Numerical Linear Algebra

Francesco Zanlungo

Contents

Chapter 1

Linear systems

1.1 Introduction

Let us suppose we are managing a Natural Park in Africa, and we are worried about the number of lions in our park. We want to increase the number of lions, but the only way we have is to regulate the number of its preys, which in our park are zebras and gazelles, by either killing them or introducing new ones. We try to write down a mathematical model that tells us how the number of animals in each species changes with time. Let us suppose we know that, if left free to reproduce, each gazelle will generate, in average, 0.5 gazelles a year[1], and that the same applies to zebras. We also know that a lion kills 6 gazelles and 4 zebras a year. We also know that gazelles are bad for zebras' health, because they tend to eat their food, and in average the presence of a gazelle in the environment will cause the death of 1 over 20 zebras a year. We also know that, if they cannot eat, lions die faster than they can reproduce, and each year in absence of preys their number will halve. Nevertheless, the presence of a gazelle in the environment (due to the probability of eating and thus surviving and successfully reproducing) will increase the number of lions, and in general we will have 3 lion births a year for every 100 gazelles in the park. Zebras are good too, but not that good.

[1] An offspring for each male-female couple. This model is not intended to be in any way a realistic one from a biological point of view; if you are interested in some population dynamics you can google "Logistic growth" or "Prey-predator (Lotcka-Volterra)" models, or check some books on the subject such as Murray's "Mathematical Biology". In general even the simplest realistic models will be non-linear and thus will not be solvable with the theory introduced in these lectures.

Just 1 lion birth every 50 zebras.

Someone suggests to kill 50 zebras every year, and to introduce 100 gazelles, in order to stabilise the number of lions to the desired level. Will that work? What will happen to the zebra and gazelle population?

Let us call the number of lions l, the number of gazelles g and the number of zebras z. The variation of the number in each species is then given by

$$\Delta l = -0.5l + 0.03g + 0.02z, \tag{1.1}$$

$$\Delta g = -6l + 0.5g + 100, \tag{1.2}$$

$$\Delta z = -4l - 0.05g + 0.5z - 50. \tag{1.3}$$

We don't bother about studying what is the time evolution of the system, but just wonder what is its "stationary state", i.e., the condition for having a stable number of animals in each population, that is

$$\begin{cases} 0 = -0.5l + 0.03g + 0.02z \\ 0 = -6l + 0.5g + 100 \\ 0 = -4l - 0.05g + 0.5z - 50 \end{cases} . \tag{1.4}$$

We put our equations together, to show that they are a system that has to be solved "as a whole". For reasons that will be clear later, we re-arrange them as

$$\begin{cases} -0.5l + 0.03g + 0.02z = 0 \\ -6l + 0.5g + 0z = -100 \\ -4l + 0.05g + 0.5z = 50 \end{cases} . \tag{1.5}$$

We will now study an efficient way to solve this kind of mathematical problem.

1.2 Informal review of systems of linear equations

1.2.1 Trivial cases

A linear or first order equation in the unknown variable x is an equation in which only first powers of x appear, namely an equation in the form

$$ax = b. \tag{1.6}$$

If $a \neq 0$, the solution is found as

$$x = a^{-1}b, \tag{1.7}$$

while obviously the inverse of 0 is not defined, and

$$0x = b \tag{1.8}$$

has no solution for $b \neq 0$, and is valid for any x in case $b = 0$. These facts are almost self evident but they may be used as a guiding light for generalising to the many variable case.

A linear equation in two variables will be in the form

$$a_1x_1 + a_2x_2 = b, \tag{1.9}$$

and (assuming $a_1 \neq 0$) solving for x_1 we have many[2] solutions depending on the choice of x_2

$$x_1 = a_1^{-1}(b - a_2x_2). \tag{1.10}$$

In other words, the equation is indeterminate, which may not be good if we were looking for some explicit numbers. It is indeed a good rule of thumb to have at least two equations if we are dealing with two variables, which leads us to a system of (linear) equations, which we may write as

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 = b_2 \end{cases}. \tag{1.11}$$

Generalising, we will write the jth equation of a system of m equations ($j = 1, \ldots, m$) in n variables as

$$\sum_{i=1}^{n} a_{j,i}x_i = b_j, \tag{1.12}$$

where the introduced notation will allow us to use the matrix formalism, as we will see soon. We may also introduce some terminology, and name *homogeneous* a system with $b_j = 0 \, \forall j$ (otherwise the system will be called *inhomogeneous*).

Obviously having n equations and n variables does not assure a unique solution. As an example let us consider

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 = b_1 \\ a_{1,1}x_1 + a_{1,2}x_2 = b_2 \end{cases}, \tag{1.13}$$

[2]Continuously infinite.

where the a coefficients are the same in the first and second equations. Clearly there is something "wrong" with this system, and we intuitively understand that if $b_1 = b_2$ the second equation provides no new information, while if $b_1 \neq b_2$ new information is available but there will be no values of x_1, x_2 satisfying both equations.

To formalise this, we may consider some basic properties of (systems of) equations. If a is a real number and we have

$$B = C, \qquad D = E \tag{1.14}$$

Also the following will hold

$$aB = aC, \qquad B + D = C + E, \qquad B + aD = C + aE. \tag{1.15}$$

Namely, an equation will be still valid if we multiply each of its terms by a constant, and if we sum two equations term by term (possibly multiplying the terms by constants), we still obtain an equivalent equation. As a result, by subtracting term by term the two equations in (1.13), we obtain the equivalent system

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 = b_1 \\ 0 = b_1 - b_2 \end{cases} \tag{1.16}$$

which has no solution if $b_1 \neq b_2$, while if $b_1 = b_2$ the second equation is just an identity in which the variables x_1, x_2 do not appear at all. In order to be able to solve a system with n variables in a unique way we will thus not just need n equations, but n *independent* equations. This concept will be formalised below, but it is clear that, in the case of two variables, systems like

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 = b_1 \\ c\,a_{1,1}x_1 + c\,a_{1,2}x_2 = b_2 \end{cases} \tag{1.17}$$

will not lead to unique solutions (multiply the first equation by c and subtract the second to obtain again $0 = c\,b_1 - b_2$).

In the same way, while in general having more equations than variables is a problem (i.e., $x = 2$ and $x = 3$) this is true only if the equations are actually independent (no problems to solve $x = 2$, $2x = 4$, just redundancy of information).

Before proceeding to analyse formally the n equation case, let us use again the two variable example to understand a property of homogeneous systems,

which will again provide insight on the general case. Let us assume

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 = 0 \\ a_{2,1}x_1 + a_{2,2}x_2 = 0 \end{cases} . \tag{1.18}$$

This system, *as any other homogeneous system*, is trivially solved by $x_i = 0\,\forall i$. Is the solution unique? Let us solve the first equation as

$$x_1 = -a_{1,1}^{-1}(a_{1,2}x_2). \tag{1.19}$$

By substitution in the second, and after multiplying each term by a_{11}, we have

$$x_2(a_{2,2}a_{1,1} - a_{1,2}a_{2,1}) = 0. \tag{1.20}$$

You may recognise the term in parentheses as the determinant of a matrix with entries $a_{j,i}$. If such a determinant is different from 0, then we have $x_2 = 0$, and as a consequence (eq. 1.19) $x_1 = 0$, as the only possible solution. Otherwise, each x_2 will work, and we have an infinity of solutions, the relation between the two variables being given by eq. (1.19). This is not by chance, and indeed matrices and determinants play a fundamental role in determining if a system has solutions or not, and if the solution is unique. We should then revise their properties.

1.2.2 Vectors, matrices, determinants

We call[3] a vector $\mathbf{x} \in \mathbb{R}^n$ a collection of n real numbers (n-tuple)

$$\{x_1, \ldots, x_n\}, \tag{1.21}$$

and we define the operations of addition of two vectors

$$\mathbf{z} = \mathbf{x} + \mathbf{y}, \quad z_i = x_i + y_i, \tag{1.22}$$

and multiplication of a vector with a scalar (real number) c

$$\mathbf{z} = c\,\mathbf{x}, \quad z_i = c\,x_i. \tag{1.23}$$

[3]No attempt of rigour or formality will be found in the following discussion, the interested reader may rely on a large number of excellent text on Linear Algebra for a deeper and better treatment

We then consider linear applications A that go from $\mathbb{R}^n$ to $\mathbb{R}^m$. By linear, we mean such that its action preserves the sum and pruduct by a scalar operation, namely

$$A(c\mathbf{x}) = cA(\mathbf{x}), \tag{1.24}$$

and

$$A(\mathbf{x}+\mathbf{y}) = A(\mathbf{x}) + A(\mathbf{y}). \tag{1.25}$$

Clearly, such applications cannot include terms like x_i^l with $l \neq 1$, or constant terms[4], and thus, if

$$\mathbf{y} = A\mathbf{x}, \tag{1.26}$$

A will be written in the form

$$y_j = \sum_{i=1}^{n} a_{j,i} x_i, \tag{1.27}$$

with $j = 1, \ldots, m$. We name A a m by n matrix, and write it as a table with m rows and n columns

$$A = \begin{pmatrix} a_{1,1} & \ldots & a_{1,n} \\ \vdots & \vdots & \vdots \\ a_{m,1} & \ldots & a_{m,n} \end{pmatrix}. \tag{1.28}$$

Let us consider now another linear application, B, this time from $\mathbb{R}^m$ to $\mathbb{R}^l$, and let B act on $\mathbf{y}$, the result of the application of A on $\mathbf{x}$. We may call

$$\mathbf{z} = B\mathbf{y} = BA\mathbf{x} \equiv C\mathbf{x}, \tag{1.29}$$

where in the last step we defined the application $C = BA$ that goes from $\mathbb{R}^n$ to $\mathbb{R}^l$, i.e. the result of applying B on the result of A. From our definition of matrix, we have

$$y_j = \sum_{i=1}^{n} a_{j,i} x_i, \quad z_k = \sum_{j=1}^{m} b_{k,j} y_j, \quad z_k = \sum_{k=1}^{n} c_{k,i} x_i. \tag{1.30}$$

We chose the name of indexes in such a way that putting together the first and second expression we get

$$z_k = \sum_{j=1}^{m} b_{k,j} \left(\sum_{i=1}^{n} a_{j,i} x_i \right). \tag{1.31}$$

[4]Consider for example $A(x) = x+b$. We have $A(x+y) = x+y+b \neq x+y+2b = A(x)+A(y)$. In a similar way, if $A(x) = x^2$, $A(x+y) = x^2+y^2+2xy \neq x^2+y^2 = A(x)+A(y)$.

These may be re-written (due to the associative, commutative and ditributive properties of sums and product, try for a simple case!) as

$$z_k = \sum_{i=1}^{n} (\sum_{j=1}^{m} b_{k,j} a_{j,i}) x_i, \tag{1.32}$$

which leads to the identification

$$c_{k,i} \equiv \sum_{j=1}^{m} b_{k,j} a_{j,i}. \tag{1.33}$$

This is indeed the "row by column" multiplication law between matrices. Since k runs between 1 and l, and i between 1 and n, the matrix has the correct number of terms, rows and columns.

As it is usual, vectors may be treated as single columns matrices. Namely, the (n) x_i component of the vector may be considered as the $x_{1,i}$ components of a matrix, and applying a $m \times n$ matrix on them will generate a single column, m component, matrix, i.e. a m component vector.

Writing a vector as a column, instead of a row, may seem a waste of space, but the notation is particularly handy when we consider our original problem, linear equation systems. By comparing eqs. (1.12) and (1.27), we may see that an inhomogeneous system of m equations with known terms b_j for the n variables x_i may be written as

$$A\mathbf{x} = \mathbf{b}. \tag{1.34}$$

Graphically, this becomes

$$\begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & \vdots & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}, \tag{1.35}$$

leaving the coefficient $a_{j,i}$ of the matrix in the same row and column structure in which we are used to see equation systems. We have thus established a connection between linear systems and matrices, with the $m \times n$ coefficients of the system generating a m row n column matrix.

Can we solve eq. (1.34) as we did for eq. (1.6)? In the solution (1.7) we used the inverse of a, a^{-1}. What is such an inverse for a matrix, supposed that it exists? For real numbers the inverse was defined $\forall a \in \mathbb{R}, a \neq 0$ as

the number a^{-1} such that $aa^{-1} = 1$. We introduced matrices as linear applications (functions or better *mappings*) between (possibly) different vector space, so if we want to "inverse" a matrix, we have to look for its inverse mapping. It is possible to define such an inverse A^{-1} for a class of square $(n \times n)$ matrices (those that have non-zero determinant) as

$$A^{-1}A = AA^{-1} = \mathbb{1}. \tag{1.36}$$

A few clarifications have to be given regarding this latter equation. The first one obviously regards the symbol on the right, $\mathbb{1}$. This represent the *identity* application on the space $\mathbb{R}^n$, i.e. the mapping that brings each vector in itself. This application is obviously linear, and may be written as a matrix. It is easy to check that we have

$$\mathbb{1} = \begin{pmatrix} 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & 0 & \ddots & 0 & \vdots \\ \vdots & \vdots & 0 & \ddots & \vdots \\ 0 & \dots & \dots & 0 & 1 \end{pmatrix}, \tag{1.37}$$

or, equivalently,

$$\mathbb{1}_{j,i} = \delta_{j,i}, \tag{1.38}$$

where the *Kronecker delta* (see appendix A) satisfies

$$\delta_{j,i} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq 0 \end{cases}. \tag{1.39}$$

In eq. (1.36) we were also careful in asking that the inverse application gives the identity both when applied on the result of A, and when A is applied on its result. In general, we know that when we are dealing with functions, the order in which they are applied is important. Indeed, for non-square matrices, we may have that while BA is defined, AB may not even be defined, since a $m \times n$ matrix cannot be multiplied on the left (i.e., applied to the result) of a $k \times m$ matrix if $k \neq n$.

For square matrices we do not have this problem, but in general the matrix product is not commutative, i.e. $BA \neq AB$.[5] As a trivial example,

[5] This trivial mathematical property was surprisingly not known, in 1925, to the 23 years old physics Ph.D. Werner Heisenberg. Maybe thanks also to this hole in his mathematical

we may check the product

$$\begin{pmatrix} 1 & 0 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}, \tag{1.40}$$

while

$$\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}. \tag{1.41}$$

The second product leads to a matrix full of zeros, i.e. a matrix that, applied on any vector, produces a vector full of zeros. We may call such a matrix $\mathbb{O}$. The fact that this matrix may be obtained as the product of two non-zero matrices shows us again how the product of matrices is less trivial than the product of real numbers.

When dealing with reals (i.e., with a single variable) we knew that $ax = b$ had no solution (for $b \neq 0$, or infinite solutions for $b = 0$) only when $a = 0$. Indeed, we still have that

$$\mathbb{O}\mathbf{x} = \mathbf{b} \tag{1.42}$$

has no solution (unless $b_i = 0\,\forall i$, in which case the system has infinite solutions; in both cases the system has no *unique* solution), but this will happen also for a larger class of square matrices, namely those that have determinant equal to zero, i.e. *singular* matrices.

Formal treatments of determinants are quite troublesome, and we do not enter here in the details of the theory[6]. For our purposes it is enough to remember the following properties:

- *If we multiply a row of matrix by a constant c, then also the matrix's determinant is multiplied by c*

background (but mainly to his genius), Heisenberg went on associating an electron's position and momentum (components) not to real numbers but to matrices, and eventually went on to discover his uncertainty principle, get a Nobel price and perform one of the greatest revolutions in the history of human thought and knowledge.

[6]You may refer again to many good Linear Algebra texts; a good practical definition of a determinant uses the concept of minor, i.e. computes the value of a determinant based on those of smaller sub-matrices, reducing eventually to 2×2 matrices for which an easy closed form is available, but the number of terms in these calculations gets extremely high for high n, unless the matrix has a particular form (many zeros). Formal proofs often use an extremely useful and elegant mathematical object, the *totally asymmetrical Levi-Civita symbol*.

- *If we sum to a row of matrix another row, possibly multiplied by a constant c, the matrix's determinant does not change*

The first property tells us that if a matrix has a row full of zeros, then its determinant is zero[7]. The second one tells us that *if one of the rows may be written as a linear combination of the other rows, we may obtain a matrix* (by subtracting other rows multiplied by proper constants) *with a row of zeros, without changing the value of the determinant.* In other words, such a matrix has determinant zero.
We finally came to the generalisation of the problem we studied at the beginning for simple (2, 3 variables) cases. We had seen that a system with n equations and n variables had a unique solutions only if all the equations where independent between them. This can be formalised by saying that none of the rows of the corresponding matrix of coefficients may be written as a linear combination of the others, and, as we just discussed, this corresponds to the matrix having determinant different from zero, i.e. being non-singular.

For such matrices the inverse may be defined (we will discuss later how to compute it), and the system $A\mathbf{x} = \mathbf{b}$ formally solved as

$$\mathbf{x} = \mathbb{1}\mathbf{x} = A^{-1}A\mathbf{x} = A^{-1}\mathbf{b}. \tag{1.43}$$

In this case, the solution is unique, both for homogeneous and inhomogeneous systems (in the latter case it is obviously $x_i = 0$).

Determinants give us also a lot of information about singular matrices, and also for the more general $m \times n$ case. Without entering in details, we may remember that the *rank* k of a matrix is the size of its largest submatrix (i.e., a matrix that is obtained by the original one eliminating a few rows and columns) with non-zero determinant. In case all the determinants of matrices that may be obtained by adding a row of coefficients and the column of known terms to this sub-matrix are zero, the system has at least a solution. The solution is unique if the rank k equals the number of variables.

[7]This can be obviously derived directly also from a definition of determinant, for example from the one that uses minors.

1.3 Algorithms to solve linear equation systems

Let us consider a simple linear equations system

$$\begin{cases} 3x_1 + 2x_2 = 8 \\ x_1 - 4x_2 = -2 \end{cases} \tag{1.44}$$

One very intuitive way of solving it is to isolate the first variable in the second equation

$$x_1 = 4x_2 - 2, \tag{1.45}$$

substitute in the first

$$3(4x_2 - 2) + 2x_2 = 8 \Rightarrow 14x_2 = 14 \Rightarrow x_2 = 1, \tag{1.46}$$

and finally again in the second

$$x_1 = 4x_2 - 2 = 4 - 2 = 2. \tag{1.47}$$

This procedure works quite well for $n = 2$ and it is obviously correct for any n. Nevertheless, as you can check already in the case of a simple $n = 3$ system,

$$\begin{cases} 4x_1 - 3x_2 + x_3 = 1 \\ x_1 + x_2 + x_3 = 6 \\ -x_1 + 2x_2 - 2x_3 = -3 \end{cases}, \tag{1.48}$$

the calculation becomes easily messy with growing n. Furthermore, the procedure gets much easier if we do "the right choices" (for example, starting from the second equation in eq. 1.44), and involves formal relations between variables, which are not easy to implement in programming languages.

By algorithm we mean a set of computational rules that will lead us to the solution of the problem (if such solution exists, and inform us in the case the solution does not exist or it is not unique), rules that may be implemented on a computer.

Mathematicians of the past have provided us with such algorithms. Let us first understand the logic behind the algorithm before stating it in a formal way. If we go back to the example (1.44), and remember (eq. 1.15) that we may add to an equation a linear combination of other equations without

changing the system, we may notice that if we add to the second equation the first one multiplied by $-1/3$, we cancel x_1 from it

$$\begin{cases} 3x_1 + 2x_2 = 8 \\ x_1 - x_1 - 4x_2 - \frac{2}{3}x_2 = -2 - \frac{8}{3} \Rightarrow -\frac{14}{3}x_2 = -\frac{14}{3} \end{cases} . \tag{1.49}$$

In this way in the second equation we have decreased the number of variables to one, and we may easily solve as

$$\begin{cases} 3x_1 = 8 - 2x_2 \\ x_2 = 1 \end{cases} \Rightarrow \begin{cases} 3x_1 = 6 \\ x_2 = 1 \end{cases} \Rightarrow \begin{cases} x_1 = 2 \\ x_2 = 1 \end{cases} . \tag{1.50}$$

The procedure is easily extended and formalised to a n variable system. We first write our system in matrix form

$$A\mathbf{x} = \mathbf{b}, \tag{1.51}$$

and then repeatedly use the above procedure (subtracting equations) until we obtain a system in the form

$$A'\mathbf{x} = \mathbf{b}', \tag{1.52}$$

where A' is a matrix in the upper triangular form, i.e. $a'_{j,i} = 0$ if $j > i$, or

$$A' = \begin{pmatrix} a'_{1,1} & a'_{2,2} & \dots & \dots & a'_{n,n} \\ 0 & a'_{2,2} & a'_{2,3} & \dots & a'_{2,n} \\ \vdots & 0 & \ddots & a'_{i,j} & \vdots \\ \vdots & \vdots & 0 & \ddots & \vdots \\ 0 & \dots & \dots & 0 & a'_{n,n} \end{pmatrix}, \tag{1.53}$$

and then proceed to solve for each variable

$$x_n = b'_n / a'_{n,n} \tag{1.54}$$

$$x_{n-1} = (b'_{n-1} - a'_{n-1,n}x_n)/a'_{n,n}, \tag{1.55}$$

and so on. Formally,

- *for* $j = 1, \dots, n$
  - *for* $l = j + 1, \dots, n$
    compute the *multiplying factor* or *multiplicator* $m_{l,j} = a^j_{l,j}/a^j_{j,j}$, where $a^j_{j,j}$ is called the jth *pivot*
    substitute $a^{j+1}_{l,j} = a^j_{l,j} - m_{l,j}a^j_{j,j} = 0$

  - *for* $i = j+1, \ldots, n$
    substitute $a^{j+1}_{l,i} = a^j_{l,i} - m_{l,j} a^j_{j,i}$
  substitute $b^{j+1}_l = b^j_l - m_{l,j} b^j_j$

- *for* $j = n, \ldots, 1$
  $x_j = \left(b^n_j - \sum_{i=j+1}^{n} a^n_{j,i} x_i \right) a^n_{j,j}$

Let us do it for the example of eq. (1.48). We have

$$A = \begin{pmatrix} 1 & 1 & 1 \\ -1 & 2 & -2 \\ 4 & -3 & 1 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} 6 \\ -3 \\ 1 \end{pmatrix}. \tag{1.56}$$

At the first step we get $m_{2,1} = -1$, $m_{3,1} = 4$, and

$$A^2 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 3 & -1 \\ 0 & -7 & -3 \end{pmatrix} \qquad \mathbf{b}^2 = \begin{pmatrix} 6 \\ 3 \\ -23 \end{pmatrix}. \tag{1.57}$$

The next multiplicator is $m_{3,2} = -7/3$ and finally we have

$$A^3 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 3 & -1 \\ 0 & 0 & -16/3 \end{pmatrix} \qquad \mathbf{b}^3 = \begin{pmatrix} 6 \\ 3 \\ -16 \end{pmatrix}. \tag{1.58}$$

With *backward substitution* we get $x_3 = 3$, $x_2 = (3+3)/3 = 2$ and $x_1 = 6 - 3 - 2 = 1$ or

$$\mathbf{x} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}. \tag{1.59}$$

1.3.1 What may go wrong?

Let us consider this system

$$\begin{cases} 0.0003 x_1 + 63.324 x_2 = 63.327 \\ 5.2922 x_1 - 7.133 x_2 = 45.789 \end{cases}, \tag{1.60}$$

whose solution is

$$\mathbf{x} = \begin{pmatrix} 10 \\ 1 \end{pmatrix}, \tag{1.61}$$

as can be easily checked (by substitution). Let us solve it according to our algorithm, but as a computer would do it, i.e. just keeping a finite number of digits (we keep 5, computers would typically do better than that[8]).

We obtain $m_{2,1} = 17640.666\ldots \approx 17641$. We then substitute and solve for $x_2 = 1.0001$ and... $x_1 = -10$!!!!
The extremely high value of m, when multiplied by $a_{1,2}$ and b_1, gave rise to numbers that where extremely big (6 order of magnitude larger) than $a_{2,2}$ and b_2, so that these latter contributions where just cancelled in the rounding. These caused a tiny error in x_2, but when replaced in the first equation, due again to the small value of $a_{1,1}$ with respect to $a_{1,2}$ and b_1, this produced a huge mistake in the value of x_1.

How to avoid this problem? The first impression would be that it is caused by the small value of the pivot $a_{1,1}$, and the algorithm could be corrected in order to look for the largest absolute value pivot, and exchange rows in order to avoid the use of small (or even worst, zero!) pivots.

But this would not work. If you read carefully the argument given above, more than the absolute value of $a_{1,1}$, was its relative value to $a_{1,2}$ to cause problems. You can convince yourself by studying, again with 5 digit precision, the following system, where the coefficients in the first line are multiplied by 10^5.

$$\begin{cases} 30x_1 + 6332400x_2 = 6332700 \\ 5.2922x_1 - 7.133x_2 = 45.789 \end{cases} . \tag{1.62}$$

We may then modify our algorithm, that we may call *Gaussian elimination with scaled partial pivoting*[9] in the following way

- *for* $j = 1, \ldots, n$
  - *for* $l = j, \ldots, n$
    compute the *scale factor* $\sigma_l = \max_i |a^j_{l,i}|$
- *for* $j = 1, \ldots, n$ find the line $l = \max_i |a^j_{i,j}|/\sigma_i$
  - *for* $i = j, \ldots, n$ replace $\bar{a}^j_{j,i} = a^j_{l,i}$
    and $\bar{a}^j_{l,i} = a^j_{j,i}$

[8] 8 digits in *single precision*, 16 in *double precision*, or much more in high precision computations.

