LECTURE NOTES ON NUMERICAL LINEAR ALGEBRA FOR DATA SCIENCE

Martín Sombra

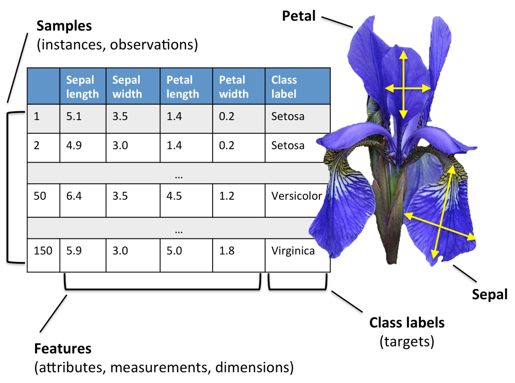
Arturo Vieiro

Contents

Introduction

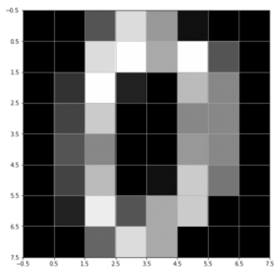
Matrices are used to represent data, and linear algebra provides mathematical tools to understand and manipulate them in order to derive useful knowledge like, for instance, linear relations. It is a building block of many of the basic techniques of data analysis, including dimensionality reduction, machine learning, image processing, and language recognition.

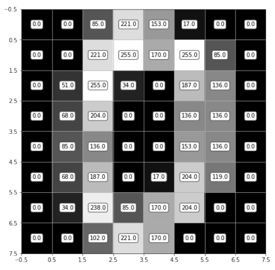
Data is usually collected in matrices, that is, numerical tables representing *samples* (or *data points*) with multiples *attributes* (or *variables*). Typically rows correspond to samples and columns to attributes, like in this representation of Fischer's iris flower dataset:


Samples
(instances, observations)
Petal
Sepal
Class labels
(targets)
Features
(attributes, measurements, dimensions)

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			…		
50	6.4	3.5	4.5	1.2	Versicolor
			…		
150	5.9	3.0	5.0	1.8	Virginica

Also digital b/w images are represented by matrices, since they are rectangular arrays of pixels, each of them coded by a value in the range from 0 (black) to 255 (white):





0.0	0.0	85.0	221.0	153.0	17.0	0.0	0.0
0.0	0.0	221.0	255.0	170.0	255.0	85.0	0.0
0.0	51.0	255.0	34.0	0.0	187.0	136.0	0.0
0.0	68.0	204.0	0.0	0.0	136.0	136.0	0.0
0.0	85.0	136.0	0.0	0.0	153.0	136.0	0.0
0.0	68.0	187.0	0.0	17.0	204.0	119.0	0.0
0.0	34.0	238.0	85.0	170.0	204.0	0.0	0.0
0.0	0.0	102.0	221.0	170.0	0.0	0.0	0.0

When confronted to a matrix, we might ask questions like:

- Are all the attributes independent?
- Can we identify the linear relationships?
- Can we reduce the size of the data matrix?

For instance, consider the 4×3 matrix

$$A = \begin{bmatrix} 1 & -1 & 0 \\ 2 & 0 & 2 \\ 3 & 2 & 5 \\ 4 & -3 & 1 \end{bmatrix}.$$

We have that $\mathrm{col}_1(A) + \mathrm{col}_2(A) - \mathrm{col}_3(A) = 0$ and so

$$\mathrm{col}_3(A) = \mathrm{col}_1(A) + \mathrm{col}_2(A),$$

that is, the third attribute can be written in terms of the first and the second one, which contain all the significant information. In general, the mathematical notion of *rank* encodes the number of linearly independent attributes. Linear algebra allows to compute it and to find the redundancies in a given data matrix, and thus to reduce its size.

More generally, we can ask how far is a matrix from being rank defective. Data coming from measurements is always inexact, and so we have to consider approximate linear relations between the different attributes. Methods like *principal components analysis (PCA)* and the closely related *singular value decomposition (SVD)* allow to detect these approximate linear relations and to estimate how much information is lost when reducing the matrix through them.

In this notes, we will study the three main problems of numerical linear algebra that are needed for these applications. *Linear equation solving* consist in solving

$$A\,x = b$$

for a given nonsingular $n \times n$ matrix A and n-vector b. The standard algorithms for this task is Gaussian elimination and its variants, like Cholesky's algorithm for positive symmetric matrices, to be used when the considered matrix is not too large and a solution is required with guaranteed stability and in a guaranteed amount of time. When the cost of applying such methods is too expensive, iterative methods can be applied instead.

The *least squares problem* consists in computing the n-vector x minimizing the Euclidean norm

$$\|A\,x - b\|_2$$

for a given $m \times n$ matrix A and m-vector b. In data analysis, it appears prominently when fitting data with prefixed functions depending on a set of parameters.

Finally, the *eigenproblem* amounts to find for a given $n \times n$ matrix A, a scalar λ and a nonzero n-vector x such that

$$A\,x = \lambda\,x,$$

that is, the directions in which the linear map associated to A is a homothecy. The results and techniques for solving it also apply to the computation of the SVD. These problems play a central role in the design of Internet search machines through the Pagerank algorithm, and in dimensionality reduction through the PCA method.

Our approach to all these problems will be based on *matrix factorizations*, that is, representations of the given matrix as a product of simpler ones. For instance, *Gaussian elimination with partial pivoting (GEPP)* is based on the factorization of the given $n \times n$ matrix A as

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

with P a permutation, L a unit lower triangular, and U upper triangular, like in Figure 0.0.1.

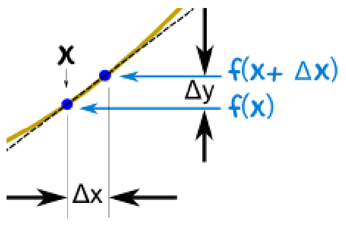
$$\underset{P}{\begin{bmatrix} & & 1 & \\ & & & 1 \\ & 1 & & \\ 1 & & & \end{bmatrix}} \underset{A}{\begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix}} = \underset{L}{\begin{bmatrix} 1 & & & \\ \frac{3}{4} & 1 & & \\ \frac{1}{2} & -\frac{2}{7} & 1 & \\ \frac{1}{4} & -\frac{3}{7} & \frac{1}{3} & 1 \end{bmatrix}} \underset{U}{\begin{bmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{6}{7} & -\frac{2}{7} \\ & & & \frac{2}{3} \end{bmatrix}}.$$

FIGURE 0.0.1. A PLU factorization

Solving the linear equation $Ax = b$ then breaks into three easier tasks, done by permuting rows and applying forward and backward substitution.

To ensure the reliability of the obtained results, we will be concerned with the *perturbation properties* of the considered problem and the *numerical stability* of our algorithms. There are two sources of numerical errors: those coming from the approximation of the input data, which is given by measurements and is subject to truncation, and those introduced by the algorithms.

Here the keyword is *condition numbers.* These are the parameters that measure the propagation of errors in a given problem, from the input to the output. For instance, let f be a real valued differentiable function. Then $f(x+\Delta x) \approx f(x) + f'(x)\,\Delta x$, and so the condition number is given by the derivative $f'(x)$.



.

Hence the first source of numerical errors is controlled by the condition number of the considered problem. If the algorithm is *backward stable*, then the second source is also controlled by this parameter.

We will also be concerned with the *efficiency* (or *speed*) of the proposed methods: before running a computation, it is useful to know how long it will take. The efficiency of an algorithm is modeled by the notion of *(time) complexity*, which is the function measuring the number of floating-point operations (flops) performed by this algorithm for a given input. For instance, GEPP solves an $n \times n$ linear system $Ax = b$ with

$$\approx \frac{2}{3} n^3 \quad \text{flops.}$$

We can then use this knowledge to decide beforehand the feasibility of a certain computation on a specific machine.

Finally, it is also important to identify and exploit any special structure that might be present, in order to both increase speed and reduce the use of storage space. For instance, when A is symmetric and positive definite, Cholesky's algorithm solves the linear equation $Ax = b$ with

$$\approx \frac{1}{3} n^3 \quad \text{flops,}$$

which is approximately half the time complexity of GEPP. If moreover A is *banded* with band width $\approx \sqrt{n}$ like in Figure 0.0.2, then Cholesky's algorithm uses only

$$\mathbf{A} = \begin{pmatrix}
a_{1,1} & a_{1,2} & a_{1,3} & & & & & & & \\
a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & & & & & & \\
a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & & & & & \\
 & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} & & & & \\
 & & a_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} & a_{5,7} & & & \\
 & & & a_{6,4} & a_{6,5} & a_{6,6} & a_{6,7} & a_{6,8} & & \\
 & & & & a_{7,5} & a_{7,6} & a_{7,7} & a_{7,8} & a_{7,9} & \\
 & & & & & a_{8,6} & a_{8,7} & a_{8,8} & a_{8,9} & a_{8,10} \\
 & & & & & & a_{9,7} & a_{9,8} & a_{9,9} & a_{9,10} \\
 & & & & & & & a_{10,8} & a_{10,9} & a_{10,10}
\end{pmatrix}$$

FIGURE 0.0.2. A banded matrix

$$O(n^2) \;\text{ flops},$$

which represents a substantial saving for large values of n.

Part 1

Linear equation solving

In this part we discuss the computational aspects of the equation

$$A\,x = b.$$

First we focus on the *direct methods* for this task, that is, algorithms that are exact in the absence of rounding. Later we will study some *iterative methods*, that produce sequences of approximations to the searched solution. As an important part of our discussion we will discuss, with more or less detail, the complexity and numerical behavior for each of the considered algorithms.

CHAPTER 1

Gaussian elimination

1.1. Preliminaries

We will deal with matrices having entries in a field $\mathbb{F}$ that can be either the field of real numbers $\mathbb{R}$ or that of complex numbers $\mathbb{C}$. The set of $m \times n$ matrices with m rows and n columns with entries in $\mathbb{F}$ is denoted by

$$\mathbb{F}^{m\times n}.$$

The (i,j)-entry of an $m \times n$ matrix A is denoted by $a_{i,j}$, so that

$$A = \begin{bmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{bmatrix} \quad \text{with } a_{i,j} \in \mathbb{F}.$$

Similarly, the i-th entry of an n-vector x is denoted by x_i, so that $x = (x_1, \dots, x_n)$. By default, these vectors will be identified with $n \times 1$ matrices:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}. \tag{1.1}$$

The *rank* of the matrix A, denoted by $\operatorname{rank}(A)$, is the maximal number of linearly independent vectors among the columns

$$\operatorname{col}_j(A) \in \mathbb{F}^m, \quad j = 1, \dots, n.$$

Equivalently, it can be defined as the dimension of the linear subspace of $\mathbb{F}^m$ generated by these columns. The matrix is said to have *full rank* if $\operatorname{rank}(A) = m$.

Multiplying A by an $n \times 1$ matrix gives an $m \times 1$ matrix, and so the identification in (1.1) allows to consider the linear map

$$L_A \colon \mathbb{F}^n \longrightarrow \mathbb{F}^m \tag{1.2}$$

defined by $L_A(x) = A\,x$. For instance, for $A = \begin{bmatrix} 2 & -1 & 3 \\ -5 & 0 & 7 \end{bmatrix}$ we have that

$$A \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2x_1 - x_2 + 3x_2 \\ -5x_1 + 7x_3 \end{bmatrix}$$

and so $L_A(x_1, x_2, x_3) = (2x_1 - x_2 + 3x_2, -5x_1 + 7x_3)$.

The *image* of L_A is the subset of vectors in $\mathbb{F}^m$ that are images of vectors in $\mathbb{F}^n$, whereas its *kernel* is the set of vectors in $\mathbb{F}^n$ where this linear map vanishes, that is

$$\operatorname{Im}(L_A) = \{L_A(x) \mid x \in \mathbb{F}^n\} \subset \mathbb{F}^m \quad \text{and} \quad \operatorname{Ker}(L_A) = \{x \in \mathbb{F}^n \mid L_A(x) = 0\} \subset \mathbb{F}^n.$$

A fundamental result in linear algebra, the *rank-nullity theorem*, says that the dimensions of these linear subspaces are complementary in the sense that

$$\dim(\operatorname{Im}(L_A)) + \dim(\operatorname{Ker}(L_A)) = n. \tag{1.3}$$

The image of L_A coincides with the *column space* of the matrix, that is, the linear span of its columns

$$\operatorname{span}(\operatorname{col}_1(A), \dots, \operatorname{col}_n(A)),$$

and so its dimension coincides with the rank of the matrix, that is $\dim(L_A) = \operatorname{rank}(A)$.

An $r \times s$ *block matrix* is an $m \times n$ matrix written as

$$A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,s} \\ \vdots & & \vdots \\ A_{r,1} & \cdots & A_{r,s} \end{bmatrix} \in \mathbb{F}^{m \times n}$$

with *blocks* $A_{i,j}$ that are $m_i \times n_j$ matrices, for given sequences of nonnegative integers m_i, n_j, $i = 1, \dots, r$, $j = 1, \dots, s$, such that

$$\sum_{i=1}^{r} m_i = m \quad \text{and} \quad \sum_{j=1}^{s} n_j = n.$$

We can also say that A is an $(m_1, \dots, m_r) \times (n_1, \dots, n_s)$ *block matrix*, specially when we want to precise the sizes of the blocks.

Block matrices with corresponding blocks of equal sizes can be added through the block structure: if

$$B = \begin{bmatrix} B_{1,1} & \cdots & B_{1,s} \\ \vdots & & \vdots \\ B_{r,1} & \cdots & B_{r,s} \end{bmatrix}$$

is also an $(m_1, \dots, m_r) \times (n_1, \dots, n_s)$ block matrix, then

$$A + B = [A_{i,j} + B_{i,j}]_{i,j} \quad \text{with } 1 \le i \le r, 1 \le j \le s.$$

Similarly, block matrices with blocks of compatible sizes can be multiplied in the natural way: for an $n \times p$ matrix written as

$$C = \begin{bmatrix} C_{1,1} & \cdots & C_{1,t} \\ \vdots & & \vdots \\ C_{s,1} & \cdots & C_{s,t} \end{bmatrix},$$

where each $C_{j,k}$ is an $n_j \times p_k$ matrix for another sequence of nonnegative integers p_k, $k = 1, \dots, t$, with $\sum_{k=1}^{t} p_k = p$, we have that

$$A\,C = \left[\sum_{j=1}^{s} A_{i,j}\, C_{j,k} \right]_{i,k} \quad \text{with } 1 \le i \le r, 1 \le k \le t,$$

which is an $r \times t$ block matrix with blocks of sizes $m_i \times p_k$, $i = 1, \dots, r$, $k = 1, \dots, t$.

Beware that not every operation or computation can be directly rewritten using blocks. For instance, the determinant of a block matrix cannot be computed with the usual formula applied to the blocks, so that if

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}$$

then in general $\det(A) \neq \det(A_{1,1})\det(A_{2,2}) - \det(A_{2,1})\det(A_{1,2})$. Nevertheless, this formula does hold when the matrix is block triangular: if $A_{2,1} = \mathbb{0}$ then

$$\det(A) = \det(A_{1,1})\det(A_{2,2}).$$

1.2. From Gaussian elimination to the PLU factorization

The first and most fundamental problem of numerical linear algebra is to solve the equation

$$A\,x = b \tag{1.4}$$

for a given $m \times n$ matrix A and m-vector b in terms of an unknown n-vector x, all of them with entries in the field $\mathbb{F}$.

In general, this equation can have either a single solution, an infinite number of them, or none at all. Indeed, it admits a solution if and only if b belongs to the image of the linear map L_A in (1.2), that is $b \in \mathrm{Im}(L_A)$. Assuming that such a solution x exists, then any other solution x' can be obtained by adding an element of the kernel of L_A, that is $x' = x + y$ with $y \in \mathrm{Ker}(L_A)$. Hence when a solution exists, it is unique if and only if this kernel is trivial, that is $\mathrm{Ker}(L_A) = \{0\}$.

Thus, the equation (1.4) has a unique solution for every $b \in \mathbb{F}^m$ if and only if

$$\mathrm{Im}(L_A) = \mathbb{F}^m \quad \text{and} \quad \mathrm{Ker}(L_A) = \{0\}$$

or equivalently, if and only if $\dim(\mathrm{Im}(L_A)) = m$ and $\dim(\mathrm{Ker}(L_A)) = 0$. By the formula in (1.3), these conditions are equivalent to the fact that $m = n$ and that A has full rank.

When $m = n$, the condition that $\mathrm{rank}(A) = n$ is equivalent to A being *nonsingular*. This means that A admits an *inverse*, that is, there is an $n \times n$ matrix A^{-1} such that $A\,A^{-1} = A^{-1}A = \mathbb{1}_n$ with

$$\mathbb{1}_n = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix},$$

the *identity* $n \times n$ matrix. Hence

$$\boxed{\text{Equation (1.4) has a unique solution for all } b \iff m = n \text{ and } A \text{ is nonsingular}}$$

This is the main case of interest, and the one that we are going to consider in this part. Of course, in this situation the solution of the equation can be written as

$$x = A^{-1}b. \tag{1.5}$$

However, this expression does not give an efficient way to obtain this solution, since computing the inverse is too expensive, as we will see later. Instead, we will consider more efficient methods for this task, both direct and iterative.

Gaussian elimination is the basis of most direct methods for solving the equation (1.4). We next recall how its standard version works in a simple example.

Example 1.2.1. Consider the system of linear equations

$$\begin{cases} x_1 + 2\,x_2 = b_1 \\ 3\,x_1 + 4\,x_2 = b_2 \end{cases}$$

Since the coefficient of the variable x_1 in the first equation is $a_{1,1} = 1$ and the corresponding coefficient in the second equation is $a_{2,1} = 3$, we have to subtract the multiple $a_{2,1}/a_{1,1} = 3$ of the first equation to the second, to eliminate x_1 in this second equation. We obtain

$$\begin{cases} x_1 + 2\,x_2 = b_1 \\ \quad\ - 2\,x_2 = -3\,b_1 + b_2 \end{cases}$$

Next we solve the modified system of linear equations by *backward substitution*: we first compute the value of x_2 using the last equation and then plug it into the first to get the value of x_1:

$$x_2 = \frac{1}{-2}(-3\,b_1 + b_2) = \frac{3}{2}\,b_1 - \frac{1}{2}\,b_2 \quad \text{and} \quad x_1 = b_1 - 2\,x_2 = -2\,b_1 + b_2.$$

In matrix notation, the system of linear equations in Example 1.2.1 writes down as

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix},$$

and the elimination procedure therein is equivalent to a left multiplication of both sides of this vector equation by a lower triangular matrix:

$$\begin{bmatrix} 1 & 0 \\ -3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -3 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \tag{1.6}$$

which gives the equivalent equation $\begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ -3\,b_1 + b_2 \end{bmatrix}$. Setting

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix},$$

we have that $L^{-1} = \begin{bmatrix} 1 & 0 \\ -3 & 1 \end{bmatrix}$ and so the above procedure implies that $L^{-1}A = U$, or equivalently the *matrix factorization*

$$A = L\,U.$$

Notice that L is *unit lower triangular* (all its diagonal entries are 1 and all its superdiagonals entries are 0) and U is *upper triangular* (all its subdiagonals entries are 0).

In Example 1.2.1, the quantities $a_{1,1}$ and $a_{1,2}/a_{1,1}$ in the elimination step are respectively called the *pivot* and the *multiplier*. In general, Gaussian elimination will perform successive elimination steps to bring the system of equations to a triangular one. At each step, we need the pivot to be nonzero to be able to define the corresponding multiplier

Hence when some pivot vanishes, we modify Gaussian elimination adding to it a *pivoting strategy*, permuting the equations so that the algorithm can continue eliminating variables, as in the next example.

Example 1.2.2. Consider the system of linear equations

$$\begin{cases} \ \ 2\,x_1 - \ \ x_2 \qquad\ = b_1 \\ \ \ 2\,x_1 - \ \ x_2 + x_3 = b_2 \\ -2\,x_1 + 3\,x_2 - x_3 = b_3 \end{cases}$$

Gaussian elimination proceeds subtracting the first equation to the second and adding the first equation to the third, thus eliminating the variable x_1 from both of them.

In the resulting system, the coefficient of x_2 in the second equation is 0, and so the algorithm permutes this second equation with the third, to obtain a system in *row echelon form*:

$$\begin{cases} 2\,x_1 - x_2 \phantom{{}-x_3} = b_1 \\ \phantom{2\,x_1 -{}} 2\,x_2 - x_3 = b_1 + b_3 \\ \phantom{2\,x_1 - 2\,x_2 -{}} x_3 = -b_1 + b_2 \end{cases}$$

As before, this system can be easily solved by backward substitution:

$$\begin{aligned} x_3 &= -b_1 + b_2, \quad x_2 = \frac{1}{2}(b_1 + b_3 + x_3) = \frac{1}{2}\,b_2 + \frac{1}{2}\,b_3, \\ x_1 &= \frac{1}{2}(b_1 + x_2) = \frac{1}{2}\,b_1 + \frac{1}{4}\,b_2 + \frac{1}{4}\,b_3. \end{aligned} \tag{1.7}$$

Consider the 3×3 matrices arising in Example 1.2.2

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

corresponding respectively to the given system of linear equations, the elimination step, the permutation of the second and the third rows, and the obtained system in row echelon form. The procedure therein can be condensed as the equality $P\,L\,A = U$. We can verify by a direct computation that $(P\,L)^{-1} = P\,L$, and so this equality can be equivalently written as

$$A = \begin{bmatrix} 2 & -1 & 0 \\ 2 & -1 & 1 \\ -2 & 3 & -1 \end{bmatrix} = P\,L\,U = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & -1 & 0 \\ 0 & 2 & -1 \\ 0 & 0 & 1 \end{bmatrix}. \tag{1.8}$$

The good news are that this matrix factorization can be extended to the general case: for any given nonsingular $n \times n$ matrix A, the Gaussian elimination algorithm can be used to factor it as

$$A = P\,L\,U \tag{1.9}$$

with P a permutation, L a unit lower triangular, and U an upper triangular matrix. With this factorization, the equation $A\,x = b$ gets reduced to similar equations for each of these factors that, thanks to their special structure, are easy to solve. Precisely, we perform the following the steps:

(1) solve $P\,z = b$ permuting the entries of b,
(2) solve $L\,y = z$ by forward substitution,
(3) solve $U\,x = y$ by backward substitution.

Then $A\,x = P\,(L\,(U\,x)) = P\,(L\,y) = P\,z = b$, and so x is the searched solution.

Example 1.2.3. Let us solve the system of linear equations in Example 1.2.2 using the PLU factorization in (1.8). We first solve $P\,z = b$, which gives

$$z_1 = b_1, \quad z_2 = b_3, \quad z_3 = b_2.$$

Next we solve $L\,y = z$ to obtain

$$y_1 = z_1 = b_1, \quad y_2 = z_2 + y_1 = b_1 + b_3, \quad y_3 = z_3 - y_1 = -b_1 + b_2.$$

Finally, the solution is computed by solving $U\,x = y$:

$$x_3 = y_3 = -b_1 + b_2, \quad x_2 = \frac{1}{2}(y_2 + x_3) = \frac{1}{2}\,b_2 + \frac{1}{2}\,b_3,$$
$$x_1 = \frac{1}{2}(y_1 + x_2) = \frac{1}{2}\,b_1 + \frac{1}{4}\,b_2 + \frac{1}{4}\,b_3,$$

in agreement with (1.7).

1.3. The GEPP algorithm

In §1.2, we discussed examples where the standard version of the Gaussian elimination algorithm applied to a given system of linear equations, produces a PLU factorization of the associated matrix as in (1.9). This is a general phenomenon and indeed, this algorithm can be modified to produce such a factorization for any given nonsingular square matrix.

Instead of going this way, we will devise a dedicated algorithm to compute such a factorization. This algorithm will make this computation in a more direct way although of course, the underlying ideas will be same as those in the standard version. More importantly, it will give us a more advanced point of view on Gaussian elimination: we will shift from an algorithm that performs a specific task (solving a linear equation) to the more general framework of matrix factorizations, which is the technique of choice in numerical linear algebra.

Let A be a nonsingular $n \times n$ matrix with entries in $\mathbb{F}$. A *PLU factorization* for A is its expression as a product of $n \times n$ matrices with entries in $\mathbb{F}$

$$A = P\,L\,U \tag{1.10}$$

with P a permutation, L unit lower triangular, and U upper triangular.

Before explaining how we can compute this factorization, we introduce and discuss the basic properties of permutations and their associated matrices.

An *n-permutation* is a bijection $\sigma\colon \{1, \dots, n\} \to \{1, \dots, n\}$. It is usually written verbose as an explicit sequence of preimage/image pairs:

$$\sigma = \begin{pmatrix} 1 & \cdots & n \\ \sigma(1) & \cdots & \sigma(n) \end{pmatrix}.$$

An *n-transposition* is an n-permutation σ whose only effect on the set $\{1, \dots, n\}$ is the swapping two different elements a and b. It is usually written using cycle notation as

$$\sigma = (a, b).$$

The set of all n-permutations, denoted by $\mathcal{S}_n$, forms a group with respect to the composition of functions. Every n-permutation can be expressed as the composition of a finite number of transpositions.

To each $\sigma \in \mathcal{S}_n$ we associate the *permutation $n \times n$-matrix* constructed permuting the columns of the identity matrix according to σ, that is

$$P_\sigma = \begin{bmatrix} e_{\sigma(1)} & \cdots & e_{\sigma(n)} \end{bmatrix}$$

where the e_i's are the vectors in the *canonical basis* of $\mathbb{F}^n$, that is

$$e_i = (0, \dots, 0, \overset{i}{1}, 0, \dots, 0),$$

plugged into P_σ as columns following our convention in (1.1). For instance, to the permutation $\begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} \in \mathcal{S}_3$ corresponds the permutation matrix

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \in \mathbb{F}^3.$$

Pre-multiplication of A by a permutation $n \times n$ matrix produces a permutation of its rows, whereas its post-multiplication produces a permutation of its columns. Precisely

$$P_\sigma^T A = \begin{bmatrix} \mathrm{row}_{\sigma(1)}(A) \\ \vdots \\ \mathrm{row}_{\sigma(n)}(A) \end{bmatrix} \quad \text{and} \quad A\, P_\sigma = \begin{bmatrix} \mathrm{col}_{\sigma(1)}(A) & \cdots & \mathrm{col}_{\sigma(n)}(A) \end{bmatrix}. \tag{1.11}$$

Finally, we note that the correspondence between n-permutations and permutation matrices is compatible with the group structure of $\mathcal{S}_n$: for $\sigma' \in \mathcal{S}_n$ we have that

$$P_\sigma\, P_{\sigma'} = P_{\sigma \circ \sigma'} \quad \text{and} \quad P_\sigma^{-1} = P_{\sigma^{-1}} = P_\sigma^T \tag{1.12}$$

with P_σ^T the transpose of P_σ.

To compute the searched factorization of A, we will apply the following iterative procedure of pivoting and elimination by columns.

In the first step, choose $i_1 \in \{1, \dots, n\}$ such that $a_{i_1,1} \neq 0$, let σ_1 be the permutation of the set $\{1, \dots, n\}$ that swaps 1 and i_1, and consider the associated permutation matrix P_{σ_1}. Relabel the entries of A so that $P_{\sigma_1}^T A = \begin{bmatrix} a_{i,j} \end{bmatrix}_{i,j}$ and write it as the $(1, n-1) \times (1, n-1)$ block matrix

$$P_{\sigma_1}^T A = \begin{array}{cc} \begin{array}{cc} \scriptstyle 1 & \scriptstyle n-1 \end{array} & \\ \begin{bmatrix} a_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} & \begin{array}{l} \scriptstyle 1 \\ \scriptstyle n-1 \end{array} \end{array}. \tag{1.13}$$

Here $a_{1,1} \neq 0$ is the $(1,1)$ entry of $P_{\sigma_1}^T A$, whereas $A_{1,2}$ and $A_{2,1}$ denote respectively the rest of its first row and column, whereas $A_{2,2}$ is the remaining $(n-1) \times (n-1)$ block. Setting

$$u_{1,1} = a_{1,1}, \quad L_{2,1} = a_{1,1}^{-1} A_{2,1}, \quad U_{1,2} = A_{1,2}, \quad A^{(1)} = A_{2,2} - L_{2,1}\, U_{1,2} \tag{1.14}$$

we obtain the block LU factorization

$$P_{\sigma_1}^T A = \begin{bmatrix} a_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} = \begin{bmatrix} 1 & \mathbb{0} \\ L_{2,1} & \mathbb{1}_{n-1} \end{bmatrix} \begin{bmatrix} u_{1,1} & U_{1,2} \\ \mathbb{0} & A^{(1)} \end{bmatrix}. \tag{1.15}$$

The $(n-1) \times (n-1)$ matrix $A^{(1)}$ is nonsingular, and is called the *Schur complement* of $P_{\sigma_1}^T A$.

