45} = 0\,, \tag{2.59}$$
$$\text{Node 5:} \qquad i_{35} + i_{45} - i_{56} = 0\,. \tag{2.60}$$

[2] The best one can do with near-singular matrices is generally *singular value decomposition*, which extracts at least *some* information from a singular system.

Ohm's law gives

$$R_1: \quad 14\, i_{12} + V_2 = 5\,, \tag{2.61}$$
$$R_2: \quad 2\, i_{23} - V_2 + V_3 = 0\,, \tag{2.62}$$
$$R_3: \quad 6\, i_{24} - V_2 + V_4 = 0\,, \tag{2.63}$$
$$R_4: \quad 0.1\, i_{34} - V_3 + V_4 = 0\,, \tag{2.64}$$
$$R_5: \quad 7\, i_{35} - V_3 + V_5 = 0\,, \tag{2.65}$$
$$R_6: \quad 15\, i_{45} - V_4 + V_5 = 0\,, \tag{2.66}$$
$$R_7: \quad 5\, i_{56} - V_5 = 0\,, \tag{2.67}$$

where we have set $V_6 = 0$, and thus $V_1 = 5$.

Gathering all of the equations in matrix format we obtain a linear system of the form $AX = B$, i.e.

$$\begin{pmatrix} 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 14 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 15 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} i_{12} \\ i_{23} \\ i_{24} \\ i_{34} \\ i_{35} \\ i_{45} \\ i_{56} \\ V_2 \\ V_3 \\ V_4 \\ V_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 5 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{2.68}$$

In engineering much larger circuits are usually considered. Fortunately rewriting the circuit equations in matrix format allows for the use of numerical techniques. We will solve this problem in the lab.

2.6 Summary of the algorithm

We can now summarize the typical steps in any algorithms involved in solving the equation $Ax = b$.

1. Set up the Matrix A and the vector b.
2. LU decompose A using Crout's algorithm [by calling the '`lu_decompose`' routine].
3. If A is ill-conditioned then you might need *pivoting*. Decompose the matrix PA where A is the relevant *permutation matrix*.
4. Then call '`lubksb`' routine to solve for y (forward substitution) then x (backward substitution).
5. Finally finish by *refining* the solution x.

2.7 Appendix

2.7.1 Permutation matrices

Let A be an $n \times n$ matrix. If the matrix P is obtained by swapping rows i and j of the $n \times n$ identity matrix I_n, then rows i and j of A will be swapped in the product PA, and columns i and j of A will be swapped in the product AP.

A permutation matrix is any matrix which can be created by rearranging the rows and/or columns of an identity matrix. It contains exactly one entry equal to 1 in each column and each row, while all other entries are zero.

The permutation matrices for $n = 2$ and $n = 3$ are:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tag{2.69}$$

and

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix},$$
$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}. \tag{2.70}$$

Multiplying a matrix A by a permutation matrix P from the left (PA) results in rearranging the *rows* of A. Multiplying by P from the right (AP) results in rearranging the *columns* of A.

2.7.2 Refinement algorithm

Let x_i denote an initially computed (approximate) solution of a linear system of equations $Ax = b$. The residuum $r_i = Ax_i - b$ is computed first. Then, one obtains the correction Δx_i by solving the original system with residuum r_i on the right-hand side: you need to pass L and U to the refinement algorithm since they are used to calculate the Δx_i.

1. Compute $r_i = b - Ax_i$;
2. solve for Δx_i with $A\Delta x_i = r_i$ (using the LU method);
3. set $x_{i+1} = x_i + \Delta x_i$;
4. calculate new residual $r_{i+1} = b - A(x_i + \Delta x_i)$;
5. if r_{i+1} is below the desired accuracy, you are done with refinement,
   if r_{i+1} is still above the desired accuracy, continue looping until the desired precision is achieved.

2.7.3 Crout's algorithm

An $N \times N$ matrix A can be written as the product of an lower and upper triangular matrix, $A = LU$. The equation leads to N^2 equations and N^2 unknowns. By solving the equations in a particular order (Crout's algorithm), they are easily solved. Crout's algorithm is:

→ **Loop over columns, j=1,N** so that:

First we solve for upper diagonal elements, $i = 1$, j**:**

$U_{ij} = A_{ij} - \sum_{k=1}^{k=i-1} L_{ik} \cdot U_{kj}$

Then we solve for lower diagonal elements, $i = j + 1, N$

$L_{ij} = [A_{ij} - \sum_{k=1}^{k=j-1} L_{ik} \cdot U_{kj}]U_{jj}$

2.7.4 Lab exercise

Question 8 *How to write a matrix in LaTeX*

Here is one way:

```
\begin{pmatrix}
 1 & 1 & 1 \\
 1 & \omega & \omega^2 \\
 1 & \omega^2 & \omega
\end{pmatrix}
```

(i) Try the above and see what you get.