[9] *Partial* because we search for the best pivot over rows, but we do not perform column exchange.

replace $\bar{\sigma}_j = \sigma_l$,
and $\bar{\sigma}_l = \sigma_j$.

replace $\bar{b}^j_j = b^j_l$
and $\bar{b}^j_l = b^j_i$.
IF($\bar{a}^j_{j,j} = 0$): EXIT (no unique solution)
Using the barred variables (even if not shown)

- *for* $l = j+1, \ldots, n$
  compute the *multiplying factor* $m_{l,j} = a^j_{l,j}/a^j_{j,j}$, where $a^j_{j,j}$ is called the jth *pivot*
  substitute $a^{j+1}_{l,j} = a^j_{l,j} - m_{l,j}a^j_{j,j} = 0$
  - *for* $i = j+1, \ldots, n$
    substitute $a^{j+1}_{l,i} = a^j_{l,i} - m_{l,j}a^j_{j,i}$

  substitute $b^{j+1}_l = b^j_l - m_{l,j}b^j_j$

IF($\bar{a}^n_{n,n} = 0$): EXIT (no unique solution)

- *for* $j = n, \ldots, 1$
  $x_j = \left(b^n_j - \sum_{i=j+1}^n a^n_{j,i}x_i\right)/a^n_{j,j}$.

In this algorithm there is no column swap, so the result will be given in the original variable order[10].

Let us use this algorithm for a modified version of eq. (1.48). We want to solve $A\mathbf{x} = \mathbf{b}$ for

$$A = \begin{pmatrix} 1 & 2 & 1 \\ -1 & 2 & -2 \\ 4 & -3 & 1 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} 8 \\ -3 \\ 1 \end{pmatrix}. \tag{1.63}$$

We start by computing a vector of scale factors, i.e. of maximal modulus in each row,

$$\sigma = \begin{pmatrix} 2 \\ 2 \\ 4 \end{pmatrix}. \tag{1.64}$$

[10]For complete pivoting, at the end we would need to keep track of all row substitutions for obtaining the result in the original form.

We have $|a_{1,1}|/\sigma_1 = 1/2$, $|a_{2,1}|/\sigma_2 = 1/2$ and $|a_{3,1}|/\sigma_3 = 4/4 = 1$. The maximum is for the third row, and thus we swap the third and first row, obtaining the system

$$\begin{pmatrix} 4 & -3 & 1 \\ -1 & 2 & -2 \\ 1 & 2 & 1 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 1 \\ -3 \\ 8 \end{pmatrix}. \tag{1.65}$$

The scale factors become

$$\sigma = \begin{pmatrix} 4 \\ 2 \\ 2 \end{pmatrix}. \tag{1.66}$$

Computing the multiplicators, at the first step we get $m_{2,1} = -1/4$, $m_{3,1} = 1/4$, and the system modifies to

$$\begin{pmatrix} 4 & -3 & 1 \\ 0 & 5/4 & -7/4 \\ 0 & 11/4 & 3/4 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 1 \\ -11/4 \\ 31/4 \end{pmatrix}. \tag{1.67}$$

We have now[11] $|a_{2,2}|/\sigma_2 = 5/8$, $|a_{3,2}|/\sigma_3 = 11/8$, with a maximum on the third row. We swap again the second and third row to obtain

$$\begin{pmatrix} 4 & -3 & 1 \\ 0 & 11/4 & 3/4 \\ 0 & 5/4 & -7/4 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 1 \\ 31/4 \\ -11/4 \end{pmatrix}. \tag{1.68}$$

There is no need of using scale factors on the last row (since we have no choice but usining it). The next multiplicator is $m_{3,2} = (5/4)/(11/4) = 5/11$ and finally we have

$$\begin{pmatrix} 4 & -3 & 1 \\ 0 & 11/4 & 3/4 \\ 0 & 0 & -23/11 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 1 \\ 31/4 \\ -69/11 \end{pmatrix}. \tag{1.69}$$

With *backward substitution* we get $x_3 = 3$, $x_2 = 4/11[31/4 - 9/4] = 2$ and $x_1 = 1/4[1 + 3(2) - 3] = 1$ or

$$\mathbf{x} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}. \tag{1.70}$$

[11] Since there is no risk of misunderstanding, we will now denote the transformed matrix A^2 simply as A.

1.4 Review of Linear Algebra: Exercises

1. **Exercise** Using analytical geometry (lines in a plane) discuss when a system of 2 equations in 2 variables has

   - 1 solution
   - ∞ solutions
   - no solution

   May we have $1 < n < \infty$ solutions?

2. **Exercise** For an homogeneous system ($b_1 = b_2 = 0$) can you provide an example with no solutions?

3. **Exercise** Rewrite the "African Park" example in matrix form.

4. **Exercise** Rewrite in matrix form

$$\begin{cases} 27 + y + 32z + x + w = 0 \\ 2x + 3y + w = 7 \\ -7x + w - 7y - z + 2y + 3 = 0 \end{cases}$$

5. **Exercise** Compute

$$\begin{pmatrix} -1 & 3 & 2 \\ 2 & 0 & 1 \\ 1 & 0 & -2 \end{pmatrix} \begin{pmatrix} 3 \\ 0 \\ 4 \end{pmatrix}$$

6. **Exercise** Compute

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 7 \\ 8 \\ 2 \\ 4 \\ 1 \\ 0 \\ 3 \end{pmatrix}$$

7. **Exercise** Compute

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 7 \\ 8 \\ 2 \\ 4 \\ 1 \\ 0 \\ 3 \end{pmatrix}$$

8. **Exercise** Compute

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} 3 \\ -2 \\ 7 \end{pmatrix}$$

9. **Exercise** Compute

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 0 \\ 2 \\ 0 \end{pmatrix}$$

   I call this matrix D for "derivation". Why? (Hint: polynomials)

10. **Exercise** Compute

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 0 \\ 2 \\ 0 \end{pmatrix}$$

    I call this matrix M for "multiplication" (by x). Why? (Hint: see above.)

11. **Exercise** Show the equivalence of

$$z_k = \sum_j b_{kj} \left(\sum_i a_{ji} x_i \right)$$

and

$$z_k = \sum_i \left(\sum_j b_{kj} a_{ji} \right) x_i$$

You may also do it only for a particular example.

12. **Exercise** Compute

$$\begin{pmatrix} -1 & 3 & 2 \\ 2 & 0 & 1 \\ 1 & 0 & -2 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ 1 & -1 \\ 0 & -3 \end{pmatrix}$$

13. **Exercise** Compute

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

14. **Exercise** Compute

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

15. **Exercise** Compute

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

16. **Exercise** Compute

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

17. **Exercise** Using the matrices D of Ex. 9 and M of Ex. 10 compute

$$DM - MD$$

18. **Exercise (Advanced)** Can you do Ex. 17 on the infinite polynomial space? (Hint: use the Kronecker δ)

19. **Exercise** Study the number of solutions of the following systems using determinant methods and compare to the graphical method

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

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

1.5 Solution of Linear Sistems: Exercises

1. **Exercise** Solve
$$\begin{cases} x_1 + x_2 + x_3 = 6 \\ -x_1 + 2x_2 - 2x_3 = -3 \\ 4x_1 - 3x_2 + x_3 = 1 \end{cases}$$

2. **Exercise** Solve the African Park model
$$\begin{cases} -0.5l + 0.03g + 0.02z = 0 \\ -6l + 0.5g + 100 = 0 \\ -4l + 0.05g + 0.5z - 50 = 0 \end{cases}$$

3. **Exercise** Solve
$$\begin{cases} x_1 + x_2 + x_3 = 10 \\ x_1 + x_2 - x_3 = -2 \\ 3x_1 - x_2 + 6x_3 = 36 \end{cases}$$

4. **Exercise** Solve with 4 digits precision (rounding)
$$\begin{cases} 0.003x_1 + 59.14x_2 = 59.17 \\ 5.291x_1 - 6.38x_2 = 46.53 \end{cases}$$

5. **Exercise** Solve with 4 digits precision (rounding)
$$\begin{cases} 30x_1 + 591400x_2 = 591700 \\ 5.291x_1 - 6.38x_2 = 46.53 \end{cases}$$

6. **Exercise** Solve
$$\begin{cases} x_1 + x_2 + x_3 + x_4 = 4 \\ x_1 - x_2 + x_3 + x_4 = 2 \\ x_1 + x_2 - x_3 + x_4 = 2 \\ x_1 - x_2 + 3x_3 + x_4 = 4 \end{cases}$$

7. **Exercise** Solve
$$\begin{cases} x_1 + x_2 + x_3 + x_4 = 4 \\ x_1 - x_2 + x_3 + x_4 = 2 \\ 2x_1 + 2x_3 + 2x_4 = 7 \\ 3x_1 - 2x_2 - x_3 + 4x_4 = 4 \end{cases}$$

8. **Exercise** Solve

$$\begin{cases} x_1 + x_2 + x_3 + x_4 = 4 \\ x_1 - x_2 + x_3 + x_4 = 2 \\ 2x_1 + 2x_3 + 2x_4 = 6 \\ 3x_1 - 2x_2 + 3x_3 + 4x_4 = 8 \end{cases}$$

Chapter 2

Numerical Linear Algebra: Determinant, Inverse and LU decomposition

2.1 Determinant

Let us recall one possible definition of the determinant, based on minors (using the first column, but different columns or rows could be used). Assuming A to be a $n \times n$ square matrix, we have

$$\det(A) = \sum_j (-)^{j+1} \det(C_{j,1}), \tag{2.1}$$

here, $C_{j,1}$ is the $n-1 \times n-1$ square matrix obtained by deleting the jth row and 1st column, for example if

$$A = \begin{pmatrix} a_{1,1} & a_{2,1} & \dots & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & \dots & a_{2,n} \\ a_{3,1} & a_{3,2} & \dots & \dots & a_{3,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & \dots & \dots & a_{n,n} \end{pmatrix}, \tag{2.2}$$

we have

$$C_{1,1} = \begin{pmatrix} a_{2,2} & \dots & \dots & a_{2,n} \\ a_{3,2} & \dots & \dots & a_{3,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n,2} & \dots & \dots & a_{n,n} \end{pmatrix}, \qquad C_{2,1} = \begin{pmatrix} a_{1,2} & \dots & \dots & a_{1,n} \\ a_{3,2} & \dots & \dots & a_{3,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n,2} & \dots & \dots & a_{n,n} \end{pmatrix}. \tag{2.3}$$

For a 2×2 matrix

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix}, \tag{2.4}$$

we have $C_{1,1} = a_{2,2}$, $C_{2,1} = a_{1,2}$ and thus

$$\det(A) = (-)^2 a_{1,1}a_{2,2} + (-)^3 a_{2,1}a_{1,2} = a_{1,1}a_{2,2} - a_{2,1}a_{1,2}, \tag{2.5}$$

as we expected.

The computation of a 2×2 determinant implies 2 multiplications[1]. For a 3×3 determinant, we will compute 3 multiplications with 2×2 determinant, and thus we overall have $3! = 6$ multiplications. For a 4×4 determinant we will have $4!$ multiplications, and by induction $n!$ multiplications for a n square matrix. Cranmer's rule uses a determinant based method to solve linear systems, involving $\sim n!$ operations, while Gaussian Elimination involves $\sim n^3$ (if optimised, roughly $n^3/3$)[2] Cranmer's rule is popular for solving by hand small systems, but it's not used by computers. Why? It would take billions of years to solve a 30×30 system with Cranmer's method even on the world's most powerful computer!

How may we compute the determinant of a matrix then? Let us first consider

$$\det \begin{pmatrix} a_{1,1} & a_{2,1} \\ 0 & a_{2,2} \end{pmatrix} = a_{1,1}a_{2,2}. \tag{2.6}$$

We may now use eq. (2.1) and obtain

$$\det \begin{pmatrix} a_{1,1} & a_{2,1} & a_{3,1} \\ 0 & a_{2,2} & a_{2,3} \\ 0 & 0 & a_{3,3} \end{pmatrix} = +a_{1,1}\left(a_{2,2}a_{3,3}\right) - 0 + 0 = a_{1,1}a_{2,2}a_{3,3}. \tag{2.7}$$

[1] Multiplications and divisions are more time consuming for computers than additions and subtractions.

[2] The n^3 order comes from the fact the main loop in which to each row the pivot row multiplied by the multiplying factor is subtracted. This loop runs over all possible pivot rows ($j = 1, \dots, n$), over all the rows behind the pivot ($l = j + 1, \dots, n$) and over all the columns in the rows ($i = j + 1, \dots, n$). Clearly the number of operations is lower than n^3, but grows as αn^3 where α is a constant.

	n^3	$n!$
3	27	6
5	125	120
10	1000	3628800
30	27000	$2.65 \cdot 10^{32}$

Table 2.1: Comparison between n^3 and $n!$

We have thus seen that for 2 and 3 upper triangular matrices (and obviously for unit matrices) the determinant is given by the product of the diagonal terms. This is true for any n, i.e. if U is an upper triangular matrix, we have

$$\det(U) = \prod_{i=1}^{n} u_{j,j} \tag{2.8}$$

The proof by induction is straightforward. If we know that (2.8) is true for $n-1$, then

$$\det \begin{pmatrix} u_{1,1} & u_{2,1} & \dots & \dots & u_{1,n} \\ 0 & u_{2,2} & \dots & \dots & \vdots \\ \vdots & 0 & \ddots & \dots & \vdots \\ \vdots & \vdots & 0 & \ddots & \vdots \\ 0 & \dots & \dots & 0 & u_{n,n} \end{pmatrix} = u_{1,1} \det\left(C_{1,1}\right) + 0 = u_{1,1} \prod_{i=2}^{n} u_{j,j}. \tag{2.9}$$

But we know a method to reduce a matrix to an upper triangular form, Gaussian Elimination, which involves $\sim n^3$ operations. It includes operations such as adding to a row an other row, which do not affect the value of the determinant[3]; and, in case the pivot is 0, swapping two rows, which changes the determinant's sign, as can be seen easily from eq. (2.1) and checked for a 2 by 2 matrix

$$\det \begin{pmatrix} a_{2,1} & a_{2,2} \\ a_{1,1} & a_{1,2} \end{pmatrix} = a_{2,1}a_{1,2} - a_{2,2}a_{1,1} = -\det \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix}. \tag{2.10}$$

[3]By using the row version of (2.1) and the appropriate row, we may rewrite the determinant as the sum of two terms, one of them the original determinant, and the other one a determinant including two linearly dependent rows. The second term is thus 0, and the determinant does not change.

So we may compute the determinant using Gaussian Elimination and keeping track of each row swap, calling n_s the number of swaps. After obtaining the triangular form for A, we compute

$$\det = (-)^{n_s} \prod_{i=1}^{n} a_{jj}. \tag{2.11}$$

This involves $\sim n^3$ operations. There are some manipulations that involve n computations (as for example eq. 2.11) or n^2 (computing the multiplicators), but the only n^3 manipulation is the subtraction $a_{li} = a_{li} - m_{lj}a_{ji}$. This is a $\sim n^3$ term since it runs over the 3 indexes j, l, i, each one running respectively over n columns, $n - j$ rows, $n - j$ columns.

To revise the method of Gaussian Elimination, let us do the following exercise.

2.1.1 Exercise

Using Gaussian Elimination compute

$$\det \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 2 & 1 & 0 & 1 \\ 2 & 0 & 3 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Solution

The multiplying factors are $m_{2,1} = 0$, $m_{3,1} = 1$, $m_{4,1} = 2$ and $m_{5,1} = 1$, and after the row subtractions the matrix modifies to

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & -1 & 1 \\ 0 & -2 & 3 & -2 & 0 \\ 0 & -1 & 0 & 0 & 0 \end{pmatrix}.$$

We cannot compute the multiplicators using $a_{2,2} = 0$. We then swap the

rows 2 and 3 to obtain

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & -1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & -2 & 3 & -2 & 0 \\ 0 & -1 & 0 & 0 & 0 \end{pmatrix}.$$

The number of swaps is now $n_s = 1$. The multiplying factors are now $m_{3,2} = 0$, $m_{4,2} = -2$ and $m_{5,2} = -1$, and after the row subtractions the matrix modifies to

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

The multiplying factors are $m_{4,3} = 5$ and $m_{5,3} = 1$. After row subtractions the matrix modifies to

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & -1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & -9 & -3 \\ 0 & 0 & 0 & -2 & 0 \end{pmatrix}.$$

The last multiplier is $m_{5,4} = 2/9$. We obtain

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

The determinant is

$$\det A = (-1)^1(1)(1)(1)(-9)(2/3) = 6 \tag{2.12}$$

2.2 Inverse

Formally we now that $A\mathbf{x} = \mathbf{b}$ is solved by $\mathbf{x} = A^{-1}\mathbf{b}$, but we did not use this property to solve a system, since Gaussian Elimination is more efficient.

But how may we compute an inverse A^{-1}? The inverse, after all, can be used if we have to solve *many* systems as

$$A\mathbf{x}_i = \mathbf{b}_i \Rightarrow \mathbf{x}_i = A^{-1}\mathbf{b}_i, \tag{2.13}$$

where A does not depend on i. After all, once we get A^{-1}, the matrix product involves only n^2 operations.

To compute A^{-1} we may use again Gaussian Elimination. Let us solve the $i = 1, .., n$ systems

$$A\mathbf{d}^i = \mathbf{e}^i \qquad e^i_j = \delta_{j,i}. \tag{2.14}$$

Namely,

$$\mathbf{e}^1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \qquad \mathbf{e}^2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad \dots . \tag{2.15}$$

We may now check that the columns of A^{-1} are given by the vectors $\mathbf{d}^i$

$$A^{-1} = \left(\mathbf{d}^1|\mathbf{d}^2|\dots|\mathbf{d}^i|\dots|\mathbf{d}^{n-1}|\mathbf{d}^n\right), \tag{2.16}$$

or

$$A^{-1}_{j,i} = d^i_j \tag{2.17}$$

Indeed

$$(AA^{-1})_{j,i} = \sum_l a_{j,l}a^{-1}_{l,i} = \sum_l a_{j,l}d^l_i = A\mathbf{d}^i_j = \mathbf{e}^i_j = \delta_{j,i}. \tag{2.18}$$

There is anyway, from a computational point of view, no need to solve n systems, since the operations on A are always the same. It may be shown that the total number of operations is of order $\sim n^3$ ($\sim 4/3n^3$, i.e. 4 times Gaussian elimination). The trick is, while reducing A to the diagonal form, to operate the substitution not on a single $\mathbf{e}^i$, but to all of them simultaneously, i.e. to a $n \times n$ identity matrix. Namely we operate on

$$(A|\mathbb{1}), \tag{2.19}$$

and finally solve the n systems by backsubstitution to find the columns $\mathbf{d}^i$ of A^{-1}.

2.2.1 Exercise

Use the above method to invert

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 1 & 2 & 0 \end{pmatrix}.$$

Solution

We use Gaussian Elimination to simultaneously solve the 3 systems

$$A\mathbf{d}^1 = \mathbf{e}^1,$$

$$A\mathbf{d}^2 = \mathbf{e}^2,$$

$$A\mathbf{d}^3 = \mathbf{e}^3,$$

the inverse matrix will be given by

$$A^{-1} = (\mathbf{d}^1|\mathbf{d}^2|\mathbf{d}^3)$$

and we may thus identify $\mathbf{d}^j \equiv (\mathbf{A}^{-1})_j$.

We write

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

The multiplicators are

$$m_{2,1} = a_{2,1}/a_{1,1} = 1,$$

$$m_{3,1} = a_{3,1}/a_{1,1} = 1.$$

Now, when we subtract from the jth row the first row multiplied by $m_{j,1}$, we need to act also on the known terms on the left. The result of the first iteration is

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

Now we have

$$m_{3,2} = a_{3,2}/a_{2,2} = 1,$$

and the next iteration of Gaussian Elimination gives

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

We now solve

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & -1 \end{pmatrix} \mathbf{A^{-1}}_1 = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$$

to find the first column of the inverse matrix. We use back substitution

$$A^{-1}_{3,1} = 0,$$

$$A^{-1}_{2,1} = -1,$$

$$A^{-1}_{1,1} = 1 + 1 = 2.$$

Then we solve

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & -1 \end{pmatrix} \mathbf{A^{-1}}_2 = \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}$$

to find the second column of the inverse matrix. We use back substitution

$$A^{-1}_{3,2} = 1,$$

$$A^{-1}_{2,2} = 1 - 1 = 0,$$

$$A^{-1}_{1,2} = 0 - 0 = 0.$$

Finally we solve

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & -1 \end{pmatrix} \mathbf{A^{-1}}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

to find the third column of the inverse matrix. We use back substitution

$$A^{-1}_{3,3} = -1,$$

$$A^{-1}_{2,3} = 1,$$

$$A^{-1}_{1,3} = -1.$$

So that

$$A^{-1} = \begin{pmatrix} 2 & 0 & -1 \\ -1 & 0 & 1 \\ 0 & 1 & -1 \end{pmatrix}.$$

The result may be checked by computing $A^{-1}A = \mathbb{1}$.

2.2.2 Exercise

Invert

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 1 & 0 & 0 \\ -8 & 0 & 0 & 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Solution

Since $a_{1,1} = 1$ we have $m_{j,1} = a_{j,1}$. After the first iteration we obtain

$$\left(\begin{array}{ccccccc|ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -3 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -4 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 8 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right).$$

The matrix has been reduced to an upper triangular (actually diagonal) form in a single iteration. Since the systems

$$\mathbb{1}\mathbf{x} = \mathbf{b}$$

is solved by

$$\mathbf{x} = \mathbf{b},$$

we find that

$$A^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ -4 & 0 & 0 & 0 & 1 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 1 & 0 \\ -2 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

2.2.3 Exercise

Invert

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

Solution

For the first 3 iterations all the multiplicators are 0, and there is no change in the matrix or known terms. For the 4th iteration we have, since $a_{4,4} = 1$ we have $m_{j,4} = a_{j,4}$. After the 4th iteration we obtain

$$\left(\begin{array}{ccccccc|ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -3 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 4 & 0 & 0 & 1 \end{array}\right).$$

The matrix has been reduced to a diagonal form. Again since the systems

$$\mathbb{1}\mathbf{x} = \mathbf{b}$$

is solved by

$$\mathbf{x} = \mathbf{b}$$

and we find that

$$A^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & -3 & 0 & 1 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 1 \end{pmatrix}.$$

2.3 LU decomposition

Is A^{-1} the best we can do to solve many systems? Let us first consider

$$LU\mathbf{x} = \mathbf{b}. \tag{2.20}$$

Here U is an upper triangular matrix, while L is lower triangular

$$L = \begin{pmatrix} l_{11} & 0 & \dots & \dots & 0 \\ l_{21} & l_{22} & 0 & \dots & \vdots \\ \vdots & \vdots & \ddots & 0 & \vdots \\ \vdots & \vdots & \dots & \ddots & 0 \\ l_{n1} & \dots & \dots & \dots & l_{nn} \end{pmatrix}, \qquad U = \begin{pmatrix} u_{11} & u_{21} & \dots & \dots & u_{1n} \\ 0 & u_{22} & \dots & \dots & \vdots \\ \vdots & 0 & \ddots & \dots & \vdots \\ \vdots & \vdots & 0 & \ddots & \vdots \\ 0 & \dots & \dots & 0 & u_{nn} \end{pmatrix}. \tag{2.21}$$

We name

$$U\mathbf{x} = \mathbf{y}. \tag{2.22}$$

We use *forward substitution* to solve

$$L\mathbf{y} = \mathbf{b}. \tag{2.23}$$

Namely, $y_1 = b_1/l_{11}$,

$$y_i = \left[b_i - \sum_{k=1}^{i-1} l_{ik} y_k \right]. \tag{2.24}$$

Once obtained $\mathbf{y}$, we solve eq. (2.22) with backward substitution. Forward and backward substitution need n^2 operations. But when can we use this method? Let us first

2.3.1 Exercise

Solve

$$\begin{pmatrix} 1 & 0 & 1 \\ 3 & 4 & 5 \\ 1 & 4 & 12 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix},$$

knowing that

$$\begin{pmatrix} 1 & 0 & 1 \\ 3 & 4 & 5 \\ 1 & 4 & 12 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 4 & 2 \\ 0 & 0 & 9 \end{pmatrix}.$$

Solution

We solve

$$\begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \mathbf{y} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

We have

$$y_1 = 1$$

$$y_2 = -3$$

$$y_3 = 1 + 3 - 1 = 3$$

Now we solve

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 4 & 2 \\ 0 & 0 & 9 \end{pmatrix} \mathbf{x} = \mathbf{y} = \begin{pmatrix} 1 \\ -3 \\ 3 \end{pmatrix}.$$

We find

$$x_3 = 3/9 = 1/3,$$

$$x_2 = 1/4(-3 - 2/3) = -11/12,$$

$$x_1 - 1/3 = 2/3.$$

or

$$\mathbf{x} = \begin{pmatrix} 2/3 \\ -11/12 \\ 1/3 \end{pmatrix}.$$

2.3.2 Exercise

Using Gaussian Elimination solve

$$\begin{pmatrix} 1 & 0 & 1 \\ 3 & 4 & 5 \\ 1 & 4 & 12 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

Solution

The multiplicators are given by $m_{2,1} = 3$, $m_{3,1} = 1$. The extended matrix representing the system transforms to

$$\left(\begin{array}{ccc|c} 1 & 0 & 1 & 1 \\ 0 & 4 & 2 & -3 \\ 0 & 4 & 11 & 0 \end{array}\right).$$

The last multiplicator is $m_{3,2} = 1$, and finally the system becomes

$$\left(\begin{array}{ccc|c} 1 & 0 & 1 & 1 \\ 0 & 4 & 2 & -3 \\ 0 & 0 & 9 & 3 \end{array}\right).$$

This is exactly the system $U\mathbf{x} = \mathbf{y}$ that we solved above.