In the second step, we proceed similarly with this latter matrix: we choose $i_2 \in \{2, \dots, n\}$ such that $a_{i_2,2}^{(1)} \neq 0$ and we denote by σ_2 the permutation of the set $\{2, \dots, n\}$ swapping 2 and i_2. Then we relabel the entries of the permuted matrix $P_{\sigma_2}^T A^{(1)}$, write it as a $(1, n-2) \times (1, n-2)$ block matrix as in (1.13) and, applying the formulae in (1.14), compute its block LU factorization

$$P_{\sigma_2}^T A^{(1)} = L_2\, U_2 \quad \text{with} \quad L_2 = \begin{bmatrix} 1 & \mathbb{0} \\ L_{3,2} & \mathbb{1}_{n-2} \end{bmatrix} \quad \text{and} \quad U_2 = \begin{bmatrix} u_{2,2} & U_{2,3} \\ \mathbb{0} & A^{(2)} \end{bmatrix}. \tag{1.16}$$

It follows from (1.15) and (1.16) together with some algebraic manipulations that

$$\begin{aligned}\begin{bmatrix}1 & \mathbb{0}\\ \mathbb{0} & P_{\sigma_2}^T\end{bmatrix} P_{\sigma_1}^T A &= \begin{bmatrix}1 & \mathbb{0}\\ \mathbb{0} & P_{\sigma_2}^T\end{bmatrix}\begin{bmatrix}1 & \mathbb{0}\\ L_{2,1} & \mathbb{1}_{n-1}\end{bmatrix}\begin{bmatrix}u_{1,1} & U_{1,2}\\ \mathbb{0} & A^{(1)}\end{bmatrix}\\ &= \begin{bmatrix}1 & \mathbb{0}\\ P_{\sigma_2}^T L_{2,1} & \mathbb{1}_{n-1}\end{bmatrix}\begin{bmatrix}u_{1,1} & U_{1,2}\\ \mathbb{0} & P_{\sigma_2}^T A^{(1)}\end{bmatrix}\\ &= \begin{bmatrix}1 & \mathbb{0}\\ P_{\sigma_2}^T L_{2,1} & \mathbb{1}_{n-1}\end{bmatrix}\begin{bmatrix}u_{1,1} & U_{1,2}\\ \mathbb{0} & L_2\, U_2\end{bmatrix}\\ &= \begin{bmatrix}1 & \mathbb{0}\\ P_{\sigma_2}^T L_{2,1} & L_2\end{bmatrix}\begin{bmatrix}u_{1,1} & U_{1,2}\\ \mathbb{0} & U_2\end{bmatrix}.\end{aligned} \tag{1.17}$$

In the subsequent steps, we choose for each $j = 3, \dots, n-1$ an index i_j such that $a_{i_j,j}^{(j-1)} \neq 0$ and denote by σ_j the permutation of the set $\{j, \dots, n\}$ swapping j with i_j. Then we relabel the entries of the permuted Schur complement $P_{\sigma_j}^T A^{(j-1)}$ and compute the corresponding $(1, n-j) \times (1, n-j)$ block LU factorization

$$P_{\sigma_j}^T A^{(j-1)} = L_j\, U_j$$

applying the formulae in (1.14), which produces the next Schur complement $A^{(j)}$ as the $(n-j) \times (n-j)$ block of U_j.

Finally, the permutation matrix in (1.10) is obtained considering for each j the permutation $n \times n$ matrix $P_j = \begin{bmatrix}\mathbb{1}_{j-1} & \mathbb{0}\\ \mathbb{0} & P_{\sigma_j}\end{bmatrix}$ and setting

$$P = \prod_{j=1}^{n-1} P_j. \tag{1.18}$$

For each j we have that P_j coincides with the permutation matrix corresponding to the transposition (j, i_j). By the compatibility formulae in (1.12), we deduce that $P = P_\sigma$ for the n-permutation

$$\sigma = (1, i_1) \circ \cdots \circ (n-1, i_{n-1}) \in \mathcal{S}_n.$$

On the other hand, the upper triangular matrix U in (1.10) is given by the first row of the U_j's, and the unit lower triangular matrix L is similarly obtained from the first column of the L_j's permuted according to the σ_k's for $k > j$ as in (1.17).

There is one further point, that is crucial for the numerical stability of the algorithm: the pivots have to be chosen wisely, because dividing by numbers that are too small can have disastrous consequences for the reliability of the computed solution. In *Gaussian elimination with partial pivoting (GEPP)*, for each step j this is done choosing the index i_j so that $|a_{i_j,j}|$ is maximal among $|a_{p,j}|$, $j \leq p \leq n$. By (1.14), this implies that all the entries of L will have absolute value bounded by 1.

We next describe the obtained algorithm in *pseudocode notation*. Therein we denote by P, L and U three $n \times n$ matrices, all of them initialized to $\mathbb{0}$, that will eventually become the searched factors. Also, for $j = 0, \dots, n-1$ we denote by

$$A^{(j)} = \left[a_{i,k}^{(j)}\right]_{j+1 \leq i,k \leq n}$$

an $(n-j) \times (n-j)$ matrix, the first one initialized to A and the others to $\mathbb{0}$, that will become the successive Schur complements.

Algorithm 1.3.1 (GEPP)

```
for $j = 1, \dots, n-1$
    choose $i_j \in \{j, \dots, n\}$ such that $|a_{i_j,j}^{(j-1)}| = \max_{j \le p \le n} |a_{p,j}^{(j-1)}|$
    swap row $j$ and row $i_j$ of $L$ and of $A^{(j-1)}$
    for $k = j+1, \dots, n$
        $u_{j,k} \leftarrow a_{j,k}^{(j-1)}$        (compute the $j$-th row of $U$)
    for $i = j+1, \dots, n$
        $l_{i,j} \leftarrow a_{i,j}^{(j-1)} / a_{j,j}^{(j-1)}$        (compute the $j$-th column of $L$)
    for $i, k = i+1, \dots, n$
        $a_{i,k}^{(j)} \leftarrow a_{i,k}^{(j-1)} - l_{i,j}\, u_{j,k}$        (compute the $j$-th Schur complement)
$u_{n,n} \leftarrow a_{n,n}^{(n-1)}$        (compute the $(n,n)$ entry of $U$)
$P \leftarrow P_\sigma$ for the $n$-permutation $\sigma = (1, i_1) \circ \cdots \circ (n-1, i_{n-1})$
```

Taking a closer look at this algorithm, we see that at the j-th step each entry $a_{j,k}^{(j-1)}$ is assigned to the corresponding entry of U and is never used again. Moreover, each entry $a_{i,j}^{(j-1)}$ is used to compute the corresponding entry of L and is neither used again, and the same happens with the entries that compute the j-th Schur complement. Hence at that step we can safely rewrite A with the nontrivial entries in the j-th column of L and the j-th row of U, and with those of the j-th Schur complement. Hence we need no extra space to store the computations done at this step.

With this strategy, at the end of the algorithm the matrix A would contain all the nontrivial entries of L and of U distributed as in Figure 1.3.1.

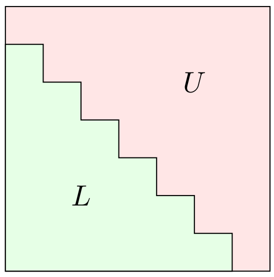


FIGURE 1.3.1. Storage distribution for GEPP

Moreover, the permutation matrix P can be coded by the corresponding element of $\mathcal{S}_n$ or even as the sequence of transpositions defining it, leading to a more efficient implementation (Algorithm 1.3.2).

As an illustration, we apply it to recompute the PLU factorization in Example 1.2.3.

EXAMPLE 1.3.1. Set

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

For $j = 1$ we choose $i_1 = 1$, and so the rows of the matrix need not be permuted at this step. We compute the first column of L and the entries of the first Schur complement, and overwrite the matrix A with the obtained values: the used operations and updated

Algorithm 1.3.2 (GEPP with overwritting)

```
for $j = 1, \dots, n-1$
    choose $i_j \in \{j, \dots, n\}$ such that $|a_{i_j,j}| = \max_{j \le p \le n} |a_{p,j}|$
    swap row $j$ and row $i_j$ of $A$
    for $i = j+1, \dots, n$
        $a_{i.j} \leftarrow a_{i,j}/a_{j,j}$
    for $i, k = j+1, \dots, n$
        $a_{i,k} \leftarrow a_{i,k} - a_{i,j}\, a_{j,k}$
$\sigma \leftarrow (1, i_1) \circ \dots \circ (n-1, i_{n-1})$
```

matrix are

$$A = \begin{bmatrix} 2 & -1 & 0 \\ 2/2 & -1 - 1 \cdot (-1) & 1 - 1 \cdot 0 \\ -2/2 & 3 - (-1) \cdot (-1) & -1 - (-1) \cdot 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 \\ 1 & 0 & 1 \\ -1 & 2 & -1 \end{bmatrix}.$$

For $j = 2$ we choose $i_2 = 3$, and then swap the second and the third rows of A. We compute the second column of L and the entries of the second Schur complement, and overwrite the matrix A with the obtained values to get

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 1 & 0/2 & 1 - 0 \cdot (-1) \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 1 & 0 & 1 \end{bmatrix}.$$

This matrix together with the chosen indexes $i_1 = 2$ and $i_2 = 3$ encode the PLU factorization of A:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 2 & -1 & 0 \\ 0 & 2 & -1 \\ 0 & 0 & 1 \end{bmatrix}.$$

Another variant of this algorithm is *Gaussian elimination with complete pivoting (GECP)*, that at the j-th step reorders the rows and all the columns of the $(j-1)$th Schur complement, so to choose a pivot whose absolute value is maximal among all the entries of this matrix. It yields a factorization of the form

$$A = P_1 \, L \, U P_2$$

where both P_1 and P_2 are permutation matrices. This variant is slower than GEPP, but can be also more numerically stable.

As discussed in §1.2, the factorization $A = PLU$ allows to solve the linear equation $A\,x = b$ by reducing it to the three analogous but simpler equations

$$Pz = b, \quad Ly = z, \quad Ux = y.$$

The equation $Pz = b$ can be solved permuting the entries of the n-vector b according to the permutation $\sigma = (1, i_1) \circ \dots \circ (n-1, i_{n-1})$, which can be done by successively swapping b_j and b_{i_j}, $j = 1, \dots, n-1$. On the other hand, $Ly = z$ and $Ux = y$ can be solved by applying Algorithms 1.3.3 and 1.3.4, respectively.

Algorithm 1.3.3 (Forward substitution)

```
for $i = 1, \dots, n$
    $y_i \leftarrow z_i - \sum_{j=1}^{i-1} l_{i,j} z_j$
```

Algorithm 1.3.4 (Backward substitution)

```
for $i = n, \dots, 1$
    $x_i \leftarrow \frac{1}{u_{i,i}}\Big(x_i - \sum_{j=i+1}^{n} u_{i,j} y_j\Big)$
```

CHAPTER 2

Complexity and numerical stability

In this chapter we discuss the efficiency of the algorithms for linear equation solving presented so far. Our focus will be on Gaussian elimination with partial pivoting (GEPP), but these discussions can also be applied, with suitable adaptations, to most other (if not all) algorithms in numerical linear algebra.

The main aspects of the efficiency of these algorithms come under two labels: complexity and numerical stability. Roughly speaking, the complexity of an algorithm measures how much space in memory and how many arithmetic operations it needs to perform a given computation. On the other hand, numerical stability concerns the propagation of numerical errors from the input to the output. In a nutshell, it is all about the *feasibility* of a given computation and the *reliability* of the computed solution.

2.1. Floating-point arithmetic

Floating-point number systems are the playground of numerical analysis and, in particular, of numerical linear algebra. All the problems we are going to consider in this book deal only with real numbers that can be represented and treated using such systems. Hence it will be important for us to understand how they are defined and how they work. The models of floating-point arithmetic that are actually implemented in machines are quite complicated in their details, but here we are going to present a simple version that is sufficient for our purposes.

Given integers $\beta \geq 2$, $p \geq 1$ and $\ell \leq u$, we consider the *floating-point number system*

$$F(\beta, p, \ell, u)$$

whose elements are 0 and the rational numbers of the form

$$f = \pm 0.d_1 d_2 \dots d_p \times \beta^e \tag{2.1}$$

with $0 < d_1 < \beta$, $0 \leq d_i < \beta$ for $i = 2, \dots, p$, and $\ell \leq e \leq u$. For this *floating-point number* f, we say that the d_i's are the *digits*, the expression $0.d_1 d_2 \dots d_p$ is the *mantissa*, and the integer e is the *exponent*.

The parameter β is the *base* the floating-point number system, and determines the possible digits: typically β is taken as 2 in machines and as 10 in humans. The parameter p gives the length of the mantissa, and so determines the precision of the system. On the other hand, ℓ is the *underflow* and u the *overflow*, and determine the range of the exponent and thus the smallest and largest representable numbers.

Precisely, the smallest and the largest positive elements of $F(\beta, p, \ell, u)$, respectively called the *underflow threshold* and the *overflow threshold*, are

$$m = 0.10 \dots 0 \times \beta^{\ell} = \beta^{\ell - 1} \quad \text{and} \quad M = 0.dd \dots d \times \beta^u = (1 - \beta^{-p})\, \beta^u \tag{2.2}$$

with $d = \beta - 1$. For every $f \in F(\beta, p, \ell, u)$ we have that $m \leq |f| \leq M$.

EXAMPLE 2.1.1. Consider the particular case when $\beta = 2$, $p = 3$, $\ell = -1$ and $u = 1$. The possible mantissas and exponents are

$$q = 0.100, 0.101, 0.110, 0.111 \quad \text{and} \quad e = -1, 0, 1,$$

and so the nonzero elements of $F(2, 3, -1, 1)$ are the rational numbers 1/2, 5/8, 3/4 and 7/8, multiplied by either $\pm 1/2$, ± 1 or ± 2. Figure 2.1.1 gives the graphical representation of these floating-point numbers.

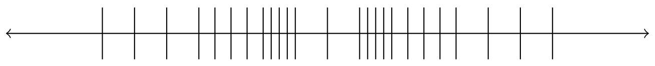


FIGURE 2.1.1. A simple floating-point number system

As we can see in the previous example, the floating-point line is plenty of holes, and so we need a rounding function to represent real numbers and operate within it. Our *rounding function* is the function

$$\text{fl} \colon \mathbb{R} \longrightarrow F(\beta, p, \ell, u) \cup \{\pm\infty\} \tag{2.3}$$

defined for $\lambda \in \mathbb{R}$ as 0 if $|\lambda| < m$, as $+\infty$ if $\lambda > M$, as $-\infty$ if $\lambda < -M$, and otherwise as the floating-point number that is closest to λ, choosing the largest one in case there are two of them.

The floating-point number $\text{fl}(\lambda)$ is an approximation of the real number λ, and the quantities

$$|\lambda - \text{fl}(\lambda)| \quad \text{and} \quad \frac{|\lambda - \text{fl}(\lambda)|}{|\lambda|} \text{ if } \lambda \neq 0$$

are called the *absolute error* and the *relative error* of this approximation, respectively. The second reflects the quality of the approximation relative to the size of the number, and will be more relevant in our considerations.

As we can also in Figure 2.1.1, the floating-point line has a lot of autosimilarities, due to its construction as a set of mantissas multiplied by a set of powers of the base. This feature allows a satisfactory control of the relative error of the produced approximations. To this end, consider the *machine epsilon* (or *macheps*) ε of $F(\beta, p, \ell, u)$, defined as the smallest real number that added up to 1, has a rounding that is larger than 1. This quantity can be computed as

$$\varepsilon = \frac{\beta^{1-p}}{2}, \tag{2.4}$$

and it gives an essentially optimal upper bound for these relative errors: for $\lambda \in \mathbb{R}$ within the *machine capacity*, that is, whose absolute value lies within the underflow and overflow threshold, we have that

$$\frac{|\lambda - \text{fl}(\lambda)|}{|\lambda|} < \varepsilon, \tag{2.5}$$

the worst relative error occurring when λ approaches from below the number $1 + \varepsilon$. Hence our rounding function verifies that

$$\text{fl}(\lambda) = \lambda + \delta\lambda \quad \text{with } \frac{|\delta\lambda|}{|\lambda|} \leq \varepsilon. \tag{2.6}$$

The floating-point versions of the arithmetic operations $+, -, \times, /$ are defined rounding their actual result. For instance, in our toy floating-point number system (Example 2.1.1) we have that

$$0.100\times 2^1 \oplus 0.101\times 2^{-1} = \mathrm{fl}(0.100\times 2^1 + 0.101\times 2^{-1}) = \mathrm{fl}(0.10101\times 2^1) = 0.101\times 2^1.$$

Currently, the IEEE standards for floating-point arithmetic are the most common in actual implementations. They include two systems: single precision and double precision.

In the *IEEE single precision standard*, each floating-point number is coded as a string of 32 bits (or 4 bytes) distributed as shown in Figure 2.1.2.

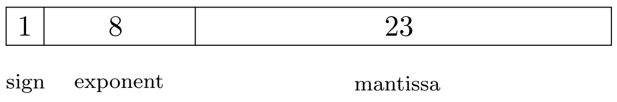


FIGURE 2.1.2. IEEE single precision encoding

As indicated therein, the sign is coded by 1 bit, the exponent by 8 bits, and the mantissa by the remaining 23 bits. Denoting these bits by b_i, $i = 1, \dots, 32$, and using the base 2 notation, the corresponding *coded number* is

$$f = (-1)^{b_1}\, 0.1b_{10}b_{11}\dots b_{32} \times 2^{b_2\dots b_9 - 126}.$$

Note that for the base 2 the first digit in the mantissa will always be 1, and so we do not need to code it. On the other hand, the cases when the digits coding the exponent are all 0's or all 1's are excluded, since they are reserved for special numbers. Hence the floating-point numbers in this standard coincide with those in $F(\beta, p, \ell, u)$ for

$$\beta = 2, \quad p = 24, \quad \ell = -125, \quad u = 128.$$

The underflow and overflow thresholds of this system are

$$m = 2^{-126} \approx 1.17\times 10^{-38} \quad \text{and} \quad M = (1 - 2^{-24})\, 2^{128} \approx 3.40\times 10^{38},$$

whereas its machine epsilon is $\varepsilon = 2^{-24} \approx 5.96\times 10^{-8}$.

In the *IEEE double precision standard*, each floating-point number is coded as a string of 64 bits (or 8 bytes), distributed as shown in Figure 2.1.3.

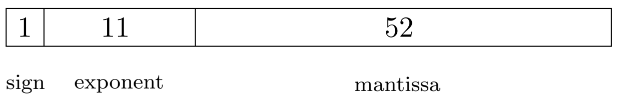


FIGURE 2.1.3. IEEE double precision encoding

The sign is coded by 1 bit, the exponent by 11 bits, and the mantissa by the remaining 52 bits. Denoting these bits by b_i, $i = 1, \dots, 64$, and using again the base 2, the coded number is

$$f = (-1)^{b_1}\, 0.1b_{13}b_{14}\dots b_{64} \times 2^{b_2\dots b_{12} - 1022}.$$

The cases when the digits coding the exponent are all 0's or all 1's are reserved for special numbers. Hence these floating-point numbers coincide with those in $F(\beta, p, \ell, u)$ for

$$\beta = 2, \quad p = 53, \quad \ell = -1021, \quad u = 1024.$$

The underflow and overflow thresholds of this system are

$$m = 2^{-1022} \approx 2.22 \times 10^{-308} \quad \text{and} \quad M = (1 - 2^{-53}) \times 2^{1024} \approx 1.79 \times 10^{308},$$

whereas its machine epsilon is $\varepsilon = 2^{-53} \approx 1.11 \times 10^{-16}$.

2.2. The complexity of GEPP

All floating-point numbers occupy the same space in the machine memory, and so the cost of a computation depends on the following factors:

(1) the number of flops $\oplus$, $\ominus$, $\otimes$, $\oslash$ and of tests $x < y$,
(2) reading and writing in memory,
(3) use of memory space.

In numerical analysis, we mostly consider (1) and (3). In practical situation, (2) is also relevant and an efficient implementation should take this issue into account seriously. Block algorithm could help in optimizing the memory resources.

The use of memory space of an algorithm measured in terms of its *space complexity*, defined as the number of floating-point numbers simultaneously needed to perform computations. For instance, GEPP with storage management (Algorithm 1.3.2) is optimal in this respect, since it runs with n^2 floating-point numbers, that is

$$\epsilon_{\text{GEPP}}(n) = n^2,$$

which is also the size of the input matrix A.

The *time complexity* of an algorithm is defined as the number of flops used through computations. This parameter can be computed from the pseudocode of the algorithm. To bound the expressions that appear, we recall from basic calculus the asymptotic formulae for power sums: for $k \geq 0$ we have that

$$\sum_{i=1}^{n} i^k = \int_0^n x^k \, dx + O(n^k) = \frac{n^{k+1}}{k+1} + O(n^k),$$

where $O(n^k)$ denotes the "big O" notation, indicating that there is a constant $c \geq 0$ such that

$$\left| \sum_{i=1}^{n} i^k - \frac{n^{k+1}}{k+1} \right| \leq c\, n^k.$$

The time complexity of the GEPP algorithm acting on $n \times n$ matrices can be computed and bounded as

$$\tau_{\text{GEPP}}(n) = \sum_{i=1}^{n-1} \left(\sum_{j=i+1}^{n} 1 + \sum_{j=i+1}^{n} \sum_{k=i+1}^{n} 2 \right) = \sum_{i=1}^{n-1} ((n-i) + 2\,(n-i)^2) = \frac{2}{3}\, n^3 + O(n^2).$$

On the other hand, the algorithms for forward and backward substitution indicated in §1.2 have each a time complexity of $n^2 + O(n)$ flops. Hence GEPP solves the equation $A\,x = b$ with an overall time complexity of

$$\frac{2}{3}\, n^3 + O(n^2) \text{ flops.}$$

As said in §1.2, inverting the matrix A is not a good strategy for solving the linear equation $A\,x = b$. Such a computation would take $2\,n^3 + O(n^2)$ flops, which is approximately three times the cost of solving this linear equation using Gauss elimination algorithm.

2.3. Vector and matrix norms

Norms are used to measure numerical errors in matrix computations. These errors may appear even when manipulating exact input data, because the operations are performed in floating-point arithmetic as explained in §2.1.

A *vector norm* is a function

$$\|\cdot\| \colon \mathbb{F}^n \longrightarrow \mathbb{R}$$

such that for all $x, y \in \mathbb{F}^n$ and $\alpha \in \mathbb{F}$ we have that

(1) $\|x\| \geq 0$ *(positivity)*,
(2) $\|x\| = 0$ if and only if $x = 0$ *(definiteness)*,
(3) $\|\alpha\, x\| = |\alpha|\, \|x\|$ *(homogeneity)*,
(4) $\|x + y\| \leq \|x\| + \|y\|$ *(triangle inequality)*.

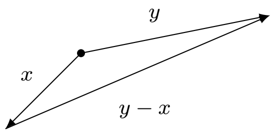


FIGURE 2.3.1. The triangle inequality

EXAMPLE 2.3.1. For $1 \leq p \leq +\infty$, the *p-norm* is the vector norm on $\mathbb{F}^n$ defined as

$$\|x\|_p = \begin{cases} \left(\displaystyle\sum_{i=1}^{n} |x_i|^p \right)^{1/p} & \text{if } 1 \leq p < +\infty, \\ \max_i |x_i| & \text{if } p = +\infty. \end{cases}$$

Any two vector norms $\|\cdot\|_1$ and $\|\cdot\|_2$ can be compared: there are constants $c_1, c_2 > 0$ such that

$$\|x\|_1 \leq c_1\, \|x\|_2 \quad \text{and} \quad \|x\|_2 \leq c_2\, \|x\|_1 \quad \text{for all } x \in \mathbb{F}^n.$$

For instance, for all $x \in \mathbb{F}^n$ we have that

$$\|x\|_2 \leq \|x\|_1 \leq n^{1/2} \|x\|_2 \quad \text{and} \quad \|x\|_\infty \leq \|x\|_1 \leq n\, \|x\|_\infty.$$

A *matrix norm* is simply a vector norm on the space of $m \times n$ matrices $\mathbb{F}^{m\times n}$. Matrix norms on the spaces of $m \times n$, $n \times p$ and $m \times p$ matrices are said to be *compatible* is they are submultiplicative, that is, if for all $A \in \mathbb{F}^{m\times n}$ and $B \in \mathbb{F}^{n\times p}$ we have that

$$\|A\, B\| \leq \|A\|\, \|B\|. \tag{2.7}$$

EXAMPLE 2.3.2. The *Frobenius norm* of an $m \times n$ matrix A is defined as its Euclidean norm when considered as a vector of $\mathbb{F}^{n\times n}$, that is,

$$\|A\|_{\mathrm{F}} = \left(\sum_{i,j} |a_{i,j}|^2 \right)^{1/2}.$$

Frobenius norms on $m \times n$, $n \times p$ and $m \times p$ matrices are compatible in the sense of (2.7).

Given norms on both $\mathbb{F}^m$ and on $\mathbb{F}^n$, the associated *operator norm* is the matrix norm defined as

$$\|A\| = \sup_{x\neq 0} \frac{\|A\,x\|}{\|x\|}.$$

If we consider a further norm on $\mathbb{F}^p$, the associated operator norms on $m\times n$, $n\times p$ and $m\times p$ matrices are also compatible in the sense of (2.7).

We list a number of basic properties of matrix norms. Let A be an $m\times n$ matrix with entries in $\mathbb{F}$. We denote by A^* the $n\times m$ matrix obtained as the transpose of the complex conjugate of A: if A has complex entries,

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix} \quad \text{then} \quad A^* = \begin{bmatrix} \overline{a_{1,1}} & \cdots & \overline{a_{1,m}} \\ \vdots & & \vdots \\ \overline{a_{n,1}} & \cdots & \overline{a_{n,m}} \end{bmatrix}.$$

On the other hand, if A is real, then $A^* = A^T$ is the transpose of A.

For the most basic vector norms, we have the next explicit formuale for their associated operator norms:

(1) $\|A\|_\infty = \max_i \sum_j |a_{i,j}|$ *(maximum row sum)*,

(2) $\|A\|_1 = \max_j \sum_i |a_{i,j}|$ *(maximum column sum)*,

(3) $\|A\|_2 = \rho(A^*A)^{1/2}$, where $\rho(A^*A)$ is the spectral radius of the matrix A^*A, that is, the maximal absolute value of its eigenvalues.

An $n\times n$ matrix Q with real entries is *orthogonal* if

$$Q^TQ = Q\,Q^T = \mathbb{1}_n.$$

More generally, if Q has complex entries then it is *unitary* if

$$Q^*Q = Q\,Q^* = \mathbb{1}_n.$$

If Q' is another $m\times m$ matrix that is orthogonal ($\mathbb{F}=\mathbb{R}$) or unitary ($\mathbb{F}=\mathbb{C}$) then

$$\|Q'\,A\,Q\|_{\mathrm{F}} = \|A\|_{\mathrm{F}} \quad \text{and} \quad \|Q'\,A\,Q\|_2 = \|A\|_2.$$

In particular $\|Q\|_{\mathrm{F}} = n^{1/2}$, $\|Q'\|_{\mathrm{F}} = m^{1/2}$ and $\|Q\|_2 = \|Q'\|_2 = 1$ The proofs of these properties can be found in [**Dem97**, Lemma 1.7].

2.4. Perturbation theory in linear equation solving

In general terms, given an input, a numerical algorithm will provide an output which may approximate the actual solution to the problem at hand, or not. Hence it is important to have criteria to decide if those outputs are accurate enough for our purposes, or not at all.

In this section we study the *forward stability* of the linear equation $A\,x = b$, that is, how errors in the input data (A, b) propagate to the solution x. Such errors are typically produced by approximate measurements and prior computations, and by rounding because of the floating-point representation. They are unavoidable, and so it is important to understand how they affect the quality of our computations.

Some matrices behave really bad with respect to perturbations.

Example 2.4.1. Set

$$A = \begin{bmatrix} 3.55 & 1.13 \\ 2.2 & 0.7 \end{bmatrix} \quad \text{and} \quad b = (3.55, 2.2).$$

The solution to the equation $A\,x = b$ is the vector $x = (1, 0)$. Taking the perturbation $\widehat{b} = b + \delta b$ with

$$\delta b = 0.00017,$$

the solution to the equation $A\,x = \widehat{b}$ is $\widehat{x} = (1.9775, -0.3111)$, far away from the solution to the original problem.

Let $(\widehat{A}, \widehat{b})$ be the perturbed data and $\widehat{x}$ the corresponding solution, so that

$$\widehat{A}\,\widehat{x} = \widehat{b}.$$

To compare the *errors*

$$\delta A = A - \widehat{A}, \quad \delta b = b - \widehat{b}, \quad \delta x = x - \widehat{x}$$

we consider a norm $\|\cdot\|$ on $\mathbb{F}^n$ and its associated operator norm on $\mathbb{F}^{n\times n}$.