(ii) Now try the following:

```
 The \emph{characteristic polynomial} $\chi(\lambda)$ of the
 $3 \times 3$~matrix
\[ \left( \begin{array}{ccc}
 a & b & c \\
 d & e & f \\
 g & h & i \end{array} \right)\]
 is given by the formula
  \[ \chi(\lambda) = \left| \begin{array}{ccc}
  \lambda - a & -b & -c \\
 -d & \lambda - e & -f \\
-g & -h & \lambda - i \end{array} \right|.\]
```

(ii) Now try the following:

```
[label=\BoxLabel{Array in Latex}]
\begin{equation}
  A = \left( \begin{array}{cccc}
      a_{11} & a_{12} & a_{13} & a_{14}\\
      a_{21} & a_{22} & a_{23} & a_{24}\\
      a_{31} & a_{32} & a_{33} & a_{34}\\
      a_{41} & a_{42} & a_{43} & a_{44}\\
    \end{array} \right)
  \quad
  =
  \quad
  \left( \begin{array}{cccc}
      1      & 0      & 0      & 0 \\
      l_{21} & 1      & 0      & 0 \\
      l_{31} & l_{32} & 1      & 0 \\
      l_{41} & l_{42} & l_{43} & 1 \\
    \end{array} \right)
  \left( \begin{array}{cccc}
      u_{11} & u_{12} & u_{13} & u_{14} \\
      0      & u_{22} & u_{23} & u_{24} \\
      0      & 0      & u_{33} & u_{34} \\
      0      & 0      & 0      & u_{44} \\
    \end{array} \right)
```

Question 9 *Gauss Elimination*

Consider the following system of equations

$$\begin{aligned} 10x_2 - 7x_3 &= 7 \\ 6x_1 + 2.099x_2 + 3x_3 &= 3.901 \\ 5x_1 - x_2 + 5x_3 &= 6 \end{aligned} \tag{2.71}$$

(i) Solve for the vector $x = (x_1, x_2, x_3)$ using 5-significant figures with chopping.

(ii) How does your solution compare to the exact one: $x_{\text{exact}} = (0, -1, 1)$? Explain the difference and suggest a remedy.

Question 10 *LU decomposition*

The upward velocity of a rocket is given at 3 different times

The velocity data is approximated by a polynomial $v(t) = a_1\ t^2 + a_2\ t + a_3 =$ for the time interval $5 \geq t \leq 12$.

Time (s)	Velocity (m/s)
5	106.8
8	177.2
12	279.2

(ii) Using the Gauss Elimination method, solve for the vector $a = (a_1, a_2, a_3)$.

(ii) Using the LU decomposition method, solve for the vector $a = (a_1, a_2, a_3)$.

(iii) Find the velocity at $t = 6, 7.5, 9, 15$ seconds.

Question 11 *LU decomposition on electric circuits*

Use LaTeX to prepare your report which should also include your makefile and all of your codes with the necessary comments.

The purpose of this exercise is:

1. to get you acquainted with matrix handling in Fortran 95;
2. write an elaborate makefile where the main code has more than one dependency;
3. to apply LU decomposition with refinement to the electric circuit given in § 2.5 above.

In Appendix H (**codes**) you will find the following routines in Section H.2. Copy and paste them into Emacs. Save them into your directory (I suggest you create a directory called 'Lab2').

- 'LU_on_circuits.f90'
- 'lup_decomposition.f90' (you should already have this file from the previous lab)
- 'refine_mod.f90'
- 'circuits.data'

(a) Check the 'circuits.data' file and compare with the matrix given in § 2.5 (application to electric circuits) of the lecture notes on the LU decomposition. Make sure the numbers match. For simplicity, we will assume that the currents and the voltage are dimensionless.

(b) Identify the main program among the files you downloaded and the modules it calls (the dependencies). Write a makefile and explain the underlying logic. Run the makefile and make sure that it compiles without any error message.

(c) Explain *in detail* what is performed by 'refine_mod.f90' [refer to § 2.4 and § 2.7.2 above]. In order to refine the solution given by the LU decomposition you will need to call the refinement module from 'LU_on_ircuits.f90'. Update your makefile by including 'refine_mod.f90' and recompile. How many iterations are needed to improve your solution? Explain.