2.3.3 The algorithm

Performing the Gaussian elimination in the exercise above we arrived exactly to the system $U\mathbf{x} = \mathbf{y}$ that we used to solve $A\mathbf{x} = \mathbf{b}$ knowing the decomposition $A = LU$. We may also note that the multiplicators are related to the form of the matrix L. The matrix L has (obviously, since it is lower triangular) $l_{j,i} = 0$ if $j < i$; $l_{j,j} = 1$ and $l_{j,i} = m_{j,i}$ for $j > i$, where m are the multiplicators.

This did not happen by chance. It is easy to check that, if there is no swap, the matrix that performs the transformation of the matrix that deletes all the $a_{l,1}$ for $l > 1$, i.e. that performs the transformation $a_{l,i} = a_{l,i} - m_{l,1}a_{1,i}$

for $l > 1$ is

$$M^1 = \begin{pmatrix} 1 & 0 & \dots & \dots & \dots & 0 \\ -m_{21} & 1 & 0 & \dots & \vdots & \vdots \\ \vdots & 0 & \ddots & 0 & \vdots & \vdots \\ \vdots & \vdots & \dots & \ddots & \vdots & 0 \\ \vdots & \vdots & \dots & \dots & \ddots & 0 \\ -m_{n1} & 0 & \dots & \dots & \dots & 1 \end{pmatrix}. \tag{2.25}$$

The transformation $a_{li} = a_{li} - m_{lk}a_{ki}$ for $l > k$, $i \geq k$ is given by

$$M^k = \begin{pmatrix} 1 & 0 & \dots & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & \vdots & \vdots \\ \vdots & 0 & 1 & 0 & \vdots & \vdots \\ \vdots & \vdots & -m_{lk} & 1 & \vdots & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & 0 \\ 0 & 0 & -m_{nk} & 0 & \dots & 1 \end{pmatrix}. \tag{2.26}$$

Namely the matrix U which is obtained as the result of Gaussian elimination is given by applying all these matrices to the original matrix A, or

$$U = M^{n-1}M^{n-2}\dots M^k \dots M^2M^1A. \tag{2.27}$$

From this follows

$$\left(M^1\right)^{-1} \dots \left(M^k\right)^{-1} \dots \left(M^{n-1}\right)^{-1} U = A. \tag{2.28}$$

But we know, generalising exercises 2.3.1 and 2.3.2, the form of the $\left(M^k\right)^{-1}$, or

$$(M^k)^{-1} = \begin{pmatrix} 1 & 0 & \dots & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & \vdots & \vdots \\ \vdots & 0 & \ddots & 0 & \vdots & \vdots \\ \vdots & \vdots & m_{lk} & \ddots & \vdots & 0 \\ \vdots & \vdots & \vdots & \dots & \ddots & 0 \\ 0 & 0 & m_{nk} & 0 & \dots & 1 \end{pmatrix}. \tag{2.29}$$

It follows that if

$$L \equiv \left(M^1\right)^{-1} \dots \left(M^k\right)^{-1} \left(M^{n-1}\right)^{-1}, \tag{2.30}$$

L is lower diagonal (verify it), given by

$$L = \begin{pmatrix} 1 & 0 & \dots & \dots & \dots & 0 \\ m_{21} & 1 & 0 & \dots & \vdots & \vdots \\ \vdots & m_{32} & \ddots & 0 & \vdots & \vdots \\ \vdots & \vdots & m_{lk} & \ddots & \vdots & 0 \\ \vdots & \vdots & \vdots & m_{n-1,n-2} & \ddots & 0 \\ m_{n1} & m_{n2} & m_{nk} & m_{n,n-2} & m_{n,n-1} & 1 \end{pmatrix}, \tag{2.31}$$

and gives the wanted decomposition. This information is given by Gaussian Elimination, provided that there are no swaps, i.e. that the pivot is never 0.

2.3.4 Exercise

Decompose LU

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

Solution

We compute

$$m_{2,1} = a_{2,1}/a_{1,1} = 2/1 = 2,$$
$$m_{3,1} = a_{3,1}/a_{1,1} = -2/1 = -2,$$
$$m_{4,1} = a_{4,1}/a_{1,1} = 4/1 = 4.$$

As a result the first column of the L matrix is

$$\mathbf{L}_1 = \begin{pmatrix} 1 \\ 2 \\ -2 \\ 4 \end{pmatrix}.$$

The transformed matrix is

$$\begin{pmatrix} 1 & 1 & 3 & 1 \\ 0 & -2 & -5 & 1 \\ 0 & 3 & 6 & 3 \\ 0 & -3 & -12 & -4 \end{pmatrix}.$$

The second step of Gaussian Elimination gives

$$m_{3,2} = a_{3,2}/a_{2,2} = 3/(-2) = -3/2,$$

$$m_{4,2} = a_{4,2}/a_{2,2} = (-3)/(-2) = 3/2.$$

The second column of L is given by

$$\mathbf{L}_2 = \begin{pmatrix} 0 \\ 1 \\ -3/2 \\ 3/2 \end{pmatrix},$$

and the matrix transforms to

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

For the last iteration we have $m_{4,3} = (-9/2)/(-3/2) = 3$, so that the third column of L is given by

$$\mathbf{L}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 3 \end{pmatrix},$$

and the matrix transforms to

$$\begin{pmatrix} 1 & 1 & 3 & 1 \\ 0 & -2 & -5 & 1 \\ 0 & 0 & -3/2 & 9/2 \\ 0 & 0 & 0 & -19 \end{pmatrix}.$$

This matrix is the U in the LU decomposition

The last column of the L matrix is *always*

$$\mathbf{L}_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

The LU decomposition is thus given by

$$\begin{pmatrix} 1 & 1 & 3 & 1 \\ 2 & 0 & 1 & 3 \\ -2 & 1 & 0 & 1 \\ 4 & 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -2 & -3/2 & 1 & 0 \\ 4 & 3/2 & 3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 3 & 1 \\ 0 & -2 & -5 & 1 \\ 0 & 0 & -3/2 & 9/2 \\ 0 & 0 & 0 & -19 \end{pmatrix}.$$

as can be verified by matrix multiplication.

2.3.5 Dealing with swaps

What happens if there are swaps? We should keep track of them. Theoretically, the problem is treated by introducing permutation matrices, such as

$$P_{\{1,2,5,4,3,6\}} = P_{[3,5]} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.32}$$

We used two different notations here. By $P_{[3,5]}$ we mean rows 3 and 5 being swapped. By $P_{\{1,2,5,4,3,6\}}$ we mean that the 5th row is going to replace the 3rd, and the 3rd replacing the 5th. The matrix is constructed in such a way that

$$(P_{\{p_j\}})_{j,i} = \delta_{p_j,i}, \tag{2.33}$$

where $\mathbf{p}$ is the vector that appears as the index of P in the curly brackets. Thus if $B = PA$ we have

$$b_{ji} = \sum_l (P_{\mathbf{p}})_{jl} a_{li} = \sum_l \delta_{p_j,l} a_{li} = a_{p_j,i}. \tag{2.34}$$

This relation obviously applies also to vectors, $\mathbf{y} = P\mathbf{x}$, and may be also verified by applying the row by column operation.

With the $P_{[]}$ notation we considered single swaps between rows, but clearly by applying many times swaps we may obtain more complex *permutation* as in

$$P_{[1,4]} P_{[1,2]} = P_{\{4,1,3,2\}}, \tag{2.35}$$

as may be verified by

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}. \tag{2.36}$$

Given the meaning of the permutation it easy to check that the transpose of a permutation is its inverse

$$P^T P = P P^T = \mathbb{1}. \tag{2.37}$$

We recall that the transpose is defined by

$$A^T_{j,i} = A_{i,j}. \tag{2.38}$$

Basically, applying a permutation and then its transpose, means just sending a row somewhere, and then back to the original position.

We know that by swapping a few times rows, we may reduce a non-singular matrix to the LU form. So, if we have to solve $A\mathbf{x} = \mathbf{b}$, and we know that there is a permutation P such that $PA = LU$, we just need to solve $PA\mathbf{x} = P\mathbf{b}$, knowing that $PA = LU$. Here LU is given as always by the Gaussian Elimination process, while P is given by the sequence of swap. There is no need to perform matrix multiplications or just use the matrix P in explicit form, since the whole information is in the the index-vector $\mathbf{p}$.

An important point is, while obtaining the matrix L (the matrix of multiplicators), to remember to swap already computed rows also in it, and not only in U.

2.3.6 Exercise

Solve, using LU decomposition, the system

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 2 & 1 & 1 & 0 \\ 0 & 3 & 2 & 1 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 5 \\ 2 \\ 3 \\ 5 \end{pmatrix}.$$

Solution

We initialise the swap vector as

$$\mathbf{s} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}.$$

We compute

$$m_{2,1} = a_{2,1}/a_{1,1} = 1/1 = 1,$$
$$m_{3,1} = a_{3,1}/a_{1,1} = 2/1 = 2$$
$$m_{4,1} = a_{4,1}/a_{1,1} = 0/1 = 0$$

and as a result the first column of the L matrix is

$$\mathbf{L}_1 = \begin{pmatrix} 1 \\ 1 \\ 2 \\ 0 \end{pmatrix}$$

The transformed matrix is

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & -1 \\ 0 & -1 & 1 & 2 \\ 0 & 3 & 2 & 1 \end{pmatrix}$$

We have now $a_{2,2} = 0$, so that a swap is needed. We swap the rows 2 and 3, and obtain the matrix

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & -1 & 1 & 2 \\ 0 & 0 & 0 & -1 \\ 0 & 3 & 2 & 1 \end{pmatrix}$$

The swap vector modifies to

$$\mathbf{s} = \begin{pmatrix} 1 \\ 3 \\ 2 \\ 4 \end{pmatrix}$$

and the first column of L is modified to

$$\mathbf{L}_1 = \begin{pmatrix} 1 \\ 2 \\ 1 \\ 0 \end{pmatrix}$$

The second step of Gaussian Elimination gives now

$$m_{3,2} = a_{3,2}/a_{2,2} = 0/(-1) = 0$$

$$m_{4,2} = a_{4,2}/a_{2,2} = 3/(-1) = -3$$

The second column of L is given by

$$\mathbf{L}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ -3 \end{pmatrix}$$

and the matrix transforms to

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & -1 & -1 & -2 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & -5 \end{pmatrix}$$

We have now $a_{3,3} = 0$, and we need to swap the 3rd and 4th row, so that

$$\mathbf{s} = \begin{pmatrix} 1 \\ 3 \\ 4 \\ 2 \end{pmatrix}$$

The matrix is now

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & -1 & -1 & -2 \\ 0 & 0 & -1 & -5 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

but we need also to keep track of the swaps in the matrix L, and re-write

$$\mathbf{L}_1 = \begin{pmatrix} 1 \\ 2 \\ 0 \\ 1 \end{pmatrix}$$

and

$$\mathbf{L}_2 = \begin{pmatrix} 0 \\ 1 \\ -3 \\ 0 \end{pmatrix}$$

Now we have $m_{4,3} = 0$ and the third column of L is given by

$$\mathbf{L}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

the matrix is unchanged

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & -1 & -1 & -2 \\ 0 & 0 & -1 & -5 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

This matrix is the U in the LU decomposition (but not of the original matrix, of the matrix obtained by swapping a few rows in the original one).

The last column of the L matrix is as always

$$\mathbf{L}_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

The LU decomposition is (notice the permutation)

$$PA = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & 1 & 0 \\ 0 & 3 & 2 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & -3 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & -1 & -1 & -2 \\ 0 & 0 & -1 & -5 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

The matrix P is given by

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

To solve a system by LU decomposition we first define

$$\mathbf{y} \equiv U\mathbf{x}$$

where $\mathbf{x}$ is the solution of the system.

We now find $\mathbf{y}$ by solving

$$L\mathbf{y} = \mathbf{b}$$

where, since

$$\mathbf{s} = \begin{pmatrix} 1 \\ 3 \\ 4 \\ 2 \end{pmatrix}$$

we have

$$\mathbf{b} = \begin{pmatrix} 5 \\ 3 \\ 5 \\ 2 \end{pmatrix}$$

The solution is found by forward substitution. Let us recall that the system is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & -3 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \mathbf{y} = \begin{pmatrix} 5 \\ 3 \\ 5 \\ 2 \end{pmatrix}$$

We have

$$y_1 = 5$$

$$y_2 = 3 - 10 = -7$$

$$y_3 = 5 - 21 = -16$$

$$y_4 = 2 - 5 = -3$$

Finally we solve $U\mathbf{x} = \mathbf{y}$ or

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & -1 & -1 & -2 \\ 0 & 0 & -1 & -5 \\ 0 & 0 & 0 & -1 \end{pmatrix} \mathbf{y} = \begin{pmatrix} 5 \\ -7 \\ -16 \\ -3 \end{pmatrix}$$

This is solved by backward substitution, namely

$$x_4 = -(-3) = 3$$

$$x_3 = -(-16 + 15) = 1$$

$$x_2 = -(-7 + 1 + 6) = 0$$

$$x_1 = 5 - 1 - 3 = 1$$

The solution is thus

$$\mathbf{x} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 3 \end{pmatrix}.$$

Chapter 3

Numerical Linear Algebra: Norms and iterative methods

3.1 Introduction

Let us consider

$$A\mathbf{x} = \mathbf{b}, \tag{3.1}$$

A non-singular. We know the solution is $\mathbf{x} = A^{-1}\mathbf{b}$, but in general, due to numerical round-off, we will have access to an approximated solution, $\tilde{\mathbf{x}} \neq \mathbf{b}$. Thus, the *residual vector*

$$\mathbf{r} = \mathbf{b} - A\tilde{\mathbf{x}}, \tag{3.2}$$

will be non-zero. Given $\mathbf{r}$, we may solve

$$A\delta\mathbf{x} = \mathbf{r}, \tag{3.3}$$

so that

$$\delta\mathbf{x} = A^{-1}A\delta\mathbf{x} = A^{-1}\mathbf{b} - A^{-1}A\tilde{\mathbf{x}} = \mathbf{x} - \tilde{\mathbf{x}}. \tag{3.4}$$

Of course we will have access again only to an approximate solution $\tilde{\delta\mathbf{x}}$ for which we may assume

$$\tilde{\delta\mathbf{x}} \approx \mathbf{x} - \tilde{\mathbf{x}}. \tag{3.5}$$

Many questions arise.

- If the residual $\mathbf{r}$ is small, does it mean that also the error $\tilde{\delta\mathbf{x}}$ is small?
- What does it mean, first of all, that a vector is small?

- If we rename $\mathbf{x}_0 \equiv \tilde{\mathbf{x}}$ and $\delta\mathbf{x}_0 \equiv \tilde{\delta\mathbf{x}}$ and use an iterative process $\mathbf{x}_{n+1} = \mathbf{x}_n + \delta\mathbf{x}_n$, does the sequence converge to $\mathbf{x} = A^{-1}\mathbf{b}$?

- And what does "convergence" mean for vectors?

3.2 Metrics and norms

3.2.1 Norm and distance

Let us first recall some notions from the familiar $\mathbb{R}$ case. We may say that our numerical solution $\tilde{x}$ is close to the exact one x, or that the error $\delta x = x - \tilde{x}$ is small if

$$\varepsilon = \frac{|x - \tilde{x}|}{|x|} = \frac{|\delta x|}{|x|} \ll 1, \tag{3.6}$$

and typically we will compare $|\varepsilon|$ to a threshold, such as a required precision.

Here we are actually using two, although related, concepts: the distance between two points on $\mathbb{R}$, $|x - y|$, and the "size" (norm) of a point on $\mathbb{R}$, $|x|$. More precisely, we are using the norm $|\ |$ to define a distance between two points as the size of their difference. As we remember from calculus, this is also related to the concepts of convergence. In an informal way, let us consider the transformation

$$x_{n+1} = Tx_n \tag{3.7}$$

where

$$Tx \equiv \alpha x + \beta, \qquad |\alpha| < 1. \tag{3.8}$$

We have

$$|Tx - Ty| = |\alpha x + \beta - \alpha y - \beta| = |\alpha||x - y|, \tag{3.9}$$

so that

$$|T^n x - T^n y| = |\alpha|^n |x - y|. \tag{3.10}$$

As a consequence

$$|x_{n+1} - x_n| = |\alpha|^n |x_1 - x_0|, \tag{3.11}$$

and

$$\lim_{n\to\infty} |x_{n+1} - x_n| = 0. \tag{3.12}$$

We say that the sequence x_n converges because the distance between its elements goes to zero[1]. The limit is x such that $x = Tx$, and it is also called a *fixed point*. As an example, we may use $\alpha = 1/2$, $\beta = 1$ and $x_0 = 0$. We have

$$x_1 = 1, \qquad x_2 = \frac{3}{2}, \qquad x_3 = \frac{7}{4}, \qquad x_4 = \frac{15}{8}, \ldots, \tag{3.13}$$

converging to the fixed point $x = 2$.

Can we do the same for vectors?

3.2.2 Norms and distances between vectors

In Euclidean space[2] points are given by 3 coordinates $\mathbf{x} = \{x, y, z\}$ (with respect to a specific frame with given origin and orientation). The distance between two points is

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}, \tag{3.14}$$

as given by Pythagoras' theorem. This is at the same time the size or norm of the vector $\mathbf{x} - \mathbf{y}$.

The n dimensional versions of these formulae define Euclidean norm and distance as

$$\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{n} x_i^2}, \tag{3.15}$$

and

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|. \tag{3.16}$$

Mathematicians found anyway that many definition of distance (or metric) and norm are possible. A metric on a vector space is an application that satisfies the following

$$d(\mathbf{x}, \mathbf{y}) \in \mathbb{R},\ d(\mathbf{x}, \mathbf{y}) \geq 0, \tag{3.17}$$

$$d(\mathbf{x}, \mathbf{y}) = 0 \Leftrightarrow \mathbf{x} = \mathbf{y}, \tag{3.18}$$

$$d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}), \tag{3.19}$$

$$d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y}) \qquad \forall \mathbf{z}. \tag{3.20}$$

[1] This is an extremely informal proof. We will later show that the sequence is a Cauchy one, and thus converges in a complete set as $\mathbb{R}$.

[2] The usual geometry of our physical space, at least ignoring any relativistic effect.

The third property is called triangular inequality, and says that by passing through a third point you may not find a shorter way than the one going directly to $\mathbf{y}$ (Figure 3.1).

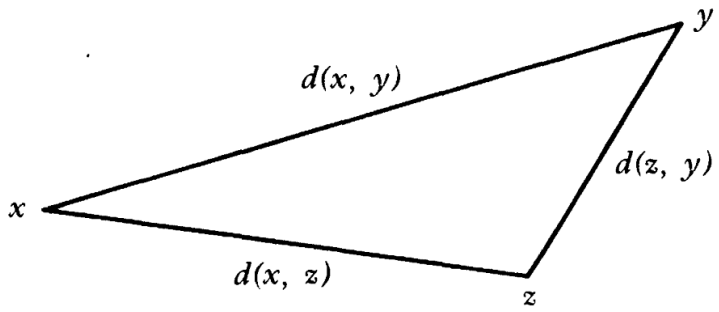

Figure 3.1: Triangular inequality, figure taken by *Functional Analysis* by Kreysig.

The metric *may* be defined through a norm[3], i.e.

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|, \tag{3.21}$$

where the norm satisfies

$$\|\mathbf{x}\| \in \mathbb{R},\ \|\mathbf{x}\| \geq 0, \tag{3.22}$$

$$\|\mathbf{x}\| = 0 \Leftrightarrow \mathbf{x} = 0, \tag{3.23}$$

$$\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|, \tag{3.24}$$

$$\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|. \tag{3.25}$$

$\|\mathbf{x} - \mathbf{y}\|$ clearly defines a metric, for example we have

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\| = \|\mathbf{x} - \mathbf{z} + \mathbf{z} - \mathbf{y}\| \leq \|\mathbf{x} - \mathbf{z}\| + \|\mathbf{z} - \mathbf{y}\| = d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y}). \tag{3.26}$$

It may be shown that a way to define a norm is to use the following formula

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} \qquad p \geq 1. \tag{3.27}$$

[3]And a norm *may* be defined through a inner product.

For $p = 1$ we have the sum of the absolute values of all components, while $p = 2$ gives the Euclidean norm. We may define also

$$\|\mathbf{x}\|_\infty = \max_i |x_i|. \tag{3.28}$$

For example, for the vector $\mathbf{x} = \{1, 1, 1\}$ we have

$$\|\mathbf{x}\|_1 = 3, \qquad \|\mathbf{x}\|_2 = \sqrt{3}, \qquad \|\mathbf{x}\|_\infty = 1. \tag{3.29}$$

Nevertheless there is a theorem that tells us that on $\mathbb{R}^n$, with any finite n, all norms are equivalent, meaning that given a succession $\mathbf{x}_n$, if there is a norm such that $\lim_{n\to\infty}\|\mathbf{x}_n\|_* = 0$, then $\lim_{n\to\infty}\|\mathbf{x}_n\| = 0$ for all norms. Equally, if there is a norm such that $\lim_{n\to\infty}\|\mathbf{x}_n\|_* = \infty$, then $\lim_{n\to\infty}\|\mathbf{x}_n\| = \infty$ for all norms. An intuitive way of understanding this theorem is to visualise the *unit balls* (sets of points with norm lower or equal to 1) for different norms (Fig. 3.2). Clearly if a ball shrinks to zero, all balls will follow.

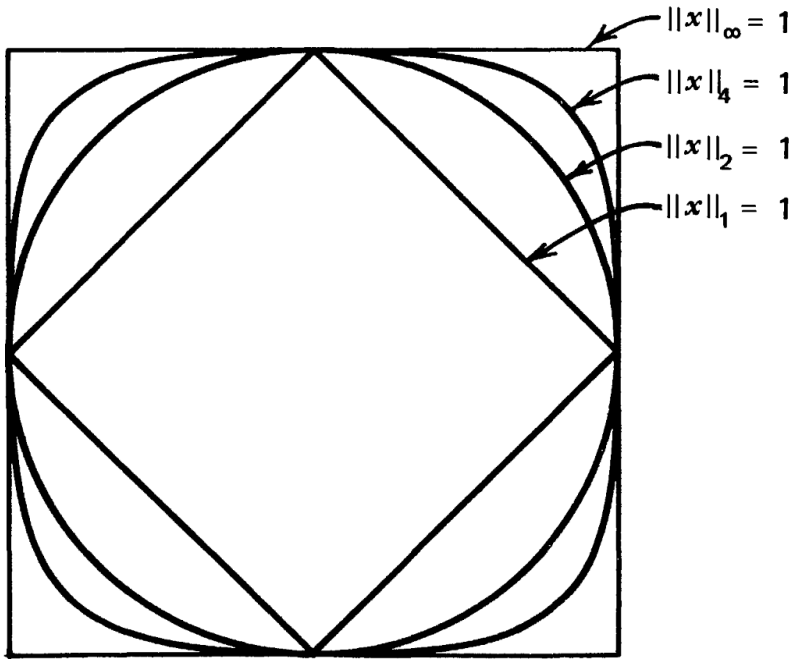

Figure 3.2: Unit balls for different norms, $n = 2$, figure taken by *Functional Analysis* by Kreysig.

3.3 Norms on matrices

A $n \times n$ matrix may be considered as a n^2 component vector, and we may use a vector norm to define a norm on matrices. Anyway, since matrices are also operators on which we may define a multiplication rule, it is convenient to look for a matrix specific norm that satisfies also

$$\|BA\| \leq \|B\|\|A\|. \tag{3.30}$$

Such a norm on operators may be defined using a norm on vectors. We may define

$$\|A\| \equiv \max_{\mathbf{x}} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}. \tag{3.31}$$

Since norms and matrices are linear,

$$\|A\| = \max_{\mathbf{x}} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \max_{\mathbf{x}} \left\| \frac{A\mathbf{x}}{\|\mathbf{x}\|} \right\| = \max_{\mathbf{x}} \left\| A\frac{\mathbf{x}}{\|\mathbf{x}\|} \right\|. \tag{3.32}$$

So, we are actually checking only norm 1 vectors[4], so an alternative definition is

$$\|A\| \equiv \max_{\|\mathbf{x}\|=1} \|A\mathbf{x}\|. \tag{3.33}$$

From the definition we clearly have

$$\|A\mathbf{x}\| \leq \|A\|\|\mathbf{x}\|, \tag{3.34}$$

so, since $\|BA\mathbf{x}\| \leq \|B\|\|A\mathbf{x}\|$, this norm satisfies eq. (3.30), since

$$\|BA\| = \max_{\mathbf{x}} \frac{\|BA\mathbf{x}\|}{\|\mathbf{x}\|} \leq \|B\| \max_{\mathbf{x}} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \|B\|\|A\|. \tag{3.35}$$

It is easy to show that

$$\|A\|_\infty = \max_j \sum_i |a_{j,i}|, \tag{3.36}$$

or the maximum over rows of the sum of absolute values.

$$\|A\|_\infty = \max_{\|\mathbf{x}\|_\infty=1} \|A\mathbf{x}\|_\infty = \max_{\|\mathbf{x}\|_\infty=1} \left(\max_j \left| \sum_i a_{j,i}x_i \right| \right) \leq \max_{\|\mathbf{x}\|_\infty=1} \left(\max_j \sum_i |a_{j,i}x_i| \right) \leq \max_j \sum_i |a_{j,i}|. \tag{3.37}$$

[4] Since $\left\| \frac{\mathbf{x}}{\|\mathbf{x}\|} \right\| = \frac{\|\mathbf{x}\|}{\|\mathbf{x}\|} = 1$.