All these matrices and vectors are coded by floating-point numbers, and a standard rule of thumb says that their norm gives the most significant exponent of these floating-point numbers, whereas the relative error gives the precision of the approximation. We can express this in a slightly more concrete way as the fact that $\log_\beta \|x\|$ and $\log_\beta \|\widehat{x}\|$ approximate the largest exponent of x and of $\widehat{x}$ respectively, and that

$$-\log_\beta \left(\frac{\|\delta x\|}{\|x\|}\right) \approx \text{ number of correct digits of } \widehat{x}, \tag{2.8}$$

and similarly for A and b. These are certainly not mathematical statements, but nevertheless can be followed in practice as guiding principles.

Since we are interested in the precision of our computations, we translate this interest to the study of the propagation of relative errors, that is, how we can control the ratio

$$\frac{\|\delta x\|}{\|x\|} \tag{2.9}$$

in terms of $\|\delta A\|/\|A\|$ and $\|\delta b\|/\|b\|$.

The behavior of the solution the equation $A\,x = b$ with respect to perturbations of the input data is quantified by the notion of condition number. The *condition number* of A with respect to the vector norm $\|\cdot\|$ is defined as

$$\kappa_{\|\cdot\|}(A) = \|A^{-1}\|\,\|A\|,$$

and also noted as $\kappa(A)$ when the norm is clear from the context. It is a real number greater or equal to 1, because

$$\|A^{-1}\|\,\|A\| \geq \|A^{-1}A\| = \|\mathbb{1}_n\| = 1$$

by the submultiplicativity of the operator norm.

To bound the relative error in the solution to (2.9), we consider the difference

$$\begin{array}{rrl} & (A + \delta A)\,(x + \delta x) & = b + \delta b \\ - & A\,x & = \ \ b \\ \hline & \delta\,A\,x + (A + \delta\,A)\,\delta x = & \ \ \delta b \end{array}$$

which readily implies that

$$\delta x = A^{-1}(-\delta A\,(x+\delta x)+\delta b)$$

Taking norms we obtain $\|\delta x\| \leq \|A^{-1}\|\,(\|\delta A\|\,(\|x\|+\|\delta\, x\|)+\|\delta b\|)$, that can be rearranged to

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A)\left(\frac{\|\delta A\|}{\|A\|}\left(1+\frac{\|\delta x\|}{\|x\|}\right)+\frac{\|\delta b\|}{\|A\|\,\|x\|}\right) \leq \kappa(A)\left(\frac{\|\delta A\|}{\|A\|}\left(1+\frac{\|\delta x\|}{\|x\|}\right)+\frac{\|\delta b\|}{\|b\|}\right)$$

because $\|b\| \leq \|A\|\,\|x\|$. This readily implies the following upper bound for the relative error of x in terms of those of A and b:

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1-\kappa(A)\,\frac{\|\delta A\|}{\|A\|}}\left(\frac{\|\delta A\|}{\|A\|}+\frac{\|\delta b\|}{\|b\|}\right). \tag{2.10}$$

The multiplier in the right-hand side of (2.10) is close to the condition number when the relative error of the matrix A is small, which is the case of interest. Disregarding the denominator in this multiplier, we can write this inequality as

$$-\log_\beta\left(\frac{\|\delta x\|}{\|x\|}\right) \geq -\log_\beta\left(\frac{\|\delta A\|}{\|A\|}+\frac{\|\delta b\|}{\|b\|}\right)-\log_\beta \kappa(A)$$

Hence following the rule of thumb (2.8), when solving the linear equation $A\,x=b$ we expect a loss of precision of $\approx \log_\beta \kappa(A)$ digits.

In linear equation solving, the matrix A is said to be *well/ill conditioned* if its condition number is small/large when compared to its norm. When A is ill conditioned, small relative changes in the data (A,b) might produce large changes in the solution x, as it was the case in Example 2.4.1.

Example 2.4.2. With notation as in Example 2.4.1, the relative errors with respect to the ∞-norm are, up to 5 decimal digits, are

$$\frac{\|\delta A\|_\infty}{\|A\|_\infty}=0,\quad \frac{\|\delta b\|_\infty}{\|b\|_\infty}=\frac{0.00017}{3.55}=0.00005,\quad \frac{\|\delta x\|_\infty}{\|x\|_\infty}=\frac{0.9775}{1}=0.9775. \tag{2.11}$$

The relative error has been propagated from the input data to the solution almost by a factor 20,000! In terms of floating-point numbers, we have that

$$b=(0.355\times 10^1, 0.22\times 10^1),\quad \widehat{b}=(0.35498\times 10^1, 0.220017\times 10^1),$$
$$x=(0.1\times 10^1, 0),\quad \widehat{x}=(0.19775\times 10^1, -0.3111\times 10^0).$$

Hence $\widehat{b}$ has 4 correct digits, whereas $\widehat{x}$ has none: all the precision has been lost.

Indeed, the inverse is $A^{-1}=\begin{bmatrix}-3550 & 2200\\ 1130 & -700\end{bmatrix}$ and so the condition number of A is

$$\kappa(A)=\|A^{-1}\|\,\|A\|=5750\times 4.68=26910.$$

The upper bound (2.10) writes down in this case as

$$0.9775=\frac{\|\delta x\|_\infty}{\|x\|_\infty}\leq \kappa(A)\,\frac{\|\delta b\|_\infty}{\|b\|_\infty}=26910\times 0.00005=1.3455,$$

reflecting the behavior in (2.11). Moreover,

$$\log_{10}\kappa(A)=3.75966,$$

and so the rule (2.8) is valid in this case.

In the exercises we will see families of matrices whose condition number increases exponentially with respect to their dimension.

Ill-conditioned matrices destroy the quality of your approximations. Recall that rounding with IEEE single precision and double precision give approximations with 24 and 53 correct bits, respectively. Hence if you have exact data truncated with IEEE single or double precision, the computed solution (with *any* algorithm!) of $A\,x = b$ will be meaningless as soon as

$$\kappa(A) > 2^{24} \approx 6 \cdot 10^8 \text{ (single precision)} \quad \text{and} \quad \kappa(A) > 2^{53} \approx 10^{16} \text{ (double precision)}.$$

For the 2-norm, the condition number has a beautiful geometric interpretation as the inverse of the distance of the matrix to the set of singular matrices:

$$\min\left\{ \frac{\|\delta A\|_2}{\|A\|_2} \;\middle|\; A + \delta A \text{ is singular} \right\} = \frac{1}{\kappa_2(A)},$$

see for instance [**Dem97**, Theorem 2.1]. This reciprocal relation between the condition number of a well-posed problem and its distance to the set of ill-posed ones, is an instance of a general principle in numerical analysis.

2.5. Error analysis in GEPP

The cost of each individual floating-point operation or *flop* is satisfactory: if $*$ is one of the arithmetic operations, then

$$\frac{|\lambda * \mu - \mathrm{fl}(\lambda * \mu)|}{|\lambda * \mu|} < \varepsilon \quad \text{if } m \le |\lambda * \mu| \le M.$$

The GEPP algorithm takes as input the pair (A, b) and proceeds by computing a factorization

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

with P a permutation, L unit lower triangular, and U upper triangular. Once this is achieved, the equation

$$A\,x = b$$

is solved permuting the entries of b and applying forward and backward substitution. In an actual implementation, computations are performed numerically, that is, using floating-point arithmetic, thus producing an approximate solution

$$x_{\mathrm{GEPP}}.$$

To control the error in this computation, we apply the two steps:

(1) analyze roundoff errors to show the existence of a matrix A_{GEPP} such that $A_{\mathrm{GEPP}}\,x_{\mathrm{GEPP}} = b$ having a small relative error with respect to A *(backward analysis)*,
(2) apply perturbation theory to bound the relative error of x_{GEPP} with respect to x in terms of the condition number of A and the precision of our floating-point number system.

We next study this strategy for the ∞-norm, although we might equally well consider any other standard norm like the 1-norm or the 2-norm. Rounding A gives a matrix

$$\widehat{A} = A + \delta A$$

whose entries approximate in relative error those of A, up to the machine epsilon ε, as explained in (2.5). Similarly rounding b gives a vector $\widehat{b}$ whose coefficients approximate those of b with a relative error bounded by ε too. Then

$$\|\delta A\|_\infty = \max_i \sum_{j=1}^n |\delta a_{i,j}| \le \max_i \sum_{j=1}^n \varepsilon\, |a_{i,j}| \le \varepsilon\, \max_i \sum_{j=1}^n |a_{i,j}| = \varepsilon\, \|A\|_\infty,$$

$$\|\delta b\|_\infty = \max_i |\delta b_i| \le \max_i \varepsilon\, |b_i| \le \varepsilon\, \max_i |b_i| = \varepsilon\, \|b\|_\infty,$$

and so

$$\frac{\|\delta A\|_\infty}{\|A\|_\infty}, \frac{\|\delta b\|_\infty}{\|b\|_\infty} < \varepsilon.$$

By the perturbation theorem (2.10), this error will be amplified to

$$\frac{\|\delta x\|_\infty}{\|x\|_\infty} \le \frac{\kappa(A)}{1-\varepsilon\,\kappa(A)}\, 2\,\varepsilon.$$

Hence to keep the quality of this bound, we need to show that $A_{\mathrm{GEPP}} = A + \delta A$ with

$$\frac{\|\delta A\|_\infty}{\|A\|_\infty} \le c\,\varepsilon$$

for a small constant $c > 0$. To this end, we must be careful about *pivoting*.

EXAMPLE 2.5.1. Consider the matrix

$$A = \begin{bmatrix} \eta & 1 \\ 1 & 1 \end{bmatrix}$$

with η a power of the base β that is smaller than ε. Its LU factorization (without pivoting) is $A = L\,U$ with

$$L = \begin{bmatrix} 1 & 0 \\ \eta^{-1} & 1 \end{bmatrix} \quad\text{and}\quad U = \begin{bmatrix} \eta & 1 \\ 0 & 1-\eta^{-1} \end{bmatrix}.$$

We have that

$$A^{-1} = \begin{bmatrix} \frac{1}{\eta-1} & \frac{-1}{\eta-1} \\ \frac{-1}{\eta-1} & \frac{\eta}{\eta-1} \end{bmatrix}, \quad L^{-1} = \begin{bmatrix} 1 & 0 \\ -\eta^{-1} & 1 \end{bmatrix}, \quad U^{-1} = \begin{bmatrix} \eta^{-1} & \frac{1}{1-\eta} \\ 0 & \frac{\eta}{\eta-1} \end{bmatrix}$$

and so

$$\kappa(A) \approx 4, \quad \kappa(L) \approx \eta^{-2}, \quad \kappa(U) \approx \eta^{-2}.$$

The disparity between the condition number of A and those of L and U indicates that applying Gauss elimination without partial pivoting (GEWP) to this matrix may be numerically unstable. We next verify that this is indeed the case.

Set $b = (1, 2)$, so that the solution to the equation $A\,x = b$ is

$$(x_1, x_2) = \Big(\frac{1}{1-\eta}, \frac{1-2\,\eta}{1-\eta}\Big) \approx (1, 1).$$

On the other hand, by the definition of the machine epsilon we have that

$$1 \ominus \eta^{-1} = \eta^{-1}(1 \ominus \eta).$$

Hence in floating-point arithmetic, the computed factors of A are

$$L_{\mathrm{GEWP}} = \begin{bmatrix} 1 & 0 \\ \eta^{-1} & 1 \end{bmatrix} \quad\text{and}\quad U_{\mathrm{GEWP}} = \begin{bmatrix} \eta & 1 \\ 0 & 1 \ominus \eta^{-1} \end{bmatrix} = \begin{bmatrix} \eta & 1 \\ 0 & -\eta^{-1} \end{bmatrix}.$$

Solving $L_{\text{GEWP}}\, y = b$ gives

$$y_1 = 1 \quad \text{and} \quad y_2 = 2 \ominus \eta^{-1} = -\eta^{-1}.$$

Then $U_{\text{GEWP}}\, x = y$ gives

$$x_2 = \frac{-\eta^{-1}}{-\eta^{-1}} = 1 \quad \text{and} \quad x_1 = \frac{1 \ominus 1}{\eta} = 0.$$

We obtain $x_{\text{GEWP}} = (1, 0)$, which is *not* close to the actual solution: the relative error

$$\frac{\|\delta x\|_\infty}{\|x\|_\infty} \approx 1$$

cannot be bounded by $c\,\varepsilon$ for a small constant $c > 0$, and so GEWP is not backward stable.

In this example, pivoting eliminates the instability just illustrated. Indeed, GEPP applied to this matrix gives the factorization is $A = P\,L\,U$ with

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 \\ \eta & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 1 \\ 0 & 1 - \eta \end{bmatrix}. \tag{2.12}$$

We have that $\kappa(P) = 1$, $\kappa(L) \approx 1$ and $\kappa(U) \approx 4$, and so all these factors are well-conditioned. Moreover,

$$P_{\text{GEPP}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad L_{\text{GEPP}} = \begin{bmatrix} 1 & 0 \\ \eta & 1 \end{bmatrix}, \quad U_{\text{GEPP}} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \ominus \eta \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix},$$

and successively solving $P_{\text{GEPP}}\, z = b$, $L_{GEPP} y = z$ and $U_{\text{GEPP}} x = y$ gives the accurate solution $x_{\text{GEPP}} = (1, 1)$.

2.6. Backward error analysis of GEPP

To see that GEPP is a numerically stable algorithm we need to show that the computed solution x_{GEPP} satisfies $A_{\text{GEPP}}\, x_{\text{GEPP}} = b$ with $A_{\text{GEPP}} = A + \delta A$ such that

$$\frac{\|\delta A\|_\infty}{\|A\|_\infty} \leq c\,\varepsilon$$

for a small constant $c > 0$ *(backward stability)*.

As shown in Example 2.5.1, crucial information might be lost when intermediate quantities of disparate size are added together. This is an actual risk for us, because GEPP does perform a lot of additions and subtractions and, as a matter of fact, GEPP is *not* backward stable. Still we can give an interesting upper bound for the relative error it produces. In the sequel we will explain this bound and give some remarks of practical interest.

We denote by $|A|$ the $n \times n$ matrix whose entries are the absolute values of those of A, that is

$$|A| = \left[|a_{i,j}|\right]_{i,j}.$$

Analyzing the roundoff errors of the involved computations, one can find an $n \times n$ matrix A_{GEPP} with $A_{\text{GEPP}}\, x = b$ such that setting $\delta A = A - A_{\text{GEPP}}$ we have that

$$|\delta A| \leq (3\,n\,\varepsilon + n^2\,\varepsilon^2)\,|L_{\text{GEPP}}|\,|U_{\text{GEPP}}|.$$

see [**Dem97**, pages 47-49] for the details. Taking norms, this readily implies that

$$\frac{\|\delta A\|_\infty}{\|A\|_\infty} \leq (3\,n\,\varepsilon + n^2\,\varepsilon^2)\,\frac{\|L_{\text{GEPP}}\|_\infty\,\|U_{\text{GEPP}}\|_\infty}{\|A\|_\infty}. \tag{2.13}$$

The practice of numerical linear solving is that GEPP *almost* always keeps

$$\|L_{\mathrm{GEPP}}\|_\infty \, \|U_{\mathrm{GEPP}}\|_\infty \approx \|A\|_\infty,$$

as in (2.12). If this were the case, then we would have

$$\frac{\|\delta_{\mathrm{GEPP}} A\|}{\|A\|} \lesssim n\,\varepsilon.$$

Thus we say that GEPP is backward stable *in practice.*

For a theoretical upper bound we can consider the *pivot growth factor*

$$g_{\mathrm{GEPP}} = \frac{\max_{i,j} |u_{i,j}|}{\max_{i,j} |a_{i,j}|}.$$

GEPP guarantees that the entries of L_{GEPP} are bounded by 1 in absolute value and so

$$\|L\|_\infty \le n \quad \text{and} \quad \|U\|_\infty \le n\, g_{\mathrm{GEPP}}\, \|A\|_\infty.$$

Together with (2.13) this implies that

$$\frac{\|\delta A\|_\infty}{\|A\|_\infty} \le 3\, n^3 \varepsilon\, g_{\mathrm{GEPP}}. \tag{2.14}$$

Backward stability amounts to the fact that g_{GEPP} is small or grows slowly as a function of n. In general, we have that

$$g_{\mathrm{GEPP}} \le 2^{n-1},$$

and unfortunately, there are rare cases where this exponential bound is attained: for instance, when $n = 4$ we have that

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & -1 & 1 & 0 \\ -1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 8 \end{bmatrix},$$

and this example can be easily generalized to an $n \times n$ matrix with pivot growth 2^{n-1}

CHAPTER 3

Special linear systems

As a general principle, we want to take advantage of any special structure that may be present in the problem under consideration, to increase the efficiency of our computations. This increase of efficiency might be translated into faster computations using less memory space, and/or more reliable results. Next we will study matrices exhibiting properties like *symmetry*, *definiteness* and *bandedness*.

3.1. Symmetric matrices

An $n \times n$ matrix A is *symmetric* if

$$A^T = A.$$

Since $a_{j,i} = a_{i,j}$ for all i, j, to represent A we only need to specify the $\frac{n\,(n+1)}{2}$ entries $a_{i,j}$ for $i \geq j$, instead of the usual n^2 ones, and so to store a symmetric matrix we need approximately half the space that for general one. Consequently, in this case we

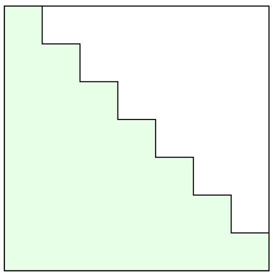


Figure 3.1.1. Storage of a symmetric matrix

would like to solve the linear equation

$$A\,x = b$$

with half the complexity of the general case, that is, using (approximately) $\frac{n^2}{2}$ memory slots instead of n^2, and $\frac{n^3}{3}$ flops instead of $\frac{2\,n^3}{3}$.

Permuting rows destroys symmetry, as can be seen already when $n = 2$:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} b & c \\ a & b \end{bmatrix}.$$

Hence to preserve symmetry, we will avoid using any pivoting strategy, and consequently we will only consider its LU factorization, whenever it exists.

From now on, suppose that A is a symmetric nonsingular $n \times n$ matrix that has a factorization

$$A = L\,U \tag{3.1}$$

with L unit lower triangular and U upper triangular. When this occurs, these factors are connected: for instance, when $n = 2$ we have that

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{b}{a} & 1 \end{bmatrix} \begin{bmatrix} a & b \\ 0 & c - \frac{b^2}{a} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{b}{a} & 1 \end{bmatrix} \begin{bmatrix} a & 0 \\ 0 & c - \frac{b^2}{a} \end{bmatrix} \begin{bmatrix} 1 & \frac{b}{a} \\ 0 & 1 \end{bmatrix},$$

and so U is a row scaling of L. This is a general fact: in the factorization (3.1) we have that

$$U = D\,L^T$$

with $D = \operatorname{diag}(u_{1,1}, \dots, u_{n,n})$, and so this factorization is equivalent to

$$A = L\,D\,L^T$$

with L unit lower triangular and D diagonal.

This LDLT factorization also allows to solve the linear equation $A\,x = b$ following the steps:

1. solve $L\,z = b$ by forward substitution,
2. solve $D\,y = z$ scaling each entry,
3. solve $L^T\,x = y$ by backward substitution.

To compute it, we proceed similarly as we did before for the PLU factorization: consider the 2×2 block decomposition

$$A = \begin{array}{c} \begin{array}{cc} \scriptstyle 1 & \scriptstyle n-1 \end{array} \\ \begin{bmatrix} a_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \end{array} \begin{array}{l} \scriptstyle 1 \\ \scriptstyle n-1 \end{array}$$

where $a_{1,1}$ is the $(1,1)$-entry of A, $A_{1,2}$ and $A_{2,1}$ are the rest of its first row and columns respectively, and $A_{2,2}$ is the remaining $(n-1) \times (n-1)$ block. Since we assume that A admits an LU factorization we have that $a_{1,1} \neq 0$, and since it is symmetric we also have that $A_{1,2} = A_{2,1}^T$. Then we can compute the the block LDLT factorization

$$\begin{bmatrix} a_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} = \begin{bmatrix} 1 & \mathbb{0} \\ L_{2,1} & \mathbb{1}_{n-1} \end{bmatrix} \begin{bmatrix} d_1 & \mathbb{0} \\ \mathbb{0} & A_1 \end{bmatrix} \begin{bmatrix} 1 & L_{2,1}^T \\ \mathbb{0} & \mathbb{1}_{n-1} \end{bmatrix}$$

by setting

$$d_1 = a_{1,1}, \quad L_{2,1} = a_{1,1}^{-1}\,A_{2,1}, \quad A_1 = A_{2,2} - A_{2,1}\,L_{2,1}^T. \tag{3.2}$$

The Schur complement A_1 is a symmetric nonsingular $(n-1) \times (n-1)$ matrix and so we can repeat the procedure on it, leading to a symmetric version of Algorithm 1.3.1.

For convenience, we write the Schur complements defined by the successive application of the formulae in (3.2) as

$$A_j = \big[a_{i,k}^{(j)}\big]_{j+1 \leq i,k \leq n} \qquad \text{for } j = 0, \dots, n-1.$$

As it stands, this algorithm is inefficient both from the point of view of memory usage and speed of execution. To fully profit from the symmetry of the input matrix A, first recall that it can be represented keeping only its (i,j)-th entries for $i \geq j$. At the j-th step of Algorithm 3.1.1, the (j,j)-entry of the $(j-1)$-th Schur complement A_{j-1} is assigned to the j-th diagonal entry of D and never used again. On the other hand, its (i,j)-th entry is used both to compute the (i,j)-th entry of L and the i-th row of the j-th Schur complement. Moreover, the (i,k)-entries of A_{j-1} are only used to compute this next Schur complement. Moreover, since it is symmetric, we only need to compute its lower triangular part.

Algorithm 3.1.1 (LDLT algorithm)

```
for $j = 1, \dots, n$
    $d_j \leftarrow a_{j,j}^{(j-1)}$    (compute the $j$-th diagonal entry of $D$)
    for $i = j+1, \dots, n$
        $l_{i,j} \leftarrow a_{i,j}^{(j-1)} / a_{j,j}^{(j-1)}$    (compute the $j$-th column of $L$)
    for $i, k = j+1, \dots, n$
        $a_{i,k}^{(j)} \leftarrow a_{i,k}^{(j-1)} - a_{i,j}^{(j-1)}\, l_{k,j}$    (compute the $j$-th Schur complement)
$d_n \leftarrow a_{n,n}^{(n-1)}$    (compute the $n$-th diagonal entry of $D$)
```

Hence we can safely rewrite the lower triangular part of A with the j-th diagonal entry of D, the nontrivial entries in the j-th column of L, and the lower triangular part of the j-th Schur complement. Hence we need no extra space to store them apart from an auxiliary $(n-1)$-th vector c to keep the values of the j-th column of A. At the end of the procedure, the lower triangular part of A would contain all the diagonal entries of D and the nontrivial entries of L.

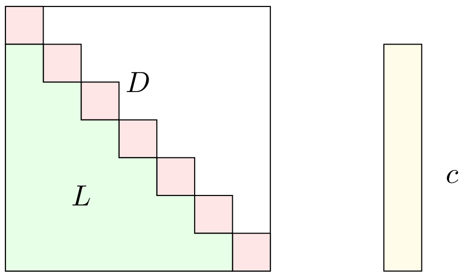


Figure 3.1.2. Storage distribution for LDLT

Algorithm 3.1.2 (LDLT algorithm with storage management)

```
for $j = 1, \dots, n$
    for $i = j+1, \dots, n$
        $c_i \leftarrow a_{i,j}$
        $a_{i,j} \leftarrow a_{i,j} / a_{j,j}$
    for $k = j+1, \dots, n$ and $i = k, \dots, n$
    $a_{i,k} \leftarrow a_{i,k} - c_i\, a_{k,j}$
```

As shown in Figure 3.1.2, this algorithm uses an (essentially) optimal amount of space, namely

$$\frac{n\,(n+1)}{2} + n - 1 = \frac{n^2}{2} \text{ memory slots.}$$

On the other hand, its time complexity is

$$\sum_{j=1}^{n-1} \Bigg(\sum_{i=j+1}^{n} 1 + \sum_{j+1 \le k \le i \le n} 2 \Bigg) = \sum_{j=1}^{n-1} ((n-j) + (n-j)\,(n-j+1))$$
$$= \sum_{j=1}^{n-1} ((n-j)^2 + O(n-j)) = \frac{n^3}{3} + O(n^2),$$

which is asymptotically half the time complexity of the PLU factorization.

Example 3.1.1. Consider the 3×3 matrix

$$A = \begin{bmatrix} 1 & -1 & 2 \\ -1 & 5 & 2 \\ 2 & 2 & 17 \end{bmatrix}.$$

Applying Algorithm 3.1.2, for $j = 1$ we have that

$$c = \begin{bmatrix} -1 \\ 2 \end{bmatrix} \quad \text{and} \quad A = \begin{bmatrix} 1 & & \\ -1/1 & 5-(-1)\cdot(-1) & \\ 2/1 & 2-2\cdot(-1) & 17-2\cdot 2 \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1 & 4 & \\ 2 & 4 & 13 \end{bmatrix}.$$

For $j = 2$ we have that

$$c = \begin{bmatrix} * \\ 4 \end{bmatrix} \quad \text{and} \quad A = \begin{bmatrix} 1 & & \\ -1 & 4 & \\ 2 & 4/4 & 13-4\cdot 1 \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1 & 4 & \\ 2 & 1 & 9 \end{bmatrix},$$

from where we extract the matrices in the LDLT factorization:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix} \quad \text{and} \quad D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 9 \end{bmatrix}.$$

In general, the LDLT of a symmetric matrix can be numerically unstable, as shown already by the matrix

$$A = \begin{bmatrix} \eta & 1 \\ 1 & 1 \end{bmatrix}$$

for $0 < \eta$ smaller than the machine epsilon (Example 2.5.1). Still this factorization can be useful in practice, whenever the successive pivots are not too small.

3.2. Symmetric positive definite systems

Let A be a symmetric $n \times n$ matrix with real coefficients. We say that A is a *positive-definite* if for all $x \in \mathbb{R}^n \setminus \{0\}$ we have that

$$x^T A\, x > 0.$$

Positive definite symmetric (PDS) matrices play an important role in convex optimization. For instance, given a function of several real variables that is twice differentiable, if its Hessian matrix (the matrix of its second partial derivatives) is positive-definite then the function is convex in a neighborhood of that point.

PDS matrices can be characterized in several ways. First of all, there is an important result from linear algebra saying that a real $n \times n$ matrix A is symmetric if and only if it is orthogonally similar to a diagonal matrix, that is

$$A = Q^T \Lambda\, Q \tag{3.3}$$

with Q orthogonal and Λ diagonal, and then A is positive-definite if and only if the diagonal entries of Λ are positive.

In the factorization (3.3), we have that $Q^{-1} = Q^T$ and that the diagonal entries of Λ coincide with the eigenvalues of the given symmetric matrix. Hence PDS matrices can also be characterized as the symmetric real matrices whose eigenvalues are positive.

A third characterization of PDS matrices is given by the existence of a *Cholesky factorization*, a variant of the LU factorization with a positivity property. Namely, a real $n \times n$ matrix A is PDS if and only if there is a lower triangular $n \times n$ matrix G with positive diagonal entries such that

$$A = G\,G^T. \tag{3.4}$$

When it exists, this lower triangular matrix is unique [**Dem97**, Proposition 2.2].

This factorization is closely related to the LDLT factorization of a symmetric matrix studied in §3.1. Indeed, consider the diagonal matrix $\Lambda = \mathrm{diag}(g_{1,1}, \dots, g_{n,n})$ and set

$$L = G\,\Lambda^{-1} \quad \text{and} \quad D = \Lambda^2.$$

Then L is unit lower triangular and D is diagonal, and the Cholesky factorization (3.4) translates into the LDLT factorization

$$A = L\,D\,L^T.$$

Indeed, one way of computing the Cholesky factorization a PDS matrix proceeds by computing its LDLT factorization with Algorithm 1.3.2. The diagonal entries of D are positive and we might then obtain the lower triangular matrix G in (3.4) by setting

$$G = L\,\mathrm{diag}(d_{1,1}^{1/2}, \dots, d_{n,n}^{1/2}).$$

As a matter of fact, it will be easier to proceed in a different way. Given the PDS $n \times n$ matrix A, the matrix G is the solution of a system of $\frac{n\,(n+1)}{2}$ equations (one per each entry of A in its lower triangular part) in $\frac{n\,(n+1)}{2}$ variables (the nontrivial entries of G).

These equations are nonlinear but nevertheless, they can be easily solved when appropriately ordered: for each $i \ge j$ we have that $a_{i,j} = \sum_{k=0}^{j} g_{j,k}\, g_{i,k}$, or equivalently

$$g_{j,j}\, g_{i,j} = a_{i,j} - \sum_{k=1}^{j-1} g_{j,k}\, g_{i,k}.$$

If the first $j-1$ columns of G are already known, we can use this equation to compute the diagonal entry $g_{j,j}$ and then the rest of the entries in this column. Similarly as with GEPP (and to a great extent with the LDLT factorization) each (i,j)-th entry of A is used only to compute the corresponding entry of G, and so we can safely overwrite it. At the end of the procedure, the matrix A would contain all the nontrivial entries of G in its lower triangular part.