The last inequality follows since $|x_i| \leq 1$ for $\|\mathbf{x}\|_\infty = 1$ vectors. But if we choose $x_i = 1$ if $a_{j^*,i} \geq 0$, $x_i = -1$ if $a_{j^*,i} < 0$ for the particular $j^* = \max_j \sum_i |a_{j,i}|$, we obtain

$$\|A\|_\infty \geq \max_j \sum_i |a_{j,i}|, \tag{3.38}$$

and thus prove eq. (3.36). It is also possible to show

$$\|A\|_1 = \max_i \sum_j |a_{j,i}|, \tag{3.39}$$

or the maximum over columns of the sum of absolute values. For the 2 norm it is more difficult. We first define the matrix $A^T A$,[5] and look for its eigenvalues λ_i (which are real and positive). We have then

$$\|A\|_2 = \max_i \sqrt{\lambda_i}. \tag{3.40}$$

Without entering to much in eigenvalue and eigenvector theory (more on this later), it is easy to understand that these are related to the norm. Let us recall that if $\mathbf{v}_{\lambda_i}$ is a eigenvector of A with eigenvalue λ_i, we have

$$A\mathbf{v}_{\lambda_i} = \lambda_i \mathbf{v}_{\lambda_i}. \tag{3.41}$$

Let us call the *spectral radius* of A

$$\rho(A) = \max_i |\lambda_i|. \tag{3.42}$$

Choosing the maximum λ^* we have

$$\rho(A)\|\mathbf{v}_{\lambda^*}\| = |\lambda^*|\|\mathbf{v}_{\lambda^*}\| = \|\lambda^* \mathbf{v}_{\lambda^*}\| = \|A\mathbf{v}_{\lambda^*}\| \leq \|A\|\|\mathbf{v}_{\lambda^*}\|, \tag{3.43}$$

and thus

$$\rho(A) \leq \|A\|. \tag{3.44}$$

3.4 Contraction theorem

We may now state a useful theorem. Let us assume $\|A\| = \alpha < 1$ is a matrix on $\mathbb{R}^k$, and consider $T\mathbf{x} \equiv A\mathbf{x} + \mathbf{b}$. We may show that $\mathbf{x}_{n+1} = T\mathbf{x}_n$ converges to the only fixed point $T\mathbf{x}^* = \mathbf{x}^*$.

[5] We are here assuming A it's a real matrix. For complex ones, the *adjoint* (complex conjugate transpose) is used.

First of all, we have

$$d(T\mathbf{x}, \mathbf{y}) = \|T\mathbf{x} - T\mathbf{y}\| = \|A(\mathbf{x} - \mathbf{y})\| \leq \|A\|\|\mathbf{x} - \mathbf{y}\| = \alpha d(x, y). \quad (3.45)$$

A transformation with this property is called a contraction. We may now show

$$d(\mathbf{x}_{n+1}, \mathbf{x}_n) \leq \alpha d(\mathbf{x}_n, \mathbf{x}_{n-1}) \leq \ldots \leq \alpha^n d(\mathbf{x}_1, \mathbf{x}_0). \quad (3.46)$$

From this, if $m > n$, and defining $\beta = d(\mathbf{x}_1, \mathbf{x}_0)$ we have

$$d(\mathbf{x}_m, \mathbf{x}_n) \leq \sum_{i=0}^{m-n-1} d(\mathbf{x}_{n+i+1}, \mathbf{x}_{n+i}) \leq \sum_{i=0}^{m-n-1} \alpha^{n+i}\beta = \beta\alpha^n \sum_{i=0}^{m-n-1} \alpha^i = \beta\alpha^n \frac{1-\alpha^{m-n}}{1-\alpha} < \beta\alpha^n \frac{1}{1-\alpha}. \quad (3.47)$$

By choosing an high enough n we may make $d(\mathbf{x}_m, \mathbf{x}_n)$ as small as we like. This is a Cauchy sequence, which in a complete space (as $\mathbb{R}^k$ is) converges to $\mathbf{x}^*$. Now, sending $m \to \infty$ in 3.47 we have

$$d(\mathbf{x}^*, \mathbf{x}_n) \leq \beta\alpha^n \frac{1}{1-\alpha}. \quad (3.48)$$

We have then

$$d(\mathbf{x}^*, T\mathbf{x}^*) \leq d(\mathbf{x}^*, \mathbf{x}_{n+1}) + d(T\mathbf{x}^*, \mathbf{x}_{n+1}) = d(\mathbf{x}^*, \mathbf{x}_{n+1}) + d(T\mathbf{x}^*, T\mathbf{x}_n) \leq \frac{2\beta\alpha^{n+1}}{1-\alpha}. \quad (3.49)$$

Again, choosing n high enough we may send the right term to zero, showing $\mathbf{x}^* = T\mathbf{x}^*$. Finally, let us assume two fixed points exist, $\mathbf{x}^*$ and $\mathbf{y}^*$. We have

$$d(\mathbf{x}^*, \mathbf{y}^*) = d(T\mathbf{x}^*, T\mathbf{y}^*) \leq \alpha d(\mathbf{x}^*, \mathbf{y}^*) < d(\mathbf{x}^*, \mathbf{y}^*) \Rightarrow \mathbf{x}^* = \mathbf{y}^*, \quad (3.50)$$

which proves the theorem.

3.5 Iterative refinement

As discussed above, let us call $\mathbf{x}_0$ the solution obtained using Gaussian Elimination for

$$A\mathbf{x} = \mathbf{b}, \quad (3.51)$$

name

$$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0, \quad (3.52)$$

and $\delta\mathbf{x}_0$ the numerical solution of

$$\delta\mathbf{x}_0 = A^{-1}\mathbf{r}_0 \approx \mathbf{x} - \mathbf{x}_0. \tag{3.53}$$

Let us assume we know that

$$\frac{\|\mathbf{r}_0\|}{\|\mathbf{b}\|} \ll 1, \tag{3.54}$$

is it true that

$$\frac{\|\delta\mathbf{x}_0\|}{\|\mathbf{x}\|} \ll 1 \qquad ? \tag{3.55}$$

Let us first write

$$\frac{\|\delta\mathbf{x}_0\|}{\|\mathbf{x}\|} \leq \frac{\|A^{-1}\|\|\mathbf{r}_0\|}{\|\mathbf{x}\|}, \tag{3.56}$$

and remember

$$\|\mathbf{b}\| \leq \|A\|\|\mathbf{x}\| \Rightarrow \frac{1}{\|\mathbf{x}\|} \leq \frac{\|A\|}{\|\mathbf{b}\|}, \tag{3.57}$$

to obtain

$$\frac{\|\delta\mathbf{x}_0\|}{\|\mathbf{x}\|} \leq \|A^{-1}\|\|A\|\frac{\|\mathbf{r}_0\|}{\|\mathbf{b}\|}. \tag{3.58}$$

We name

$$\kappa(A) = \|A^{-1}\|\|A\|, \tag{3.59}$$

the *conditioning number* of A. It may be very high, as a trivial[6] example may show

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 10^{-8} \end{pmatrix} \|A\|_{1,2,\infty} = 1 \qquad A^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & 10^{8} \end{pmatrix} \|A\|_{1,2,\infty} = 10^8$$
$$\Rightarrow \kappa(A) = 10^8. \tag{3.60}$$

Eq. (3.58) shows that a high κ may be a problem.

An estimate for κ (without computing the inverse) may be obtained as follows. If computations are performed using t digits, the following estimate holds

$$\|\mathbf{r}_0\| \approx 10^{-t}\|A\|\|\mathbf{x}_0\|. \tag{3.61}$$

We then have, from $\delta\mathbf{x}_0 = A^{-1}\mathbf{r}_0$,

$$\|\delta\mathbf{x}_0\| \leq \|A^{-1}\|\|\mathbf{r}_0\| \approx 10^{-t}\|A^{-1}\|\|A\|\|\mathbf{x}_0\|, \tag{3.62}$$

[6] and obviously non-problematic from a numerical standpoint, since diagonal

or

$$\|\delta \mathbf{x}_0\| \approx 10^{-t} \|\kappa(A)\mathbf{x}_0\|. \tag{3.63}$$

This equation tells us that *the numerical solution of a system is reliable only if the conditioning number is considerably lower than the used computing precision*, or

$$\kappa(A) \ll 10^t. \tag{3.64}$$

From eq. (3.63) we have

$$\kappa(A) \approx 10^t \frac{\|\delta \mathbf{x}_0\|}{\|\mathbf{x}_0\|}. \tag{3.65}$$

The equation above assumes that $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$ *is computed using an higher precision than the one used to solve for* $\mathbf{x}_0$. For example, $\mathbf{x}_0$ may be computed using floats (8 digits) and $\mathbf{r}_0$ computed using doubles (16 digits).

In general, the sequence $\mathbf{x}_{n+1} = \mathbf{x}_n + \delta \mathbf{x}_n$ should approach the exact solution until precision 10^{-t} is reached. Each iteration should improve the solution of $t - \mathrm{Log}_{10}\kappa(A)$, provided that $\kappa(A)^{-1} > 10^{-t}$. This process is called *iterative refinement*.

3.5.1 Exercise

Compute the conditioning number of the matrix

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 2 & 0 & 1 \end{pmatrix}$$

using the infinite norm.

Solution

Using our algorithm for matrix inversion (see Chapter 2) we find

$$A^{-1} = \begin{pmatrix} -1/2 & -1/2 & 1 \\ 1/2 & -1/2 & 0 \\ 1 & 1 & -1 \end{pmatrix}$$

Looking for the maximum over rows, we find $\|A\|_\infty = 3$, $\|A^{-1}\|_\infty = 3$, and thus $\kappa(A) = 9$.

3.5.2 Exercise

Compute the conditioning number of the matrix

$$A = \begin{pmatrix} 0.3601 & 0.4799 \\ 0.4799 & 0.6401 \end{pmatrix}$$

using the infinite norm.

Solution

Using our algorithm for matrix inversion (see Chapter 2) we find

$$A^{-1} \approx \begin{pmatrix} 3265.82 & -2448.47 \\ -2448.47 & 1837.24 \end{pmatrix}$$

Looking for the maximum over rows, we find $\|A\|_\infty = 1.12$, $\|A^{-1}\|_\infty \approx 5714.29$, and thus $\kappa(A) \approx 6400$.

3.6 Iterative methods

3.6.1 Jacobi method

Iterative refinement uses as first step Gaussian Elimination, a process that would be correct if infinite information could be used, and uses iterations to improve the numerical accuracy. For some large systems with many 0 entries, it may be useful to use a completely iterative procedure.

As usual, our task is to solve

$$A\mathbf{x} = \mathbf{b}, \tag{3.66}$$

but this time we start from an arbitrary vector $\mathbf{x}^{(0)}$. If we write eq. (3.66) in components, we have

$$\sum_{i=1}^{n} a_{j,i}x_i = b_j \Rightarrow a_{j,j}x_j + \sum_{i\neq j} a_{j,i}x_i = b_j \Rightarrow x_j = \frac{1}{a_{j,j}}\left(b_j - \sum_{i\neq j} a_{j,i}x_i\right). \tag{3.67}$$

The idea of the Jacobi method is to improve on the current estimate $\mathbf{x}^{(k)}$ using eq. (3.67), i.e.

$$x_j^{(k+1)} = \frac{1}{a_{j,j}}\left(b_j - \sum_{i\neq j} a_{j,i}x_i^{(k)}\right). \tag{3.68}$$

When does the method converge? Any square matrix may be written as $A = D + L + U$. Here D is a matrix that is zero everywhere but on the diagonal, where it has $d_{j,j} = a_{j,j}$. L is zero everywhere but under the diagonal, where $l_{j,i} = a_{j,i}$, while U is non-zero only over the diagonal, where $l_{j,i} = a_{j,i}$. Namely

$$d_{j,i} = a_{j,i}\delta_{j,i}, \tag{3.69}$$

$$l_{j,i} = \begin{cases} a_{j,i} & \text{if } j > i \\ 0 & \text{if } j \leq i \end{cases}, \tag{3.70}$$

$$u_{j,i} = \begin{cases} a_{j,i} & \text{if } j < i \\ 0 & \text{if } j \geq i \end{cases}, \tag{3.71}$$

and $A = D + U + L$ trivially follows from the matrix addition rule[7].

Eq. (3.68) may then be written in matrix form as

$$\mathbf{x}^{(k+1)} = D^{-1}\left(-L - U\right)\mathbf{x}^{(\mathbf{k})} + D^{-1}\mathbf{b}, \tag{3.72}$$

as can be easily verified writing the latter equation component by component and recalling that the inverse of a diagonal matrix is again diagonal with $d_{jj}^{-1} = 1/d_{jj}$. Eq. (3.72) is in the form required by the theorem of section 3.4 ($D^{-1}\mathbf{b}$ being the constant term), so the procedure will converge to a fix point if we find a matrix norm with $\|D^{-1}\left(-L - U\right)\| < 1$. Such a fixed point satisfies eq. (3.67) and thus is a solution of our system.

We may easily prove that if the matric A is *diagonally dominant*, i.e. if

$$|a_{j,j}| > \sum_{i \neq j} |a_{j,i}| \quad \forall j, \tag{3.73}$$

then the sequence converges. Let us use the $\| \ \|_\infty$ norm. Then

$$\left[D^{-1}\left(-L - U\right)\right]_{j,i} = \begin{cases} 0 & \text{if } j = i \\ -\frac{a_{j,i}}{a_{j,j}} & \text{if } j \neq i \end{cases}. \tag{3.74}$$

As a result

$$\|D^{-1}\left(-L - U\right)\|_\infty = \max_j \frac{\sum_{i \neq j} |a_{j,i}|}{|a_{j,j}|} < 1. \tag{3.75}$$

Eq. (3.48) may be used to estimate the error.

[7]This decomposition, based on a sum, should not be confused with the LU decomposition, based on the matrix product.

3.6.2 Gauss-Seidel method and SOR

In eq. (3.68) we update the new $\mathbf{x}^{(k+1)}$ vector using components of $\mathbf{x}^{(k)}$, even if the components of the new vector are available. If we try to use the new information as soon as it is available we have the Gauss-Seidel method

$$x_j^{(k+1)} = \frac{1}{a_{jj}} \left(b_j - \sum_{i<j} a_{ji} x_i^{k+1} - \sum_{i>j} a_{ji} x_i^{(k)} \right). \tag{3.76}$$

In matrix form

$$(D + L)\, \mathbf{x}^{(k+1)} = -U\mathbf{x}^{(k)} + \mathbf{b}, \tag{3.77}$$

or

$$\mathbf{x}^{(k+1)} = -(D + L)^{-1} U\mathbf{x}^{(k)} + (D + L)^{-1}\mathbf{b}. \tag{3.78}$$

It may be shown that also this method converges for diagonally dominant matrices (it is important to notice that some matrices may be done diagonally dominant by rearranging rows).

We may finally consider Successive Over Relaxation methods. Eq. (3.76) may be re-written as

$$x_j^{(k+1)} = \frac{1}{a_{jj}} \left(b_j - \sum_{i<j} a_{ji} x_i^{(k+1)} - \sum_{i>j} a_{ji} x_i^{(k)} - a_{jj} x_i^{(k)} + a_{jj} x_i^{(k)} \right), \tag{3.79}$$

or

$$x_j^{(k+1)} = \frac{1}{a_{jj}} \left(b_j - \sum_{i<j} a_{ji} x_i^{(k+1)} - \sum_{i>j} a_{ji} x_i^{(k)} - a_{jj} x_i^{(k)} \right) + x_i^{(k)}. \tag{3.80}$$

The term in the parenthesis is the residual. So the method may be generalised to

$$x_j^{(k+1)} = \omega \frac{r_j}{a_{jj}} + x_j^{(k)}, \tag{3.81}$$

where $\omega = 1$ gives the Gauss-Seidel method, while $\omega > 1$ ($\omega < 1$) gives and over (under) relaxation method.

3.6.3 Exercise

Starting from the seed

$$\mathbf{x}_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix},$$

use the Jacobi method to find the solution of

$$\begin{pmatrix} 1 & 2 & 5 \\ 0 & -3 & 1 \\ 5 & 1 & 1 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 8 \\ -2 \\ 7 \end{pmatrix}.$$

Put the system in a form that assures convergence, perform 3 iterations and compare to the exact result

$$\mathbf{x} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Solution

First of all we have to put the system in a form that assures convergence. The first row has the component of maximum modulus on the third column, while the third row has the maximum modulus on the first column. If we swap the first and third row (including the known terms) we have

$$\begin{pmatrix} 5 & 1 & 1 \\ 0 & -3 & 1 \\ 1 & 2 & 5 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 7 \\ -2 \\ 8 \end{pmatrix}.$$

Now the matrix is diagonally dominant, since

$$|5| > |1| + |1|,$$

$$|-3| > |0| + |1|,$$

$$|5| > |1| + |2|,$$

and we know that if

$$|a_{j,j}| > \sum_{i \neq j} |a_{j,i}|$$

then the Jacobi method converges to the solution of

$$A\mathbf{x} = \mathbf{b}.$$

The first iteration gives

$$\mathbf{x}_1 = \begin{pmatrix} 7/5 \\ 2/3 \\ 8/5 \end{pmatrix}.$$

The second iteration gives

$$\mathbf{x}_2 = \begin{pmatrix} (7 - 2/3 - 8/5)/5 \\ (-2 - 8/5)/(-3) \\ (8 - 7/5 - 2(2/3))/5 \end{pmatrix} \approx \begin{pmatrix} 0.946667 \\ 1.2 \\ 1.05333 \end{pmatrix}.$$

The third iteration gives

$$\mathbf{x}_3 \approx \begin{pmatrix} 0.949333 \\ 1.01778 \\ 0.930667 \end{pmatrix}.$$

The error with respect to the correct solution is

$$\varepsilon = \frac{\|\mathbf{x_3} - \mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty}.$$

We have

$$\|\mathbf{x}\|_\infty = 1,$$

and

$$\mathbf{x_3} - \mathbf{x} \approx \begin{pmatrix} -0.05 \\ 0.02 \\ -0.07 \end{pmatrix},$$

and thus

$$\varepsilon \approx 0.07.$$

3.6.4 Exercise

Starting from the seed

$$\mathbf{x}_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

Use the Gauss-Seidel method to find the solution of

$$\begin{pmatrix} 1 & 2 & 5 \\ 0 & -3 & 1 \\ 5 & 1 & 1 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 8 \\ -2 \\ 7 \end{pmatrix}.$$

Put the system in a form that assures convergence, perform 3 iterations and compare to the exact result

$$\mathbf{x} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Solution

First of all we have to put the system in a form that assures convergence. We proceed in the same way as for question 3.6.3. The first row has the component of maximum modulus on the third column, while the third row has the maximum modulus on the first column. If we swap the first and third row (including the known terms) we have

$$\begin{pmatrix} 5 & 1 & 1 \\ 0 & -3 & 1 \\ 1 & 2 & 5 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 7 \\ -2 \\ 8 \end{pmatrix}.$$

Now the matrix is diagonally dominant, since

$$|5| > |1| + |1|,$$
$$|-3| > |0| + |1|,$$
$$|5| > |1| + |2|,$$

and we know that a theorem assures that if

$$|a_{j,j}| > \sum_{i \neq j} |a_{j,i}|$$

then the Gauss-Seidel converges to the solution of

$$A\mathbf{x} = \mathbf{b}.$$

The first iteration gives

$$\mathbf{x}_1 = \begin{pmatrix} 7/5 \\ 2/3 \\ (8 - 7/5 - 2(2/3))/5 = 79/75 \approx 1.05333 \end{pmatrix}.$$

The second iteration gives

$$\mathbf{x}_2 \approx \begin{pmatrix} 1.056 \\ 1.01778 \\ 0.981689 \end{pmatrix}.$$

The third iteration gives

$$\mathbf{x}_3 \approx \begin{pmatrix} 1.00011 \\ 0.993896 \\ 1.00242 \end{pmatrix}.$$

The error with respect to the correct solution is

$$\varepsilon = \frac{\|\mathbf{x_3} - \mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty}.$$

We have

$$\|\mathbf{x}\|_\infty = 1,$$

and

$$\mathbf{x_3} - \mathbf{x} \approx \begin{pmatrix} 1 \cdot 10^{-4} \\ -6 \cdot 10^{-3} \\ 2 \cdot 10^{-3} \end{pmatrix},$$

and thus

$$\varepsilon \approx 0.006.$$

Chapter 4

Eigenvalues and the like

4.1 Egeinvalues and Eigenvectors: definition

"Eigen" is the German word for "proper", "own". By eigenvector, we mean a vector on which a matrix acts by changing its size, but not its direction. Namely, given a matrix A, we say that $\mathbf{v} \neq 0$ is a *eigenvector* of A with *eigenvalue* λ if

$$A\mathbf{v} = \lambda\mathbf{v}. \tag{4.1}$$

We may rewrite this as

$$(A - \lambda\mathbb{1})\,\mathbf{v} = 0, \tag{4.2}$$

where $\mathbb{1}$ is the identity matrix. From the equation above follows

$$\det(A - \lambda\mathbb{1}) = 0. \tag{4.3}$$

Eq. (4.3) may be used to obtain all the eigenvalues of A. The determinant will be written as a polynomial of order n, and its n roots (possibly with multiplicity higher than one) will give the eigenvalues λ_i. Eq. (4.1) may then be used to find the corresponding eigenvectors $\mathbf{v}^i$. Obviously the solution will not be unique (as to be expected due to 4.2), since also $\alpha\mathbf{v}^i$ will be an eigenvalue for arbitrary α.

4.1.1 Exercise

Let us find the eigenvalues and eigenvectors of

$$A = \begin{pmatrix} 5 & 2 \\ 2 & 2 \end{pmatrix}.$$

Solution

We compute

$$\det\left(A-\lambda\mathbb{1}\right)=\det\begin{pmatrix}5-\lambda & 2\\ 2 & 2-\lambda\end{pmatrix}=(\lambda-6)(\lambda-1).$$

From (4.3) we find that the eigenvalues are $\lambda_1 = 6$, $\lambda_2 = 1$. To solve for the eigenvectors, we may fix the first component to an arbitrary $v_1^i = \alpha$, and using the fact that only $n - 1 = 1$ equations of the system are independent, we may use the first one to get

$$(5-\lambda_i)\alpha+2v_2^i=0,$$

and

$$\mathbf{v}^1=\begin{pmatrix}\alpha\\ \alpha/2\end{pmatrix},\qquad \mathbf{v}^2=\begin{pmatrix}\alpha\\ -2\alpha\end{pmatrix}.$$

It may useful to get "normalised" (i.e., norm 1) vectors. For reasons that will be clear later we may be interested in using either $\|\ \|_\infty$ or $\|\ \|_2$. We have

$$\|\mathbf{v}^1\|_\infty=|\alpha|,\quad \|\mathbf{v}^2\|_\infty=2|\alpha|,\quad \|\mathbf{v}^1\|_2=|\alpha|\frac{\sqrt{5}}{2},\quad \|\mathbf{v}^2\|_2=|\alpha|\sqrt{5}.$$

As a result, $\|\ \|_\infty = 1$ vectors are

$$\mathbf{v}^1=\begin{pmatrix}1\\ \frac{1}{2}\end{pmatrix},\qquad \mathbf{v}^2=\begin{pmatrix}\frac{1}{2}\\ -1\end{pmatrix},$$

and $\|\ \|_2 = 1$ vectors are

$$\mathbf{v}^1=\begin{pmatrix}\frac{2}{\sqrt{5}}\\ \frac{1}{\sqrt{5}}\end{pmatrix},\qquad \mathbf{v}^2=\begin{pmatrix}\frac{1}{\sqrt{5}}\\ -\frac{2}{\sqrt{5}}\end{pmatrix}.$$

That these are eigenvectors with the proper eigenvalues may be checked by matrix multiplication.

4.1.2 The importance of complex numbers

Equation (4.3) gives a polynomial of *nth* order for a n square matrix. The fundamental theorem of algebra assures that we may find n roots *in the complex field* $\mathbb{C}$. It is important to remember that roots may have an imaginary

part even if all the entries of the matrix are real. For example, if we look for eigenvalues of

$$A = \begin{pmatrix} 1 & \sqrt{2} \\ -\sqrt{2} & -1 \end{pmatrix}, \tag{4.4}$$

we find

$$\det(A - \lambda \mathbb{1}) = \lambda^2 + 1, \tag{4.5}$$

and

$$\lambda_{1,2} = \pm i. \tag{4.6}$$

The corresponding (un-normalised) eigenvectors are

$$\mathbf{v}^1 = \begin{pmatrix} \alpha \\ \alpha \frac{i-1}{\sqrt{2}} \end{pmatrix} \qquad \mathbf{v}^2 = \begin{pmatrix} \alpha \\ -\alpha \frac{i+1}{\sqrt{2}} \end{pmatrix}. \tag{4.7}$$

As a consequence, the theory of eigenvalues needs considerations related to complex numbers, matrices and vectors. Anyway, we will consider a kind of matrices for which it may be shown that all eigenvalues and eigenvectors are real, and our main discussion will be confined to reals. The more general theory is to be found at the end of the chapter, section 4.7.

4.2 Scalar (inner) product

We know that

$$\|\mathbf{x}\|_2 = \sqrt{\left(\sum_i x_i^2\right)}. \tag{4.8}$$

Let us introduce a *inner product*

$$\langle \mathbf{x}, \mathbf{y} \rangle \equiv \sum_i x_i y_i, \tag{4.9}$$

so that

$$\|\mathbf{x}\|_2 = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}. \tag{4.10}$$

4.2.1 Geometrical meaning

You may be familiar from basic physics and geometry to the definition of a scalar product between two vectors **a** and **b** as

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos\theta, \tag{4.11}$$

θ being the angle between the two vectors. If we put the two vectors in an arbitrary reference frame with versors $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$ (limiting ourselves for simplicity to the 2D case), we have

$$\mathbf{a} = a_x\hat{\mathbf{i}} + a_y\hat{\mathbf{j}}, \tag{4.12}$$

and

$$\mathbf{b} = b_x\hat{\mathbf{i}} + b_y\hat{\mathbf{j}}. \tag{4.13}$$

Calling θ_a the angle between $\mathbf{a}$ and $\hat{\mathbf{i}}$, and θ_b the angle between $\mathbf{b}$ and $\hat{\mathbf{i}}$, we have the following relations (figure 4.1)

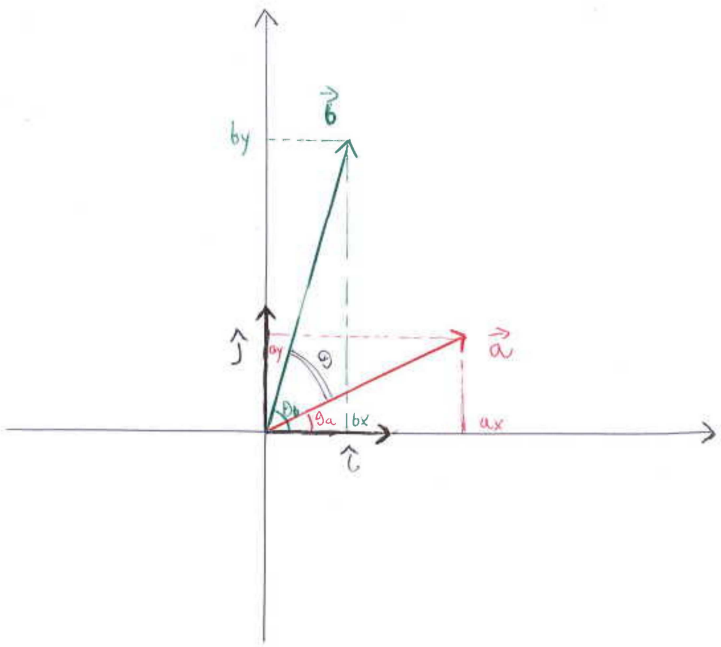

Figure 4.1: Geometrical meaning of scalar product

$$\begin{aligned} \theta &= \theta_b - \theta_a, \\ a_x &= |\mathbf{a}|\cos\theta_a, \\ a_y &= |\mathbf{a}|\sin\theta_a, \\ b_x &= |\mathbf{b}|\cos\theta_b, \\ b_y &= |\mathbf{b}|\sin\theta_b. \end{aligned} \tag{4.14}$$

From

$$\cos(\theta_a - \theta_b) = \cos\theta_a\cos\theta_b + \sin\theta_a\sin\theta_b, \tag{4.15}$$

we get

$$\mathbf{a}\cdot\mathbf{b} = |\mathbf{a}||\mathbf{b}|(\cos\theta_a\cos\theta_b + \sin\theta_a\sin\theta_b) = a_x b_x + a_y b_y, \tag{4.16}$$

explaining the geometrical meaning of the product (4.9).

It is easy to see that the product defined by eq. (4.9) satisfies

$$\langle \mathbf{a}, \mathbf{b}+\mathbf{c}\rangle = a_x(b_x+c_x) + a_y(b_y+c_y) = \langle \mathbf{a},\mathbf{b}\rangle + \langle \mathbf{a},\mathbf{c}\rangle. \tag{4.17}$$

By writing the scalar product using the explicit expressions (4.12,4.13), the linear property (4.17) and comparing to the result (4.16) we get

$$\langle \hat{\mathbf{i}}, \hat{\mathbf{i}}\rangle = \langle \hat{\mathbf{j}}, \hat{\mathbf{j}}\rangle = 1, \tag{4.18}$$

and

$$\langle \hat{\mathbf{i}}, \hat{\mathbf{j}}\rangle = \langle \hat{\mathbf{j}}, \hat{\mathbf{i}}\rangle = 0. \tag{4.19}$$

The vectors $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$ are a *orthonormal set*. Their norm is 1, and they are reciprocally orthogonal (or perpendicular),

$$\langle \mathbf{x}, \mathbf{y}\rangle = 0. \tag{4.20}$$

The product of two non zero vectors may thus be zero, and in this case the vectors are said to be orthogonal. The geometrical meaning of this statement should be obvious from eq. (4.11) and it should not be surprising that applies to Cartesian unit vectors.

4.2.2 Formal definition

The inner product on vectors in $\mathbb{R}^n$ is an application from a pair of vectors to the real field, i.e. $\langle \mathbf{x}, \mathbf{y}\rangle \in \mathbb{R}$ satisfying the following properties:

$$\langle \mathbf{x}, \mathbf{y}\rangle = \langle \mathbf{y}, \mathbf{x}\rangle, \tag{4.21}$$

$$\langle \mathbf{x}, \alpha\mathbf{y}\rangle = \alpha\langle \mathbf{y}, \mathbf{x}\rangle, \quad \alpha \in \mathbb{R}, \tag{4.22}$$

$$\langle \mathbf{x}, \mathbf{y}+\mathbf{z}\rangle = \langle \mathbf{x}, \mathbf{y}\rangle + \langle \mathbf{x}, \mathbf{z}\rangle, \tag{4.23}$$

$$\langle \mathbf{x}, \mathbf{x}\rangle \geq 0, \text{ with } \langle \mathbf{x}, \mathbf{x}\rangle = 0 \Leftrightarrow \mathbf{x} = 0. \tag{4.24}$$

It is easily verified that eq. (4.9) satisfies the formal inner product properties.

4.2.3 Proof that an inner product defines a norm

The formal definition of a norm was introduced in 3.2.2. If we define

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}, \tag{4.25}$$

we clearly have

$$\|\mathbf{x}\| \geq 0, \tag{4.26}$$

$$\|\mathbf{x}\| = 0 \Leftrightarrow \mathbf{x} = 0, \tag{4.27}$$

and

$$\|\alpha\mathbf{x}\| = \sqrt{\langle \alpha\mathbf{x}, \alpha\mathbf{x} \rangle} = \sqrt{\alpha^2 \langle \mathbf{x}, \mathbf{x} \rangle} = |\alpha| \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} = |\alpha| \|\mathbf{x}\|. \tag{4.28}$$

To show that

$$\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|, \tag{4.29}$$

is more tricky. Since the norm is positive, we may equivalently show

$$\left(\|\mathbf{x} + \mathbf{y}\|\right)^2 \leq \left(\|\mathbf{x}\| + \|\mathbf{y}\|\right)^2. \tag{4.30}$$

For the right hand side we have

$$\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 + 2\|\mathbf{x}\|\|\mathbf{y}\|. \tag{4.31}$$

For the left side, we write

$$\begin{aligned} \left(\|\mathbf{x} + \mathbf{y}\|\right)^2 = \langle \mathbf{x} + \mathbf{y}, \mathbf{x} + \mathbf{y} \rangle &= \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 + \langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{y}, \mathbf{x} \rangle \\ &\leq \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 + 2|\langle \mathbf{x}, \mathbf{y} \rangle| \end{aligned} \tag{4.32}$$

From the geometrical interpretation of the inner product, you may expect that

$$|\langle \mathbf{x}, \mathbf{y} \rangle| \leq \|\mathbf{x}\|\|\mathbf{y}\|, \tag{4.33}$$

and thus the triangular inequality holds. For the proof, you may see section 4.7.2.

4.3 Adjoint and transpose

Let us now consider matrices and write

$$\langle A\mathbf{x}, \mathbf{y} \rangle. \tag{4.34}$$

By components, we have

$$\langle A\mathbf{x}, \mathbf{y}\rangle = \sum_j \left(A\mathbf{x}\right)_j \left(\mathbf{y}\right)_j = \sum_j \left(\sum_i A_{j,i} x_i \right) y_j. \tag{4.35}$$

This may be rewritten as

$$\sum_{j,i} A_{j,i} x_i y_j. \tag{4.36}$$

Now, inside the sum, all matrix and vector components are just numbers, and we may re-arrange them as

$$\sum_{j,i} x_i A_{j,i} y_j = \sum_{j,i,l} x_i A^T_{i,j} y_j, \tag{4.37}$$

where we have used the definition of transpose matrix,

$$A^T_{j,i} = A_{i,j}. \tag{4.38}$$

We have finally

$$\langle A\mathbf{x}, \mathbf{y}\rangle = \sum_i \left(\sum_i x_i A^T_{i,j} y_j \right) = \langle \mathbf{x}, A^T\mathbf{y}\rangle. \tag{4.39}$$

Before moving a matrix from the right to the left side of the inner product, we have to take its transpose. In general we say that if

$$\langle A\mathbf{x}, \mathbf{y}\rangle = \langle \mathbf{x}, A^\dagger\mathbf{y}\rangle, \tag{4.40}$$

$A^\dagger$ is the adjoint of A. For real matrices, we have $A^\dagger = A^T$.

As a consequence, we have

$$\langle AB\mathbf{x}, \mathbf{y}\rangle = \langle B\mathbf{x}, A^\dagger\mathbf{y}\rangle = \langle \mathbf{x}, B^\dagger A^\dagger\mathbf{y}\rangle. \tag{4.41}$$

This means that

$$(AB)^T = (AB)^\dagger = B^\dagger A^\dagger = B^T A^T. \tag{4.42}$$

The relation for the transpose may be shown directly

$$(AB)^T_{j,i} = (AB)_{i,j} = \sum_l A_{i,l} B_{l,j} = \sum_l A^T_{l,i} B^T_{j,l} = \sum_l B^T_{j,l} A^T_{l,i} = (B^T A^T)_{j,i}. \tag{4.43}$$

4.4 Orthogonal matrix and change of basis

Let us consider a matrix O such that $O^T = O^{-1}$. Such a matrix is said to be orthogonal, and satisfies

$$\sum_l O_{l,j} O_{l,i} = \sum_l O^T_{j,l} O_{l,i} = \delta_{j,i}. \tag{4.44}$$

This means that if we consider a set of vectors $\mathbf{v}^i$ given by the columns of O, we have

$$\langle \mathbf{v}^i, \mathbf{v}^j \rangle = \delta_{i,j}, \tag{4.45}$$

i.e., the columns of O are given by orthonormal vectors. In the same way

$$\sum_l O_{j,l} O_{i,l} = \sum_l O_{j,l} O^T_{l,i} = \delta_{j,i}, \tag{4.46}$$

so that also the rows of O are orthonormal vectors.

Since O has an inverse, its rows and columns are independent, and they are n. They form thus a base for the vector space $\mathbb{R}^n$, meaning that any $\mathbf{x} \in \mathbb{R}^n$ may be written as

$$\mathbf{x} = \sum_k \chi_k \mathbf{v}^k, \tag{4.47}$$

or, in components,

$$x_i = \sum_k \chi_k \mathbf{v}^k_i. \tag{4.48}$$

Since the vectors are orthonormal (eq. 4.45) given a vector

$$\mathbf{y} = \sum_k \psi_k \mathbf{v}^k, \tag{4.49}$$

we have

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_k \chi_k \psi_k, \tag{4.50}$$

which resembles eq. (4.9) in the new "components".

Defining $\mathbf{v}^k_i = O_{k,i}$, i.e. considering the rows of O as the independent vectors, eq. (4.48) becomes

$$x_i = \sum_k O^T_{i,k} \chi_k. \tag{4.51}$$

Multiplying both sides by O we get

$$\sum_i O_{j,i}x_i = \sum_{i,k} O_{j,i}O^T_{i,k}\chi_k = \sum_k \left(\sum_i O_{j,i}O^T_{i,k}\right)\chi_k = \sum_k \delta_{j,k}\chi_k, \tag{4.52}$$

or

$$\chi_j = \sum_k O_{j,i}x_i. \tag{4.53}$$

We thus have that the matrix O operates a "change of basis", from our usual basis to a new one based on the vectors given by the rows of O. O^T operates the reverse change, going from the new basis to the original one.

From the properties of the determinant

$$\det(A^T) = \det(A), \qquad \det(AB) = \det(A)\det(B), \tag{4.54}$$

we get

$$\det(\mathbb{1}) = \det(O^T O) = (\det(O))^2 \Rightarrow \det(O) = \pm 1. \tag{4.55}$$

4.5 Symmetric matrices

A matrix is symmetric if $A^T = A$, or

$$A_{j,i} = A_{i,j}. \tag{4.56}$$

As it is shown in eq. (4.116), a symmetric matrix has real eigenvalues. Furthermore, if $\mathbf{v}^1$ and $\mathbf{v}^2$ are eigenvectors corresponding to different eigenvalues $\lambda_1 \neq \lambda_2$, they are orthogonal, $\langle \mathbf{v}^1, \mathbf{v}^2\rangle = 0$. To prove it we write

$$\begin{aligned}\lambda_1\langle \mathbf{v}^2, \mathbf{v}^1\rangle = \langle \mathbf{v}^2, \lambda_1\mathbf{v}^1\rangle = \langle \mathbf{v}^2, A\mathbf{v}^1\rangle = \langle A^\dagger\mathbf{v}^2, \mathbf{v}^1\rangle = \langle A\mathbf{v}^2, \mathbf{v}^1\rangle \\ = \langle \lambda_2\mathbf{v}^2, \mathbf{v}^1\rangle = \lambda_2\langle \mathbf{v}^2, \mathbf{v}^1\rangle,\end{aligned} \tag{4.57}$$

thus

$$(\lambda_2 - \lambda_1)\langle \mathbf{v}^1, \mathbf{v}^2\rangle = 0. \tag{4.58}$$

Since the eigenvalues are different, we necessarily have $\langle \mathbf{v}^1, \mathbf{v}^2\rangle = 0$. If A has n different eigenvalues, the corresponding eigenvectors are thus orthogonal

$$\langle \mathbf{v}^i, \mathbf{v}^j\rangle = \delta_{i,j}\|\mathbf{v}^j\|^2. \tag{4.59}$$

They are also necessarily independent, since if

$$\mathbf{v}^j = \sum_{i\neq j} \alpha_i \mathbf{v}^i, \tag{4.60}$$

with at least a $\alpha_k \neq 0$, we have

$$\langle \mathbf{v}^j, \mathbf{v}^k \rangle = \alpha_k \|\mathbf{v}^k\|^2, \tag{4.61}$$

leading to a contradiction. It may be shown that even if the matrix has independent eigenvectors corresponding to the same eigenvalue, a basis of n linearly independent eigenvetors may always be found[1]. By dividing such vectors by their norm, eq. (4.59) may be simplified to

$$\langle \mathbf{v}^i, \mathbf{v}^j \rangle = \delta_{i,j}. \tag{4.62}$$

The eigenvectors form thus a orthonormal basis, and the orthogonal matrix $O_{j,i} = v_i^j$ may be constructed.

4.5.1 Matrix diagonalisation

As discussed above (eq. 4.48), given the orthonormal basis of eigenvectors we may write for each $\mathbf{x}$

$$x_i = \sum_k \chi_k \mathbf{v}_i^k, \tag{4.63}$$

from which follows (eqs. 4.51, 4.53)

$$x_i = \sum_k O_{i,k}^T \chi_k \qquad \chi_j = \sum_k O_{j,i} x_i. \tag{4.64}$$

We also recall that, from $A\mathbf{v}^k = \lambda_k \mathbf{v}^k$ we have

$$\sum_i A_{j,i} O_{k,i} = \sum_i A_{j,i} v_i^k = \lambda_k v_j^k. \tag{4.65}$$

Using the usual index manipulation we get

$$\begin{aligned}(A\mathbf{x})_j = \sum_l A_{j,l} x_l = \sum_{l,k} A_{j,l} O_{l,k}^T \chi_k = \sum_{l,k,i} A_{j,l} O_{l,k}^T O_{k,i} x_i = \sum_{l,k,i} A_{j,l} v_l^k O_{k,i} x_i = \\ \sum_{k,i} \lambda_k v_j^k O_{k,i} x_i = \sum_{l,k,i} v_j^l \lambda_l \delta_{l,k} O_{k,i} x_i = \sum_{l,k,i} O_{j,l}^T \lambda_l \delta_{l,k} O_{k,i} x_i = (O^T D O \mathbf{x})_j,\end{aligned} \tag{4.66}$$

[1] The physicists' trick to show that is to add a small perturbation to the matrix so that the symmetry is broken and the basis is present (all eigenvalues different), and then send the perturbation to zero.

or

$$A = O^T D O, \tag{4.67}$$

where D is a diagonal matrix with entries $D_{j,i} = \lambda_i \delta_{j,i}$

$$D = \begin{pmatrix} \lambda_1 & 0 & \dots & \dots & 0 \\ 0 & \lambda_2 & 0 & \dots & 0 \\ \vdots & 0 & \lambda_3 & 0 & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & \lambda_n \end{pmatrix} \tag{4.68}$$

The meaning should be clear. O brings us to a basis in which A acts diagonally, and O^T brings us back to the original one[2].

4.5.2 Example

Diagonalise

$$A = \begin{pmatrix} 5 & 2 \\ 2 & 2 \end{pmatrix}. \tag{4.69}$$

From 4.1.1, where we found the orthonormal (check!) eigenvectors, we have

$$A = \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} \end{pmatrix} \begin{pmatrix} 6 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} \end{pmatrix}. \tag{4.70}$$

4.5.3 Determinant and trace of symmetric matrices

From the rule for the product of determinants and eq. (4.55) we find

$$\det(A) = \det(O^T D O) = \det(O^T)\det(O)\det(D) = \prod_i \lambda_i. \tag{4.71}$$

The trace of a matrix is defined as

$$\mathrm{Tr}(A) = \sum_i A_{ii}. \tag{4.72}$$

We have

$$\mathrm{Tr}(AB) = \sum_i \sum_j A_{i,j} B_{j,i} = \sum_j \sum_i B_{j,i} A_{i,j} = \mathrm{Tr}(BA). \tag{4.73}$$

[2] D and A are two different representations of the same linear operator. The Bra-Ket formalism, itroduced by the British Physicist and Noble Prize Paul Dirac, is very useful to handle all the linear algebra computations.

Thus

$$\mathrm{Tr}(A) = \mathrm{Tr}(O^T DO) = \mathrm{Tr}(OO^T D) = \mathrm{Tr}(D) = \sum_i \lambda_i. \tag{4.74}$$

These results may also be used to check if the diagonalisation process was performed in a correct way.

4.6 Power method to find numerically eigenvalues and vectors

4.6.1 Theory

We now see an algorithm to compute numerically eigenvectors and eigenvalues. Let us assume that our matrix has n orthonormal real eigenvectors, as it would be the case for a symmetric one. *We also assume that it has n different modulus eigenvalues* λ_k, i.e. that there are no 2 eigenvalues such that $|\lambda_i| = |\lambda_k|$ if $i \neq k$. Let us then list the eigenvalues starting from the higher modulus value, i.e. $|\lambda_k| > |\lambda_i|$ if $k < i$. The corresponding normalised vectors will be $\mathbf{v}^k$.

We know that for each $\mathbf{x}$ we may write

$$\mathbf{x} = \sum_k \alpha_k \mathbf{v}^k \tag{4.75}$$

For simplicity's sake, let us assume $\alpha_k \neq 0\ \forall k$ (we will generalise later). Let us apply A^n (i.e., n times A) to $\mathbf{x}$, and obtain the normalised vector

$$\frac{A^n \mathbf{x}}{\|A^n \mathbf{x}\|} \tag{4.76}$$

This may be written as

$$\frac{A^n(\sum_k \alpha_k \mathbf{v}^k)}{\|A^n \mathbf{x}\|} = \frac{\sum_k \alpha_k A^n \mathbf{v}^k}{\|A^n \mathbf{x}\|} = \frac{\sum_k \alpha_k \lambda_k^n \mathbf{v}^k}{\|A^n \mathbf{x}\|} = \frac{\lambda_1^n}{\|A^n \mathbf{x}\|}\left(\alpha_1 \mathbf{v}^1 + \sum_{k=2}^{n} \alpha_k \left(\frac{\lambda_k}{\lambda_1}\right)^n \mathbf{v}^k\right) \tag{4.77}$$

Since we have

$$\lim_{n\to 0} \left(\frac{\lambda_k}{\lambda_1}\right)^n = 0 \quad \forall k \neq 1 \tag{4.78}$$

we obtain

$$\lim_{n\to 0} \frac{A^n \mathbf{x}}{\|A^n \mathbf{x}\|} = \frac{\lambda_1^n \alpha_1}{\|A^n \mathbf{x}\|} \mathbf{v}^1 = \pm \mathbf{v}^1 \tag{4.79}$$

The last equation is due to the fact that the resulting vector is normalised to one and parallel to $\mathbf{v}^1$, which for a real vector means that it is either $\mathbf{v}^1$ or $-\mathbf{v}^1$. In the second case we may just rename $-\mathbf{v}^1$ as $\mathbf{v}^1$. Now that we know (with a sufficient good approximation for high enough n) $\mathbf{v}^1$, we obtain λ_1 from

$$\lambda_1 = \frac{(A\mathbf{v}^1)_j}{\mathbf{v}^1_j} \tag{4.80}$$

for any component j.

We may now proceed and look for λ_2. Let us first compute

$$\langle \mathbf{x}, \mathbf{v}^1 \rangle \tag{4.81}$$

i.e., the scalar product between two known vectors. This is equal to

$$\left\langle \sum_k \alpha_k \mathbf{v}^k, \mathbf{v}^1 \right\rangle = \sum_k \alpha_k \langle \mathbf{v}^k, \mathbf{v}^1 \rangle = \sum_k \alpha_k \delta_{1,k} = \alpha_1 \tag{4.82}$$

Thus,

$$\mathbf{x}_2 \equiv \mathbf{x} - \langle \mathbf{x}, \mathbf{v}^1 \rangle \mathbf{v}^1 = \sum_{k=2}^{n} \alpha_k \mathbf{v}^k \tag{4.83}$$

It should now be obvious that

$$\lim_{n\to 0} \frac{A^n \mathbf{x}_2}{\|A^n \mathbf{x}_2\|} = \pm \mathbf{v}^2 \tag{4.84}$$

In the same way, after we have found $\mathbf{v}_k$ for all $k < j$ we may define

$$\mathbf{x}_j \equiv \mathbf{x} - \sum_{k=1}^{j-1} \langle \mathbf{x}, \mathbf{v}^k \rangle \mathbf{v}^k = \sum_{k=j}^{n} \alpha_k \mathbf{v}^k \tag{4.85}$$

and we have

$$\lim_{n\to 0} \frac{A^n \mathbf{x}_j}{\|A^n \mathbf{x}_j\|} = \pm \mathbf{v}^j \tag{4.86}$$

A special treatment has to be done only for $\lambda_j = 0$, since in this case $A\mathbf{x}_j$ is 0, and not parallel to $\mathbf{v}^j$. Anyway, this case may be trivially handled by checking that $\|A\mathbf{x}^j\|$ is zero for a non-zero $\|\mathbf{x}^j\|$, so that the latter is an eigenvector of $\lambda = 0$.

At the beginning of this discussion we assumed $\alpha_k \neq 0$ for all k. This will not be the case in general. Anyway, since $\{v^k\}$ is a basis, we will have *some* $\alpha_k \neq 0$, and thus we will identify, starting from an arbitrary $\mathbf{x} \neq 0$, a subset

$j < n$ of our basis of eigenvectors, $\mathbf{v}^i$ with $i \leq j$. We know another basis of $\mathbb{R}^n$, namely the Cartesian one $\mathbf{e}^i$ with $\mathbf{e}^i_j = \delta_{i,j}$. It is clear that we cannot have

$$\mathbf{e}^i - \sum_{k=1}^{j} \langle \mathbf{e}^i, \mathbf{v}^k \rangle \mathbf{v}^k = 0 \tag{4.87}$$

otherwise each element of the basis would be written as a linear combination of the $\mathbf{v}^k$ with $k < j$, so that this $j < n$ vectors would be a basis of $\mathbb{R}^n$, leading to a contradiction. Thus by trying different elements of the Cartesian basis we may apply the procedure again until we eventually find the whole eigenvector basis.

What happens if the matrix has eigenvalues with the same module, $|\lambda_i| = |\lambda_j|$ for $i \neq j$? Two cases are possible. If the two eigenvalues are actually the same, i.e. if we have 2 or more independent eigenvectors with the same eigenvalue, the procedure will converge without problem. Let us assume for example $\lambda_1 = \lambda_2 = \lambda$. We have

$$\lim_{n \to 0} \frac{A^n \mathbf{x}_j}{\|A^n \mathbf{x}_j\|} = \frac{\alpha_1 \mathbf{v}^1 + \alpha_2 \mathbf{v}^2}{\sqrt{\alpha_1^2 + \alpha_2^2}} \tag{4.88}$$

This is a legit eigenvector with eigenvalue λ. Our procedure will eventually lead to another eigenvector *for the 2D subspace corresponding to* λ, that in the original (but unknown) basis $\mathbf{v}^i$ would be

$$\pm \frac{\alpha_2 \mathbf{v}^1 - \alpha_1 \mathbf{v}^2}{\sqrt{\alpha_1^2 + \alpha_2^2}} \mathbf{v}^j \tag{4.89}$$

(to be orthonormal to the first). The same situation applies to a more general l dimensional subspace corresponding to the eigenvalue λ. Our procedure correctly converges to a basis if the eiegnvalues with equal modulus are actually the same eigenvalue.

But if the two eigenvalues are actually different, as for example $\lambda_1 = -\lambda_2$, we have then

$$\lim_{n \to 0} \frac{A^n \mathbf{x}_j}{\|A^n \mathbf{x}_j\|} = \frac{\alpha_1 \mathbf{v}^1 (-)^n \alpha_2 \mathbf{v}^2}{\sqrt{\alpha_1^2 + \alpha_2^2}} \tag{4.90}$$

which clearly does not converge.

We may nevertheless use the power method to study the *spectrum* (eigenvalue set) of any A in the following way. The matrix A^2 is symmetric and has eigenvalues $\lambda_i^2 > 0$. The procedure above will converge on A^2, being

all eigenvalues positive, so that we may find all possible eigenvalues of A in the form $\pm\lambda_i$. If then the procedure does not converge for A, it means both positive and negative values were present.