Algorithm 3.2.1 (Cholesky algorithm))

```
for $j = 1, \dots, n$
    $a_{j,j} \leftarrow (a_{j,j} - \sum_{k=1}^{j-1} a_{j,k}^2)^{1/2}$
    for $i = j+1, \dots, n$
        $a_{i,j} \leftarrow (a_{i,j} - \sum_{k=1}^{j-1} a_{i,k}\, a_{j,k})/a_{j,j}$
```

Thanks to the characterization of PDS matrices in (3.4), this algorithm might be applied to an arbitrary real symmetric A: if this matrix is PDS then we would obtain its Cholesky factorization as explained and if it is not, then the computation algorithm would break down because of a forbidden operation, like taking the square root of a negative number or dividing by zero. Indeed, this is the cheapest way to test if a given real symmetric matrix is definite-positive or not.

EXAMPLE 3.2.1. Consider again the 3×3 matrix

$$A = \begin{bmatrix} 1 & -1 & 2 \\ -1 & 5 & 2 \\ 2 & 2 & 17 \end{bmatrix}$$

as in Example 3.1.1. Applying the Cholesky algorithm, for $j = 1$ we obtain

$$A = \begin{bmatrix} 1^{1/2} & & \\ -1/1 & 5 & \\ 2/1 & 2 & 17 \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1 & 5 & \\ 2 & 2 & 17 \end{bmatrix}.$$

For $j = 2$ we have that

$$A = \begin{bmatrix} 1 & & \\ -1 & (5 - (-1) \cdot (-1))^{1/2} & \\ 2 & (2 - (-1) \cdot 2)/2 & 17 \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1 & 2 & \\ 2 & 2 & 17 \end{bmatrix}$$

Finally, for $j = 3$ we obtain

$$A = \begin{bmatrix} 1 & & \\ -1 & 2 & \\ 2 & 2 & (17 - (2 \cdot 2 + 2 \cdot 2))^{1/2} \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1 & 2 & \\ 2 & 2 & 3 \end{bmatrix}.$$

Hence A is positive-definite, and the matrix in its Cholesky factorization is

$$G = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 2 & 0 \\ 2 & 2 & 3 \end{bmatrix}.$$

Algorithm 3.2.1 uses an optimal amount of space, namely

$$\frac{n\,(n+1)}{2} \text{ memory slots,}$$

exactly the space needed to represent A.

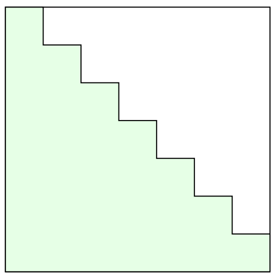


FIGURE 3.2.1. Storage distribution for the Cholesky algorithm

Its time complexity is

$$\begin{aligned}\sum_{j=1}^{n}\Big(2\,j-1+\sum_{i=j+1}^{n}(2\,j-1)\Big)&=\sum_{j=1}^{n}(n-j+1)\,(2\,j-1)\\&=\sum_{j=1}^{n}2\,n\,j-\sum_{j=1}^{n}2\,j^2+O(n^2)\\&=(n^3+O(n^2))-\Big(\frac{2}{3}\,n^3+O(n^2)\Big)+O(n^2)\\&=\frac{1}{3}\,n^3+O(n^2),\end{aligned}$$

which is also asymptotically half the time complexity of the PLU factorization.

Furthermore, the Cholesky algorithm is also backward stable. Indeed, let A be a PDS matrix and b a real n-vector, and let x_{Ch} be the computed solution of the equation $A\,x=b$ using the Cholesky factorization, combined with forward and backward substitution for solving the equations $G\,y=b$ and $G^T x=y$ respectively. The same analysis of GEPP in § 2.6 shows that this solution satisfies

$$(A+\delta A)\,x_{\mathrm{Ch}}=b$$

with

$$|\delta A|\le 3\,n\,\varepsilon\,|G|\,|G^T| \tag{3.5}$$

with ε the machine epsilon and where the bars indicate the matrices whose entries are the absolute values of those in the indicated ones, and where the inequality occurs at every entry.

By the Cauchy-Schwartz inequality, for each i,j we have that

$$(|G|\,|G^T|)_{i,j}\le\sum_{k=1}^{n}|g_{i,k}|\,|g_{j,k}|\le\Big(\sum_{k=1}^{n}g_{i,k}^2\Big)^{1/2}\Big(\sum_{k=1}^{n}g_{j,k}^2\Big)^{1/2}=a_{i,i}^{1/2}a_{j,j}^{1/2}\le\max_{i,j}|a_{i,j}|.$$

Hence $\||G|\,|G^T|\|_\infty\le n\,\|A\|_\infty$, and so we deduce from (3.5) the upper bound for the relative backward error

$$\frac{\|\delta\,A\|_\infty}{\|A\|_\infty}\le 3\,n^2\,\varepsilon.$$

Properly done, all the elements in this section can be extended to the complex case. An $n\times n$ matrix A with complex entries is *Hermitian* if it coincides with its conjugate transpose that is, if

$$A^*=A.$$

A Hermitian matrix is *positive-definite* if for all $x\in\mathbb{C}^n\setminus\{0\}$ we have that

$$x^*A\,x>0.$$

Hermitian matrices enjoy similar properties as those of symmetric real matrices: for instance, a complex $n\times n$ matrix A is Hermitian if and only if

$$A=Q^*\Lambda\,Q$$

with Q a unitary matrix and Λ a diagonal matrix with real entries. When this is the case, the Hermitian matrix A is positive-definite if and only if the diagonal entries of Λ are positive.

The Cholesky factorization also extends to this setting: a complex $n \times n$ matrix A is positive-definite Hermitian if and only if it can be factored as

$$A = G\, G^*$$

with G lower triangular with positive diagonal entries, and Algorithm 3.2.1 can be easily modified to compute this matrix.

3.3. Band matrices

Roughly speaking, a *band matrix* is a matrix whose entries are concentrated around the diagonal. Such matrices arise from systems of (scalar) linear equations that can be ordered in such a way that each i-th variable only appears in a neighborhood of the i-th equation, like those appearing when discretizing ordinary differential equations (ODE's).

More precisely, the *lower bandwidth* and the *upper bandwidth* of an $n \times n$ matrix A are respectively defined as the smallest integers b_L and b_U such that

$$a_{i,j} = 0$$

whenever $i > j - b_L$ or $i < j + b_U$. For instance, a nonsingular lower (respectively upper) triangular matrix has lower (respectively upper) bandwidth equal to zero, a tridiagonal matrix with nonzero subdiagonal and supdiagonal entries has lower and upper bandwidths equal to one, and so on.

A nonsingular matrix A with lower bandwidth b_L and upper band with b_U has the shape

$$\begin{bmatrix} a_{1,1} & \cdots & a_{1,b_U+1} & & \\ \vdots & & & \ddots & \\ a_{b_L+1,1} & & & & a_{n-b_U,n} \\ & \ddots & & & \vdots \\ & & a_{n,n-b_L} & \cdots & a_{n,n} \end{bmatrix}$$

and so we need at most $(b_L + b_U)\, n$ entries to represent it.

Its LU factorization (whenever it exists!) preserves its band structure: we have that

$$A = L\, U$$

with L unit lower triangular with lower bandwidth b_L and U upper triangular with upper bandwidth b_U.

Indeed, this factorization can be computed iteratively using Algorithm 1.3.2 skipping the pivoting strategy. At the j-th step, the j-th Schur complement is computed with the formula

$$a_{i,k} \leftarrow a_{i,k} - a_{i,j}\, a_{j,k} \quad \text{for } i, k = j+1, \ldots, n.$$

Hence the lower and upper bandwidths of this Schur complement are bounded by those of the previous one, and so by those of the input matrix A.

Thus this factorization can be computed by overwriting the nontrivial entries of A and so with at most $(b_L + b_U)\, n$ memory slots, and this can be done using at most

$$2\, n\, b_L\, b_U + O(n\, (b_L + b_U)) \text{ flops.}$$

EXAMPLE 3.3.1. Consider the 4×4 matrix

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \end{bmatrix}$$

which has lower and upper bandwidths equal to 1. Applying Algorithm 1.3.2 without pivoting, for $j = 1$ we obtain

$$A = \begin{bmatrix} 2 & -1 & & \\ 4/2 & -1-2\cdot(-1) & 3 & \\ & -1 & -2 & 1 \\ & & 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & -1 & & \\ 2 & 1 & 3 & \\ & -1 & -2 & 1 \\ & & 3 & 4 \end{bmatrix}.$$

For $j = 2$ we have that

$$A = \begin{bmatrix} 2 & -1 & & \\ 2 & 1 & 3 & \\ & -1/1 & -2-(-1)\cdot 3 & 1 \\ & & 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & -1 & & \\ 2 & 1 & 3 & \\ & -1 & 1 & 1 \\ & & 3 & 4 \end{bmatrix}.$$

Finally, for $j = 3$ we obtain that

$$A = \begin{bmatrix} 2 & -1 & & \\ 2 & 1 & 3 & \\ & -1 & 1 & 1 \\ & & 3/1 & 4-3\cdot 1 \end{bmatrix} = \begin{bmatrix} 2 & -1 & & \\ 2 & 1 & 3 & \\ & -1 & 1 & 1 \\ & & 3 & 1 \end{bmatrix}.$$

We conclude that

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 3 & 1 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} 2 & -1 & 0 & 0 \\ 0 & 1 & 3 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

GEPP can exploit band structure, but the band properties of the factors L and U are less simple. Indeed, GEPP produces a factorization

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

where U is banded with upper bandwidth $b_L + b_U$, and L has at most $b_L + 1$ nonzero entries per column.

Indeed, at the j-th step of Algorithm 1.3.2, pivoting can only be done within the first b_L rows, because the others have only zeros on the j-th column. Hence the corresponding permutation can increase the upper bandwidth by b_L, and it can also reorder the entries of the previously computed columns of L.

EXAMPLE 3.3.2. Consider again the matrix in Example 3.3.1

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \end{bmatrix}$$

Applying GEPP (Algorithm 1.3.2), for $j = 1$ we first swap rows 1 and 2 and then we compute the first column of L, the first row of U, and the first Schur complement:

$$A = \begin{bmatrix} 4 & -1 & 3 & 0 \\ 2/4 & -1 - \frac{1}{2} \cdot (-1) & 0 - \frac{1}{2} \cdot 3 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 4 & -1 & 3 & 0 \\ \frac{1}{2} & \frac{-1}{2} & \frac{-3}{2} & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \end{bmatrix}.$$

For $j = 2$ we swap the rows 2 and 3 of A and then compute the second column of L, row of U, and Schur complement:

$$A = \begin{bmatrix} 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ \frac{1}{2} & \frac{-1}{2}/(-1) & \frac{-3}{2} - \frac{1}{2} \cdot (-2) & 0 - \frac{1}{2} \cdot 1 \\ 0 & 0 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ \frac{1}{2} & \frac{1}{2} & \frac{-1}{2} & \frac{-1}{2} \\ 0 & 0 & 3 & 4 \end{bmatrix}.$$

Finally, for $j = 3$ we swap rows 3 and 4 and we compute the third column of L, row of U and Schur complement to obtain

$$A = \begin{bmatrix} 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \\ \frac{1}{2} & \frac{1}{2} & \frac{-1}{2}/3 & \frac{-1}{2} - \frac{-1}{6} \cdot 4 \end{bmatrix} = \begin{bmatrix} 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \\ \frac{1}{2} & \frac{1}{2} & \frac{-1}{6} & \frac{1}{6} \end{bmatrix}.$$

We conclude that

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & & & \\ 0 & 1 & & \\ 0 & 0 & 1 & \\ \frac{1}{2} & \frac{1}{2} & \frac{-1}{6} & 1 \end{bmatrix}, \quad U \begin{bmatrix} 4 & -1 & 3 & 0 \\ & -1 & -2 & 1 \\ & & 3 & 4 \\ & & & \frac{1}{6} \end{bmatrix}.$$

In this case, U has upper bandwidth 2 and L has at most 2 nonzero elements per column.

CHAPTER 4

Iterative methods

Iterative methods for solving the linear equation $A\,x = b$ are used when the direct methods, like GEPP, require either too much time or too much space. They are usually based on matrix-vector multiplications, and so they are particularly convenient when the cost of this operation is low, like it happens for sparse matrices.

These methods do not produce an exact answer after a finite number of steps, but rather decrease the error by some amount after each step. The final error depends on how many iterations are done, and on the properties of the considered method and the matrix to which it is applied.

4.1. Splittings and iterative methods

Let A be a nonsingular $n \times n$ matrix and b an n-vector. Given an initial n-vector x_0, an iterative method generate a sequence of n-vectors

$$(x_l)_{l\geq 0}$$

hopefully converging to the solution $x = A^{-1}\,b$, and where each vector is easy to compute from the previous one.

A *splitting* of A is a decomposition

$$A = M - K$$

with M nonsingular. Such a decomposition produces an iterative method as above in the following way: the equation $A\,x = b$ is equivalent to $M\,x = K\,x + b$ or still to

$$x = M^{-1}K\,x + M^{-1}b.$$

Decoupling both sides of this equality we obtain the iteration

$$x_l = R\,x_{l-1} + c \quad \text{for } l \geq 1 \tag{4.1}$$

with $R = M^{-1}K$ and $c = M^{-1}b$. When this iteration converges, its limit x_∞ satisfies the equation $x_\infty = R\,x_\infty + c$ or equivalently

$$A\,x_\infty = b,$$

and so this limit *is* the solution of the linear equation.

The convergence of this method depends on the *spectral radius* of the iteration matrix, denoted by $\rho(R)$ and defined as the maximum absolute value of its eigenvalues. Indeed, the iteration (4.1) converges for every choice of the initial vector x_0 if and only if

$$\rho(R) < 1. \tag{4.2}$$

To see this, let x be the solution of the linear equation $A\,x = b$. If $\rho(R) \geq 1$ then choose x_0 such that $x - x_0$ is an eigenvector for an eigenvalue λ of R of absolute value ≥ 1. Hence

$$x - x_l = R^l\,(x - x_0) = \lambda^l\,(x - x_0),$$

and so x_l does not converge to x in this case.

For the converse, suppose for simplicity that R is diagonalizable, so that we can factor this matrix as

$$R = S\,\Lambda\,S^{-1}$$

with S nonsingular and $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ diagonal. Then for each $l \geq 0$ we have that

$$\begin{array}{rrcl} & x &=& R\,x + c \\ - & x_l &=& R\,x_{l-1} + c \\ \hline & x - x_l &=& R\,(x - x_{l-1}) \end{array}$$

which readily implies that

$$x - x_l = R^l(x - x_0) = S\,\Lambda^l\,S^{-1}(x - x_0).$$

Taking ∞-norms we obtain that

$$\begin{aligned} \|x - x_l\|_\infty &= \|S\,\Lambda^l\,S^{-1}(x - x_0)\|_\infty \\ &\leq \|S\|_\infty \|\Lambda\|_\infty^l \|S^{-1}\|_\infty \|x - x_0\|_\infty = \kappa(S)\,\rho(R)^l \|x - x_0\|_\infty \end{aligned} \tag{4.3}$$

because $\|\Lambda\|_\infty = \max_j |\lambda_j| = \rho(R)$. Hence if the condition (4.2) holds, then the upper bound in (4.3) tends to 0 when $l \to +\infty$ and to the iteration converges.

Moreover, if $x \neq 0$ or equivalently $b \neq 0$, then we deduce from (4.3) the upper bound for the relative error

$$\frac{\|x - x_l\|_\infty}{\|x\|_\infty} \leq \kappa(S)\,\rho(R)^l\,\frac{\|x - x_0\|_\infty}{\|x\|_\infty}.$$

Setting $\gamma(R) = -\log_\beta \rho(R) > 0$ with β the base of our floating point system, this upper bound can be rewritten as

$$-\log_\beta\Big(\frac{\|x - x_l\|_\infty}{\|x\|_\infty}\Big) \geq \gamma(R)\,l - \log_\beta \kappa(S) - \log_\beta\Big(\frac{\|x - x_0\|_\infty}{\|x\|_\infty}\Big)$$

which, following the rule of thumb (2.8), shows that when the condition (4.2) holds, the precision of the approximations increases *linearly* with rate $\gamma(R)$: the smaller the spectral radius is, the higher is the rate of convergence.

To design an efficient iterative scheme for a given problem, we need to find a splitting $A = M - K$ verifying the conditions:

(1) $R = M^{-1}K$ and $c = M^{-1}b$ are easy to compute,
(2) $\rho(R)$ is small.

4.2. The Richardson iteration

This is a simple iterative method that will mainly serve us as an example where the relevant aspects can be fully understood.

Given a nonsingular $n \times n$ matrix A and an n-vector b, the *Richardson iteration* aims at improving an approximate solution $\widehat{x}$ of the equation $A\,x = b$ by adding to it multiple of the residual $b - A\,\widehat{x}$. Precisely, for a parameter $\omega > 0$ this iteration starts from an initial n-vector x_0 and constructs the sequence $(x_l)_{l \geq 0}$ by setting

$$x_l = x_{l-1} + \omega\,(b - A\,x_{l-1}) \qquad \text{for } l \geq 1. \tag{4.4}$$

In the context of the previous section, it corresponds to the splitting $A = M_\omega - K_\omega$ with $M_\omega = \omega^{-1}\,\mathbb{1}_n$ and $K_\omega = \omega^{-1}\,\mathbb{1}_n - A$, and so we can rewrite it as

$$x_l = R_\omega\,x_{l-1} + c_\omega$$

with $R_\omega = M_\omega^{-1} K_\omega = \mathbb{1}_n - \omega\, A$ and $c_\omega = M^{-1} b = \omega\, b$.

The eigenvalues of this iteration matrix are of the form $1 - \omega\,\lambda$ with $\lambda \in \mathbb{C}$ an eigenvalue of A. Hence if we denote by $\Lambda(A)$ the *spectrum* of A, that is, the set of its eigenvalues, then

$$\rho(R_\omega) = \max_{\lambda \in \Lambda(A)} |1 - \omega\,\lambda|.$$

Hence by the criterion in (4.2), the Richardson iteration (4.4) converges for every choice of initial vector x_0 if and only if

$$|1 - \omega\,\lambda| < 1 \quad \text{for all } \lambda \in \Lambda(A) \tag{4.5}$$

or equivalently, if the spectrum of A lies in the open disk centered at the point ω^{-1} and of radius ω^{-1}:

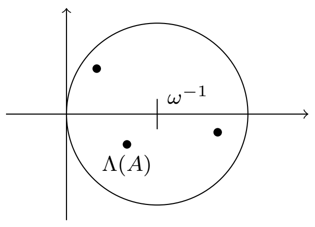


Figure 4.2.1. Convergence of the Richardson iteration

To make the full convergence analysis, we assume for simplicity that the eigenvalues of A are real. We respectively denote by $\lambda_{\min}$ and $\lambda_{\max}$ the minimal and the maximal ones, so that $\lambda_{\min} \le \lambda \le \lambda_{\max}$ for all $\lambda \in \Lambda(A)$.

The condition for convergence in (4.5) then translates into $-1 < 1 - \omega\,\lambda_{\max}$ and $1 - \omega\,\lambda_{\min} < 1$, which are equivalent to

$$0 < \lambda_{\min} \quad \text{and} \quad \lambda_{\max} < \frac{2}{\omega}.$$

Hence in this situation, the iteration converges if and only if all the eigenvalues are positive and the parameter satisfies

$$\omega < \frac{2}{\lambda_{\max}}.$$

When this is the case, the spectral radius is given by the piecewise affine function

$$\rho(R_\omega) = \max(|1 - \omega\,\lambda_{\min}|, |1 - \omega\,\lambda_{\max}|),$$

whose graph is shown in Figure 4.2.2.

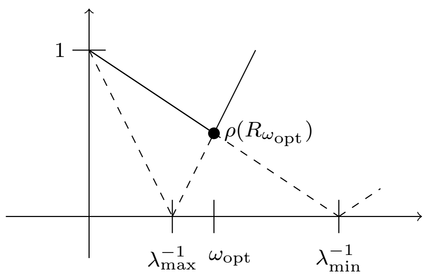


Figure 4.2.2. The spectral radius of R_ω

As shown in this figure, the best value ω_{opt}, that is the value that minimizes the spectral radius, is reached at the point where the graph of $\omega \mapsto |1-\omega\,\lambda_{\max}|$ with positive slope crosses the graph of $\omega \mapsto |1-\omega\,\lambda_{\min}|$ with negative slope, that is when $-1+\lambda_1\,\omega_{\mathrm{opt}} = 1-\lambda_n\omega_{\mathrm{opt}}$. This gives the value

$$\omega_{\mathrm{opt}} = \frac{2}{\lambda_{\max}+\lambda_{\min}},$$

whose corresponding spectral radius is

$$\rho(\omega_{\mathrm{opt}}) = \frac{\lambda_{\max}-\lambda_{\min}}{\lambda_{\max}+\lambda_{\min}}.$$

If A has very large and very small eigenvalues, then the Richardson iteration will be extremely slow, even when using the optimal parameter ω_{opt}. Moreover, the determination of this optimal parameter requires knowledge of the maximal and minimal eigenvalue, which is not easily accessible in realistic problems.

4.3. Basic iterative methods

For the methods to be discussed in the section, we assume that all the diagonal entries of A are nonzero and we decompose it as

$$A = D - \widetilde{L} - \widetilde{U} = D\,(\mathbb{1}_n - L - U)$$

with D diagonal, $\widetilde{L}$ and L strictly lower triangular, and $\widetilde{U}$ and U strictly upper triangular.

Jabobi's method can be interpreted as going successively through each scalar linear equation, so that at the l-th iteration the updated value of j-th component of the approximate solution satisfies the j-th scalar linear equation when evaluated at the previous values of the other components. Precisely, for each $l \geq 1$ we want to have

$$a_{j,1}\,x_{l-1,1} + \cdots + a_{j,j-1}\,x_{l-1,j-1} + a_{j,j}\,x_{l,j} + a_{j,j+1}\,x_{l-1,j+1} + \cdots + a_{j,n}\,x_{l-1,n} = b_j$$

for $j = 1,\dots,n$. This leads to an iterative method starting with a given initial vector x_0 and where the l-th iteration is defined by the rule:

Algorithm 4.3.1 (Jacobi's method)

```
for $j = 1, \dots, n$
    $x_{l,j} \leftarrow \frac{1}{a_{j,j}}\left(b_j - \sum_{k\neq j} a_{j,k}\,x_{l-1,k}\right)$
```

It can be expressed in matrix notation as $x_l = R_{\mathrm{J}}\,x_{l-1} + c_{\mathrm{J}}$ with

$$R_{\mathrm{J}} = D^{-1}(\widetilde{L}+\widetilde{U}) = L + U \quad \text{and} \quad c_{\mathrm{J}} = D^{-1}b. \tag{4.6}$$

The *Gauss-Seidel method* is motivated by the observation that at the j-th step of Jacobi's method we have already computed improved values for the first $j-1$ components of the approximate solution, and so it is reasonable to take advantage of them when updating the j-th component: for each $l \geq 0$ we aim at

$$a_{j,1}\,x_{l,1} + \cdots + a_{j,j-1}\,x_{l,j-1} + a_{j,j}\,x_{l,j} + a_{j,j+1}\,x_{l-1,j+1} + \cdots + a_{j,n}\,x_{l-1,n} = b_j$$

for $j = 1,\dots,n$. The corresponding iterative method is defined by the rule:

Algorithm 4.3.2 (Gauss-Seidel method)

```
for $j = 1, \dots, n$
    $x_{l,j} \leftarrow \frac{1}{a_{j,j}} \Big( b_j - \underbrace{\sum_{k<j} a_{j,k}\, x_{l,k}}_{\text{updated } x\text{'s}} - \underbrace{\sum_{k>j} a_{j,k}\, x_{l-1,k}}_{\text{older } x\text{'s}} \Big)$
```

In matrix notation, we have that $x_{l+1} = R_{\mathrm{GS}}\, x_l + c_{\mathrm{GS}}$ with

$$\begin{aligned} R_{\mathrm{GS}} &= (D - \widetilde{L})^{-1}\widetilde{U} = (\mathbb{1}_n - L)^{-1}U, \\ c_{\mathrm{GS}} &= (D - \widetilde{L})^{-1}b = (\mathbb{1}_n - L)^{-1}D^{-1}b. \end{aligned} \tag{4.7}$$

Successive overrelaxation for a parameter $\omega \in \mathbb{R}$ is a weighted average of the vectors x_{l+1} and x_l from the Gauss-Seidel method: it is defined by

$$x_l^{\mathrm{SOR}(\omega)} = (1-\omega)\, x_{l-1}^{\mathrm{GS}} + \omega x_l^{\mathrm{GS}} \qquad \text{for } l \geq 1.$$

This yields the following iterative scheme:

Algorithm 4.3.3 (SOR(ω))

```
for $j = 1, \dots, n$
    $x_{l,j} \leftarrow (1-\omega) x_{l-1,j} + \frac{\omega}{a_{j,j}} \big( b_j - \sum_{k<j} a_{j,k}\, x_{l,k} - \sum_{k>j} a_{j,k}\, x_{l-1,k} \big)$
```

In matrix notation, we have that $x_l = R_{\mathrm{SOR}(\omega)}\, x_{l-1} + c_{\mathrm{SOR}(\omega)}\, b$ with

$$\begin{aligned} R_{\mathrm{SOR}(\omega)} &= (D - \omega\, \widetilde{L})^{-1}((1-\omega)\, D + \omega\, \widetilde{U}) = (\mathbb{1}_n - \omega\, L)^{-1}((1-\omega)\, \mathbb{1}_n + \omega\, U), \\ c_{\mathrm{SOR}(\omega)} &= \omega\, (D - \omega\, \widetilde{L})^{-1}\, b = \omega\, (\mathbb{1}_n - \omega\, L)^{-1}\, D^{-1} b. \end{aligned} \tag{4.8}$$

We distinguish three cases, depending on the value of the relaxation parameter ω: $\omega = 1$ is equivalent to the Gauss-Seidel method, $\omega < 1$ it is called *underrelaxation*, and $\omega > 1$ is called *overrelaxation*. A somewhat superficial motivation for overrelaxation is that if going from x_{l-1} to x_l is a good direction towards the solution, then moving $\omega > 1$ times as far should be even better.

Example 4.3.1. Let

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \quad \text{and} \quad b = (1, 0).$$

The solution of the linear equation $A\, x = b$ is the 2-vector $x = (\frac{3}{2}, \frac{-1}{3})$.

The decompositions $A = D - \widetilde{L} - \widetilde{U} = D\, (\mathbb{1}_2 - L - U)$ are respectively given by

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ -1 & 0 \end{bmatrix} - \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ \frac{-1}{2} & 0 \end{bmatrix} - \begin{bmatrix} 0 & \frac{-1}{2} \\ 0 & 0 \end{bmatrix} \right)$$

Hence

$$R_{\mathrm{J}} = L + U = \begin{bmatrix} 0 & \frac{-1}{2} \\ \frac{-1}{2} & 0 \end{bmatrix} \quad \text{and} \quad c_{\mathrm{J}} = D^{-1} b = \begin{bmatrix} \frac{1}{2} \\ 0 \end{bmatrix}$$

and so the corresponding Jacobi iteration writes down as

$$\begin{bmatrix} x_{l,1} \\ x_{l,2} \end{bmatrix} = \begin{bmatrix} 0 & \frac{-1}{2} \\ \frac{-1}{2} & 0 \end{bmatrix} \begin{bmatrix} x_{l-1,1} \\ x_{l-1,2} \end{bmatrix} + \begin{bmatrix} \frac{1}{2} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{-x_{l-1,2}}{2} + \frac{1}{2} \\ \frac{x_{l-1,1}}{2} \end{bmatrix}.$$

The corresponding characteristic polynomial is

$$\chi_{R_{\mathrm{J}}} = \det \begin{bmatrix} -t & \frac{-1}{2} \\ \frac{-1}{2} & -t \end{bmatrix} = t^2 - \frac{1}{4}.$$

Its spectrum is $\Lambda(R_{\mathrm{J}}) = \{t \mid \chi_{R_{\mathrm{J}}}(t) = 0\} = \big\{ \pm \frac{1}{2}\big\}$ and so its spectral radius is

$$\rho(R_{\mathrm{J}}) = \frac{1}{2},$$

showing that the method converges to the solution x for every choice of initial vector.