4.6.2 Practice

A few precautions have to be taken before numerically applying the procedure. The limit for $n \to \infty$ will obviously replaced with a computation for finite n. The normalisation will be performed at each step, and the difference between two iterations

$$\left\| \frac{A^n \mathbf{x}}{\|A^n \mathbf{x}\|} - \frac{A^{n-1} \mathbf{x}}{\|A^{n-1} \mathbf{x}\|} \right\| \tag{4.91}$$

computed to check iteration. While the norm 2 is necessary for the scalar product, at each step the normalisation may be done using the infinite norm, reducing both numerical cost and rounding due to the computation of square roots.

Furthermore, the subtraction of the orthogonal components eq. (4.85) should be computed at each step, otherwise the errors due to round off could easily propagate.

The convergence of the process relating the λ_k eigenvalue depends on the ratio $|\lambda_k/\lambda_{k+1}|$, so that if these two number are very similar in modulus we will have a slow convergence.

4.6.3 Exercise

Starting from the seed

$$\mathbf{x}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Use the power method to find the highest modulus eigenvalue of the matrix

$$A = \begin{pmatrix} 5 & 2 \\ 2 & 2 \end{pmatrix}$$

and the corresponding eigenvalue. Stop after 4 iterations, and compare to the exact value.

4.6.4 Solution

The exact value has been found in section 4.1.1. Since the matrix has eigenvalues with different modulus, we know that the method will converge. In the first iteration we compute

$$\mathbf{y}_1 = A\mathbf{x}_0 = \begin{pmatrix} 5 & 2 \\ 2 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 2 \end{pmatrix}$$

Dividing the maximum modulus component of $\mathbf{y}_1$ by the maximum modulus component of $\mathbf{x}_0$ we get the first estimate for λ

$$\lambda_1 = 5$$

Normalising to infinite norm 1 (i.e. dividing each component of $\mathbf{y}_1$ by its maximum modulus component) we obtain

$$\mathbf{x}_1 = \begin{pmatrix} 1 \\ 2/5 \end{pmatrix}$$

The second iteration gives

$$\mathbf{y}_2 = A\mathbf{x}_1 = \begin{pmatrix} 29/5 \\ 14/5 \end{pmatrix}$$

$$\lambda_2 = 29/5$$

and

$$\mathbf{x}_2 \approx \begin{pmatrix} 1 \\ 0.482759 \end{pmatrix}$$

The third iteration gives

$$\mathbf{y}_3 = A\mathbf{x}_2 \approx \begin{pmatrix} 5.96552 \\ 2.96552 \end{pmatrix}$$

$$\lambda_3 \approx 5.96552$$

and

$$\mathbf{x}_3 \approx \begin{pmatrix} 1 \\ 0.49711 \end{pmatrix}$$

The fourth iteration gives

$$\mathbf{y}_4 = A\mathbf{x}_3 \approx \begin{pmatrix} 5.99422 \\ 2.99422 \end{pmatrix}$$

$$\lambda_4 \approx 5.99422$$

and

$$\mathbf{x}_4 \approx \begin{pmatrix} 1 \\ 0.499518 \end{pmatrix}$$

After 4 iterations, the error on the eigenvalue with respect to the exact result 6 is

$$|\lambda_4 - 6| \approx 6 \cdot 10^{-3}$$

with a relative error of

$$\frac{|\lambda_4 - \lambda_+|}{6} \approx 10^{-3}$$

The error on the eigenvector is

$$\|\mathbf{x_4} - \mathbf{v}_6\|_\infty \approx 5 \cdot 10^{-4}$$

Since the vectors are normalised to 1, this is also the relative error. In the 2 dimensional case, the other eigenvector may be simply (numerically) found by requiring orthogonality, namely

$$\pm \begin{pmatrix} 0.499518 \\ -1 \end{pmatrix}$$

4.7 Complex vectors and matrices

4.7.1 Inner product

Let us consider the vector space $\mathbb{C}^n$ (vector with complex components). If we were to define the 2 norm on the space as

$$\|\mathbf{x}\|_2 = \sqrt{\left(\sum_i x_i^2\right)}. \tag{4.92}$$

we would face a problem. If $c = \alpha + i\beta$, we have $c^2 = \alpha^2 - \beta^2 + 2i\alpha\beta$, which may be not positive (for example for $c = i$). On the opposite, remembering the definition of complex conjugate $\overline{c} = \alpha - i\beta$, we have

$$|c| \equiv c\overline{c} = \alpha^2 + \beta^2. \tag{4.93}$$

To obtain the correct form of the 2 norm we thus define the scalar product on $\mathbb{C}^n$ as[3]

$$\langle \mathbf{x}, \mathbf{y} \rangle \equiv \sum_i \overline{x_i} y_i. \tag{4.94}$$

Formally, the product on a complex space satisfies

$$\langle \mathbf{x}, \mathbf{y} \rangle = \overline{\langle \mathbf{y}, \mathbf{x} \rangle}, \tag{4.95}$$

$$\langle \mathbf{x}, \alpha \mathbf{y} \rangle = \alpha \langle \mathbf{y}, \mathbf{x} \rangle, \quad \alpha \in \mathbb{C}, \tag{4.96}$$

$$\langle \mathbf{x}, \mathbf{y} + \mathbf{z} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{x}, \mathbf{z} \rangle, \tag{4.97}$$

$$\langle \mathbf{x}, \mathbf{x} \rangle \geq 0, \text{ with } \langle \mathbf{x}, \mathbf{x} \rangle = 0 \Leftrightarrow \mathbf{x} = 0. \tag{4.98}$$

From these properties derives that

$$\langle \alpha \mathbf{x}, \mathbf{y} \rangle = \overline{\langle \mathbf{y}, \alpha \mathbf{x} \rangle} = \overline{\alpha \langle \mathbf{y}, \mathbf{x} \rangle} = \overline{\alpha} \overline{\langle \mathbf{y}, \mathbf{x} \rangle}, \tag{4.99}$$

and thus

$$\langle \alpha \mathbf{x}, \mathbf{y} \rangle = \overline{\alpha} \langle \mathbf{x}, \mathbf{y} \rangle. \tag{4.100}$$

It is easy to verify that eq. (4.94) satisfies the above properties. In particular

$$\langle \mathbf{y}, \mathbf{x} \rangle = \sum_i \overline{y_i} x_i = \overline{\sum_i y_i \overline{x_i}} = \overline{\langle \mathbf{x}, \mathbf{y} \rangle}. \tag{4.101}$$

4.7.2 Norm from inner product and Cauchy-Schwartz inequality

The proof follows the one for the real case, taking in account that

$$\|\alpha \mathbf{x}\| = \sqrt{\langle \alpha \mathbf{x}, \alpha \mathbf{x} \rangle} = \sqrt{\alpha \langle \alpha \mathbf{x}, \mathbf{x} \rangle} = \sqrt{\overline{\alpha} \alpha \langle \mathbf{x}, \mathbf{x} \rangle} = |\alpha| \|\mathbf{x}\|. \tag{4.102}$$

We may prove explicitly the Cauchy-Schwartz inequality that leads to the triangular one, namely

$$|\langle \mathbf{x}, \mathbf{y} \rangle| \leq \|\mathbf{x}\| \|\mathbf{y}\|. \tag{4.103}$$

If $\mathbf{y} = 0$ the inequality is true. Let assume $\mathbf{y} \neq 0$. We may then subtract to $\mathbf{x}$ its projection along y

$$\mathbf{x}_{\mathbf{y}} = \mathbf{x} - \frac{\langle \mathbf{y}, \mathbf{x} \rangle}{\|\mathbf{y}\|^2} \mathbf{y}. \tag{4.104}$$

[3]Some authors prefer $\sum_i x_i \overline{y_i}$

Now (it is basically Pythagoras's theorem for 2 orthogonal components)

$$0 \leq \|\mathbf{x}_\mathbf{y}\|^2 = \langle \mathbf{x}_\mathbf{y}, \mathbf{x}_\mathbf{y} \rangle = \|\mathbf{x}\|^2 + \frac{|\langle \mathbf{y}, \mathbf{x} \rangle|^2}{\|\mathbf{y}\|^2} - 2\frac{\langle \mathbf{y}, \mathbf{x} \rangle \langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{y}\|^2}, \tag{4.105}$$

from which we have

$$\|\mathbf{x}\|^2 - \frac{|\langle \mathbf{y}, \mathbf{x} \rangle|^2}{\|\mathbf{y}\|^2} \geq 0, \tag{4.106}$$

and finally

$$\|\mathbf{x}\|^2 \|\mathbf{y}\|^2 \geq |\langle \mathbf{y}, \mathbf{x} \rangle|^2. \tag{4.107}$$

4.7.3 Adjoint

Let us write

$$\langle A\mathbf{x}, \mathbf{y} \rangle. \tag{4.108}$$

By components, we have

$$\langle A\mathbf{x}, \mathbf{y} \rangle = \sum_j \overline{(A\mathbf{x})_j} (\mathbf{y})_j = \sum_j \left(\sum_i \overline{A_{j,i}} \overline{x_i} \right) y_j = \sum_{j,i} x_i \overline{A_{j,i}} y_j = \sum_{j,i,l} x_i \overline{A^T_{i,j}} y_j. \tag{4.109}$$

For complex matrices, the adjoint operator is thus given by the complex conjugate of the transpose,

$$A^\dagger_{j,i} = \overline{A_{i,j}}. \tag{4.110}$$

Again we have

$$\langle AB\mathbf{x}, \mathbf{y} \rangle = \langle B\mathbf{x}, A^\dagger \mathbf{y} \rangle = \langle \mathbf{x}, B^\dagger A^\dagger \mathbf{y} \rangle, \tag{4.111}$$

or

$$(AB)^\dagger = B^\dagger A^\dagger. \tag{4.112}$$

4.7.4 Unitary matrix

Everything we said about the orthogonal matrices applies in the complex case to unitary ones, i.e. matrices with $U^\dagger = U^{-1}$. For them we have

$$\sum_l \overline{U_{j,l}} U_{i,l}, \tag{4.113}$$

so that the rows may be again considered as orthonormal vectors and thus a basis, and so on.

4.7.5 Hermitian matrices

A matrix is *Hermitian* if it is its own adjoint, $A^\dagger = A$ or

$$\overline{A_{i,j}} = A_{j,i}. \tag{4.114}$$

A real symmetric matrix is thus Hermitian. It is easy to show that an Hermitian matrix has real eigenvalues. Let us assume

$$A\mathbf{v} = \lambda\mathbf{v}. \tag{4.115}$$

We have then

$$\lambda\langle\mathbf{v},\mathbf{v}\rangle = \langle\mathbf{v},\lambda\mathbf{v}\rangle = \langle\mathbf{v},A\mathbf{v}\rangle = \langle A^\dagger\mathbf{v},\mathbf{v}\rangle = \langle A\mathbf{v},\mathbf{v}\rangle = \langle\lambda\mathbf{v},\mathbf{v}\rangle = \overline{\lambda}\langle\mathbf{v},\mathbf{v}\rangle, \tag{4.116}$$

from which we obtain $\lambda = \overline{\lambda}$.

The discussion leading to eq. (4.57) is not changed for Hermitian matrices. Furthermore, it may be shown that also in the case of Hermitian matrices we have a basis of orthonormal vectors, that may be used to build a unitary matrix an diagonalise the matrix $A = U^{-1}DU$. The results for determinant and trace follow.

Chapter 5

Non linear equations

When dealing with linear equations, our task has been the one of generalising the trivial $n = 1$ situation

$$ax = b \Rightarrow x = \frac{b}{a}, \tag{5.1}$$

to the general finite n case

$$A\mathbf{x} = b \Rightarrow \mathbf{x} = A^{-1}\mathbf{b} \text{ if } \det(A) \neq 0. \tag{5.2}$$

In the process, we developed algorithms to solve such a mathematical problem (which in general *do not* involve computing the inverse matrix, whose importance is more in the development of the theory than in specific computations).

What about non linear equations? They are actually already not trivial in the 1D case, and it is hard to find a *one size fits all* method that solves any problem. Knowledge of the nature of the problem itself will be needed, and since in general for each method we have a trade-off between generality (i.e., ability to be applied to any problem) and precision, a mixed approach is often needed. In these notes we will just give a short introduction to some $n = 1$ methods, and then spend a few words on their generalisation to arbitrary finite n.

5.1 $n = 1$

In a $n = 1$ linear equation only terms like ax and b could enter, so we wrote $ax = b$. Now these terms generalise to arbitrary functions

$$g(x) = h(x) \Rightarrow f(x) \equiv g(x) - h(x) = 0, \tag{5.3}$$

so that we may express a non-linear equation as $f(x) = 0$ without loss of generality. How do we handle them?

In particular for a single variable function, the first step is always to try to study it analytically. Let us consider

$$f(x) \equiv e^x - \sin x = 0, \tag{5.4}$$

or $e^x = \sin x$. We know that $|\sin x| \leq 1 \ \forall x$, and $e^x > 1$ for $x > 0$, so the equation has no solutions for $x > 0$. $x = 0$ is not a solution either, since $e^0 = 1$ and $\sin 0 = 0$. We also know that $e^x > 0 \ \forall x$, and $e^x < 1$ for $x < 0$. Since $\sin x \leq 0$ if $x \in [-2n\pi, (-2n-1)\pi]$, $n \in \mathbb{N}$, again we cannot have any solution in these intervals. Now for some positive result: we have $\sin x = 1$, and thus $f(x) < 0$, in $x = (-2n - 3/2)\pi$. Since $f(x)$ is continuous, and $f(x) > 0$ in the zeros of $\sin x$, we have a solution (exactly one, since the two functions are monotone in such intervals) in each of the following intervals

$$\big((-2n-1)\pi, (-2n-3/2)\pi\big) \qquad \big((-2n-3/2)\pi, (-2n-2)\pi\big). \tag{5.5}$$

Furthermore, since $\lim_{x\to-\infty} e^x = 0$, in the limit of large n the solutions will be located in $-n\pi$. (See figure 5.1.)

In this way we already obtained an information that would not be available to any numerical procedure, i.e. the existence of an infinity of solutions! Anyway, if we want to know, for example, with a given approximation where the solution with maximum x is located, we need to rely on numerical methods. Still, these methods will need a starting point, or a starting interval, and this may be chosen based on our analytical results.

5.1.1 "Monte Carlo" simulations

Before proceeding to examine the numerical methods to solve non-linear equations, we may wonder what can we do if we are not able to study the analytical properties of the function. In this situation, we will choose an interval of interest $[a, b]$ on which we want to study our function[1]. An example for our function is shown below in Figure 5.1.

[1]Since it is impossible to numerically compute the function on the whole $\mathbb{R}$. Assuming we have computational time to evaluate our function on N points, we define $h = (b - a)/(N - 1)$, and compute the function in $f(a + ih)$, $i = 0, \ldots, N - 1$.

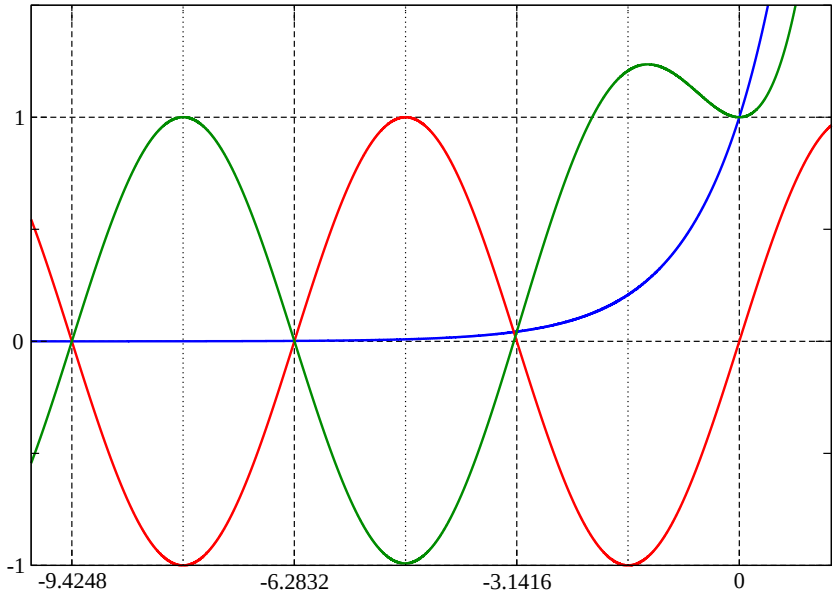

Figure 5.1: e^x in blue, $\sin x$ in red, $e^x - \sin x$ in green

To know the behaviour of a function $f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^n$, we would need to compute f on a grid

$$f(x_1 + i_1 h_1, \ldots, x_j + i_j h_j, \ldots, x_n + i_n h_n), \tag{5.6}$$

where possibly the grid is un-even, $h_k \neq h_j$ if $j \neq k$, and the index i_j assumes values in $1, \ldots, M_j$ If we use a regular grid, and divide the interval corresponding to each variable by M, we get to evaluate the function $N = M^n$ times. As a result, for large n (i.e. in a large *configuration space*), N has to be small, and the grid is extremely sparse. In this situation, it may be better to evaluate the function in random points[2]. Such a method, which may be used for example to evaluate multi dimensional integrals, is called "Monte Carlo"[3].

[2]More exactly, *pseudo* random points

[3]Since it relies on chance, like playing in a Casino. Nevertheless, since it may be shown that the fluctuations behave as $1/\sqrt{N}$, the method becomes reliable for large enough N.

5.1.2 Bisection method

Let us suppose that we know, by analytical or "Monte Carlo" methods, that the function $f(x)$ changes of sign between two points a and b, i.e.

$$f(a)f(b) < 0. \tag{5.7}$$

If we know or assume that the function is continuous, then we have a solution of $f(x) = 0$ in (a, b). We may use the *bisection* method to compute such a solution with the desired precision.

Let us call x the point half the way between a and b, $x = a + (b - a)/2$. We compute $f(x)$ and check if $f(x)f(a) < 0$. In such case the solution (or at least a solution) is located between x and a, and we can set $a_1 = a$, $b_1 = x$, and compute $x_1 = a_1 + (b_1 - a_1)/2$. On the other end, if we had $f(x)f(a) = 0$, then x is a solution and our method converged. Finally, if $f(x)f(a) > 0$, then $f(x)f(b) < 0$, so we may set $a_1 = x$, $b_1 = b$ and proceed (Figure 5.2). At each step we localise an interval of length $(b - a)/2^n$ in which a solution is located (our original interval may have contained more possible solutions, and the algorithm converges to one of them). If we want a precision ε then (assuming $b > a$)

$$\log_2(b - a) - n < \log_2(\varepsilon) \Rightarrow n > \log_2(b - a) - \log_2(\varepsilon). \tag{5.8}$$

The method may also be called "binary" since assuming at a given step we have in binary notation $b_n - a_n = 10000\ldots0000$ then at the following step we will have $b_{n+1} - a_{n+1} = 01000\ldots0000$, and so on. As a result, in a few tens iterations, the machine precision concerning the root position is reached.

5.1.3 Secant method

But we can do better, since in the bisection method we did not even use the value of the function in a and b. The *secant* method assumes that the solution may be found close to the intercept between the line passing for $(a, f(a))$ and $(b, f(b))$ and the x axis (Figure 5.2). Let us write the equation of the line

$$y = f(a) + \frac{f(b) - f(a)}{b - a}(x - a), \tag{5.9}$$

so that asking $y = 0$ we get

$$x = a - f(a)\frac{b - a}{f(b) - f(a)}. \tag{5.10}$$

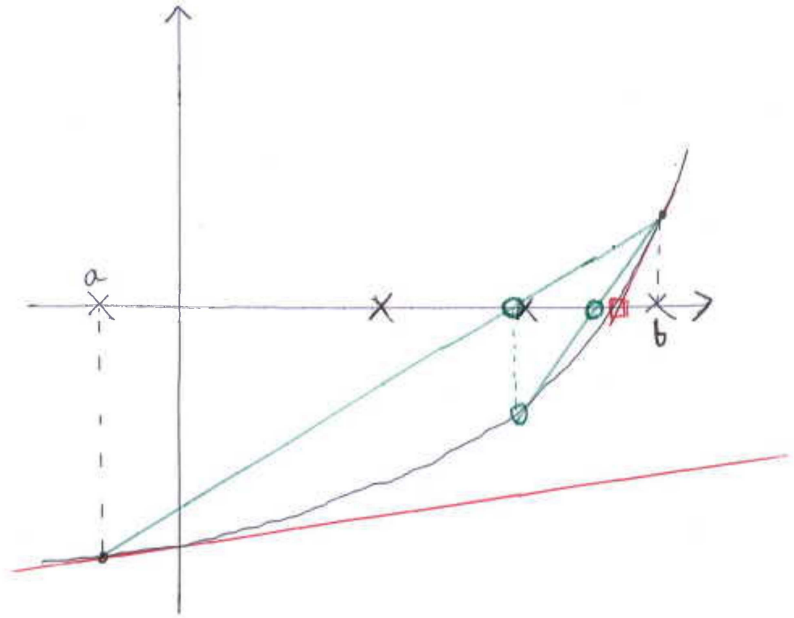

Figure 5.2: Black crosses give the first two iterations of the bisection method, green circles the first two iterations of the secant method. The red square is the first iteration of the Newton method from b. From a the Newton method does not converge.

The computation of x is the only difference between the bisection and secant method. After obtaining x we check for the sign of $f(x)f(a)$ and proceed as before. The method converges again to one of the solutions located in the interval (a, b), but in general it converges faster.

5.1.4 Newton's method

The secant method improves on the bisection one by using the value of the function. We may think that, *in particular if we are close to the solution*, a further improvement may be obtained by using also the function's variation, i.e. its derivative. Let us $b \to a$ in eq. (5.10), or if you prefer define $h \equiv b - a$, $b = a + h$, $h \to 0$ and obtain

$$x = a - \frac{f(a)}{f'(a)}. \tag{5.11}$$

While before we used an interval to find a solution, now a starting point is enough, and it will give us a sequence

$$x_n = g(x_{n-1}) \qquad g(x) = x - \frac{f(x)}{f'(x)}. \tag{5.12}$$

Clearly a solution of $f(x) = 0$ is also a fixed point for $g(x)$. Figure 5.2 gives an idea of how effective the method may be *if we start close enough to the solution*. Indeed for our problem, using $b = -\pi$, $a = -3/2\pi$, with the bisection method we need 10 iterations to get $|f(x)| < 10^{-4}$, while with the secant method at the third iteration we have $|f(x)| < 10^{-7}$. Using Newton from b, we get $|f(x)| < 10^{-15}$ in only 3 iterations.

But this comes at a cost. While the previous methods knew that a solution was present and always converged to it, the Newton method will not converge in general, as shown in figure 5.2. Indeed, if we chose to apply Newton's method from $-3/2\pi$, the method would not have converged.

Newton's method may then be used to refine a solution selected with another method, such as the secant one. Another possibility may be to do a Monte Carlo search for possible Newton starting points, and stop the iteration process if the method is not converging (i.e. if the function absolute value does not decrease).

Newton's method with numerical derivative

The derivative in eq. (5.11) may be also computed numerically, as

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}. \tag{5.13}$$

Using $h = 10^{-8}$ and 16 digit precision, we find again $|f(x)| < 10^{-15}$ in only 3 iterations for $e^x - \sin x$ starting from $-\pi$.

A theorem for the convergence of Newton's method

We have seen in the lecture on iterative methods (sec. 3.4) that if in a metric space we have $d(T\mathbf{x}, T\mathbf{y}) < d(\mathbf{x}, \mathbf{y})$, the sequence converges to the only fixed point of T.

Let us consider on $\mathbb{R}$ a function $g(x)$ with $|g'(x)| \leq \alpha < 1$. We have using

a basic calculus theorem[4]

$$\begin{aligned} |g(y) - g(x)| &= |g(x) + g'(x + \xi(y - x))(y - x) - g(x)| \\ &\leq |g'(x + \xi(y - x))||y - x| < |y - x|, \end{aligned} \tag{5.14}$$

where $0 \leq \xi \leq 1$.

For the Newton method we have, assuming f, f' and f'' to be continuous, $f(\overline{x}) = 0$ *and* $f'(\overline{x}) \neq 0$

$$g'(x) = \frac{f(x)f''(x)}{(f'(x))^2} \tag{5.15}$$

If $\overline{x}$ is such as $f(\overline{x}) = 0$, then $g'(\overline{x}) = 0$. Being g continuous, we have a neighbourhood of $\overline{x}$ in which $|g'(x)| < 1$. If we apply g to a point in the neighbour, we obtain $y = g(x)$, with

$$|y - \overline{x}| = |g(x) - g(\overline{x})| < |x - \overline{x}|. \tag{5.16}$$

g is thus an application on the neighbourhood, and the theorem applies.

5.2 Non linear systems

Let us suppose to have n non linear equations in n variables, such as

$$\begin{aligned} f_1(x_1, \ldots, x_n) = f_1(\mathbf{x}) &= 0, \\ &\vdots, \\ f_i(x_1, \ldots, x_n) = f_i(\mathbf{x}) &= 0, \\ &\vdots, \\ f_n(x_1, \ldots, x_n) = f_n(\mathbf{x}) &= 0. \end{aligned} \tag{5.17}$$

This may be rewritten by considering the f_i as components of a vector $\mathbf{F}$ (i.e., $\mathbf{F}$ is from $\mathbb{R}^n$ to $\mathbb{R}^n$), and write

$$\mathbf{F}(\mathbf{x}) = 0. \tag{5.18}$$

Considering the Jacobian matrix $J(\mathbf{x})$,

$$J(\mathbf{x})_{j,i} = \frac{\partial f_j}{\partial x_i}, \tag{5.19}$$

[4]Lagrange's formula or Mean Value Theorem.

we may generalise eq. 5.11 to the n dimensional case by writing

$$\mathbf{x}_{n+1} = \mathbf{x}_n - J^{-1}(\mathbf{x}_n)\mathbf{F}(\mathbf{x}_n). \tag{5.20}$$

The matrix J and function $\mathbf{F}$ will be computed at any $\mathbf{x}_n$. There is no need to compute the inverse of the Jacobian; since we know that inverse computation is time consuming with respect to the solution of a system, we may define

$$\mathbf{y} = J^{-1}(\mathbf{x})\mathbf{F}(\mathbf{x}), \tag{5.21}$$

solve the system

$$J(\mathbf{x})\mathbf{y} = \mathbf{F}(\mathbf{x}), \tag{5.22}$$

and compute

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{y}_n. \tag{5.23}$$

The techniques used for solving linear algebra problems (i.e., Gaussian Elimination) find thus an application in the non-linear problems.

Chapter 6

Practice exercises

1. **Exercise**

   (a) Write explicitly the following matrices in the 5×5 case ($j = 1, ..., 5$; $i = 1, ..., 5$)

   i. $a_{j,i} = i + j^2$

   ii. $a_{j,i} = \frac{1}{ij}$

   iii. $a_{j,i} = i\delta_{j,i)} + 2\delta_{j,i-2} + 3j\delta_{j,i+1}$

   iv. $a_{j,i} = \delta_{j,(i^2-3i-1)}$

   Here δ is the Kronecker symbol (see also notes online).

   (b) Which of these matrices is symmetrical $a^T = a$ and why you could tell that without any computation?

2. **Exercise** Consider the matrix $a_{j,i} = \sqrt{j}\,\delta_{j,i-1}$.

   (a) Compute the transpose matrix a^T, the matrix $N = a^T a$ and the commutator $[a, a^T] \equiv aa^T - a^T a$. Here we have $i, j \in \mathbb{N}$ i.e. they may assume any positive value (infinite matrix). Anyway, if you are not able to manage the symbolic sums, you may help yourself using a finite example, and then generalise.

(b) Particle physicists call a vector

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ \alpha \\ 0 \\ \vdots \end{pmatrix} = |3>$$

i.e. a vector with all entries equal to 0 except the fourth (and regardless of the explicit value of $\alpha \neq 0$) a "3 particle state", i.e. a physical state with 3 particles. In general, a n particle state has only the $(n+1)$th entry as non-zero

$$|n> \text{ has components } x_j = \alpha\delta_{n,j-1}$$

They also call the matrix a a particle annihilator, a^T a particle creator, and the matrix N the "particle number". Do you understand why?

3. **Exercise** Consider the matrices

$$J_z = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix} \quad J_x = \begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & 0 \end{pmatrix} \quad J_y = \begin{pmatrix} 0 & \frac{-i}{\sqrt{2}} & 0 \\ \frac{i}{\sqrt{2}} & 0 & \frac{-i}{\sqrt{2}} \\ 0 & \frac{i}{\sqrt{2}} & 0 \end{pmatrix}$$

where i is the imaginary unit $i^2 = -1$.

(a) Compute the matrix given by the commutators $[J_x, J_y] \equiv J_xJ_y - J_yJ_x$, $[J_x, J_z]$, $[J_y, J_z]$ and write the results as a function of the J matrices (i.e., $[J_x, J_y] = \alpha J_x + \beta J_y + \gamma J_z$. Hint: the result should be trivial, i.e. the commutator of two matrices gives another matrix multiplied by a constant).

(b) Compute the matrix $J^2 = J_xJ_x + J_yJ_y + J_zJ_z$. Then write down the commutators $[J^2, J_x]$, $[J^2, J_y]$, $[J^2, J_z]$. You should be able to do the commutator part by glance, i.e. without any computation...[1]

[1] Also this matrices are important in Quantum Physics. They represent angular momentum in a spin 1 system.

4. **Exercise** Compute the determinant of the following matrices

$$A_1 = \begin{pmatrix} -2 & 1 & -1 & 7 \\ -1 & 4 & 1 & 1 \\ 5 & 2 & 0 & 1 \\ 2 & 3 & 6 & 0 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 2 & 1/2 & 1/3 & 0 \\ 1 & 4 & 0 & 1 \\ 1/10 & 1/20 & 1 & 1/2 \\ 1 & 1 & 1 & 7/2 \end{pmatrix}$$

$$A_3 = \begin{pmatrix} -1 & 0 & 2 & 0 \\ 1 & 1 & 5 & 1 \\ 3 & 0 & 0 & 1 \\ 2 & -1 & 1 & 4 \end{pmatrix}$$

$$A_4 = \begin{pmatrix} 3 & 2 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 4 & 0 & 1 & 2 \\ 0 & 9 & 0 & -6 \end{pmatrix}$$

$$A_5 = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 2 & 1 & -1 & -1 \\ 1 & 2 & 0 & 1 \\ 1 & 3 & 5 & 2 \end{pmatrix}$$

5. **Exercise** If possible, invert the matrices at the previous point. If not possible, explain why. Check if the result is correct in each case.

6. **Exercise** Write the $\| \ \|_\infty$ norm for each of the matrices above, and compute the conditioning number $\kappa(A) = \|A\|_\infty \|A^{-1}\|_\infty$

7. **Exercise**

   (a) Perform the $A = LU$ decomposition for all matrices above. If the matrix may not be written in the LU form, write it as $PA = LU$ using the proper permutation matrix.

   (b) For each matrix, write P^{-1}, and check that $A = P^{-1}LU$

8. **Exercise** Given

$$\mathbf{b}_1 = \begin{pmatrix} 21 \\ 5 \\ 10 \\ -1 \end{pmatrix}$$

$$\mathbf{b}_2 = \begin{pmatrix} 23/6 \\ 6 \\ 0.85 \\ -12 \end{pmatrix}$$

$$\mathbf{b}_3 = \begin{pmatrix} 6 \\ 12 \\ -2 \\ -4 \end{pmatrix}$$

$$\mathbf{b}_5 = \begin{pmatrix} 5/4 \\ 5/4 \\ 0 \\ 17/4 \end{pmatrix}$$

solve the systems $A_1\mathbf{x}_1 = \mathbf{b}_1$, $A_2\mathbf{x}_2 = \mathbf{b}_2$, $A_3\mathbf{x}_3 = \mathbf{b}_3$, $A_5\mathbf{x}_5 = \mathbf{b}_5$ using partial scaled pivoting. Check the results.

9. **Exercise** For each of the systems above, write permutation matrices that would allow you to solve the systems $PA\mathbf{x} = P\mathbf{b}$ using partial scaled pivoting without row swaps, and the corresponding inverse matrices P^{-1}.

10. **Exercise** Solve the systems above using LU decomposition.

11. **Exercise** Solve the systems above using the inverse matrix.

12. **Exercise** In case it may be theoretically proved that one of the systems above may be solved using the Jacobi method, perform a few iterations starting with the seed vector $\mathbf{x_0}$ of your choice. Use the infinite norm to compute at each step the approximate error $\frac{\|\mathbf{x}_n - \mathbf{x}_{n-1}\|}{\|\mathbf{x}_n\|}$, and the actual error $\frac{\|\mathbf{x} - \mathbf{x}_{n-1}\|}{\|\mathbf{x}\|}$. (Hints: the exact solution is known by the exercises above; row manipulation may be necessary before applying the Jacobi method).

13. **Exercise** Repeat the point above for the Gauss-Seidel method.

14. **Exercise**

    (a) Write the exact solution for the eigenvalues and (normalised) eigenvectors of the following matrices.

$$A_1 = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$A_3 = \begin{pmatrix} 2 & \sqrt{2} \\ \sqrt{2} & 3 \end{pmatrix}$$

    (b) Diagonalise the matrices, i.e. rewrite them in the $A = O^T D O$ form, where O is a orthogonal matrix and D a diagonal one.

    (c) Explain for which of these matrices you may use the power method to find the maximum eigenvalue and the corresponding eigenvector. Perform a few iterations of the power method (starting from a $(1, 1)^T$ vector) and compare to the exact result using the infinite norm (measure both the progress of the solution, i.e. the distance from the previous step, and the distance from the exact solution. Check both the eigenvalue and eigenvector).

Appendix A

Kronecker δ and symbolic sums

A.1 δ symbol

The Kronecker δ symbol is defined as

$$\delta_{j,i} = \begin{cases} 1 \text{ if } j = i \\ 0 \text{ if } j \neq i \end{cases} , \tag{A.1}$$

where j and i assume integer values. For example, we have $\delta_{3,0} = 0$, and $\delta_{-1,-1} = 1$.

The identity matrix has clearly $\mathbb{1}_{ij} = \delta_{ij}$, or

$$\mathbb{1} = \begin{pmatrix} 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & 0 & \ddots & 0 & \vdots \\ \vdots & \vdots & 0 & \ddots & \vdots \\ 0 & \dots & \dots & 0 & 1 \end{pmatrix} . \tag{A.2}$$

Similarly, for $j = 1, ..., m$, $i = 1, ..., n$ we may write the entries of the δ

symbol as a matrix. We have

$$\begin{pmatrix} 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & 0 & \ddots & 0 & \vdots \\ \vdots & \vdots & 0 & \ddots & \vdots \\ 0 & \dots & \dots & 0 & 1 \\ 0 & \dots & \dots & \dots & 0 \\ \vdots & \dots & \dots & \dots & \vdots \\ 0 & \dots & \dots & \dots & 0 \end{pmatrix} \qquad \text{if } m > n, \tag{A.3}$$

and

$$\begin{pmatrix} 1 & 0 & \dots & \dots & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 & \vdots & \dots & \vdots \\ \vdots & 0 & \ddots & 0 & \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & 0 & \ddots & \vdots & \vdots & \dots & \vdots \\ 0 & \dots & \dots & 0 & 1 & 0 & \dots & 0 \end{pmatrix} \qquad \text{if } m < n. \tag{A.4}$$

We may also write expressions such as $\delta_{j,i+1}$, which means

$$\delta_{j,i+1} = \begin{cases} 1 \text{ if } j = i+1 \text{ (or } i = j-1) \\ 0 \text{ if } j \neq i+1 \end{cases} . \tag{A.5}$$

As an example, if $j = 1$ and $i = 1, ..., n$, then $\delta_{j,i+1} = 0 \ \forall i$, while if $j = 2$ we have $\delta_{j,i+1} = 1$ if $i = 1$. In other words, and using $n = 7$ just for explicative purposes,

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} . \tag{A.6}$$

In a similar way,

$$\delta_{j,i+2} = \begin{cases} 1 \text{ if } j = i+2 \text{ (or } i = j-2) \\ 0 \text{ if } j \neq i+2 \end{cases} \tag{A.7}$$

which may be written as

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} . \tag{A.8}$$

And

$$\delta_{j,i-3} = \begin{cases} 1 \text{ if } j = i-3 \text{ (or } i = j+3) \\ 0 \text{ if } j \neq i-3 \end{cases} , \tag{A.9}$$

represented by

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} . \tag{A.10}$$

We may also write expressions including functions of i or j

$$j\ \delta_{j,i} = \begin{cases} j\ (= i) \text{ if } j = i \\ 0 \text{ if } j \neq i \end{cases} \tag{A.11}$$

or

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 7 \end{pmatrix} . \tag{A.12}$$

We clearly have

$$j\ \delta_{j,i} = i\ \delta_{j,i}. \tag{A.13}$$

Also,

$$i^2\ \delta_{j,i} = \begin{cases} i^2\ (= j^2) \text{ if } j = i \\ 0 \text{ if } j \neq i \end{cases} , \tag{A.14}$$

represented by

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 16 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 25 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 36 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 49 \end{pmatrix}, \tag{A.15}$$

for which

$$i^2\ \delta_{j,i} = j^2\ \delta_{j,i}. \tag{A.16}$$

And more in general we have

$$f(j)\ \delta_{j,i} = f(i)\ \delta_{j,i}. \tag{A.17}$$

Let us consider

$$j\ \delta_{j,i+1} = \begin{cases} j\ (= i+1) \text{ if } j = i+1 \\ 0 \text{ if } j \neq i+1 \end{cases}, \tag{A.18}$$

which for $j = 1, ..., 7$, $i = 1, ..., 7$ is

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 & 0 \end{pmatrix}. \tag{A.19}$$

In this case,

$$j\ \delta_{j,i+1} = (i+1)\ \delta_{j,i+1} \tag{A.20}$$

applies.

If we write

$$j^2\ \delta_{j,i-1} = \begin{cases} j^2\ (= (i-1)^2) \text{ if } j = i-1 \\ 0 \text{ if } j \neq i-1 \end{cases}, \tag{A.21}$$

we have

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 16 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 36 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{A.22}$$

And in general

$$f(j)\ \delta_{j,i-1} = f(i-1)\ \delta_{j,i-1}, \tag{A.23}$$

and so on.

A.2 Symbolic sums

When we sum over an index, that index becomes "dummy" and dependence on that index disappears from the expression. For example, if $\alpha_{k,n}$ has a dependence on 2 indexes, the expression

$$\sum_n \alpha_{k,n} = \alpha_{k,1} + \alpha_{k,2} + ... + \alpha_{k,n} \equiv \beta_k \tag{A.24}$$

depends on the only index k.

We already know this from matrices

$$\sum_n \alpha_{k,n}\beta_{n,i} = \gamma_{k,i}. \tag{A.25}$$

When these sums involve δ symbols, the summation brings a simplification. Let us consider

$$\sum_{i=1}^{4} f(i)\ \delta_{3,i} = f(1)\delta_{3,1} + f(2)\delta_{3,2} + f(3)\delta_{3,3} + f(4)\delta_{3,4} = f(3). \tag{A.26}$$

Not only the i index disappears, but also the δ symbol, and all is in the summation are fixed to the value that accompanies i in the δ. Formally

$$\sum_i f(i)\ \delta_{j,i} = f(j). \tag{A.27}$$

In a similar way

$$\sum_{i=1}^{4} \alpha_{j,l}\delta_{l,3} = \alpha_{j,1}\delta_{1,3} + \alpha_{j,2}\delta_{2,3} + \alpha_{j,3}\delta_{3,3} + \alpha_{j,4}\delta_{4,3} = \alpha_{j,3}. \tag{A.28}$$

Formally

$$\sum_{i} \alpha_{j,l}\delta_{l,i} = \alpha_{j,i}. \tag{A.29}$$

And

$$\sum_{i=1}^{4} f(l)\alpha_{j,l}\delta_{l,3} = f(1)\alpha_{j,1}\delta_{1,3}+f(2)\alpha_{j,2}\delta_{2,3}+f(3)\alpha_{j,3}\delta_{3,3}+f(4)\alpha_{j,4}\delta_{4,3} = f(3)\alpha_{j,3}, \tag{A.30}$$

generalising to

$$\sum_{i} f(l)\alpha_{j,l}\delta_{l,i} = f(i)\alpha_{j,i}. \tag{A.31}$$

Even if more than one δ symbol is present, a summation can cancel only one of them, since the correct index dependence has to persist, so we have

$$\sum_{l} l\ \delta_{j,l}\delta_{l,i} = j\ \delta_{j,i}, \tag{A.32}$$

if we use the summation to cancel the first δ, and

$$\sum_{l} l\ \delta_{j,l}\delta_{l,i} = i\ \delta_{j,i}, \tag{A.33}$$

if we use it to cancel the second one, nevertheless the two results are equivalent (eq. A.13). To better clarify, if for example $l = 1, ..., 3$, we have

$$\sum_{l} \delta_{j,l}\delta_{l,i} = \delta_{j,1}\delta_{1,i} + \delta_{j,2}\delta_{2,i} + \delta_{j,3}\delta_{1,i}, \tag{A.34}$$

which is an expression that survives only if $j = i$, i.e. a $\delta_{j,i}$.

Nevertheless two summations may cancel two δs, as in

$$\sum_{l}\sum_{n} l\ n^2\ \alpha_{k,l}\delta_{l,n}\delta_{n,i} = \sum_{l} l\ i^2\ \alpha_{k,l}\delta_{l,i} = i^3\ \alpha_{k,i}. \tag{A.35}$$

A.3 Application: the Derivative-Multiplication Commutator

We have seen that derivation on polynomials may be written as an infinite matrix

$$D_{j,i} = j\ \delta_{j,i-1}, \tag{A.36}$$

acting on the infinite vector of the coefficient of the polynomial $\mathbf{c}$

$$\sum_i c_i\ x^i. \tag{A.37}$$

In a similar way, the multiplication by x is given by the matrix

$$M_{j,i} = \delta_{j,i+1}. \tag{A.38}$$

Let us define the commutator C

$$C = [D, M] \equiv DM - MD. \tag{A.39}$$

We have

$$\begin{aligned} C_{j,i} = \sum_l D_{j,l} M_{l,i} - \sum_l M_{j,l} D_{l,i} = \\ = \left(\sum_l j\ \delta_{j,l-1} \delta_{l,i+1} \right) - \left(\sum_l \delta_{j,l+1}\ (l)\ \delta_{l,i-1} \right) = \\ = j\ \delta_{j,(l=i+1)-1} - (l = i-1)\ \delta_{j,(l=i-1)+1} = \\ = j\ \delta_{j,i} - (i-1)\ \delta_{j,i} = (j - j + 1)\delta_{j,i} \end{aligned} \tag{A.40}$$

$$\Rightarrow C_{j,i} = \delta_{j,i} \tag{A.41}$$

Appendix B

Programs

B.1 Matrix multiplication

```
#include<iostream>
#include<fstream>
#include<math.h>
#include<cstdlib>

//Matrix multiplication


using namespace std;

/*
The files with matrices have to be writte as
2 3
1 2 3
2 3 4

it means 2 rows, 3 columns and then the matrix
*/


int main()
```

```
{
  ifstream in1("matrice1.dat");    //matrix on the left
  ifstream in2("matrice2.dat");    //matrix on the right
  int r1,r2,c1,c2;    //corresponding rows and columns
  in1 >> r1;//reads from each file the size of the matrices
  in1 >> c1;
  in2 >> r2;
  in2 >> c2;
  if(c1!=r2){cout << "wrong rows columns!" << endl;return 0;}
//check
  double m1[r1][c1];
  double m2[r2][c2];
  double mf[r1][c2];
  for(int j=0;j<r1;j++) //and then reads the matrix
    {
      for(int i=0;i<c1;i++) in1 >> m1[j][i];
    }
  for(int j=0;j<r2;j++)
    {
      for(int i=0;i<c2;i++) in2 >> m2[j][i];
    }
  for(int j=0;j<r1;j++)  //computes
    for(int i=0;i<c2;i++)
      {
        mf[j][i]=0;
        for(int l=0;l<c1;l++) mf[j][i]+=m1[j][l]*m2[l][i];
      }
  cout << "LEFT MATRIX" << endl;
  for(int j=0;j<r1;j++)
    {
      for(int i=0;i<c1;i++) cout << m1[j][i] << " ";
      cout << endl;
    }
  cout << "RIGHT MATRIX" << endl;
  for(int j=0;j<r2;j++)
    {
      for(int i=0;i<c2;i++) cout << m2[j][i] << " ";
```

```
      cout << endl;
    }
  cout << "PRODUCT MATRIX" << endl;
  for(int j=0;j<r1;j++)
    {
      for(int i=0;i<c2;i++) cout << mf[j][i] << " ";
      cout << endl;
    }
  return 0;
}
```

B.2 Gaussian elimination

```
#include<iostream>
#include<fstream>
#include<math.h>
#include<cstdlib>

//Gaussian Elimination with Partial Pivoting


using namespace std;

/*
takes the system
1x + 2y =0
2x +8y =1

in the form

2
1 2 0
2 8 1

*/


int main()
{
  ifstream in("matrice.dat");     //file with the matrix
  int n;        //assumed square, no checks
  in >> n;
  double a[n][n];    //matrix
  double b[n];    //constant term
  double x[n];     //variables
  double m[n][n];   //multipliers
  double s[n];    //scaling factors
  for(int j=0;j<n;j++) //reads matrix and constant term
```

```
  {
    for(int i=0;i<n;i++) in >> a[j][i];
    in >> b[j];
  }
cout << "System" << endl;
for(int j=0;j<n;j++) //shows the system on console
  {
    for(int i=0;i<n;i++)
      {
        cout << " ";
        if(a[j][i]>=0) cout << "+";
        cout << a[j][i] << "x" << i;
      }
    cout << " = " << b[j] << endl;
  }
for(int j=0;j<n;j++) //computes scaling factors
  {
    s[j]=fabs(a[j][0]);
    for(int i=1;i<n;i++)
      {
        double ls=fabs(a[j][i]);
        if(ls>s[j]) s[j]=ls;
      }
    if(!s[j])
      {
        cout << "No unique solution, row of zeros!" << endl;
        return 0;
      }
  }
cout << "Scaling factors" << endl;
for(int j=0;j<n;j++) cout << "s[" << j << "]=" << s[j] << endl;
for(int j=0;j<n-1;j++)
  {
    cout << "Step " << j << endl;
    double max=fabs(a[j][j])/s[j];
    int ml=j;
    for(int l=j+1;l<n;l++)
```

```
    {
      double ls=fabs(a[l][j])/s[l];
      if(ls>max)
        {
          max=ls;
          ml=l;
        }
    }
  if(!max)
    {
      cout << "No unique solution! (zero pivot)" << endl;
      return 0;
    }
  else
    {
      if(ml!=j)  //swaps
        {
          cout << "Swap of row " << j << " and " << ml << endl;
          double swap;
          for(int i=j;i<n;i++)
            {
              swap=a[j][i];
              a[j][i]=a[ml][i];
              a[ml][i]=swap;
            }
          swap=s[j];
          s[j]=s[ml];
          s[ml]=swap;
          swap=b[j];
          b[j]=b[ml];
          b[ml]=swap;
          cout << "After swapping" << endl;
          for(int k=0;k<n;k++) //shows the system on console
            {
              for(int i=0;i<n;i++)
                {
                  cout << " ";
```

```
                    if(a[k][i]>=0) cout << "+";
                    cout << a[k][i] << "x" << i;
                  }
                cout << " = " << b[k] << endl;
              }
          }
        for(int l=j+1;l<n;l++)  //computes multiplying factors
          {
            m[j][l]=a[l][j]/a[j][j];
            a[l][j]=0;
            for(int i=j+1;i<n;i++)
              {
                a[l][i]-=a[j][i]*m[j][l];
              }
            b[l]-=b[j]*m[j][l];
          }
        cout << "Multiplying factors" << endl;
        for(int l=j+1;l<n;l++) cout << "m[" << l << "]="
              << m[j][l] << endl;
      }
    cout << "After subtraction" << endl;
    for(int k=0;k<n;k++) //shows the system on console
      {
        for(int i=0;i<n;i++)
          {
            cout << " ";
            if(a[k][i]>=0) cout << "+";
            cout << a[k][i] << "x" << i;
          }
        cout << " = " << b[k] << endl;
      }
  }
cout << "After manipulation" << endl;  //writes the triangular system
for(int k=0;k<n;k++) //shows the system on console
  {
    for(int i=0;i<n;i++)
      {
```

```
            cout << " ";
            if(a[k][i]>=0) cout << "+";
            cout << a[k][i] << "x" << i;
          }
        cout << " = " << b[k] << endl;
      }
    if(!a[n-1][n-1])
      {
        cout << "No unique solution! (last row of zeros)" << endl;
        return 0;
      }
    for(int j=n-1;j>=0;j--)    //backward substitution
      {
        x[j]=b[j];
        for(int i=j+1;i<n;i++)
          {
            x[j]-=a[j][i]*x[i];
          }
        x[j]/=a[j][j];
      }
    cout << "Solution:" << endl;    //writes the result
    for(int i=0;i<n;i++) cout << "x" << i << "=" << x[i] << endl;
    return 0;
}
```