We also have that

$$R_{\mathrm{GS}} = (\mathbb{1}_2 - L)^{-1} U = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 & \frac{-1}{2} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & \frac{-1}{2} \\ 0 & \frac{1}{4} \end{bmatrix},$$

$$c_{\mathrm{GS}} = (\mathbb{1}_2 - L)^{-1} D^{-1} b = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{bmatrix}^{-1} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{-1}{4} \end{bmatrix}$$

and so the corresponding Gauss-Seidel iteration writes down as

$$\begin{bmatrix} x_{l,1} \\ x_{l,2} \end{bmatrix} = \begin{bmatrix} 0 & \frac{-1}{2} \\ 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} x_{l-1,1} \\ x_{l-1,2} \end{bmatrix} + \begin{bmatrix} \frac{1}{2} \\ \frac{-1}{4} \end{bmatrix} = \begin{bmatrix} \frac{-x_{l,2}}{2} + \frac{1}{2} \\ \frac{x_{l,2}}{4} - \frac{1}{4} \end{bmatrix}.$$

We have that

$$\chi_{R_{\mathrm{GS}}} = \det \begin{bmatrix} -t & \frac{-1}{2} \\ 0 & -t + \frac{1}{4} \end{bmatrix} = t^2 - \frac{1}{4}\, t.$$

Hence $\Lambda(R_{\mathrm{GS}}) = \big\{0, \frac{1}{4}\big\}$ and so $\rho(R_{\mathrm{GS}}) = \frac{1}{4}$. We have that

$$\rho(R_{\mathrm{GS}}) = \rho(R_{\mathrm{J}})^2$$

and so in this example, the Gauss-Seidel method converges with the *double of the speed* of the Jacobi method.

Now for $\omega \in \mathbb{R}$ we have that

$$\begin{aligned} R_{\mathrm{SOR}(\omega)} &= (\mathbb{1}_2 - \omega\, L)^{-1}((1-\omega)\, \mathbb{1}_2 + \omega\, U) \\ &= \begin{bmatrix} 1 & 0 \\ \omega/2 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1-\omega & -\omega/2 \\ 0 & 1-\omega \end{bmatrix} = \begin{bmatrix} 1-\omega & -\frac{\omega}{2} \\ \frac{\omega^2}{2} - \frac{\omega}{2} & \frac{\omega^2}{4} - \omega + 1 \end{bmatrix} \end{aligned}$$

and

$$c_{\mathrm{SOR}(\omega)} = \omega\, (\mathbb{1}_2 - \omega\, L)^{-1} D^{-1} b = \omega \begin{bmatrix} 1 & 0 \\ \frac{\omega}{2} & 1 \end{bmatrix}^{-1} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{\omega}{2} \\ \frac{-\omega^2}{4} \end{bmatrix},$$

and so the corresponding SOR(ω) iteration writes down as

$$\begin{aligned} \begin{bmatrix} x_{l,1} \\ x_{l,2} \end{bmatrix} &= \begin{bmatrix} 1-\omega & -\frac{\omega}{2} \\ \frac{\omega^2}{2} - \frac{\omega}{2} & \frac{\omega^2}{4} - \omega + 1 \end{bmatrix} \begin{bmatrix} x_{l-1,1} \\ x_{l-1,2} \end{bmatrix} + \begin{bmatrix} \frac{\omega}{2} \\ \frac{-\omega^2}{4} \end{bmatrix} \\ &= \begin{bmatrix} (1-\omega)\, x_{l-1,1} - \frac{\omega}{2}\, x_{l-1,2} + \frac{\omega}{2} \\ (\frac{\omega^2}{2} - \frac{\omega}{2})\, x_{l-1,1} + (\frac{\omega^2}{4} - \omega + 1)\, x_{l-1,2} - \frac{\omega^2}{4} \end{bmatrix}. \end{aligned}$$

We have that

$$\chi_{R_{\mathrm{SOR}(\omega)}} = \det \begin{bmatrix} 1-\omega-t & -\frac{\omega}{2} \\ \frac{\omega^2}{2} - \frac{\omega}{2} & \frac{\omega^2}{4} - \omega + 1 - t \end{bmatrix} = t^2 + \Big(\frac{-\omega^2}{4} + 2\,\omega - 2\Big)\, t + (\omega^2 - 2\,\omega + 1).$$

The spectral radius $\rho(R_{\mathrm{SOR}(\omega)})$ is the maximum of the absolute value of the zeros of this polynomial, and its graph is shown for $\omega \in [0,2]$ is shown in Figure 4.3.1.

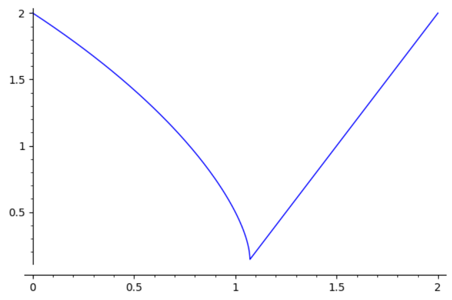


FIGURE 4.3.1. The spectral radius of SOR(ω)

The optimal value of the relaxation parameter is $\omega_{\mathrm{opt}} = 1.0717$ up to 4 decimal digits, and the corresponding spectral radius is $\rho(R_{\mathrm{SOR}(1.0717)}) = 0.1535$.

4.4. Convergence of the basic iterative schemes

We will give some criteria that, in some specific situation, guarantee the convergence of the basic iterative schemes. Since the spectral radius is difficult to compute, these convergence criteria can be useful in practice.

As in §4.3, we assume that all the diagonal entries of A are nonzero and we consider the decomposition

$$A = D - \widetilde{L} - \widetilde{U}$$

with D diagonal, $\widetilde{L}$ strictly lower triangular and $\widetilde{U}$ strictly upper triangular.

The matrix A is *(column) diagonally dominant* if the absolute value of each diagonal entry is greater than the sum of the absolute values of the rest of the entries in its column, that is

$$|a_{j,j}| > \sum_{i\neq j} |a_{i,j}|, \quad j = 1, \dots, n.$$

If A is diagonally dominant then both the Jacobi and the Gauss-Seidel iterations will converge for every choice of initial vector x_0. To see this, let

$$R_{\mathrm{J}} = D^{-1}(\widetilde{L} + \widetilde{U}) \quad \text{and} \quad R_{\mathrm{GS}} = (D - \widetilde{L})^{-1}\widetilde{U}$$

be the corresponding iteration matrices as in (4.6) and (4.7). For each $\lambda \in \Lambda(R_{\mathrm{J}})$ let x be a corresponding eigenvector for it, and k the index of its largest component. Up to a normalization, we can suppose that

$$x_k = 1 \quad \text{and} \quad |x_j| \leq 1 \text{ for all } j.$$

Then the equality $R_{\mathrm{J}}\, x = \lambda\, x$ implies that $D^{-1}(\widetilde{L} + \widetilde{U})\, x = \lambda\, x$. The k-th entry of this later equality is

$$-\sum_{j\neq k} \frac{a_{k,j}}{a_{k,k}}\, x_j = \lambda,$$

which implies that

$$|\lambda| \leq \sum_{j\neq k} \frac{|a_{k,j}|}{|a_{k,k}|}\, |x_j| < 1,$$

because $|x_j| \leq 1$ for all j and A is diagonally dominant. Since this inequality holds for every eigenvalue of R_{J}, we deduce that $\rho(R_{\mathrm{J}}) < 1$, which gives the result for the Jacobi method.

Similarly, for each $\lambda \in \Lambda(R_{\mathrm{GS}})$ let x a corresponding eigenvector, and k the index of the largest component of x. Again, we assume that $x_k = 1$ and $|x_j| \le 1$ for all j. From the equality $R_{\mathrm{GS}}\, x = \lambda\, x$ we deduce that $\widetilde{U}\, x = \lambda\, (D - \widetilde{L})\, x$ and so

$$\sum_{j<k} a_{k,j}\, x_j = \lambda \Big(a_{k,k} + \sum_{j>k} a_{k,j}\, x_j \Big).$$

This implies that

$$|\lambda| \le \frac{\sum_{j<k} |a_{k,j}|\, |x_j|}{|a_{k,k}| - \sum_{j>k} |a_{k,j}|\, |x_j|} \le \frac{\sum_{j<k} |a_{k,j}|}{|a_{k,k}| - \sum_{j>k} |a_{,j}|} < 1,$$

again because $|x_j| \le 1$ for all j and A is diagonally dominant, which implies that $|a_{k,k}| - \sum_{j>k} |a_{k,j}| > \sum_{j<k} |a_{k,j}|$. Since this inequality holds for every eigenvalue of R_{GS}, we deduce the convergence of the Gauss-Seidel method in this case.

For the successive overrelaxation method, the condition on the relaxation parameter

$$0 < \omega < 2 \tag{4.9}$$

is necessary for the convergence of the method. Indeed, by (4.8) the corresponding iteration matrix is

$$R_{\mathrm{SOR}(\omega)} = (\mathbb{1}_n - \omega\, L)^{-1}((1-\omega)\, \mathbb{1}_n + \omega\, U)$$

and so its determinant can be computed as

$$\begin{aligned} \det(R_{\mathrm{SOR}(\omega)}) &= \det((\mathbb{1}_n - \omega\, L)^{-1}((1-\omega)\, \mathbb{1}_n + \omega\, U)) \\ &= \det(\mathbb{1}_n - \omega\, L)^{-1} \det((1-\omega)\, \mathbb{1}_n + \omega\, U) = (1-\omega)^n. \end{aligned} \tag{4.10}$$

The determinant of a matrix coincides with the product of its eigenvalues, repeated with the corresponding multiplicity, and so

$$|\det(R_{\mathrm{SOR}(\omega)})| \le \rho(R_{\mathrm{SOR}(\omega)})^n. \tag{4.11}$$

The inequalities (4.10) and (4.11) readily imply the lower bound

$$\rho(R_{\mathrm{SOR}(\omega)}) \ge |1-\omega|,$$

and so the condition for the spectral radius in (4.2) can only be satisfied whenever the condition (4.9) holds.

On the other hand, if A is a symmetric and positive definite matrix with real coefficients then SOR(ω) converges for every $0 < \omega < 2$ [**Dem97**, Theorem 6.4]. In particular, the Gauss-Seidel method ($\omega = 1$) always converges in this situation.

Part 2

The least squares problem

The *least squares (LS) problem* consists in, given an $m \times n$ matrix A and an m-vector b over the reals, finding an n-vector x_{LS} over the reals that minimizes the quantity

$$\|A\,x - b\|_2. \tag{4.12}$$

In other terms, the LS problem seeks the linear combination of the columns of A

$$A\,x = \sum_{j=1}^{n} x_j \operatorname{col}_j(A) \in \mathbb{R}^m$$

that best approaches b with respect to the 2-norm.

In data analysis, the pair (A, b) arises from measurements on a series of samples. The data obtained from the samples is arranged in the rows of A, whereas the columns of A correspond to the different attributes or variables measured and b represents a further variable that purportedly depends on the other ones. Since the n-vector x_{LS} gives the linear combination of the variables in A that best approaches b, it can be used as a predictor to extrapolate the measurements on these samples to other cases.

Example 4.4.1. Suppose that we are doing medical research on the effect of a drug on the level of a certain toxin in the blood. To this aim, we extract the following data from a set of m patients:

age, weight, initial toxin level, given amount of drug, final toxin level.

Then we consider the LS problem for the $m \times 4$ matrix A whose (i, j)-entry is the measurement for the i-th patient of the j-th variable, and together with the m-vector b whose i-th entry is the measurement for the i-th patient of the last variable.

For this dataset, the solution x_{LS} gives the best linear approximation for final toxin level in terms of the other variables. For a further patient with age, weight, initial toxin level and amount of drug stored in a vector $p \in \mathbb{R}^4$, his/her final toxin level q would be predicted as

$$q = p^T\, x_{\mathrm{LS}}.$$

A similar situation arises when fitting a 2-dimensional plot with polynomials of a certain degree.

Example 4.4.2. Suppose that we have a sequence of points in the plane

$$(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_m, y_m)$$

and we want to find the polynomial of degree $\leq d$ that best fits the y_i's as a function of the x_i's.

This problem boils down to computing a $d + 1$-vector α such that the polynomial

$$p_\alpha(x) = \sum_{j=0}^{d} \alpha_{j-1}\, x^j$$

minimizes the *residuals* $p(x_i) - y_i$, $i = 1, \ldots, m$. Minimizing the 2-norm of the vectors of these residuals is an LS problem for the data

$$A = \begin{bmatrix} 1 & x_1 & \cdots & x_1^d \\ \vdots & \vdots & & \vdots \\ 1 & x_m & \cdots & x_m^d \end{bmatrix} \in \mathbb{R}^{m \times (d+1)} \quad \text{and} \quad b = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \in \mathbb{R}^m,$$

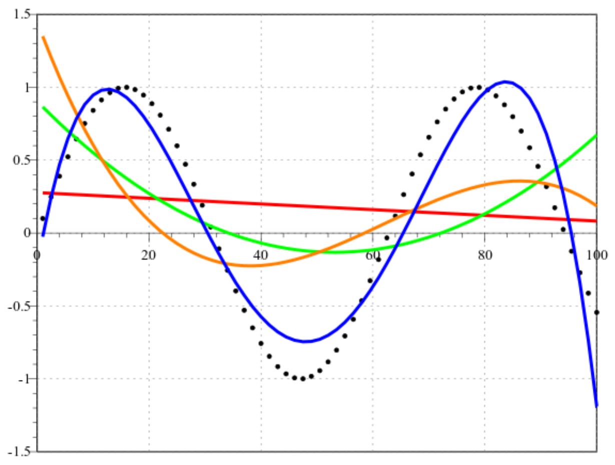


FIGURE 4.4.1. Curve fitting by polynomials of degree ≤ 4

since

$$\|A\,\alpha - b\|_2 = \Big(\sum_{i=1}^{m} (p_\alpha(x_i) - y_i)^2\Big)^{1/2}.$$

If $m = n$ and A is nonsingular then $x_{\rm LS}$ is the unique solution of $A\,x = b$. However, if $m > n$ then typically this linear equation has no solution, and the minimum in (4.12) is positive. It should be clear from the previous applications that this is our case of interest and as a matter of fact, in these application the number of rows m is usually much larger than the number of columns n.

The choice of the 2-norm for measuring the residual $A\,x - b$ is due to mathematical reasons: it places the problem within the realm of Euclidean geometry, which allows us to use notions and tools like angles and inner products, orthogonal projections, orthonormal basis and so on. However, other norms might be equally valid and interesting from the point of view of applications. In particular, the minimization of the residual vector with respect to the 1-norm is also very relevant for applications like compressed sensing for sparse signal reconstruction.

We will study three methods for solving the LS problem:

- normal equations,
- QR factorization,
- singular value decomposition.

Each of them has its own interest and range of applicability, indicated by its time and space complexities, and its numerical stability. We will concentrate on the full rank case, but we will also explain how to treat the rank deficient case.

CHAPTER 5

The QR factorization

5.1. Normal equations

Let $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, and consider the LS problem consisting in finding a vector $x_{\rm LS} \in \mathbb{R}^n$ minimizing the 2-norm of the residual, that is

$$\|A\,x - b\|_2.$$

For the moment we concentrate on the *full rank case*, which occurs when $m \geq n$ and $\operatorname{rank}(A) = n$.

Figure 5.1 illustrates this problem and its solution. For $x \in \mathbb{R}^n$ the point $p = Ax$ lies in the image of the linear map L_A, and $e = b - p$ is the residual. The norm of this vector is minimal when p is the orthogonal projection of b into $\operatorname{Im}(L_A)$, in which case e is orthogonal to this linear subspace.

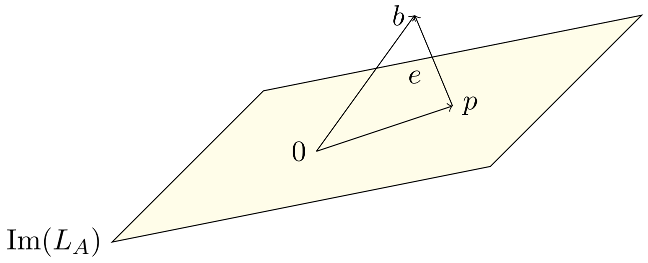


Figure 5.1.1. The least squares problem

Hence for $x \in \mathbb{R}^n$ we have that

$$0 = \langle Ax, e\rangle = (A\,x)^T e = (A\,x)^T(b - A\,x_{\rm LS}) = x^T A^T (b - A\,x_{\rm LS}).$$

Since this holds for all x, it implies that

$$A^T A\,x_{\rm LS} = A^T b. \tag{5.1}$$

The $n \times n$ matrix $A^T A$ is symmetric and positive definite *(check it!)*. In particular, it is nonsingular and so the solution of the *normal equations* (5.1) is unique. By Pythagoras' theorem, the norm of the residual is

$$\|e\|_2 = (\|b\|_2^2 - \|A\,x_{\rm LS}\|_2^2)^{1/2}. \tag{5.2}$$

Since $A^T A$ is an SPD matrix, we can use its Cholesky factorization to solve the normal equations (Algorithm 5.1.1). This procedure is convenient since it relies on standard algorithms.

Sinnce the matrix $C = A^T A$ is symmetric, we only need to compute its lower triangular part. Taking this into account, the complexity of this procedure is

$$\Big(m + \frac{n}{3}\Big)\, n^2 + O(n^2) \text{ flops.}$$

Algorithm 5.1.1 (Normal equations)

```
Compute $C \leftarrow A^T A$,
Compute $d \leftarrow A^T b$,
Compute the Cholesky factorization $C = G\,G^T$,
Solve $G\,y = d$ and $G^T x_{\mathrm{LS}} = y$.
```

For $m \gg n$ the complexity $m\,n^2$ of computing the lower triangular part of the product $C = A^T A$ dominates. The procedure is reliable when A is far from rank deficient, but unstable otherwise.

5.2. The QR factorization

Let $a_j = \mathrm{col}_j(A) \in \mathbb{R}^m$, $j = 1, \dots, n$, be the columns of $m \times n$ matrix A. If these vectors happen to be orthonormal, that is if

$$\langle a_i, a_k \rangle = \begin{cases} 1 & \text{if } i = k, \\ 0 & \text{otherwise,} \end{cases}$$

then $A^T A = \mathbb{1}_n$ and so by (5.1) the solution of the LS problem is easily computed as $x_{\mathrm{LS}} = A^T b$.

In general, the columns of A are linearly independent because this matrix has full rank, but typically they are not orthogonal. However, these vectors can be orthogonalized with the *Gram-Schmidt orthogonalization (GSO) process.* This algorithm produces an orthonormal family of m-vectors q_j, $j = 1, \dots, n$, such that

$$\mathrm{span}(q_1, \dots, q_j) = \mathrm{span}(a_1, \dots, a_j) \quad \text{for } j = 1, \dots, n. \tag{5.3}$$

Thus these vectors give orthonormal bases for the linear subspaces incrementally generated by the columns of A.

It runs as follows:

$\boxed{\text{First step}}$

$$r_{1,1} \leftarrow \|a_1\|_2, \quad q_1 \leftarrow \frac{a_1}{r_{1,1}},$$

so that this latter is a unit vector in the direction of the first column of A. Next:

$\boxed{\text{Second step}}$

$$r_{1,2} \leftarrow \langle q_1, a_2 \rangle, \quad \widetilde{a}_2 \leftarrow a_2 - r_{1,2}\,q_1, \quad r_{2,2} \leftarrow \|\widetilde{a}_2\|_2, \quad q_2 \leftarrow \frac{\widetilde{a}_2}{r_{2,2}}.$$

Indeed, $a_2 - r_{1,2}\,q_1$ is orthogonal to q_1 *(prove it!)*. Hence q_2 is a unit vector that is orthogonal to q_1, and so these two vectors form an orthonormal basis of $\mathrm{span}(a_1, a_2)$ (Figure 5.2).

When $n > 2$ this process continues similarly:

$\boxed{\text{Third step}}$

$$r_{1,3} \leftarrow \langle q_1, a_3 \rangle,\ r_{2,3} \leftarrow \langle q_2, a_3 \rangle,\ \widetilde{a}_3 \leftarrow a_3 - r_{1,3}\,q_1 - r_{2,3}\,q_2,\ r_{3,3} \leftarrow \|\widetilde{a}_3\|_2,\ q_3 \leftarrow \frac{\widetilde{a}_3}{r_{3,3}}$$

and so on.

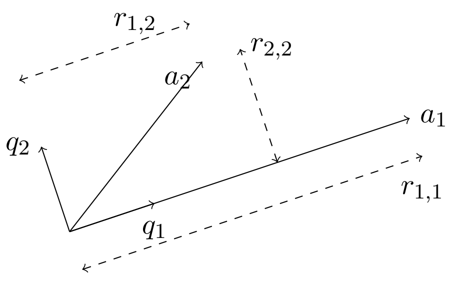


FIGURE 5.2.1. The Gram-Schmidt orthogonalization process

We can express the columns of A in terms of this orthonormal family:

$$\begin{aligned} a_1 &= r_{1,1}\, q_1, \\ a_2 &= r_{1,2}\, q_1 + r_{2,2}\, q_2, \\ a_3 &= r_{1,3}\, q_1 + r_{2,3}\, q_2 + r_{3,3}\, q_3, \ \ldots \end{aligned}$$

This can be written in matrix form as

$$\begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix} = \begin{bmatrix} q_1 & \cdots & q_n \end{bmatrix} \begin{bmatrix} r_{1,1} & \cdots & \cdots & r_{1,n} \\ 0 & r_{2,2} & \cdots & r_{2,n} \\ \vdots & & \ddots & \vdots \\ 0 & \ldots & 0 & r_{n,n} \end{bmatrix} \tag{5.4}$$

with $r_{i,j} = \langle q_i, a_j \rangle$ for $1 \le i \le j \le n$, which leads us directly to the all-important QR factorization.

Recall that a matrix $Q \in \mathbb{R}^{m\times n}$ is *orthogonal* if

$$Q^T Q = \mathbb{1}_n$$

or equivalently, if its columns form an orthonormal family of vectors of $\mathbb{R}^m$. Then the *(thin) QR factorization* of our full rank $m \times n$ matrix A is

$$A = Q\, R \tag{5.5}$$

with $Q \in \mathbb{R}^{m\times n}$ orthogonal and $R \in \mathbb{R}^{n\times n}$ upper triangular with positive diagonal entries. The *(full) QR factorization* of A is a factorization

$$A = \widetilde{Q}\, \widetilde{R} \tag{5.6}$$

with $\widetilde{Q} \in \mathbb{R}^{m\times m}$ orthogonal and $\widetilde{R} \in \mathbb{R}^{m\times n}$ of the form $\widetilde{R} = \begin{bmatrix} R \\ \mathbb{0} \end{bmatrix} \begin{matrix} {\scriptstyle n} \\ {\scriptstyle m-n} \end{matrix}$ where $R \in \mathbb{R}^{n\times n}$ is an upper triangular matrix having positive diagonal entries.

The thin QR factorization of A coincides with that in (5.4), and so the GSO algorithm both shows its existence and gives a method to compute it. Moreover, this factorization is unique because it is equivalent to the conditions in (5.3).

The full QR factorization of A can be derived from the thin by extending Q to an $m \times m$ orthogonal matrix or equivalently, by extending the orthonormal family of vectors q_j, $j = 1, \ldots, n$, to an orthonormal basis of $\mathbb{R}^m$. Computationally, this can be done extending A to a nonsingular $m \times m$ matrix $\widetilde{A}$ by adding $m - n$ further columns that are linearly independent from those of A, and applying the GSO algorithm to $\widetilde{A}$.

Example 5.2.1. Set

$$A = \begin{bmatrix} 1 & -3 \\ 0 & 2 \\ -1 & -1 \end{bmatrix} \in \mathbb{R}^{3\times 2}.$$

GSO applied this matrix gives that

$$q_1 = \frac{a_1}{\|a_1\|_2} = (0.71, 0, -0.71)$$

whereas $\widetilde{a}_2 = a_2 - \langle a_2, q_1\rangle\, q_1 = (-2, 2, -2)$ and so

$$q_2 = \frac{\widetilde{a}_2}{\|\widetilde{a}_2\|_2} = (-0.58, 0.58, -0.58).$$

Thus q_1 and q_1, q_2 are orthonormal bases of $\mathrm{span}(a_1)$ and of $\mathrm{span}(a_1, a_2)$, respectively. We also have that

$$\langle q_1, a_1\rangle = 1.41, \quad \langle q_1, a_2\rangle = -1.41, \quad \langle q_2, a_2\rangle = 3.49$$

and so the (thin) QR factorization of A is

$$A = Q\,R = \begin{bmatrix} 0.71 & -0.58 \\ 0 & 0.58 \\ -0.71 & -0.58 \end{bmatrix} \begin{bmatrix} 1.41 & -1.41 \\ 0 & 3.49 \end{bmatrix}.$$

Once the QR factorization is computed, we can easily solve the normal equations (5.1) and so the LS problem:

$$x_{\mathrm{LS}} = (A^T A)^{-1} A^T b = ((QR)^T Q\,R)^{-1} (Q\,R)^T b = (R^T R)^{-1} R^T Q\, b = R^{-1} Q^T\, b.$$

The GSO algorithm is not numerically stable when the columns of A are nearly linearly dependent or equivalently, when A is close to rank deficient.

5.3. Householder reflections

The key to a numerically stable algorithm for computing the QR factorization is to apply a sequence of basic elementary orthogonal transformations that approach the matrix to the desired upper triangular form. These basic orthogonal transformations of $\mathbb{R}^m$ can be:

- symmetries with respect to a linear subspace,
- rotations around a linear subspace of codimension 2.

In this section we will consider *Householder reflections*, which are the symmetries of $\mathbb{R}^m$ with respect to a hyperplane. Given a unit vector $u \in \mathbb{R}^m$, its the reflection with respect to the orthogonal hyperplane $u^\perp$ is given by the $m \times m$ matrix

$$P_u = \mathbb{1}_m - 2\,u\,u^T \in \mathbb{R}^{m\times m}$$

This matrix is symmetric ($P^T = P$) and orthogonal ($P^T P = \mathbb{1}_m$) *(prove it!)*. The hyperplane $u^\perp$ acts like a mirror, since this linear map sends each vector to its opposite with respect to this hyperplane.

We claim that for any given nonzero m-vector there is a reflection that zeroes all but the first entry. That is, given $y \in \mathbb{R}^m \setminus \{0\}$ we want to find a unit vector u such

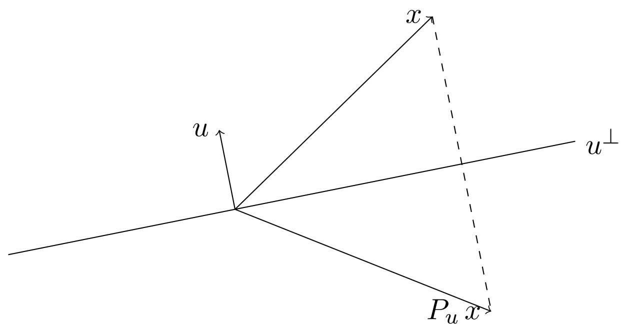


FIGURE 5.3.1. Reflection on $\mathbb{R}^2$ with respect to the line $u^{\perp}$

that

$$P_u\,y = \begin{bmatrix} c \\ 0 \\ \vdots \\ 0 \end{bmatrix} = c\,e_1 \tag{5.7}$$

where $e_1 = (1, 0, \dots, 0)$ is the first vector in the standard basis of $\mathbb{R}^m$. To compute this unit vector, note that

$$P_u\,y = (\mathbb{1}_m - 2\,u\,u^T)\,y = y - 2\,\langle u, y\rangle\,u = c\,e_1. \tag{5.8}$$

Hence $2\,\langle u, y\rangle\,u = y - c\,e_1$. Since P_u is orthogonal, necessarily $|c| = \|P_u\,y\|_2 = \|y\|_2$ and so $c = \pm\|y\|_2$. To avoid an undesired cancellation, we choose $c = -\operatorname{sign}(y_1)\,\|y\|_2$, which implies that u is a scalar multiple of

$$\widetilde{u} := y + \operatorname{sign}(y_1)\|y\|_2\,e_1 = \begin{bmatrix} y_1 + \operatorname{sign}(y_1)\,\|y\|_2 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}.$$

Thus we set

$$u = \operatorname{House}(y) = \frac{\widetilde{u}}{\|\widetilde{u}\|_2}$$

which satisfies (5.7) for $c = -\operatorname{sign}(y_1)\,\|y\|_2$.

The principles for computing the QR factorization via Householder reflections can be already illustrated in the 4×3 case. Let then

$$A = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}$$

where each symbol $\times$ indicates an unspecified scalar.

(1) Choose $P_1 \in \mathbb{R}^{4\times 4}$ such that all entries of $P_1\,[a_{1,1}\,a_{2,1}\,a_{3,1}\,a_{4,1}]^T$ are zero except the first one, and set

$$A^{(1)} \leftarrow P_1\,A = \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix}.$$

(2) Choose $P_2' \in \mathbb{R}^{3\times 3}$ such that the second and third entries of the product $P_2'\,[a_{2,2}^{(1)}\,a_{3,2}^{(1)}\,a_{4,2}^{(1)}]^T$ are zero, set $P_2 = \begin{bmatrix} 1 & \mathbb{0} \\ \mathbb{0} & P_2' \end{bmatrix} \in \mathbb{R}^{4\times 4}$ and

$$A^{(2)} \leftarrow P_2\,A^{(1)} = \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{bmatrix}.$$

(3) Choose $P_3' \in \mathbb{R}^{2\times 2}$ such that the second entry of $P_3'\,[a_{3,3}^{(2)}\,a_{3,4}^{(2)}]^T$ is zero, set $P = \begin{bmatrix} \mathbb{1}_2 & \mathbb{0} \\ \mathbb{0} & P_3' \end{bmatrix} \in \mathbb{R}^{4\times 4}$ and

$$A^{(3)} \leftarrow P_3\,A^{(2)} = \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \end{bmatrix}.$$

The product $A^{(3)} := P_3\,P_2\,P_1\,A$ is upper triangular, and we can obtain the full QR factorization

$$A = \widetilde{Q}\,\widetilde{R}$$

by setting $\widetilde{Q} = P_1^T P_2^T P_3^T \Lambda$ and $\widetilde{R} = \Lambda A^{(3)}$ with $\Lambda = \operatorname{diag}(\pm 1, \dots, \pm 1) \in \mathbb{R}^{4\times 4}$ that is chosen so that the diagonal entries of $\widetilde{R}$ as positive, as required.

The thin QR factorization $A = Q\,R$ is obtained by suitably cutting these matrices, taking Q as the left 4×3 submatrix of $\widetilde{Q}$ and R as the upper 3×3 submatrix of $\widetilde{R}$.

Example 5.3.1. Consider the 3×2 matrix

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

Set $\widetilde{u}_1 = \begin{bmatrix} 1 + 2^{1/2} \\ 0 \\ -1 \end{bmatrix}$ so that $u_1 = \operatorname{House}\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} = \frac{\widetilde{u}_1}{\|\widetilde{u}_1\|_2} = \begin{bmatrix} 0.92 \\ 0 \\ -0.38 \end{bmatrix}$. Then

$$P_1 = \mathbb{1}_3 - 2\,u_1\,u_1^T = \begin{bmatrix} -0.71 & 0 & 0.71 \\ 0 & 1 & 0 \\ 0.71 & 0 & 0.71 \end{bmatrix} \quad \text{and} \quad A^{(1)} = P_1\,A = \begin{bmatrix} -1.41 & 1.41 \\ 0 & 2 \\ 0 & -2.83 \end{bmatrix}.$$

Set then $\widetilde{u}_2 = \begin{bmatrix} 2+(2^2+(-2.83)^2)^{1/2} \\ -2.83 \end{bmatrix}$ so that $u_2 = \text{House}\begin{bmatrix} 2 \\ -2.83 \end{bmatrix} = \frac{\widetilde{u}_2}{\|\widetilde{u}_2\|_2} = \begin{bmatrix} 0.89 \\ -0.46 \end{bmatrix}$. Then

$$P_2 = \begin{bmatrix} 1 & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_2 - 2\,u_2\,u_2^T \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -0.58 & 0.82 \\ 0 & 0.82 & 0.58 \end{bmatrix} \text{ and } A^{(2)} = P_2\,A^{(1)} = \begin{bmatrix} -1.41 & 1.41 \\ 0 & -3.49 \\ 0 & 0 \end{bmatrix}.$$

Then setting

$$\widetilde{Q} = -P_1^T P_2^T = \begin{bmatrix} 0.71 & -0.58 & -0.41 \\ 0 & 0.58 & -0.82 \\ -0.71 & -0.58 & -0.41 \end{bmatrix} \quad \text{and} \quad \widetilde{R} = -A^{(2)} = \begin{bmatrix} 1.41 & -1.41 \\ 0 & 3.49 \\ 0 & 0 \end{bmatrix}$$

we obtain the full QR factorization $A = \widetilde{Q}\,\widetilde{R}$. The thin QR factorization can be obtained from this by taking the appropriate submatrices, as explained before.

As in other previous algorithms, the entries of A are no longer needed after each computation, and so this matrix can be safely overwritten. Here is the algorithm for the QR factorization using Householder reflections, using overwritting and the Matlab notation.

Algorithm 5.3.1 (QR factorization *via* Householder reflections)

```
for $i = 1$ to $\min(m-1, n)$
    $u_i \leftarrow \text{House}(A(i:m, i))$
    $P_i' \leftarrow \mathbb{1}_{m-i+1} - 2\,u_i\,u_i^T$
    $A_i(i:m, i:n) \leftarrow P_i'\,A(i:m, i:n)$
```

There are some further details that can speed up its implementation. We do not need to form the $(m-i+1)\times(m-i+1)$ matrix P_i' but just need to compute the multiplication

$$(\mathbb{1}_{m-i+1} - 2\,u_i\,u_i^T)\,A(i:m, i:n) = A(i:m, i:n) - 2\,u_i\,(u_i^T A(i:m, i:n)), \tag{5.9}$$

which costs less.

Moreover, we do not need to form the product $\prod_{j=1}^{\min(m-1,n)} P_j$, and instead we can code it with the sequence of Householder vectors u_j, $j = 1, \dots, \min(m-1, n)$. These can be stored in the lower part of the matrix plus an extra array of length $\min(m-1, n)$ for their first entries.

Recall also that to solve the LS problem $\min_x \|A\,x - b\|_2$ we need to compute the product

$$Q^T b = P_n \cdots P_1 b.$$

This can be done without explicitly forming these orthogonal matrices applying iteratively the formula in (5.9).

The complexity of this algorithm is

$$2\,n^2\,m - \frac{2}{3}\,n^2 \text{ flops}$$

if the product $Q = P_1^T \cdots P_n^T$ is not required. When $m \gg n$ this is about twice the complexity of solving the normal equations via Cholesky's algorithm, but is more numerically stable.

5.4. Givens rotations

A variant of the previous approach for computing the QR factorization can be obtained by considering rotations instead of symmetries. A rotation on the plane of angle θ is the linear map $R(\theta)\colon \mathbb{R}^2 \to \mathbb{R}^2$ given by the orthogonal matrix

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}.$$

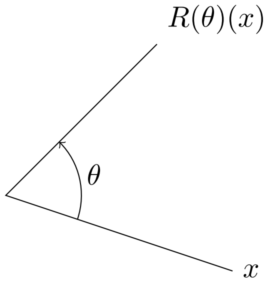


Figure 5.4.1. Rotation of angle θ

A *Givens rotation* on $\mathbb{R}^m$ is a linear map that rotates the vectors in one of the standard planes of $\mathbb{R}^m$ with a certain angle, and that fixes the vectors in the standard complementary subspace. Precisely, the Givens rotation associated to a pair of indexes $1 \leq i < j \leq m$ and an angle $\theta \in \mathbb{R}$ is the linear map on $\mathbb{R}^m$ corresponding to the matrix

$$R(i,j,\theta) = \begin{array}{c} \\ \\ i \\ \\ j \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} & i & & j & \end{array} \\ \begin{bmatrix} \mathbb{1}_{i-1} & & & & \\ & \cos(\theta) & & -\sin(\theta) & \\ & & \mathbb{1}_{j-i-1} & & \\ & \sin(\theta) & & \cos(\theta) & \\ & & & & \mathbb{1}_{n-j-1} \end{bmatrix} \end{array}$$

Given a vector $y \in \mathbb{R}^m$ and indexes $i > j$, we can apply a Givens rotation to zero the j-th entry of y by choosing θ such that

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} y_i \\ y_j \end{bmatrix} = \begin{bmatrix} (y_i^2 + y_j^2)^{1/2} \\ 0 \end{bmatrix}.$$

Indeed, we do not need to compute the angle but just its cosinus and sinus, which are given by the formulae

$$\cos(\theta) = \frac{y_i}{(y_i^2 + y_j^2)^{1/2}} \quad \text{and} \quad \sin(\theta) = \frac{-y_j}{(y_i^2 + y_j^2)^{1/2}}.$$

Thus the QR factorization of an $m \times n$ matrix A can be computed with Givens rotations similarly as it is done with Householder reflections, but zeroing one entry of this matrix at each step.

Example 5.4.1. Let $m = 3$, $n = 2$ and

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

Set $R_1 = \begin{bmatrix} 0.71 & 0 & -0.71 \\ 0 & 1 & 0 \\ 0.71 & 0 & 0.71 \end{bmatrix}$ so that

$$A^{(1)} = R_1\, A = \begin{bmatrix} 1.41 & -1.41 \\ 0 & 2 \\ 0 & -2.82 \end{bmatrix}.$$

Then set $R_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.58 & -0.82 \\ 0 & 0.82 & 0.58 \end{bmatrix}$ so that

$$A_2 = R_2\, A_1 = \begin{bmatrix} 1.41 & -1.41 \\ 0 & 3.49 \\ 0 & 0 \end{bmatrix} = \widetilde{R}.$$

Setting

$$\widetilde{Q} = R_1^T R_2^T = \begin{bmatrix} 0.71 & -0.58 & 0.41 \\ 0 & 0.58 & 0.82 \\ -0.71 & -0.58 & 0.41 \end{bmatrix} \quad \text{and} \quad \widetilde{R} = \begin{bmatrix} 1.41 & -1.41 \\ 0 & 3.49 \\ 0 & 0 \end{bmatrix}$$

we obtain the full QR factorization $A = \widetilde{Q}\,\widetilde{R}$.

The complexity of the QR factorization using Givens rotations is about twice that of doing this task with Householder reflections. However, it is useful in special situations where the input matrix has already many zeros, like in the Hessenberg case:

$$A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}$$

5.5. Rank deficient LS problem *via* the QR factorization

So far we have assumed so that A is full rank when minimizing the quantity $\|Ax - b\|_2$. What happens when A is rank deficient?

Precisely, let A be an $m \times n$ matrix with $r = \operatorname{rank}(A) < n$ and $b \in \mathbb{R}^m$. The solution of the associated LS problem is not unique: for a particular solution x_0 and every $x \in \operatorname{Ker}(L_A) \simeq \mathbb{R}^{n-r}$ we have that

$$A\,(x_0 + x) = A\,x_0 + A\,x = A\,x_0,$$

and so $x_0 + x$ also solves it.

In spite of this, in practice we need to pick a single solution $x_{\mathrm{LS}} = x_0 + x$ having a reasonable size. Such a solution is typically used as a predictor for a certain variable, but when this solution is too large it might produce an unrealistic prediction when applied to an instance outside the dataset.

Example 5.5.1. Suppose that we are doing medical research on the effect of a drug on the level of a certain toxin in the blood as in Example 4.4.1. Then for a set of m patients we want to consider the data therein but for some reason we measure their respective weights over five consecutive days, instead of just once. Then there are chances that these weights coincide, so the corresponding dataset will be encoded by a pair (A, b) where A is an $m \times 8$ matrix of rank 4.

As explained in Example 4.4.1, the solution x_{LS} would be used to predict the final toxin level

$$q = p^T x_{\mathrm{LS}}$$

for a further patient with known age, weights, initial toxin level and given amount of drug stored in a vector $p \in \mathbb{R}^8$. In might happen, though, that this patient does have a variation in the measurement of his weight, and an abnormally large choice of x_{LS} might have given him/her an unrealistic prediction.

We can solve a rank deficient LS problem applying the QR factorization: if A has rank $r < n$ and its first r columns are linearly independent, its thin QR factorization would look like

$$A = Q\,R = \begin{array}{c} \begin{array}{cc} \scriptstyle r & \scriptstyle n-r \end{array} \\ \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \end{array} \begin{array}{cc} \begin{array}{cc} \scriptstyle r & \scriptstyle n-r \end{array} & \\ \begin{bmatrix} R_{1,1} & R_{1,2} \\ 0 & 0 \end{bmatrix} & \begin{array}{c} \scriptstyle r \\ \scriptstyle n-r \end{array} \end{array}$$

with Q_1 and Q_2 orthogonal and $R_{1,1}$ nonsingular.

We can minimize the quantity $\|A\,x - b\|_2$ as follows: write $x = (x_1, x_2)$ with $x_1 \in \mathbb{R}^r$ and $x_2 \in \mathbb{R}^{n-r}$ and complete Q to an orthogonal $m \times m$ matrix $\widetilde{Q} = [Q_1\, Q_2\, Q']$ so that

$$\|A\,x - b\|_2^2 = \|\widetilde{Q}^T(A\,x - b)\|_2^2 = \|R_{1,1}\,x_1 + R_{1,2}\,x_2 - Q_1^T b - Q_2^T b - Q'^T b\|_2^2 \tag{5.10}$$

where the first equality comes from the fact that multiplying by an orthogonal $m \times m$ matrix does not change the 2-norm.

The first three terms in the right-hand side of (5.10) lie in $\mathbb{R}^r \oplus \mathbb{0}_{n-r} \oplus \mathbb{0}_{m-n}$, whereas the fourth and the fifth lie in $\mathbb{0}_r \oplus \mathbb{R}^{n-r} \oplus \mathbb{0}_{m-n}$ and in $\mathbb{0}_r \oplus \mathbb{0}_{n-r} \oplus \mathbb{R}^{m-n}$, respectively. We deduce that

$$\|A\,x - b\|_2^2 = \|R_{1,1}\,x_1 + R_{1,2}\,x_2 - Q_1^T b\|_2^2 + \|Q_2^T b\|_2^2 + \|Q'^T b\|_2^2. \tag{5.11}$$

Hence this expression is minimized by

$$x = \begin{bmatrix} R_{1,1}^{-1}(Q_1^T b - R_{1,2}\,x_2) \\ x_2 \end{bmatrix}$$

for any $(n-r)$-vector x_2. The typical choice is $x_2 = \mathbb{0}$.

CHAPTER 6

The singular value decomposition

6.1. Singular values and singular vectors

Let S be a symmetric positive definite $n \times n$ matrix over the reals. Then we can factor it as

$$S = Q\,\Lambda\,Q^{-1} \tag{6.1}$$

where Λ is a diagonal matrix with positive entries and Q is orthogonal, so that $Q^{-1} = Q^T$. Thus S has positive eigenvalues, and its diagonalization is realized by an orthogonal similarity, which is the best that one can hope.

Such a factorization does not extend directly to the general situation: an arbitrary $n \times n$ matrix is not necessarily diagonalizable and when it is, its eigenvalues might be arbitrary complex numbers and the similarity an arbitrary nonsingular $n \times n$ matrix.

In spite of this, there is a factorization that actually extends (6.1) to the general situation, even beyond the square case: the singular value decomposition (SVD). The idea behind it is to decouple the similarity and its inverse into two different matrices, which allows enough flexibility to reach the sought properties.

More precisely, for an $m \times n$ matrix A over the reals with $m \geq n$ one can construct two sets of vectors:

$$u_1, \dots, u_m \in \mathbb{R}^m \quad \text{and} \quad v_1, \dots, v_n \in \mathbb{R}^n,$$

respectively called the *left singular vectors* and the *right singular vectors.* These form orthogonal bases of $\mathbb{R}^m$ and of $\mathbb{R}^n$ respectively, and are connected by the relation

$$A\,v_i = \sigma_i\,u_i, \quad i = 1, \dots, n, \tag{6.2}$$

for the *singular values* $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$.

To express this relations in matrix form, set

$$U = \begin{bmatrix} u_1 & \cdots & u_m \end{bmatrix} \in \mathbb{R}^{m\times m} \quad \text{and} \quad V = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}.$$

These are *orthogonal* square matrices, that is they are both nonsingular and verify that $U^{-1} = U^T$ and $V^{-1} = V^T$. Set also

$$\Sigma = \begin{array}{cc} n & \\ \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma \\ & \mathbb{0} & \end{bmatrix} & \begin{array}{c} n \\ \\ m-n \end{array} \end{array}.$$

The matrix form of the relations in (6.2) is the factorization $A\,V = U\,\Sigma$, and the *(full) SVD* of A is the equivalent factorization

$$A = U\,\Sigma\,V^T. \tag{6.3}$$

Example 6.1.1. For the matrix $A = \begin{bmatrix} 3 & 0 \\ 4 & 5 \end{bmatrix}$, its full SVD is

$$\begin{bmatrix} 3 & 0 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 0.32 & -0.95 \\ 0.95 & 0.32 \end{bmatrix} \begin{bmatrix} 6.71 & 0 \\ 0 & 2.24 \end{bmatrix} \begin{bmatrix} 0.71 & 0.71 \\ -0.71 & 0.71 \end{bmatrix}.$$

In the general rectangular case, we can reduce the size of the matrices in the full SVD without loosing essential information. For instance, we can avoid the 0's in the lower part of Σ and use a diagonal matrix for the singular values by setting

$$U_n = \begin{bmatrix} u_1 & \cdots & u_n \end{bmatrix} \in \mathbb{R}^{m\times n} \quad \text{and} \quad \Sigma_n = \operatorname{diag}(\sigma_1, \dots, \sigma_n) \in \mathbb{R}^{n\times n},$$

which leads to the *thin SVD*:

$$A = U_n \, \Sigma_n \, V^T.$$

Here U is an $m \times n$ orthogonal matrix and Σ_n is a diagonal $n \times n$ matrix, whereas V is an orthogonal $n \times n$ matrix as before.

Note that the rank of A coincides with its number of nonzero nonsingular values: setting $r = \operatorname{rank}(A)$ we have that $\sigma_1 \geq \cdots \geq \sigma_r > 0$ and $\sigma_{r+1} = \cdots = \sigma_n = 0$. Hence we can further reduce the thin SVD by removing the parts that are going to produce zeros for sure: setting

$$U_r = \begin{bmatrix} u_1 & \cdots & u_r \end{bmatrix} \in \mathbb{R}^{m\times r}, \ \Sigma_r = \operatorname{diag}(\sigma_1, \dots, \sigma_r) \in \mathbb{R}^{r\times r}, \ V_r = \begin{bmatrix} v_1 & \cdots & v_r \end{bmatrix} \in \mathbb{R}^{n\times r}$$

leads to the *reduced SVD*:

$$A = U_r \, \Sigma_r V_r^T.$$

Here U_r is an orthogonal $m \times r$ matrix, Σ_r is a diagonal $r \times r$ matrix, and V_r is an orthogonal $n \times r$ matrix. We still have that $U_r^T U_r = \mathbb{1}_r$ and $V_r^T V_r = \mathbb{1}_r$ but these matrices might be not invertible since they are rectangular.

Now consider again the full SVD $A = U \, \Sigma \, V^T$. To identify the singular values and vectors of A one can consider the symmetric semipositive definite matrices

$$A^T A = (V \, \Sigma^T U^T) \, (U \, \Sigma \, V^T) = V \, \Sigma^T \Sigma \, V^T = V \Lambda_n V^T \in \mathbb{R}^{n\times n}, \tag{6.4}$$

$$A \, A^T = (U \, \Sigma \, V^T) \, (V \, \Sigma^T U^T) = U \Sigma \, \Sigma^T U^T = U \Lambda_m U^T \in \mathbb{R}^{m\times m} \tag{6.5}$$

with

$$\Lambda_n = \operatorname{diag}(\sigma_1^2, \dots, \sigma_n^2) \in \mathbb{R}^{n\times n} \quad \text{and} \quad \Lambda_m = \operatorname{diag}(\sigma_1^2, \dots, \sigma_n^2, 0, \dots, 0) \in \mathbb{R}^{m\times m}.$$

Thus the equalities in (6.4) and (6.5) give the diagonalizations of $A^T A$ and of $A \, A^T$, respectively. Hence the left and the right singular vectors of A give orthonormal bases of eigenvectors for $A \, A^T$ and for $A^T A$ respectively, whereas the squares of the singular values of A give the eigenvalues of both $A^T A$ and $A \, A^T$.

Conversely, one can deduce from the factorization (6.4) the existence of the SVD and a way to compute it. To this aim, start with the diagonalization

$$A^T A = Q \operatorname{diag}(\lambda_1, \dots, \lambda_n) \, Q^T$$

with the eigenvalues in decreasing order, so that $\lambda_1 \geq \cdots \geq \lambda_r > \lambda_{r+1} = \cdots = \lambda_n = 0$

Let $v_1, \dots, v_n$ be the columns of Q, so that $v_1, \dots, v_r$ are the eigenvectors of $A^T A$ which correspond to the nonzero eigenvalues $\lambda_1, \cdots, \lambda_r$. Set then

$$\sigma_k = \lambda_k^{1/2} \quad \text{and} \quad u_k = \frac{1}{\sigma_k} A \, v_k, \quad k = 1, \dots, r.$$

These latter vectors form an orthonormal system, since for $j,k = 1,\dots,r$ we have that

$$\langle u_j, u_k\rangle = u_j^T u_k = \Big(\frac{1}{\sigma_j} A\, v_j\Big)^T \Big(\frac{1}{\sigma_k} A\, v_k\Big)$$
$$= \frac{1}{(\lambda_j\lambda_k)^{1/2}}\, v_j^T A^T A\, v_k = v_j^T v_k = \langle v_j, v_k\rangle = \begin{cases} 1 & \text{if } j = k,\\ 0 & \text{if } j \neq k.\end{cases}$$

By construction $A\, v_i = \sigma_i u_i$, $i = 1,\dots,r$, and since both $u_1,\dots,u_r$ and $v_1,\dots,v_r$ are orthonormal we obtain the reduced SVD of A by setting

$$A = \begin{bmatrix} u_1 & \cdots & u_r \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \begin{bmatrix} v_1^T \\ \vdots \\ v_r^T \end{bmatrix},$$

Now to compute a full SVD $A = U\,\Sigma\, V^T$, one just need to take $v_{r+1},\dots,v_n \in \mathbb{R}^n$ and $u_{r+1},\dots,u_m \in \mathbb{R}^m$ completing $v_1,\dots,v_r$ and $u_1,\dots,u_r$ to orthonormal bases and set

$$U = \begin{bmatrix} u_1 & \cdots & u_m \end{bmatrix}, \quad V = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_r & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \\ & & & \mathbb{0} & & \end{bmatrix}.$$

Example 6.1.2. Set as before $A = \begin{bmatrix} 3 & 0 \\ 4 & 5 \end{bmatrix}$. Then $A^TA = \begin{bmatrix} 25 & 20 \\ 20 & 25 \end{bmatrix}$, and we have that

$$\begin{bmatrix} 25 & 20 \\ 20 & 25 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = 45 \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad\text{and}\quad \begin{bmatrix} 25 & 20 \\ 20 & 25 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix} = 5 \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

Normalizing these eigenvectors we obtain the right singular vectors

$$v_1 = \frac{1}{2^{1/2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.71 \\ 0.71 \end{bmatrix} \quad\text{and}\quad v_2 = \frac{1}{2^{1/2}}\begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.71 \\ 0.71 \end{bmatrix}.$$

The singular values are $\sigma_1 = 45^{1/2} = 6.71$ and $\sigma_2 = 5^{1/2} = 2.24$, whereas the left singular vectors are given by

$$u_1 = \frac{1}{\sigma_1} A\, v_1 = \begin{bmatrix} 0.32 \\ 0.95 \end{bmatrix} \quad\text{and}\quad u_2 = \frac{1}{\sigma_2} A\, v_2 = \begin{bmatrix} -0.95 \\ 0.32 \end{bmatrix}.$$

We conclude that $A = U\,\Sigma\, V^T$ with

$$U = \begin{bmatrix} 0.32 & -0.95 \\ 0.95 & 0.32 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 6.71 & \\ & 2.24 \end{bmatrix}, \quad V = \begin{bmatrix} 0.71 & -0.71 \\ 0.71 & 0.71 \end{bmatrix}.$$

As a first application, we can compute the 2-norm and the Frobenius norm of a matrix in terms of its singular values. Indeed, both norms are invariant with respect to multiplication by orthogonal matrices and so

$$\|A\|_2 = \|U^T A\, V\|_2 = \|\Sigma\|_2 = \sigma_1, \ \|A\|_{\mathrm{F}} = \|U^T A\, V\|_{\mathrm{F}} = \|\Sigma\|_{\mathrm{F}} = \Big(\sum_{i=1}^r \sigma_i^2\Big)^{1/2}. \tag{6.6}$$

Here are some important special cases where the SVD can be easily be obtained.

Symmetric semipositive definite matrices. Let A be symmetric semipositive definite $n \times n$ matrix, and let (λ_i, v_i), $i = 1, \dots, n$, be its eigenpairs. These eigenvalues are nonnegative real numbers and we suppose that they are in decreasing order, so that

$$\lambda_1 \geq \cdots \geq \lambda_n \geq 0.$$

The eigenvectors $v_1, \dots, v_n$ can be chosen as an orthonormal basis of $\mathbb{R}^n$. Setting

$$Q = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix} \quad \text{and} \quad \Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$$

we obtain the diagonalization $A = Q\,\Lambda\,Q^T$, which can be interpreted as the SVD $A = U\Sigma\,V^T$ by setting $U = V = Q$ and $\Sigma = \Lambda$.

Orthonormal matrices. For an orthonormal $n\times n$ matrix A we have that $A^T A = \mathbb{1}_n$. Hence its singular values are all equal to 1, and its SVD is $A = U\,\Sigma\,V$ for $U = A$ and $\Sigma = V = \mathbb{1}_n$.

Rank 1 *matrices.* Let $A = x\,y^T$ be an $m \times n$ matrix of rank 1, with $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$. Its reduced SVD is $A = U_1\,\Sigma_1\,V_1^T$ with

$$U_1 = \frac{x}{\|x\|_2}, \quad \Sigma_1 = \begin{bmatrix} \|x\|_2\,\|y\|_2 \end{bmatrix}, \quad V_1 = \frac{y}{\|y\|_2}.$$

6.2. The geometry of the SVD

The SVD helps to visualize how a given linear map deforms the space. For instance, for a given $m \times n$ matrix A the associated linear map L_A transforms the unit sphere $\mathbb{S}_n$ of $\mathbb{R}^n$ into an ellipsoid $L_A(\mathbb{S}_n)$ of $\mathbb{R}^m$ centered at the origin, that can be precised as we next explain.

The factorization $A = U\Sigma V^T$ implies that the associated linear map decomposes as

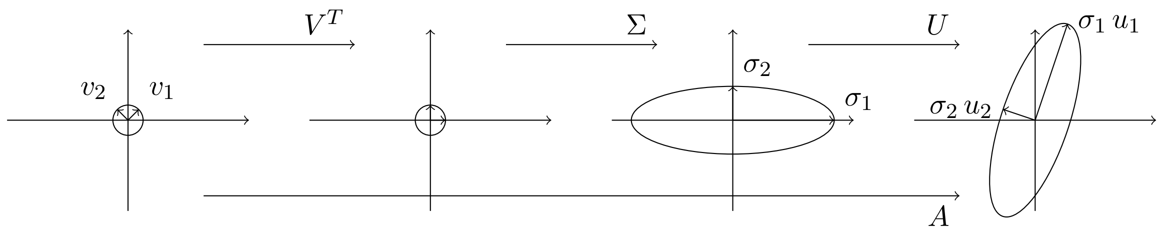
$$L_A = L_U \circ L_\Sigma \circ L_{V^T}.$$

Since V^T is orthogonal, $L_{V^T}(\mathbb{S}_n) = \mathbb{S}_n$. On the other hand, the diagonal map L_Σ stretches the space along the standard axes, and so $L_\Sigma(\mathbb{S}_n)$ is the ellipsoid of $\mathbb{R}^m$ with principal nonzero axes $\sigma_i e_i$, $i = 1, \dots, r = \operatorname{rank}(A)$. Finally $L_U(L_\Sigma(\mathbb{S}_n))$ rotates this ellipsoid according to the orthogonal map L_U. Thus

$$L_A(\mathbb{S}_n) = L_U(L_\Sigma(L_{V^T}(\mathbb{S}_n)))$$

is the ellipsoid of $\mathbb{R}^m$ with principal nonzero axes $\sigma_i u_i$, $i = 1, \dots, r$.

Example 6.2.1. Here is the scheme for the matrix $A = \begin{bmatrix} 3 & 0 \\ 4 & 5 \end{bmatrix}$:



6.3. The best low rank approximation

Given the full SVD $A = U\,\Sigma\,V^T$, the column-row multiplication of $U\,\Sigma$ and V^T splits A into r pieces of rank 1:

$$A = \sum_{i=1}^{r} \sigma_i\, u_i\, v_i^T.$$

For instance, for the matrix in Example 6.1.1 we have that

$$\begin{bmatrix} 3 & 0 \\ 4 & 5 \end{bmatrix} = 6.71 \begin{bmatrix} 0.32 \\ 0.95 \end{bmatrix} \begin{bmatrix} 0.71 & 0.71 \end{bmatrix} + 2.24 \begin{bmatrix} -0.95 \\ 0.32 \end{bmatrix} \begin{bmatrix} -0.71 & 0.71 \end{bmatrix}.$$

The first piece is more representative of A because $\sigma_1 = 6.71 > \sigma_2 = 2.24$.

In the general situation, the *Eckart-Young theorem* says that setting $r = \operatorname{rank}(A)$, for each $1 \le k \le r$ the partial sum

$$A_k = \sum_{i=1}^{k} \sigma_i\, u_i\, v_i^T = U_k\, \Sigma_k\, V_k^T$$

is the best rank k approximation of A with respect to both the 2-norm and the Frobenius norm, that is

$$\|A - B\| \ge \|A - A_k\|$$

for any of these two norms and any $m \times n$ matrix B of rank $\le k$.