B.3 Inverse matrix

```
#include<iostream>
#include<fstream>
#include<math.h>
#include<cstdlib>

//Matrix inversion with Partial Pivoting


using namespace std;


int main()
{
  ifstream in("matricei.dat");     //file with the matrix
  int n;       //assumed square, no checks
  in >> n;
  double a[n][n];    //matrix
  double ac[n][n];  //original matrix
  double B[n][n];  //for the solution of the n systems
  double a_1[n][n]; //inverse
  double m[n][n];  //multipliers
  double s[n];    //scaling factors
  for(int j=0;j<n;j++) //reads matrix
    {
      for(int i=0;i<n;i++) {in >> a[j][i];ac[j][i]=a[j][i];}
    }
  for(int j=0;j<n;j++) //computes scaling factors
    {
      s[j]=fabs(a[j][0]);
      for(int i=1;i<n;i++)
        {
          double ls=fabs(a[j][i]);
          if(ls>s[j]) s[j]=ls;
        }
      if(!s[j]) {cout << "No inverse, row of zeros!" << endl; return 0;}
```

```
    }
  for(int j=0;j<n;j++)
     //the know terms are initially a unity matrix
    for(int i=0;i<n;i++)
      {
        if(i==j) B[j][i]=1;
        else B[j][i]=0;
      }
  cout << "Before manipulation " << endl;
     //shows the system before manipulation
  for(int j=0;j<n;j++)
    {
      for(int i=0;i<n;i++)
        {
          if(a[j][i]>=0) cout << "+";
          cout << a[j][i] << "x" << i << " ";
        }
      cout << "= ";
      for(int i=0;i<n;i++) cout << " " << B[j][i];
      cout << endl;
    }
  for(int j=0;j<n-1;j++)    //pivot
    {
      double max=fabs(a[j][j])/s[j];
      int ml=j;
      for(int l=j+1;l<n;l++)
        {
          double ls=fabs(a[l][j])/s[l];
          if(ls>max)
            {
              max=ls;
              ml=l;
            }
        }
      if(!max)
      {cout << "No inverse! (zero pivot)" << endl; return 0;}
      else
```

```
        {
          if(ml!=j)   //swaps
            {
              double swap;
              for(int i=j;i<n;i++)
                {
                  swap=a[j][i];
                  a[j][i]=a[ml][i];
                  a[ml][i]=swap;
                }
              for(int k=0;k<n;k++)
                {
                  swap=B[j][k];
                  B[j][k]=B[ml][k];
                  B[ml][k]=swap;
                }
              swap=s[j];
              s[j]=s[ml];
              s[ml]=swap;
            }
          for(int l=j+1;l<n;l++)  //computes multiplying factors
            {
              m[j][l]=a[l][j]/a[j][j];
              a[l][j]=0;
              for(int i=j+1;i<n;i++)
                {
                  a[l][i]-=a[j][i]*m[j][l];
                }
              for(int k=0;k<n;k++) B[l][k]-=B[j][k]*m[j][l];
            }
        }
    }
cout << "After manipulation" << endl;
for(int j=0;j<n;j++)
  {
    for(int i=0;i<n;i++)
      {
```

```
            if(a[j][i]>=0) cout << "+";
            cout << a[j][i] << "x" << i << " ";
          }
        cout << "= ";
        for(int i=0;i<n;i++) cout << " " << B[j][i];
        cout << endl;
      }
    if(!a[n-1][n-1])
      {
        cout << "No inverse! (last row of zeros)" << endl;
        return 0;
      }
      //checks that the inverse exists
    for(int k=0;k<n;k++)
      {
        for(int j=n-1;j>=0;j--)    //backward substitution
          {
            a_1[j][k]=B[j][k];
            for(int i=j+1;i<n;i++)
              {
                a_1[j][k]-=a[j][i]*a_1[i][k];
              }
            a_1[j][k]/=a[j][j];
          }
      }
    cout << "Inverse matrix, solution:" << endl;//writes result
    for(int j=0;j<n;j++)
      {
        for(int i=0;i<n;i++) cout << a_1[j][i] << " ";
        cout << endl;
      }
    double p[n][n];
    for(int j=0;j<n;j++)
      {
        for(int i=0;i<n;i++)
          {
            p[j][i]=0;
```

```
            for(int l=0;l<n;l++)
              {
                p[j][i]+=a_1[j][l]*ac[l][i];
              }
          }
      }
    cout << "Check product:" << endl;//check the result
    for(int j=0;j<n;j++)
      {
        for(int i=0;i<n;i++) cout << p[j][i] << " ";
        cout << endl;
      }
    double x[n];
    double b[n];
    //solves a system, it has to be included in the matrix file
    for(int i=0;i<n;i++) {in >> b[i];x[i]=0;}
    for(int j=0;j<n;j++)
      {
        for(int i=0;i<n;i++) x[j]+=a_1[j][i]*b[i];
      }
    cout << "System" << endl;
    for(int j=0;j<n;j++) //shows the system on console
      {
        for(int i=0;i<n;i++)
          {
            cout << " ";
            if(a[j][i]>=0) cout << "+";
            cout << a[j][i] << "x" << i;
          }
       cout << " = " << b[j] << endl;
     }
    cout << "Solution:" << endl;    //writes the result
    for(int i=0;i<n;i++) cout << "x"
      << i << "=" << x[i] << endl;
    return 0;
}
```

B.4 LU decomposition

```
#include<iostream>
#include<fstream>
#include<math.h>
#include<cstdlib>

//LU decomposition with permutation and solution of a system


using namespace std;



int main()
{
  ifstream in("matricelu2.dat");
    //file with the matrix and known terms
  int n;       //assumed square, no checks
  in >> n;
  double a[n][n];    //matrix
  double m[n][n];   //multipliers
  int P[n][n];   //permutation matrix
  int s[n];    //swap vector
  double b[n];    //known vector
  for(int j=0;j<n;j++)
    {
      s[j]=j;  //no swap at beginning
      m[j][j]=1;    //there will be ones on the diag
      for(int i=j+1;i<n;i++) m[j][i]=0;//starts with 0 elsewhere
    }
  for(int j=0;j<n;j++) //reads matrix
    {
      for(int i=0;i<n;i++) {in >> a[j][i];P[j][i]=0;}
      in >> b[j];
    }
  cout << "Matrix" << endl;
```

```
 for(int j=0;j<n;j++) //shows the matrix on console
   {
     for(int i=0;i<n;i++)
       {
         cout << "␣";
         if(a[j][i]>=0) cout << "+";
         cout << a[j][i];
       }
     cout << endl;
   }
cout << "System" << endl;
 for(int j=0;j<n;j++) //shows the system on console
   {
     for(int i=0;i<n;i++)
       {
         cout << "␣";
         if(a[j][i]>=0) cout << "+";
         cout << a[j][i] << "x" << i;
       }
     cout << "␣=␣" << b[j] << endl;
   }
 for(int j=0;j<n-1;j++)
   {
     if(!a[j][j]) //if the pivot is 0
       {
         bool found=false;
          //looks for another anche tracks if there is one
         double swap;
         for(int l=j+1;l<n;l++)
           {
             if(a[l][j]) //if non-zeo
               {
                 swap=s[j];  //swaps s, a,
                 s[j]=s[l];
                 s[l]=swap;
                 for(int i=j;i<n;i++)
                   {
```

```
                    swap=a[j][i];
                    a[j][i]=a[l][i];
                    a[l][i]=swap;
                  }
                for(int k=0;k<j;k++)
                  {
                    swap=m[j][k];
                    m[j][k]=m[l][k];
                    m[l][k]=swap;
                  }
                found=true;
                break;
              }
          }
        if(!found) //then sungular
          {
            cout << "Singular matrix" << endl;
            return 0;
          }
      }//computes multiplying factors (does not act on b!)
    for(int l=j+1;l<n;l++)
      {
        m[l][j]=a[l][j]/a[j][j];
        a[l][j]=0;
        for(int i=j+1;i<n;i++)
          {
            a[l][i]-=a[j][i]*m[l][j];
          }
      }
  }
cout << "Lower" << endl;  //writes the triangular system
for(int k=0;k<n;k++) //shows the system on console
  {
    for(int i=0;i<n;i++)
      {
        cout << " ";
        if(m[k][i]>=0) cout << "+";
```

```
        cout << m[k][i];
      }
    cout << endl;
  }
cout << "Upper" << endl;  //writes the triangular system
for(int k=0;k<n;k++) //shows the system on console
  {
    for(int i=0;i<n;i++)
      {
        cout << " ";
        if(a[k][i]>=0) cout << "+";
        cout << a[k][i];
      }
    cout << endl;
  }
cout << "Swap" << endl;//vector of swaps and permutation matrix
for(int i=0;i<n;i++)
  {
    cout << "s[" << i << "]=" << s[i] << endl;
    P[i][s[i]]=1;
  }
double b_swap[n];//swaps b
double b_swap2[n];
for (int j=0;j<n;j++)
  {
    b_swap[j]=b[s[j]];
    b_swap2[j]=0;
    for(int i=0;i<n;i++)
      {
        b_swap2[j]+=P[j][i]*b[i];
      }
  }
cout << "b swapped" << endl;//checks the equivalence
for(int i=0;i<n;i++) cout << "b_swap[" << i << "]="
            << b_swap[i] << " " << b_swap2[i] << endl;
double o[n][n];
for(int j=0;j<n;j++)
```

```
    {
      for(int i=0;i<n;i++)
        {
          o[j][i]=0;
          for(int l=0;l<n;l++) o[j][i]+=m[j][l]*a[l][i];
        }
    }
  double orig[n][n];
  for(int j=0;j<n;j++)
    {
      for(int i=0;i<n;i++)
        {
          orig[s[j]][i]=o[j][i];
        }
    }
  double permut[n][n];
  for(int j=0;j<n;j++)
    {
      for(int i=0;i<n;i++)
        {
          permut[j][i]=0;
            for(int l=0;l<n;l++)
              {
                permut[j][i]+=P[j][l]*orig[l][i];
              }
        }
    }
  cout << "Product" << endl;  //writes the triangular system
  for(int k=0;k<n;k++) //shows the system on console
    {
      for(int i=0;i<n;i++)
        {
          cout << " ";
          if(o[k][i]>=0) cout << "+";
          cout << o[k][i];
        }
      cout << endl;
```

```
  }
cout << "Original" << endl;  //writes the triangular system
for(int k=0;k<n;k++) //shows the system on console
  {
    for(int i=0;i<n;i++)
      {
        cout << " ";
        if(orig[k][i]>=0) cout << "+";
        cout << orig[k][i];
      }
    cout << endl;
  }
cout << "Permutation" << endl;
for(int k=0;k<n;k++) //shows the system on console
  {
    for(int i=0;i<n;i++)
      {
        cout << " ";
        cout << P[k][i];
      }
    cout << endl;
  }
cout << "Permutated" << endl;  //writes the triangular system
for(int k=0;k<n;k++) //shows the system on console
  {
    for(int i=0;i<n;i++)
      {
        cout << " ";
        if(permut[k][i]>=0) cout << "+";
        cout << permut[k][i];
      }
    cout << endl;
  }
double y[n];
for(int i=0;i<n;i++)//forward sostitution, uses diag=1
  {
    y[i]=b_swap[i];
```

```
        for(int k=0;k<i;k++)
          {
            y[i]-=m[i][k]*y[k];
          }
      }
    cout << "Y:" << endl;  //writes the result
    for(int i=0;i<n;i++) cout << "y" << i << "=" << y[i] << endl;
    double x[n];
    for(int j=n-1;j>=0;j--)   //backward substitution
      {
        x[j]=y[j];
        for(int i=j+1;i<n;i++)
          {
            x[j]-=a[j][i]*x[i];
          }
        x[j]/=a[j][j];
      }
    cout << "Solution:" << endl;   //writes the result
    for(int i=0;i<n;i++) cout << "x" << i << "=" << x[i] << endl;
    return 0;
}
```

B.5 Iterative methods

```
#include<iostream>
#include<fstream>
#include<math.h>
#include<cstdlib>



using namespace std;

//Jacobi and Gauss-seidel methods

int main()
{
  ifstream in("matricejg.dat");     //file with the matrix
  ofstream out("outjacog.dat");
  int iter;
  int n;       //assumed square, no checks
  in >> n;
  double a[n][n];    //matrix
  double b[n];    //constant term
  for(int j=0;j<n;j++) //reads matrix and constant term
    {
      for(int i=0;i<n;i++) in >> a[j][i];
      in >> b[j];
    }
  double x[n];      //variables
  for(int i=0;i<n;i++) x[i]=0;
  double x0[n];    //copy of vector
  for(int j=0;j<n;j++) //shows the system on console
    {
      for(int i=0;i<n;i++)
        {
          out << " ";
          if(a[j][i]>=0) out << "+";
          out << a[j][i] << "x" << i;
```

```
      }
    out << " = " << b[j] << endl;
  }
out << "Starting seed" << endl;//shows the starting seed
for(int i=0;i<n;i++) out << x[i] << endl;
double err=1;//initialises error and iteration
int maxiter=10000;
iter=0;
while((err>1e-8)&&(iter<maxiter))
  {
    iter++;
    out << "iter=" << iter << endl;//prints iteration
    for(int j=0;j<n;j++) x0[j]=x[j];//copies the vector
    for(int j=0;j<n;j++)
      {
        x[j]=b[j];//jacobi method iteration
        for(int i=0;i<j;i++) x[j]-=a[j][i]*x0[i];
        for(int i=j+1;i<n;i++) x[j]-=a[j][i]*x0[i];
        x[j]/=a[j][j];
      }
    err=0;
    double norm=0;
    for(int j=0;j<n;j++)//computes the change
      {
        double le=fabs(x[j]-x0[j]);
        double ln=fabs(x[j]);
        if(le>err) err=le;
        if(ln>norm) norm=ln;
      }
    if(iter<4)//prints first iterations for reference
      {
        out << "approximate solution" << endl;
        for(int j=0;j<n;j++)
          {
            out << x[j] << endl;
          }
      }
```

```
      out << "abserr=" << err << endl;//absolute error
      err/=norm;
      out << "err=" << err << endl;
    }
  out << "Solution" << endl;
  for(int j=0;j<n;j++)
    {
      out << x[j] << endl;
    }
  out << "Gauss-Seidel" << endl;
   //the same as above for Gauss-seidel
  for(int j=0;j<n;j++)
    {
      x[j]=1;
    }
 for(int j=0;j<n;j++)
    {
      for(int i=0;i<n;i++)
        {
          out << " ";
          if(a[j][i]>=0) out << "+";
          out << a[j][i] << "x" << i;
        }
      out << " = " << b[j] << endl;
    }
  out << "Starting seed" << endl;
  for(int i=0;i<n;i++) out << x[i] << endl;
  err=1;
  iter=0;
  while((err>1e-8)&&(iter<maxiter))
    {
      iter++;
      out << "iter=" << iter << endl;
      for(int j=0;j<n;j++) x0[j]=x[j];
       //save only for comparison
      for(int j=0;j<n;j++)
        {
```

```
            x[j]=b[j];
            for(int i=0;i<j;i++) x[j]-=a[j][i]*x[i];
             //only difference
            for(int i=j+1;i<n;i++) x[j]-=a[j][i]*x0[i];
            x[j]/=a[j][j];
          }
        err=0;
        double norm=0;
        for(int j=0;j<n;j++)
          {
            double le=fabs(x[j]-x0[j]);
            double ln=fabs(x[j]);
            if(le>err) err=le;
            if(ln>norm) norm=ln;
          }
        if(iter<4)
          {
            out << "approximate solution" << endl;
            for(int j=0;j<n;j++)
              {
                out << x[j] << endl;
              }
          }
        out << "abserr=" << err << endl;
        err/=norm;
        out << "err=" << err << endl;
      }
    out << "Solution" << endl;
    for(int j=0;j<n;j++)
      {
        out << x[j] << endl;
      }
    return 0;
}
```

B.6 Power method

```
#include<iostream>
#include<fstream>
#include<math.h>
#include<cstdlib>

//power method, assumes symmetric matrix

using namespace std;

double r2()      //a random function
{
    return (double)rand() / (double)RAND_MAX ;
}



int main()
{
  ifstream in("matricep.dat");
    //file with the matrix
  ofstream out("outpower.dat");
  int iter;
  int n;       //assumed square, no checks
  in >> n;
  double a[n][n];    //matrix
  out << "Matrix" << endl;
  for(int j=0;j<n;j++) //reads matrix
    {
      for(int i=0;i<n;i++)
        {in >> a[j][i]; out << a[j][i] << " ";}
      out << endl;
    }
  double x[n];//normalised eigenvector
  double y[n];//not normalised
  double v[n][n];//basis
```

```
double lambda[n];//eigenvalues
double errval;//error on eigenvalue
double errvect;//error on eigenvector
int maxiter=100000;//maximum iterations
double old_lambda;//previous eigenvector estimate
double a2[n][n];//square matrix
for(int i=0;i<n;i++)
  for(int k=0;k<n;k++)
    {
      a2[i][k]=0;
      for(int l=0;l<n;l++)
        {
          a2[i][k]+=a[i][l]*a[l][k];
          //computes square matrix
        }
    }
out << "Square matrix" << endl;
for(int i=0;i<n;i++)
  {
    for(int k=0;k<n;k++)
      {
        out << a2[i][k] << " ";
      }
    out << endl;
  }
for(int j=0;j<n;j++)//looks for the basis
  {
    iter=0;//initialises
    errval=1.;
    errvect=1.;
    old_lambda=0;
    bool start=false;
   //tries a seed which is not a linear
    //combination of found eigenvectors
    while(!start)
      {
        for(int i=0;i<n;i++) x[i]=r2();
```

```
            //with randomness, but not necessary
          x[j]=1.;//so that it is norm=1
          for(int i=0;i<j;i++)
            {
              double scal=0;
              for(int k=0;k<n;k++) scal+=x[k]*v[i][k];
              for(int k=0;k<n;k++) x[k]-=scal*v[i][k];
            }
          for(int k=0;k<n;k++) if(fabs(x[k])>1e-12) start=true;
        }
      while(((errval>1e-12)||(errvect>1e-12))&&(iter<maxiter))
        {
          if(iter)
            {
              for(int i=0;i<j;i++)
                {
                  double scal=0;
                  for(int k=0;k<n;k++) scal+=x[k]*v[i][k];
                  for(int k=0;k<n;k++) x[k]-=scal*v[i][k];
//does the subtraction at each step to avoid error propagation
                }
            }
          double maxx=0;
          for(int i=0;i<n;i++)
            {
              double lmx=fabs(x[i]);
              if(lmx>maxx) maxx=lmx;
            }
          for(int i=0;i<n;i++) x[i]/=maxx;
            //normalises (infinite norm)
          iter++;
          double fabslambda=0;//absolute value of eigenvalue
          lambda[j]=0;//eigenvalue
          for(int i=0;i<n;i++)
            {
              y[i]=0;
              for(int k=0;k<n;k++)
```

```
            {
              y[i]+=a2[i][k]*x[k];
              //does it first for square, should converge
            }
          double ln=fabs(y[i]);
          if(ln>fabslambda)
           {lambda[j]=y[i]/x[i];fabslambda=ln;}
          //the eigenvalue is approximated by the
          //ratio of maximum modulus components
        }
      if(fabslambda<1e-12)
        {
          errval=errvect=0;
          lambda[j]=0;
        }
      else
        {
          errval=fabs(lambda[j]-old_lambda)/fabslambda;
          old_lambda=lambda[j];
          errvect=0;
          for(int i=0;i<n;i++)
            {
              y[i]/=lambda[j];
              double ln=fabs(y[i]-x[i]);
              if(ln>errvect) errvect=ln;
              x[i]=y[i];
            }
        }
    }
  if((iter>=maxiter)&&(errval>1e-8)&&(errvect>1e-8))
     {out << "Did not converge at eigenvalue "
          << j << endl;return 0;}
  double n2=0;
  for(int i=0;i<n;i++)
    {
      n2+=x[i]*x[i];
    }
```

```
      n2=sqrt(n2);
      for(int i=0;i<n;i++)//norm 2 eigenvector
        {
          v[j][i]=x[i]/n2;
        }
      out << "Iterations=" << iter << endl;
      out << "Eigenvalue" << " " << j << "=" << lambda[j] << endl;
      out << "Error on eigenvalue=" << errval << endl;
      out << "Infinite norm eigenvector" << endl;
      for(int i=0;i<n;i++)
        {
          out << x[i] << endl;
        }
      out << "Error on eigenvector=" << errvect << endl;
      out << "2 norm eigenvector" << endl;
      for(int i=0;i<n;i++)
        {
          out << v[j][i] << endl;
        }
    }
  out << "Matrix" << endl;
   //the same for the original matrix, could not converge
  for(int j=0;j<n;j++)
    {
      iter=0;
      errval=1.;
      errvect=1.;
      old_lambda=0;
      bool start=false;
      while(!start)
        {
          for(int i=0;i<n;i++) x[i]=r2();
          x[j]=1.;
          for(int i=0;i<j;i++)
            {
              double scal=0;
              for(int k=0;k<n;k++) scal+=x[k]*v[i][k];
```

```
            for(int k=0;k<n;k++) x[k]-=scal*v[i][k];
          }
        for(int k=0;k<n;k++) if(fabs(x[k])>1e-12) start=true;
      }
    while(((errval>1e-12)||(errvect>1e-12))&&(iter<maxiter))
      {
        for(int i=0;i<j;i++)
          {
            double scal=0;
            for(int k=0;k<n;k++) scal+=x[k]*v[i][k];
            for(int k=0;k<n;k++) x[k]-=scal*v[i][k];
          }
        double maxx=0;
        for(int i=0;i<n;i++)
          {
            double lmx=fabs(x[i]);
            if(lmx>maxx) maxx=lmx;
          }
        for(int i=0;i<n;i++) x[i]/=maxx;
        iter++;
        double fabslambda=0;
        lambda[j]=0;
        for(int i=0;i<n;i++)
          {
            y[i]=0;
            for(int k=0;k<n;k++)
              {
                y[i]+=a[i][k]*x[k];
              }
            double ln=fabs(y[i]);
            if(ln>fabslambda)
              {lambda[j]=y[i]/x[i];fabslambda=ln;}
          }
        if(fabslambda<1e-12)
          {
            errval=errvect=0;
            lambda[j]=0;
```

```
        }
      else
        {
          errval=fabs(lambda[j]-old_lambda)/fabslambda;
          old_lambda=lambda[j];
          errvect=0;
          for(int i=0;i<n;i++)
            {
              y[i]/=lambda[j];
              double ln=fabs(y[i]-x[i]);
              if(ln>errvect) errvect=ln;
              x[i]=y[i];
            }
        }
    }
  if((iter>=maxiter)&&(errval>1e-8)&&(errvect>1e-8))
     {out << "Did not converge at eigenvalue "
          << j << endl;return 0;}
  double n2=0;
  for(int i=0;i<n;i++)
    {
      n2+=x[i]*x[i];
    }
  n2=sqrt(n2);
  for(int i=0;i<n;i++)
    {
      v[j][i]=x[i]/n2;
    }
  out << "Iterations=" << iter << endl;
  out << "Eigenvalue" << " " << j << "="
      << lambda[j] << endl;
  out << "Error on eigenvalue=" << errval << endl;
  out << "Infinite norm eigenvector" << endl;
  for(int i=0;i<n;i++)
    {
      out << x[i] << endl;
    }
```

```
        out << "Error on eigenvector=" << errvect << endl;
        out << "2 norm eigenvector" << endl;
        for(int i=0;i<n;i++)
          {
            out << v[j][i] << endl;
          }
      }
  return 0;
}
```

B.7 Non linear equations

```
#include<iostream>
#include<fstream>
#include<math.h>
#include<cstdlib>

//bisection, secant, newton and newton with numerical derivative

using namespace std;

double F(double x)//function
{
  return exp(x)-sin(x);
}

double Fp(double x)//exact derivative
{
  return exp(x)-cos(x);
}

double NFp(double x)//numerical derivative
{
  double eps=1e-8;
  return (F(x+eps)-F(x))/eps;
}

int main()
{
  ofstream out("newtons.dat");
  cout << "Wanted solution" << endl;
  int n;
  cin >> n;
  n=n-1;
  out << "#" << n+1
      << " negative solution of exp(x)-sin(x)" << endl;
  double eps=1e-12;
```

```
  out << "wanted precision=" << eps << endl;
  out.precision(16);
  double x1,x2;
  double xstart1,xstart2;
  int np=n%2;
  n=n/2;
//we know the position of zeros (approximate) studying the graph,
//the following points have opposite sign
  if(np)
    {
      x2=-(2*n+2)*M_PI;
      x1=-(2*n+3./2.)*M_PI;
    }
  else
    {
      x1=-(2*n+1)*M_PI;
      x2=-(2*n+3./2.)*M_PI;
    }
  xstart1=x1;
  xstart2=x2;
  double f1=F(x1);
  double f2=F(x2);
  if(!f1)
    {
      cout << "Your starting point " << x1
           << " was a solution!" << endl;
      return 0;
    }
  if(!f2)
    {
      cout << "Your starting point " << x2
        << " was a solution!" << endl;
      return 0;
    }
  out << "Bisection" << endl;
  out << "x1=" << x1 << " " << "F(x1)=" << f1 << endl;
  out << "x2=" << x2 << " " << "F(x2)=" << f2 << endl;
```

```
double delta=x1-x2;
double f,x;
int iter=0;
int maxiter=100;
while((delta>eps)&&(iter<maxiter))
  {
    iter++;
    x=delta/2.+x2;//point in the middle
    f=F(x);
    if((f*f1)>0) {x1=x;f1=f;}
    else {x2=x;f2=f;}
      //checks for signs and chooses opposite signs
    delta=x1-x2;
    out << "iter=" << iter << endl;
    out << "x=" << x << " x+" << 2*n+np+1 << "PI="
        << x+(2*n+np+1)*M_PI << " f(x)=" << f << endl;
    if(!f) {out << "Solution found!" << endl; break;}
  }
out << "Secant" << endl;
x1=xstart1;
x2=xstart2;
f1=F(x1);
f2=F(x2);
iter=0;
delta=x1-x2;
while((delta>eps)&&(iter<maxiter))
  {
//as in bisection, but looking for the secant intercept
    iter++;
    double deltaf=f1-f2;
    x=x2-f2*delta/deltaf;
    f=F(x);
    if((f*f1)>0) {x1=x;f1=f;}
    else {x2=x;f2=f;}
    delta=x1-x2;
    out << "iter=" << iter << endl;
    out << "x=" << x << " x+" << 2*n+np+1
```

```
            << "PI=" << x+(2*n+np+1)*M_PI << " f(x)=" << f << endl;
        if(!f) {out << "Solution found!" << endl; break;}
      }
    out << "Newton" << endl;
    x=xstart1;
    f=F(x);
    iter=0;
    //just one point is needed, looks for the derivative intercept
    while((fabs(f)>eps)&&(iter<maxiter))
      {
        iter++;
        double df=Fp(x);
        x=x-f/df;
        f=F(x);
        out << "iter=" << iter << endl;
        out << "x=" << x << " x+" << 2*n+np+1
              << "PI=" << x+(2*n+np+1)*M_PI << " f(x)=" << f << endl
      }
    out << "Newton" << endl;
    x=xstart1;
    f=F(x);
    iter=0;
    while((fabs(f)>eps)&&(iter<maxiter))
      {//computes derivative numerically
        iter++;
        double df=NFp(x);
        x=x-f/df;
        f=F(x);
        out << "iter=" << iter << endl;
        out << "x=" << x << " x+" << 2*n+np+1
            << "PI=" << x+(2*n+np+1)*M_PI
            << " f(x)=" << f << endl;
      }
    return 0;
}
```