By the orthogonal invariance we have that $\|A - A_k\| = \|\Sigma - \Sigma_k\|$, and so the residuals of this approximation can be computed as

$$\|A - A_k\|_2 = \sigma_{k+1} \quad \text{and} \quad \|A - A_k\|_{\mathrm{F}} = \Big(\sum_{i=k+1}^{r} \sigma_i\Big)^{1/2}.$$

6.4. Image compression

A B/W image of $m \times n$ pixels can be coded by an $m \times n$ matrix A whose entries $a_{i,j} \in [0,1]$ indicate the brightness of the (i,j)-pixel:

$$0 \text{ (black)} \cdots \text{ gray } \cdots 1 \text{ (white)}.$$

The size of the image is the number of entries that code it, that is $m\,n$. To save space, instead of fully transmitting/storing it, one can replace it by its k-th rank approximation

$$A_k = \sum_{i=1}^{k} \sigma_i\, u_i\, v_i^T,$$

which is coded by $k\,(m+n+1)$ entries (or $k\,(m+n)$ if we store $\sigma_i\, u_i$).

The relative error of this approximation with respect to the 2-norm is

$$\frac{\|A - A_k\|_2}{\|A\|_2} = \frac{\sigma_{k+1}}{\sigma_1}$$

whereas its *compression ratio* is

$$\frac{k\,(m+n)}{m\,n}.$$

Here is a 320×200-picture of a clown and some of its approximations:

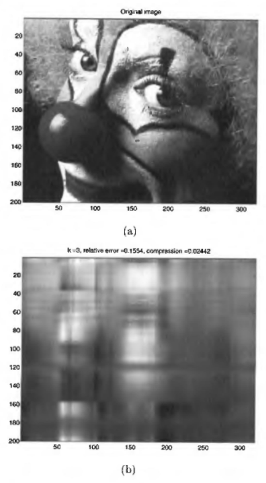


Fig. 3.3. *Image compression using the SVD.* (a) *Original image.* (b) *Rank $k = 3$ approximation.*

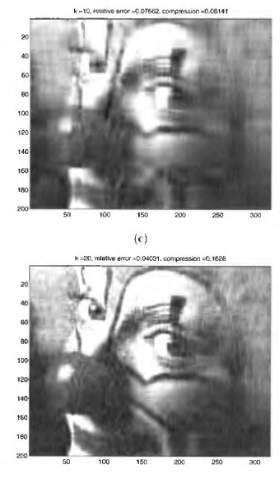

k =10, relative error =0.07662, compression =0.08141
(c)
k =20, relative error =0.04031, compression =0.1628

Fig. 3.3. *Continued.* (c) *Rank $k = 10$ approximation.* (d) *Rank $k = 20$ approximation.*

6.5. The rank deficient LS problem *via* the SVD

The SVD also applies to the LS problem, and it is particularly appropriate in the rank deficient case. Let $A \in \mathbb{R}^{m\times n}$ and set $r = \operatorname{rank}(A)$. If $r < n$ then the solution x_{LS} of the LS problem $\min_{x\in\mathbb{R}^n} \|A\,x - b\|_2$ is not unique since

$$A\,x_{\mathrm{LS}} = A\,(x_{\mathrm{LS}} + y)$$

for any $y \in \operatorname{Ker}(L_A) \simeq \mathbb{R}^{n-r}$.

To distinguish a particular solution, consider the reduced SVD $A = U_r\,\Sigma_r V_r^T$ and set

$$x_{\mathrm{LS}} = A^+ b \in \mathbb{R}^m$$

where $A^+ = V_r\,\Sigma_r^{-1}\,U_r^T \in \mathbb{R}^{n\times m}$ denotes the *Moore-Penrose pseudo-inverse* of A. This m-vector satisfies the normal equations (5.1) and so it is a solution of the LS problem:

$$A^T A\,x_{\mathrm{LS}} = (V_r\,\Sigma_r^2\,V_r^T)\,(V_r\,\Sigma_r^{-1}U_r^T)\,b = V_r\,\Sigma_r\,U_r^T b = A^T b.$$

Indeed, it is the solution with the smallest 2-norm.

This solution is well-conditioned whenever the smallest nonzero singular value of A is not too small: changing b to $b + \delta b$ changes x to $x + \delta x$ with $\delta x = V_r\,\Sigma_r^{-1}U_r^T\delta b$, which implies that

$$\|\delta x\|_2 \le \|\Sigma_r^{-1}\|_2\,\|\delta b\|_2 = \frac{\|\delta b\|_2}{\sigma_r}.$$

In practice, the difficulty is that the rank is not continuous and so it might be affected by small perturbations. The next example gives a situation where a perturbation of size $O(\varepsilon)\,\|A\|_2$ increases the condition number of the rank-deficient LS problem from σ_r^{-1} to ε^{-1}.

Example 6.5.1. Let $A = \begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}$ and $b = \begin{bmatrix}1\\ 1\end{bmatrix}$. Then

$$A^+ = \begin{bmatrix}1\\ 0\end{bmatrix}\,[1]\,\begin{bmatrix}1 & 0\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}$$

and so

$$x_{\text{LS}} = A^+ b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

with condition number $\sigma_1^{-1} = 1$.

On the other hand, if we set $A_\varepsilon = \begin{bmatrix} 1 & 0 \\ 0 & \varepsilon \end{bmatrix}$ for some an $\varepsilon > 0$ then

$$x_{\text{LS}} = \begin{bmatrix} 1 \\ \frac{1}{\varepsilon} \end{bmatrix}$$

with condition number ε^{-1}.

The SVD is *backward stable*: rounding off with machine epsilon ε gives

$$(U + \delta U)(\Sigma + \delta\Sigma)(V + \delta V)^T = A + \delta A$$

with $\|\delta A\|_2 \leq O(\varepsilon)\,\|A\|_2$. Hence the computed singular values $\sigma_i + \delta\sigma_i$ verify

$$|\delta\sigma_i| \leq O(\varepsilon)\,\|A\|_2.$$

When the matrix is very close to a rank-deficient matrix, it is advisable to replace our matrix with this one to avoid numerical instability in the solution of the LS problem.

To this aim, let tol > 0 be a user supplied measure of uncertainty in A. Rounding implies that

$$\text{tol} \geq \varepsilon\,\|A\|_2$$

but this parameter might be larger, taking into account for instance errors in measurements. Now given the computed factors $\widetilde{U}$, $\widetilde{\Sigma}$ and $\widetilde{V}$ for the SVD of A set

$$\widehat{\sigma}_i = \begin{cases} \widetilde{\sigma}_i & \text{if } \widetilde{\sigma}_i \geq \text{tol}, \\ 0 & \text{else,} \end{cases}$$

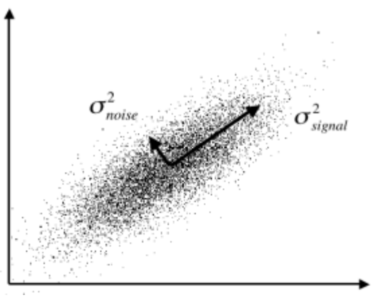
and replace $\widetilde{\Sigma}$ by the *truncated SVD*

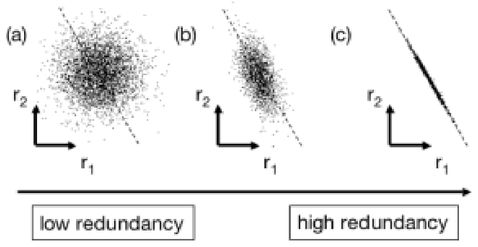
$$\widehat{\Sigma} = \begin{bmatrix} \widetilde{\sigma}_1 & & & & & \\ & \ddots & & & & \\ & & \widetilde{\sigma}_r & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \\ & & & \mathbb{0} & & \end{bmatrix}.$$

Finally set

$$\widehat{x} = \widetilde{V}_r\,\widehat{\Sigma}_r^{-1}\widetilde{U}_r^T.$$

As a result, the numerical error will be bounded by $O(\text{tol})$ and the condition number by σ_r^{-1}.





6.6. Principal component analysis

Let M be a data $m \times n$ matrix, whose rows collect the *samples* and whose columns indicate the different *variables* collected.

How can one simplify the description of this data? Plotting the samples in $\mathbb{R}^n$ we might find that they are *correlated*: This correlation might be higher *(more significant)* or lower *(less significant)*.

The key parameters in probability and statistics are the *mean* and the *variance.* Denoting by $M_i \in \mathbb{R}^n$ the i-th row of M for each i, their mean is

$$\mu = \frac{1}{n}\sum_{i=1}^{m} M_i \in \mathbb{R}^n$$

and so the *centered data matrix* is

$$A = M - \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \mu^T = \begin{bmatrix} M_1 - \mu^T \\ \vdots \\ M_m - \mu^T \end{bmatrix} = \begin{bmatrix} A_1 \\ \vdots \\ A_m \end{bmatrix}.$$

Now the mean of each variable *(column)* in A is 0.

The *covariance matrix* then is defined as

$$\operatorname{cov}(A) = \frac{1}{m-1} A^T A.$$

It is a semipositive definite symmetric $n \times n$ matrix that contains the variances and covariances of the variables in A, that is the columns $a_j \in \mathbb{R}^m$, $j = 1, \ldots, n$, of this matrix.

The *total variance* of these variables is given by the trace of the covariance matrix, and it coincides with the (normalized) Frobenius norm of the matrix A.

$$\operatorname{var}(A) = \sum_{j=1}^{n} \frac{\langle a_j, a_j\rangle}{m-1} = \operatorname{tr}(\operatorname{cov}(A)) = \frac{1}{m-1}\sum_{i,j} a_{i,j}^2 = \frac{1}{m-1}\|A\|_{\mathrm{F}}.$$

By (6.6), this total variance can also be computed from the singular values of A:

$$\operatorname{var}(A) = \frac{1}{m-1}\sum_{j=1}^{n} \sigma_j^2.$$

EXAMPLE 6.6.1. As a running example, we can think of a 6×2 matrix collecting the ages and weights in a group of six people:

$$M = \begin{bmatrix} 15 & 58 \\ 26 & 64 \\ 20 & 62 \\ 14 & 57 \\ 10 & 58 \\ 23 & 61 \end{bmatrix}$$

The mean of these variables is $\frac{1}{6}\sum_{i=1}^{6} M_i = (18, 60)$ and so the centered data matrix is

$$A = \begin{bmatrix} -3 & -2 \\ 8 & 4 \\ 2 & 2 \\ -4 & -3 \\ -8 & -2 \\ 5 & 1 \end{bmatrix}.$$

The covariance matrix is $\operatorname{cov}(A) = \frac{1}{6-1}A^T A = \begin{bmatrix} 36.40 & 15.00 \\ 15.00 & 7.60 \end{bmatrix}$ and so the total variance is $\operatorname{var}(A) = \operatorname{tr}(\operatorname{cov}(A)) = 44.40$.

Now for each $k = 1, \dots, n$ we want to find a k-th dimensional linear subspace

$$L_k \subset \mathbb{R}^n$$

such that the projection of our data on it gives us the most accurate approximation among all the linear subspace of that dimension. There are several ways of measuring the quality of this approximation: for instance, one might try to maximize the total variance of this projected data, or to minimize the distance of this projected data to the samples. The SVD allows to compute a k-dimensional subspace that will be optimal in both senses.

To this aim, denote by v_j, $j = 1, \dots, n$, the different columns of the orthogonal $n \times n$ matrix V in the thin SVD of A. For $i = 1, \dots, m$ the projection $A_i \mapsto \operatorname{span}(v_1, \dots, v_k)$ is

$$A_i \longmapsto \sum_{j=1}^{k} \langle A_i, v_j\rangle v_j.$$

Consider the orthogonal $n \times k$ matrix

$$V_k = \begin{bmatrix} v_1 & \cdots & v_k \end{bmatrix} \in \mathbb{R}^{n\times k}$$

and set

$$B_k = A\,V_k = \big[\langle A_i, v_j\rangle\big]_{\substack{1\le i\le m\\ 1\le j\le n}} = U_k\,\Sigma_k = \big[\sigma_j u_{i,j}\big]_{\substack{1\le i\le m\\ 1\le j\le n}}.$$

The variable

$$b_j = \begin{bmatrix} \langle A_1, v_j\rangle \\ \vdots \\ \langle A_m, v_j\rangle \end{bmatrix} = \begin{bmatrix} \sigma_j u_{j,1} \\ \vdots \\ \sigma_j u_{j,m} \end{bmatrix}.$$

is the j-th *principal component* of A, and it is an m-vector containing the j-th coordinates of the projection of the data with respect to the basis v_j, $j = 1, \dots, n$.

The covariance matrix of this matrix is

$$\operatorname{cov}(B_k) = \frac{1}{m-1} B_k^T B_k = \frac{1}{m-1}\Sigma_k^2 = \begin{bmatrix} \frac{\sigma_1^2}{m-1} & & \\ & \ddots & \\ & & \frac{\sigma_k^2}{m-1} \end{bmatrix}.$$

Hence these variables are not correlated and they are ordered according to their variances, since $\langle b_i, b_j\rangle = 0$ for all $i \neq j$ and $\langle b_i, b_i\rangle = \sigma_i^2$ for each i. Their total variance is

$$\operatorname{var}(B_k) = \frac{1}{m-1}\sum_{j=1}^{k}\sigma_j^2,$$

and so they account for $(\sum_{j=1}^k \sigma_j^2)/(\sum_{j=1}^n \sigma_j^2)$ of the total variance of the data.

Also the sum of the squares of the distances between the samples and their projections into this k-th linear subspace is

$$\sum_{i=1}^{m}\Big\| A_i - \sum_{j=1}^{k}\langle A_i, v_j\rangle\, v_j\Big\|_2^2 = \sum_{i=1}^{m}\sum_{j=k+1}^{n}\langle A_i, v_j\rangle^2 = \sum_{j=k+1}^{n}\sigma_j^2.$$

It can be shown that for the centered data matrix A, these principal components attain both the maximal total variance for the orthogonal projection into a k-th linear subspace of $\mathbb{R}^n$, and the minimal sum of the squares of the distances between the samples and their orthogonal projections into such a linear subspace.

EXAMPLE 6.6.2. For the matrix in Example 6.6.1 we have that

$$A = \begin{bmatrix} -3 & -2 \\ 8 & 4 \\ 2 & 2 \\ -4 & -3 \\ -8 & -2 \\ 5 & 1 \end{bmatrix} = U\,\Sigma\,V^t = \begin{bmatrix} -0.24 & 0.27 \\ 0.61 & -0.22 \\ 0.18 & -0.43 \\ -0.33 & 0.49 \\ -0.56 & -0.53 \\ 0.34 & 0.42 \end{bmatrix} \begin{bmatrix} 14.63 & 0 \\ 0 & 2.46 \end{bmatrix} \begin{bmatrix} 0.92 & 0.39 \\ 0.39 & -0.92 \end{bmatrix},$$

The first principal component is

$$b_1 = A\,V_1 = \sigma_1 u_1 = \begin{bmatrix} -3.54 \\ 8.93 \\ 2.62 \\ -4.86 \\ -8.14 \\ 4.99 \end{bmatrix},$$

so that the projected data into the line $\operatorname{span}(v_1)$ are the vectors $b_{1,j} v_1$, $j = 1, \dots, m$.

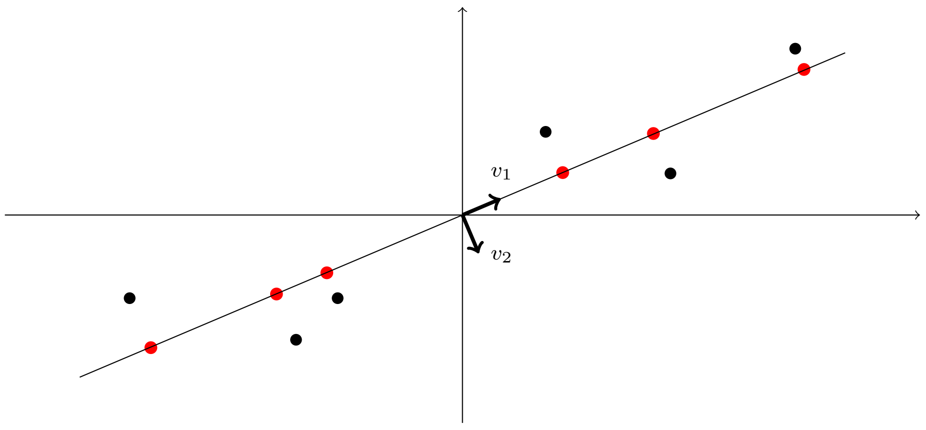


FIGURE 6.6.1. PCA for the matrix A

Its total variance is $\sigma_1^2/5 = 42.81$ and since $42.81/44.40 = 0.96$, this principal component accounts for 96% of the total variance of the centered data.

Part 3

Eigenproblems

CHAPTER 7

Eigenvalues and invariant subspaces

Let A be an $n \times n$ matrix with complex coefficients. An element $\lambda \in \mathbb{C}$ is an *eigenvalue* of A if there is $x \in \mathbb{C}^n \setminus \{0\}$ such that

$$A\,x = \lambda\,x$$

that is, if A is a homothecy in the direction of this n-vector. This happens if and only if $A - \lambda\,\mathbb{1}_n$ is singular or equivalently, if and only if $\det(A - \lambda\,\mathbb{1}_n) = 0$ or still in other terms, if and only if λ is a root of the *characteristic polynomial*

$$\chi_A = \det(A - z\,\mathbb{1}_n) \in \mathbb{C}[z]_n,$$

where $\mathbb{C}[z]_n$ denotes the space of polynomials of degree n with complex coefficients. The *spectrum* of A is

$$\lambda(A) = \{\text{eigenvalues of } A\} = \{\lambda \in \mathbb{C} \mid \chi_A(\lambda) = 0\}.$$

The eigenvalues can be complex even if A is real: for instance, if

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

then $\lambda(A) = \{\pm i\}$.

For each $\lambda \in \lambda(A)$, the associate *eigenspace* is

$$V_\lambda(A) = \{x \in \mathbb{C}^n \mid A\,x = \lambda\,x\}.$$

This an *invariant subspace*, that is $L_A(V_\lambda(A)) \subset V_\lambda(A)$. Its dimension

$$\dim(V_\lambda(A))$$

is called the *geometric multiplicity* of λ in A. On the other hand, the *algebraic multiplicity* of λ in A, denoted by $e_\lambda(A)$, is the exponent of the factor $\lambda - z$ in the irreducible factorization of χ_A. These quantities are related by

$$1 \leq \dim(V_\lambda(A)) \leq e_\lambda(A).$$

An $n \times n$ matrix B is *similar* to A if there is a nonsingular $n \times n$ matrix S such that

$$A = S\,B\,S^{-1}.$$

The matrices A and B have the same eigenvalues, and the eigenvectors of A can be read from those of B: an n-vector y is an eigenvector of B for λ if and only if $S\,y$ is an eigenvector of A for λ.

For instance, if $B = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ is a diagonal matrix, then the i-th vector in the standard basis of $\mathbb{C}^n$

$$e_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \\ \\ \\ i \\ \\ \\ \\ \end{matrix}$$

is the eigenvector of B for λ_i and so $s_i = S\,e_i$, the i-th column in the matrix S, is the eigenvector of A for λ_i.

A matrix A is *diagonalizable* if there is a nonsingular matrix S and a diagonal matrix $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ such that

$$A = S\,\Lambda\,S^{-1}.$$

This is equivalent to the fact that the geometric and the algebraic multiplicities of each of the eigenvalues of A coincide, that is,

$$\dim V_\lambda(A) = e_\lambda(A) \qquad \text{for all } \lambda \in \lambda(A).$$

Indeed, writing $S = [s_1 \cdots s_n]$, the eigenspace $V_\lambda(A)$ is generated by the vectors s_i corresponding to the eigenvalues λ_i.

In general, the matrix A admits a *Jordan decomposition*:

$$A = S\,J\,S^{-1}$$

with J a block diagonal matrix $J = \operatorname{diag}(J_1, \dots, J_q)$ where each block is of the form

$$J_i = \begin{bmatrix} \lambda_i & 1 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 0 & \cdots & 0 & \lambda_i \end{bmatrix} \in \mathbb{C}^{n_i \times n_i}$$

and $n_1 + \cdots + n_q = n$.

This decomposition gives full information about the eigenvalues and the eigenvectors of A: for each i, the eigenvalue of J_i is λ_i with eigenspace of dimension 1, generated by the vector $e_1 \in \mathbb{C}^{n_i}$. Hence the eigenvalues of A are the λ_i's and for each of them, the basis of the eigenspace $V_\lambda(A)$ is given by the columns of S corresponding to the first columns of the Jordan blocks with eigenvalue λ_i.

In spite of its convenience, this factorization has little interest from the point of view of numerical computations. For instance, the Jordan decomposition of a defective matrix, that is a matrix that is not diagonalizable, is difficult to compute numerically, because this decomposition is not continuous.

EXAMPLE 7.0.1. The Jordan form of the matrix

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

is $J = A$. For $\varepsilon_1 \neq \varepsilon_2$ small, the perturbed matrix $\widetilde{A} = \begin{bmatrix} \varepsilon_1 & 1 \\ 0 & \varepsilon_2 \end{bmatrix}$ has two different eigenvalues ε_1 and ε_2, and so its Jordan form is

$$\widetilde{J} = \begin{bmatrix} \varepsilon_1 & 0 \\ 0 & \varepsilon_2 \end{bmatrix},$$

which is far from J.

Moreover, in many situations this factorization cannot be computed stably: after computing S and J, we cannot guarantee that

$$A + \delta A = S\, J\, S^{-1}$$

with δA small, because S might have a large condition number.

Example 7.0.2. Let

$$A = \begin{bmatrix} 1+\varepsilon & 1 \\ 0 & 1-\varepsilon \end{bmatrix}$$

with ε small. Up to a scalar, the eigenvectors of A are

$$s_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \text{and} \quad s_2 = \begin{bmatrix} 1 \\ -2\,\varepsilon \end{bmatrix}.$$

The Jordan decomposition of A is

$$A = \begin{bmatrix} 1 & 1 \\ 0 & -2\,\varepsilon \end{bmatrix} \begin{bmatrix} 1+\varepsilon & 0 \\ 0 & 1-\varepsilon \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{2\varepsilon} \\ 0 & \frac{-1}{2\varepsilon} \end{bmatrix},$$

and so $\kappa_\infty(S) \approx \varepsilon^{-1}$, which can be arbitrarily large.

7.1. The Schur decomposition

A matrix $Q \in \mathbb{C}^{n\times n}$ is *unitary* if its *conjugate transpose* $Q^* := \overline{Q}^T$ coincides with its inverse, that is $Q^* = Q^{-1}$.

For reasons of numerical stability, we prefer to consider similarities given by unitary matrices. This requirement leads to the *Schur decomposition*, which is a factorization of the form

$$A = Q\, T\, Q^*. \tag{7.1}$$

for a unitary matrix Q and an upper triangular matrix T.

Its existence can be verified by induction on n. When $n = 1$ it is obvious: $Q = [1]$ and $T = A$.

When $n > 1$ take an eigenvalue $\lambda \in \lambda(A)$ and a unit eigenvector $u \in \mathbb{C}^n$ of it, which exists because of the fundamental theorem of algebra. Choose then an $n \times (n-1)$ matrix $\widetilde{U}$ such that $U = [u\, \widetilde{U}]$ is unitary. Then

$$U^* A\, U = \begin{array}{c} {\scriptstyle 1} \\ {\scriptstyle n-1} \end{array} \begin{bmatrix} u^* \\ \widetilde{U}^* \end{bmatrix} A \begin{array}{cc} {\scriptstyle 1} & {\scriptstyle n-1} \\ \big[u & \widetilde{U}\,\big] \end{array} = \begin{array}{c|cc} & {\scriptstyle 1} & {\scriptstyle n-1} \\ {\scriptstyle 1} & u^* A\, u & u^* A\, \widetilde{U} \\ {\scriptstyle n-1} & \widetilde{U}^* A\, u & \widetilde{U}^* A\, \widetilde{U} \end{array}.$$

We have that $u^* A\, u = u^* \lambda\, u = \lambda$ and $\widetilde{U}^* A\, u = \widetilde{U}^* \lambda\, u = \mathbb{0}$ and so

$$U^* A\, U = \begin{bmatrix} \lambda & \widetilde{a}_{1,2} \\ \mathbb{0} & \widetilde{A}_{2,2} \end{bmatrix}.$$

By induction, there is an $(n-1)\times(n-1)$-unitary matrix P and an $(n-1)\times(n-1)$-upper triangular matrix $\widetilde{T}$ such that $\widetilde{A}_{2,2} = P\,\widetilde{T}\,P^*$. Hence

$$A = U \begin{bmatrix} \lambda & \widetilde{a}_{1,2} \\ \mathbb{0} & \widetilde{A}_{2,2} \end{bmatrix} U^* = Q\,T\,Q^*$$

with

$$Q = U \begin{bmatrix} 1 & \mathbb{0} \\ \mathbb{0} & P \end{bmatrix} \quad \text{and} \quad T = \begin{bmatrix} \lambda & \widetilde{a}_{1,2}\,P \\ \mathbb{0} & \widetilde{T} \end{bmatrix},$$

which gives the Schur decomposition of A.

As shown by this proof, the Schur decomposition is *not unique*: the eigenvalues can appear in the diagonal of T in any possible order. The columns

$$Q = [q_1 \cdots q_n]$$

are called the *Schur vectors* of A. Writing $T = [t_{i,j}]_{i,j}$, for $k = 1,\dots,n$ we have that

$$A\,q_k = \sum_{i=1}^{k} t_{i,k}\,q_i$$

and so the linear subspace $\operatorname{span}(q_1,\dots,q_k)$ is invariant.

The Schur decomposition $A = Q\,T\,Q^*$ allows to compute all the eigenpairs of the matrix. Indeed, the eigenvalues of A coincide with those of T, and for $\lambda \in \mathbb{C}$ and $y \in \mathbb{C}^n \setminus \{0\}$ such that $T\,y = \lambda\,y$ we have that

$$A\,Q\,y = Q\,T\,y = \lambda\,Q\,y$$

and so $Q\,y$ is an eigenvalue of A with eigenvalue λ.

Since T is upper triangular, its eigenvalues coincide with its diagonal entries. For each i, the eigenspace of $\lambda = t_{i,i}$ can be computed as the kernel of the linear map defined by the matrix $A - \lambda\mathbb{1}_n$. For simplicity, suppose that this eigenvalue is *simple*, in the sense that its geometric multiplicity is equal to 1. Write the equation $(T - \lambda\,\mathbb{1}_n)\,y = 0$ as

$$\begin{array}{cc} & \begin{array}{ccc} \scriptstyle i-1 & \scriptstyle 1 & \scriptstyle n-i \end{array} \\ \begin{array}{c} \scriptstyle i-1 \\ \scriptstyle 1 \\ \scriptstyle n-i \end{array} & \begin{bmatrix} T_{1,1} - \lambda\mathbb{1}_{i-1} & T_{1,2} & T_{1,3} \\ \mathbb{0} & 0 & T_{2,3} \\ \mathbb{0} & \mathbb{0} & T_{3,3} - \lambda\mathbb{1}_{n-i} \end{bmatrix} \end{array} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$
$$= \begin{bmatrix} (T_{1,1} - \lambda\mathbb{1}_{i-1})\,y_1 + T_{1,2}\,y_2 + T_{1,3}\,y_3 \\ T_{2,3}\,y_3 \\ (T_{3,3} - \lambda\mathbb{1}_{n-i})\,y_3 \end{bmatrix} = \begin{bmatrix} \mathbb{0} \\ 0 \\ \mathbb{0} \end{bmatrix}.$$

Since λ is simple, both $T_{1,1} - \lambda\mathbb{1}_{i-1}$ and $T_{3,3} - \lambda\mathbb{1}_{n-i}$ are nonsingular, which implies that $y_3 = 0 \in \mathbb{R}^{i-1}$. The component y_2 can be arbitrarily taken as any element of $\mathbb{C}$. We fix it as $y_2 = 1$, which implies

$$y_1 = -(T_{1,1} - \lambda\mathbb{1}_{i-1})^{-1} T_{1,2} \in \mathbb{R}^{n-i}.$$

The computation of this vector amounts to the resolution of an upper triangular system. The resulting eigenvector is

$$y = \begin{bmatrix} -(T_{1,1} - \lambda\mathbb{1}_{i-1})^{-1} T_{1,2} \\ 1 \\ \mathbb{0} \end{bmatrix}.$$

EXAMPLE 7.1.1. For the upper triangular matrix $T = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}$, an eigenvector for the eigenvalue $\lambda = 3$ is given by any nonzero solution of the equation

$$(T - \lambda \mathbb{1}_2) \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} -2 & 2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = 0,$$

for instance $y = (1, 1)$. Hence given a Schur decomposition $A = Q\,T\,Q^*$, the vector $Q\,y = q_1 + q_2$ is an eigenvector of A for $\lambda = 3$.

CHAPTER 8

Computing eigenvalues and eigenvectors

Let A be an $n \times n$ matrix and consider the algorithm:

Algorithm 8.0.1 (QR iteration)

```
$H_0 \leftarrow Q_0^* A Q_0$ *(initialization)*
for $k = 1, 2, \dots$
    $H_{k-1} = Q_k R_k$ *(QR factorization)*
    $H_k \leftarrow R_k Q_k$
```

Since

$$H_k = R_k Q_k = Q_k^* \overbrace{Q_k R_k}^{H_{k-1}} Q_k$$

we have that $A = (Q_0 \cdots Q_k) H_k (Q_0 \cdots Q_k)^*$ and so H_k is unitarily similar to A. In favorable situations, we have that

$$H_k \longrightarrow_{k\to\infty} H \quad \text{upper triangular } \textit{(Schur form)},$$

but this is certainly less obvious.

To motivate this method and understand its convergence, we will first study two other eigenvalue iterations: the power method and the orthogonal iteration.

8.1. The power method

This is the iteration:

Algorithm 8.1.1 (Power method)

```
Choose $x_0 \in \mathbb{C}^n$ with $\|x_0\|_2 = 1$
for $k = 0, 1, 2, \dots$
    $y_{k+1} \leftarrow A x_k$
    $x_{k+1} \leftarrow \dfrac{y_{k+1}}{\|y_{k+1}\|_2}$
```

When stopping the iteration at an integer l we set

$$(\widetilde{x}, \widetilde{\lambda}) \leftarrow (x_{l+1}, x_{l+1}^* A x_{l+1})$$

for the obtained approximate eigenpair.

To analyze this algorithm, suppose that A is diagonalizable, that is $A = S \Lambda S^{-1}$ with

$$S = [s_1 \cdots s_n] \quad \text{and} \quad \Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_n),$$

so that s_i is the eigenvalue of A for λ_i for each i. Suppose also that

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|.$$

To this end, write the initial vector as a linear combination of these eigenvectors

$$x_0 = a_1\, s_1 + \cdots + a_n\, s_n$$

and suppose furthermore that $a_1 \neq 0$. Then for $k \geq 1$ we have that

$$A^k x_0 = a_1\, \lambda_1^k\, s_1 + \cdots + a_n\, \lambda_n^k s_n = a_1\, \lambda_1^k \Big(s_1 + \sum_{j=2}^{n} \frac{a_j}{a_1} \Big(\frac{\lambda_j}{\lambda_1}\Big)^k s_j \Big).$$

By construction, we have that $x_k = A^k x_0 / \|A^k x_0\|_2$ and so

$$\|x_k - s_1\|_2 \leq O\Big(\Big|\frac{\lambda_2}{\lambda_1}\Big|^k\Big).$$

It can also be shown that for $\widetilde{\lambda}_k = x_k^* A\, x_k$ we have that

$$\|\widetilde{\lambda}_k - \lambda_1\|_2 \leq O\Big(\Big|\frac{\lambda_2}{\lambda_1}\Big|^k\Big).$$

Thus the number of correct digits in base β of these approximations can be estimated as

$$-\log_\beta \|x_k - s_1\|_2, -\log_\beta \|\widetilde{\lambda}_k - \lambda_1\|_2 \geq k\, \log_\beta \Big(\Big|\frac{\lambda_1}{\lambda_2}\Big|^k\Big) + \text{constant}.$$

The assumption that $a_1 \neq 0$ is a very minor drawback of this method. If x_0 is random, this assumption holds with very high probability. Even if this is not the case, the roundoff errors produces during the iteration ensure a nonzero component in the direction of s_1. Moreover, many times we do have a reasonable guess for x_0 which is not far from an the actual eigenvector, and so its component in its direction is nonzero.

On the other hand, a major drawback of this method is that it only converges to an eigenpair when there is a dominant one, and if this is the case, then the convergence is only linear with ratio depending on the quotient $\frac{|\lambda_1|}{|\lambda_2|}$.

The power method does not converge in many situations, including orthogonal matrices (all eigenvalues have absolute value 1), and real matrices with complex eigenvalues (the eigenvalues come in pairs of conjugate complex numbers).

A clear advantage of this method is that it only uses matrix-vector multiplications, and as a matter of fact it does play an important role in several practical situations.

8.2. The PageRank algorithm

PageRank is the basic algorithm of the Google search machine. It has had a huge influence in the development and structure of the internet, since it determines which kind of information and services are accessed more often.

There are three main steps for ranking web pages:

(1) Crawl the web and locate all public pages,
(2) Index the data from (1) to allow the search of keywords and phrases within it,
(3) Rate the importance of these pages.

We assume that (1) and (2) are given, and focus on (3). How can we define and quantify the "importance" of these pages?

Suppose that our web of interest contains n pages, each of them indexed by an integer $1 \leq k \leq n$. This web can be interpreted as a *directed graph*, where the pages correspond to the nodes and the links to the arrows. A typical example is illustrated by Figure 8.2.1.

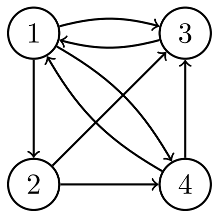


FIGURE 8.2.1. A web with four nodes

Our aim is to define a *score vector*

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n_{\geq 0}$$

deciding the relative importance of the pages: for each k the component $x_k \geq 0$ would indicate the score of the page k, and an inequality $x_j > x_k$ would indicate that the page j is more important than the page k. The score vector is normalized so that $\sum_{k=1}^{n} x_k = 1$.

The web is understood as a "democracy" where pages "vote" for the importance of the other pages by linking to them. The simplest approach would consist in *counting links*. In the example, this would give

$$x_1 = \frac{2}{8}, \quad x_2 = \frac{1}{8}, \quad x_3 = \frac{3}{8}, \quad x_4 = \frac{2}{8}. \tag{8.1}$$

But this forgets an important aspect: receiving a link from the important page should count more. For instance, the pages 1 and 4 have the same number of backlinks (links pointing to them), but the page 1 is linked by the important page 3, whereas the page 4 is linked by the less important page 1.

Another aspect to be taken into account is that a page should not be allowed to increase its influence by increasing its number of outgoing links. If the page j has n_j links, then each of them should contribute with the score x_j/n_j to the page they link. In this way, the overall influence of the page j is its score x_j.

These requirements imply that the the score vector should satisfy the linear equations

$$x_k = \sum_{j \in L_k} \frac{x_j}{n_j} \quad \text{for } k = 1, \dots, n, \tag{8.2}$$

where $L_k \subset \{1, \dots, n\}$ denotes the set of the pages linking to the page k. The corresponding *link matrix* $A = [a_{j,k}]_{j,k} \in \mathbb{R}^{n \times n}$ is defined by

$$a_{j,k} = \begin{cases} \dfrac{1}{n_j} & \text{if the page } j \text{ links to the page } k, \\ 0 & \text{else.} \end{cases} \tag{8.3}$$

Then the system of linear equations in (8.2) is equivalent to $A\,x = x$. In particular, the score vector is an eigenvector of A with eigenvalue 1.

In the example we have

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

and its normalized eigenvector for the eigenvalue 1 is

$$x = \begin{bmatrix} 12/31 \\ 4/31 \\ 9/31 \\ 6/31 \end{bmatrix} = \begin{bmatrix} 0.387 \\ 0.129 \\ 0.290 \\ 0.194 \end{bmatrix}.$$

With this more realistic model, the page 3 (linked by all the others) is less important than the page 1. This apparently contradiction is explained by the fact that the page 3 gives all its vote to the page 1 and together with the link from the page 2, gives the page 1 the highest score.

For simplicity, from now on we will assume that the web has no *dangling nodes*, that is pages without outgoing links. With this assumption, the link matrix A is *column stochastic*: its entries are nonnegative real numbers and each column sums up to 1:

$$\sum_{i=1}^{n} a_{i,j} = 1, \quad j = 1, \dots, n.$$

This condition ensures that the matrix A has $\lambda = 1$ as one of its eigenvalues. Indeed, A and its transposed A^T have the same *characteristic polynomial*:

$$\chi_{A^T} = \det(A^T - t\,\mathbb{1}_n) = \det(A - t\,\mathbb{1}_n)^T = \det(A - t\,\mathbb{1}_n) = \chi_A,$$

where the second equality follows from the fact that the determinant of a matrix equals that of its transpose. The eigenvalues of A and A^T are the zeros of their respective characteristic polynomial, and so they coincide. Moreover the fact that A is column stochastic implies that A^T is row stochastic, and so

$$A^T \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}.$$

Hence $\lambda = 1$ is an eigenvalue of A^T, and so it is also an eigenvalue of A.

There are several other desirable conditions that are needed for this idea to work properly. First of all, we want that the eigenspace $V_1(A)$ has dimension 1, so that it defines at most one normalized eigenvector. Unfortunately, this is not always the case.

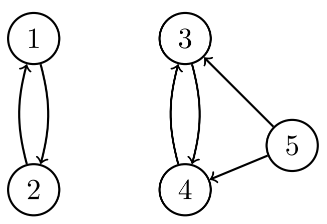


FIGURE 8.2.2. A disconnected web

EXAMPLE 8.2.1. The web in Figure 8.2.2 gives the link matrix

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1/2 \\ 0 & 0 & 1 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Then both

$$x = \begin{bmatrix} 1/2 \\ 1/2 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad y = \begin{bmatrix} 0 \\ 0 \\ 1/2 \\ 1/2 \\ 0 \end{bmatrix}$$

are eigenvectors of A for the eigenvalue 1, and so $V_1(A)$ has dimension at least 2. It is not clear which vector in this subspace should be reasonably used as a score.

The phenomenon observed in Example 8.2.1 always arises when the web is not connected. Indeed, if the web is composed of t subwebs without links between them, then the corresponding link matrix A splits into t blocks, and we have that

$$\dim V_1(A) \geq t.$$

To avoid this undesirable situation, the strategy is to modify the link matrix by adding to it a multiple of the *uniform matrix*

$$S = \left[1/n\right]_{i,j} = \begin{bmatrix} 1/n & \cdots & 1/n \\ \vdots & & \vdots \\ 1/n & \cdots & 1/n \end{bmatrix}. \tag{8.4}$$

We then set $M = (1-\alpha)\, A + \alpha\, S$ for certain parameter $0 \leq \alpha \leq 1$. If $\alpha > 0$ then

$$\dim V_1(M) = 1.$$

The value reportedly used by Google was $\alpha = 0.15$.

EXAMPLE 8.2.2. For the graph in Figure 8.2.1, the modified link matrix M gives the scores

$$x_1 = 0.368, \quad x_2 = 0.142, \quad x_3 = 0.288, \quad x_4 = 0.202.$$

These values are slightly different to those in (8.1) but they give the same order of importance to the pages.

On the other hand, for the graph in Figure 8.2.2 it gives

$$x_1 = 0.2, \quad x_2 = 0.2, \quad x_3 = 0.285, \quad x_4 = 0.285, \quad x_5 = 0.003,$$

allowing to compare the different pages.

The following is the result that justifies this method [**Eld04**]: let

$$\lambda_1 = 1, \lambda_2, \dots, \lambda_n$$

be the eigenvalues of A, repeated according to their algebraic multiplicities, that is the exponent of the corresponding linear factor in the characteristic polynomial of A. Then

$$|\lambda_i| \leq 1 \quad \text{for all } i \tag{8.5}$$

and the eigenvalues of M are

$$\lambda_1 = 1, (1-\alpha)\,\lambda_2, \dots, (1-\alpha)\,\lambda_n. \tag{8.6}$$

PROOF. For each i choose $x \in \mathbb{R}^n \setminus \{0\}$ such that $A\,x = \lambda_i\,x$. Then

$$\|A\,x\|_1 \geq |\lambda_i|\,\|x\|_1 \tag{8.7}$$

and so

$$\|A\,x\|_1 = \sum_{i=1}^{n} \Big|\sum_{j=1}^{n} a_{i,j}\,x_j\Big| \leq \sum_{j=1}^{n} \Big(\sum_{i=1}^{n} a_{i,j}\Big)\,|x_j| \leq \sum_{j=1}^{n} |x_j| = \|x\|_1 \tag{8.8}$$

because A is column stochastic. From (8.7) and (8.8) we get $|\lambda_i|\,\|x\|_1 \leq \|x\|_1$ and so $|\lambda_i| \leq 1$, as stated in (8.5). For (8.6) set

$$e = \begin{bmatrix} 1/\sqrt{n} \\ \vdots \\ 1/\sqrt{n} \end{bmatrix} \in \mathbb{R}^n$$

Note that this is a unit n-vector such that $S = e\,e^T$. Let U_1 be an $n \times (n-1)$ matrix completing e to an $n \times n$-orthonormal matrix $U = [e\,U_1]$. Then

$$U^T A\,U = \begin{array}{c} \scriptstyle 1 \\ \scriptstyle n-1 \end{array}\!\begin{bmatrix} e^T A \\ U_1^T A \end{bmatrix} \begin{array}{c} \begin{array}{cc} \scriptstyle 1 & \scriptstyle n-1 \end{array} \\ \begin{bmatrix} e & U_1 \end{bmatrix} \end{array} = \begin{bmatrix} e^T \\ U_1^T A \end{bmatrix} \begin{bmatrix} e & U_1 \end{bmatrix} = \begin{array}{c} \scriptstyle 1 \\ \scriptstyle n-1 \end{array}\!\begin{array}{c} \begin{array}{cc} \scriptstyle 1 & \scriptstyle n-1 \end{array} \\ \begin{bmatrix} 1 & e^T U_1 \\ U_1^T A\,e & U_1^T A\,U_1 \end{bmatrix} \end{array} = \begin{bmatrix} 1 & \mathbb{0} \\ w & T \end{bmatrix},$$

where the second equality follows from the fact that A is column stochastic, and the fourth from the fact that e is orthogonal to U_1.

Since $U^T A\,U$ is similar to A, it has the same eigenvalues. Hence the eigenvalues of T are $\lambda_2, \dots, \lambda_n$. We have that

$$U^T e = \begin{array}{c} \scriptstyle 1 \\ \scriptstyle n-1 \end{array}\!\begin{bmatrix} e^T \\ U_1^T \end{bmatrix} e = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^n$$

and so

$$\begin{aligned} U^T M\,U &= (1-\alpha)\,U^T A\,U + \alpha\,U^T e\,e^T U \\ &= (1-\alpha)\begin{bmatrix} 1 & \mathbb{0} \\ w & T \end{bmatrix} + \alpha \begin{bmatrix} 1 \\ \mathbb{0} \end{bmatrix} \begin{bmatrix} 1 & \mathbb{0} \end{bmatrix} = \begin{bmatrix} 1 & \mathbb{0} \\ (1-\alpha)\,w & (1-\alpha)\,T \end{bmatrix}. \end{aligned}$$

Hence the eigenvalues of M are $1, (1-\alpha)\,\lambda_2, \dots, (1-\alpha)\,\lambda_n$, as stated. $\square$

We can then compute the score vector x by applying the power method, to the matrix M starting from a well chosen initial vector x_0 and iterating

$$x_k \leftarrow \frac{M\,x_{k-1}}{\|M\,x_{k-1}\|_1}, \quad k \geq 1.$$

This iterative method converges towards x, the normalized eigenvector of A corresponding to the largest eigenvalue $\lambda = 1$.

In the PageRank algorithm, a reasonable choice for x_0 is the score vector of the previous web. The rate of convergence of the iteration is *linear* and depends on the gap between the largest eigenvalue and the other ones: the number of correct digits in base β of the k-th approximant x_k is bounded below by

$$-\log_\beta \|x - x_k\|_1 \geq k \log_\beta \Big(\frac{1}{|\lambda_2|}\Big) + \text{constant}.$$

In the present situation, Elden's theorem implies that $|\lambda_2| \leq 1 - \alpha = 0.85$. Taking $\beta = 10$ we have that

$$\log_\beta \Big(\frac{1}{|\lambda_2|}\Big) \geq 0.07$$

and so

$$-\log_\beta \|x - x_k\|_1 \geq 0.07\, k + \text{constant}.$$

The bulk in each iteration of the power method is a matrix-vector multiplication. In a practical implementation, this multiplication is computed, for $v \in \mathbb{R}^n_{\geq 0}$ with $\|v\|_1 = 1$ as

$$M\, v = (1-\alpha)\, A\, v + \alpha \begin{bmatrix} 1/n \\ \vdots \\ 1/n \end{bmatrix}$$

because the multiplication of v by the uniform matrix S gives

$$S\, v = \begin{bmatrix} 1/n \\ \vdots \\ 1/n \end{bmatrix}.$$

Since the link matrix A is very sparse, this computation is efficient in spite of the fact that the dimension n can be very large.

8.3. The orthogonal iteration

This is a generalization of the power method aimed to compute higher dimensional invariant subspaces. Given $1 \leq r \leq n$ the algorithm can be written as follows.

Algorithm 8.3.1 (Orthogonal iteration)

```
Choose a unitary $n \times r$ matrix $Q_0$
for $k = 0, 1, 2, \dots$
    $Y_{k+1} \leftarrow A\, Q_k$
    $Y_{k+1} = Q_{k+1}\, R_{k+1}$ (QR factorization)
```

When $r = 1$ the algorithm coincides with the power method, because

$$y_{k+1} = x_{k+1}\, \|A\, y_{k+1}\|_2$$

is the QR factorization of the vector y_{k+1}. Hence in this case $Q_0, Q_1, Q_2, \dots \in \mathbb{C}^{n\times 1}$ coincides with the sequence of unit vectors produced by the power method.

Now let $1 \leq r \leq n$. To analyze this iteration, for simplicity suppose that A is diagonalizable and that $A = S\, \Lambda\, S^{-1}$ with $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ satisfying

$$|\lambda_1| > \cdots > |\lambda_n|. \tag{8.9}$$

For a matrix B denote by $\operatorname{span}(B)$ the linear subspace generated by its columns. Then for all $k \geq 0$ we have that

$$\operatorname{span}(Q_k) = \operatorname{span}(Y_k) = \cdots = \operatorname{span}(A^k Q_0),$$

where the second equality comes from the fact that $Y_k = Q_k R_k$. Hence Q_k gives an orthonormal basis of the linear subspace $\operatorname{span}(A^k Q_0)$, preventing phenomena like underfull, overfull and collapsing dimensions. Under the hypothesis in (8.9), for a generic choice of initial unitary $r \times n$ matrix Q_0 this iteration converges for $k \to +\infty$ to a basis of the invariant subspace generated by the *dominant eigenvectors* $s_1, \dots, s_r$, that is

$$\operatorname{span}(Q_k) = \operatorname{span}(A^k Q_0) \longrightarrow_{k\to\infty} \operatorname{span}(s_1, \dots, s_r), \tag{8.10}$$

see [**Dem97**, Theorem 4.8 in page 158] for details.

Note that the QR factorization is compatible with restricting columns. Namely, let

$$B = P\,S$$

be the QR factorization of an $n \times n$ matrix B. If we respectively denote by B_r and P_r the $n \times r$ matrices composed of the first r columns of B and P, and by R_r the first $r \times r$ submatrix of R, we have that

$$B_r = P_r\,S_r \tag{8.11}$$

is the QR factorization of B_r.

Now for $1 \le r \le n$ complete Q_0 to a unitary $n \times n$ matrix $\widetilde{Q}_0$. The previous observation implies that for each $k \ge 0$ the unitary $r \times n$ matrix Q_{k+1} coincides with the orthogonal iteration $\widetilde{Q}_{k+1}$ of rank n on A starting with $\widetilde{Q}_0$.

Hence this rank n iteration contains all those corresponding to the intermediate ranks, and so we restrict our attention to this case. Then the convergence in (8.10) implies that Q_k converges to a unitary $n \times n$ matrix Q such that $Q_k^* A\,Q_k$ converges to an upper triangular matrix T giving the Schur decomposition

$$A = Q\,T\,Q^*.$$

8.4. The QR iteration

The orthogonal iteration on rank n (Algorithm 8.3.1) produces a sequence of unitary $n \times n$ matrices Q_k, $k \ge 0$, so that

$$H_k = Q_k^*\,A\,Q_k, \quad k \ge 0, \tag{8.12}$$

is a sequence of matrices that are unitarily similar to A and that purportedly converge to the Schur form of this matrix. The QR iteration (Algorithm 8.0.1) arises when considering how to compute this latter sequence without actually computing the Q_k's.

From the definition of Q_k and R_k from the orthogonal iteration we have that the relation $A\,Q_{k-1} = R_k\,Q_k$, from where we deduce that

$$H_{k-1} = Q_{k-1}^* A\,Q_{k-1} = (Q_{k-1}^* Q_k)\,R_k \tag{8.13}$$

$$H_k = Q_k^*\,A\,Q_k = (Q_k^*\,A\,Q_{k-1})\,Q_{k-1}^* Q_k = R_k\,(Q_{k-1}^* Q_k). \tag{8.14}$$

Since $Q_{k-1}^* Q_k$ is unitary and (8.13) is the QR factorization of H_{k-1}, this shows that the QR iteration actually computes the sequence.

This first version of the algorithm has some drawbacks:

- a single iteration costs $O(n^3)$ flops,
- convergence (when it exists!) is only linear.

We will next study how to overcome these practical difficulties.

8.5. The real Schur form and the Hessenberg reduction

Matrices arising from applications have *real* entries. Hence we will concentrate in the real version of the QR iteration: for an $n \times n$ matrix A with real entries we choose an orthogonal $n \times n$ matrix Q_0 and we set

Algorithm 8.5.1 (QR iteration)

```
$H_0 \leftarrow Q_0^T A\, Q_0$ (initialization)
for $k = 1, 2, \dots$
    $H_{k-1} = Q_k\, R_k$ (QR factorization)
    $H_k \leftarrow R_k\, Q_k$
```

If the eigenvalues of A are not real, then H_k cannot converge to an upper triangular matrix. In this case we must content ourselves with convergence to the *real Schur form*:

$$A = Q\, T\, Q^T$$

with Q orthogonal and T block upper triangular with 1×1 and 2×2 diagonal blocks, that is

$$T = \begin{bmatrix} T_{1,1} & T_{1,2} & \cdots & T_{1,n} \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & & T_{n-1,n} \\ 0 & \cdots & 0 & T_{n,n} \end{bmatrix} \in \mathbb{R}^{n \times n}$$

where each $T_{i,i}$ is a 1×1 block or a 2×2 block with complex conjugate eigenvalues.

To implement the QR iteration, we have to choose Q_0 carefully so that

$$H_0 = Q_0^T A\, Q_0$$

is an upper Hessenberg matrix, that is, such that $h_{i,j} = 0$ whenever $i > j + 1$. In this way, the complexity of each iteration is lowered to $O(n^2)$ flops.

This *Hessenberg reduction process* is a variation of the QR factorization, and can be done with a sequence of Householder reflections. We illustrate it in the 5×5 case:

(1) Choose $P_1 = \begin{bmatrix} 1 & \mathbb{0} \\ \mathbb{0} & \widetilde{P}_1 \end{bmatrix}$ with $\widetilde{P}_1$ a Householder 4×4 matrix such that

$$P_1 A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} \quad \text{and} \quad A_1 = P_1 A\, P_1^T = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix}$$

so that P_1 leaves the first row of $P_1 A$ unchanged and P_1^T leaves the first column of $(P_1\, A)\, P_1^T$ unchanged, including the zeros.

(2) Choose $P_2 = \begin{bmatrix} \mathbb{1}_2 & \mathbb{0} \\ \mathbb{0} & \widetilde{P}_2 \end{bmatrix}$ with $\widetilde{P}_2$ a Householder 3×3 matrix such that

$$P_2 A_1 = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{bmatrix} \quad \text{and} \quad A_2 = P_2 A_1 P_2^T = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{bmatrix}$$

so that P_1 leaves the first and second rows of $P_2 A_1$ unchanged and P_2^T leaves the first and second columns of $(P_2 A_2)\, P_2^T$ unchanged.

(3) Finally choose $P_3 = \begin{bmatrix} \mathbb{1}_3 & \mathbb{0} \\ \mathbb{0} & \widetilde{P}_3 \end{bmatrix}$ with $\widetilde{P}_2$ a Householder 2×2 matrix such that

$$P_3 A_2 = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix} \quad \text{and} \quad A_3 = P_3 A_2 P_3^T = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}.$$

In general, this procedure constructs an orthogonal $n \times n$ matrix

$$Q_0 = P_{n-2} \cdots P_1$$

such that $H_0 = Q_0^T A\, Q_0$ is Hessenberg. The complexity of computing it is $5\, n^3 + O(n^2)$ flops.

Now we turn to each QR step. It is important to note that the Hessenberg form is preserved during the QR iteration, so that if the initialization matrix H_0 is Hessenberg, this is also the case for all the H_k's.

To see this, let H be a Hessenberg $n \times n$ matrix and $H = Q\, R$ its QR factorization, and set

$$H^+ = R\, Q. \tag{8.15}$$

Hence $Q = H\, S$ with $S = R^{-1}$ and so for each j we have that

$$\mathrm{col}_j(Q) = s_{1,j}\, \mathrm{col}_1(H) + \cdots + s_{n,j}\, \mathrm{col}_n(H).$$

Now since R is upper triangular, this is also the case for S. Hence $s_{i,j} = 0$ when $i > j$ and so $\mathrm{col}_j(Q)$ is a linear combination of the first j columns of H, which implies that Q is also Hessenberg.

In turn, the factorization (8.15) implies that for each i we have that

$$\mathrm{row}_i(H^+) = r_{i,1}\, \mathrm{row}_1(Q) + \cdots + r_{i,n}\, \mathrm{row}_n(Q).$$

Since R is upper triangular we have that $r_{i,j} = 0$ whenever $j < i$, and so $\mathrm{row}_i(H_+)$ is a linear combination of the last i rows of Q. Since Q is Hessenberg this is also the case for H_+, as stated.

The factorization $H = Q\, R$ is better computed with $n - 1$ Givens rotations, that is

$$Q = G_1 \cdots G_{n-1}$$

with $G_i = \mathrm{Givens}(i, i+1, \theta_i)$. Then

$$H_+ = R\, Q = R\, (G_1 \cdots G_{n-1}).$$

This procedure requires $6\, n^2 + O(n)$ flops.

8.6. Deflation

A Hessenberg matrix $H \in \mathbb{R}^{n \times n}$ is said to be *reduced* if it has a zero subdiagonal entry, that is when there is $0 < p < n$ such that

$$H = \begin{array}{c c} & \begin{array}{cc} p & n-p \end{array} \\ \begin{array}{c} p \\ n-p \end{array} & \begin{bmatrix} H_{1,1} & H_{2,2} \\ \mathbb{0} & H_{2,2} \end{bmatrix} \end{array}.$$

When this is the case, the eigenproblem problem for H decouples into the eigenproblems problems for the smaller matrices $H_{1,1}$ and $H_{2,2}$ *(deflation).*

In practice, this phenomenon arises when the subdiagonal entry $h_{p+1,p}$ is small enough to be considered numerically zero, for instance when

$$|h_{p+1,p}| \leq c\,\varepsilon\,(|h_{p,p}| + |h_{p+1,p+1}|)$$

for a small constant c and where ε denotes the machine epsilon. As we will see later on, this typically occurs for $p = n-1$ and $p = n-2$.

8.7. The shifted QR iteration

For $\mu \in \mathbb{R}$ consider the iteration

Algorithm 8.7.1 (single shift QR iteration)

```
$H_0 \leftarrow Q_0^T A\, Q_0$ (Hessenberg reduction)
for $k = 1, 2, \dots$
    $H_{k-1} - \mu\, \mathbb{1}_n = Q_k\, R_k$ (QR factorization)
    $H_k \leftarrow R_k\, Q_k + \mu\, \mathbb{1}_n$
```

Each H_k is Hessenberg and orthogonally similar to A because

$$H_k = R_k\, Q_k + \mu\, \mathbb{1}_n = Q_k^T (Q_k\, R_k + \mu\, \mathbb{1}_n)\, Q_k = Q_k^T H_k\, Q_k.$$

If we shift by an eigenvalue $\mu \in \lambda(A)$, then deflation occurs in one step [**GVL13**, Theorem 7.5.1], that is

$$h_{n,n-1}^{(1)} = 0 \quad \text{and} \quad \widetilde{h}_{n,n}^{(1)} = \lambda$$

for the matrix $H_1 = [h_{i,j}]_{i,j}$.

In practice, we recognize that the QR iteration is converging when $h_{n,n-1}$ is sufficiently small. We then use the shift $\mu = h_{n,n}$, which gives a *quadratically* converging algorithm. More precisely, if

$$|h_{n,n-1}^{(k)}| \leq \eta\, \|H\| \qquad \text{with } 0 \leq \eta \ll 1$$

then for $\mu = h_{n,n}^{(k)}$ we have that

$$|h_{n,n-1}^{(k+1)}| \leq O(\eta^2 \|H\|),$$

and so the number of correct digits of the (n,n)-th entries $h_{n,n}^k \approx \lambda \in \lambda(A)$ doubles at each step.

Bibliography

[Dem97] J. W. Demmel, *Applied numerical linear algebra*, SIAM, 1997.

[Eld04] L. Eldén, *A note on the eigenvalues of the Google matrix*, e-print arXiv:math/0401177, 2004.

[GVL13] G. H. Golub and Ch. F. Van Loan, *Matrix computations*, fourth ed., Johns Hopkins Stud. Math. Sci., Johns Hopkins Univ. Press, 2013.