NOTES ON A BASIC (PARTIAL) COURSE ON NUMERICAL LINEAR ALGEBRA

FERNANDO DE TERÁN[†]

1. PRESENTATION OF THE COURSE. The main four problems in numerical linear algebra (NLA) are the following (we set $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$):

1. Solution of systems of linear equations (SLE): Compute x such that:

$$Ax = b, \qquad \text{with } A \in \mathbb{F}^{n\times n}, b \in \mathbb{F}^n.$$

2. Least squares problems (LSP): Compute:

$$\min_x \|Ax - b\|_2 \qquad A \in \mathbb{F}^{m\times n}, b \in \mathbb{F}^n.$$

3. Eigenvalue problems (EP): Compute eigenvalues (and eigenvectors) of matrices, namely, pairs (λ, v) such that:

$$Av = \lambda v, \qquad \lambda \in \mathbb{F}, v \in \mathbb{F}^n, A \in \mathbb{F}^{n\times n}.$$

4. Compute the *singular value decomposition* (SVD) of a matrix $A \in \mathbb{F}^{m\times n}$:

$$A = U\Sigma V^*,$$

with

$$U, V \text{ being } \begin{cases} \text{unitary (if } \mathbb{F} = \mathbb{C}\text{), or} \\ \text{orthogonal (if } \mathbb{F} = \mathbb{R}\text{)} \end{cases},$$

and

$$\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_r, 0, \ldots, 0).$$

The values $\sigma_1 \geq \ldots \geq \sigma_r > 0$ are the *singular values of* A.

(For the notion of unitary and orthogonal matrix see Definition 4.1)

1.1. Background. Throughout this course we will use the following notions and notations.

Given a matrix $A \in \mathbb{C}^{m\times n} = [a_{ij}]$, the notation $|A| := [|a_{ij}|]$ stands for the matrix whose entries are the absolute values of the entries of A, $A^\top$ denotes the transpose of A, and A^* denotes its conjugate transpose. The identity $n \times n$ matrix is denoted by I_n. Sometimes we introduce some specific notation for the entries of a matrix, like above, but otherwise we denote by $A(i, j)$ the (i, j) entry of A.

1.1.1. Matrix norms and spectral radius. Matrix norms will be used throughout this course. For a nice introduction to this issue we refer to [6, Lecture 3].

DEFINITION 1.1. (Consistent family of matrix norms) *A map* $\|\cdot\| : \mathbb{C}^{m\times n} \longrightarrow \mathbb{R}$ *is a* consistent family of matrix norms *if is satisfies the following properties:*

(i) $\|A\| \geq 0$*, and* $\|A\| = 0$ *if and only if* $A = 0$.

[2]Departamento de Matemáticas, Universidad Carlos III de Madrid, Avenida de la Universidad 30, 28911 Leganés, Spain (fteran@math.uc3m.es)

(ii) $\|\alpha A\| = |\alpha| \cdot \|A\|$, *for any* $A \in \mathbb{C}^{m\times n}$ *and* $\alpha \in \mathbb{C}$.
(iii) $\|A + B\| \leq \|A\| + \|B\|$, *for any* $A, B \in \mathbb{C}^{m\times n}$.
(iv) $\|AB\| \leq \|A\| \cdot \|B\|$, *for any* $A \in \mathbb{C}^{m\times n}, B \in \mathbb{C}^{n\times p}$.

EXAMPLE 1. *The standard matrix norms in this course are the following: Let* $[x_{ij}] \in \mathbb{C}^{m\times n}$. *Then:*

- *The* 1*-norm:* $\|[x_{ij}]\|_1 := \max\limits_{j=1,\ldots,n} \sum\limits_{i=1}^{m} |x_{ij}|$.
- *The* infinite norm*:* $\|[x_{ij}]\|_\infty := \max\limits_{i=1,\ldots,m} \sum\limits_{j=1}^{n} |x_{ij}|$.
- *The* Frobenius norm*:* $\|[x_{ij}]\|_F := \sqrt{\sum_{ij} |x_{ij}|^2}$.
- *The* 2*-norm (or* spectral norm*):*

$$\|X\|_2 := \max\{\sqrt{|\lambda|} : \ \lambda \text{ is an eigenvalue of } X^*X\}.$$

These maps define also norms in $\mathbb{C}^n$, *since* $\mathbb{C}^n \equiv \mathbb{C}^{n\times 1}$. *Note, in particular, that if* $x = \begin{bmatrix} x_1 & \ldots & x_n \end{bmatrix}^\top \in \mathbb{R}^n$, *then* $\|x\|_F = \sqrt{|x_1|^2 + \cdots + |x_n|^2} = \|x\|_2$ *(the standard Euclidean norm). It is also straightforward to see that this also coincides with the spectral norm.*

Note also that $\|[x_{ij}]\|_1$ *is the maximum* 1*-norm of the columns of* $[x_{ij}]$, *and that* $\|[x_{ij}]\|_\infty$ *is the maximum* 1*-norm of the rows of* $[x_{ij}]$.

The reason to term the map $\|\cdot\|$ in Definition 1.1 as a "family" of norms is because we want to introduce a map valid for **all values** m, n, so it is not a single map from a single space $\mathbb{C}^{m\times n}$, but a "family" of maps from infinitely many spaces. In particular, property (iv) in Definition 1.1 is valid for all matrices A, B such that the product is well-defined.

An *absolute norm* is a matrix norm $\|\cdot\|$ such that $\|A\| = \||A|\|$, for any $A \in \mathbb{C}^{m\times n}$.

EXERCISE 1. *Which of the norms in Definition* 1 *are absolute norms?*

DEFINITION 1.2. (Spectral radius). *The* spectral radius *of the matrix* $A \in \mathbb{C}^{n\times n}$ *is the nonnegative real number:*

$$\rho(A) := \max\{|\lambda| : \quad \lambda \text{ is an eigenvalue of } A\}.$$

The spectral radius is not a matrix norm, but is closely related to matrix norms. In particular, we have the following result.

LEMMA 1.3. *Let* $\|\cdot\|$ *be a matrix norm and* $A \in \mathbb{C}^{n\times n}$. *Then* $\rho(A) \leq \|A\|$.

Proof. By definition, there is some eigenvalue λ of A such that $|\lambda| = \rho(A)$. By definition of eigenvalue, there is some $v \neq 0$ such that $Av = \lambda v$. Then $\|Av\| = \|\lambda v\|$ and, using properties (ii) and (iv) in Definition 1.1, we arrive at $|\lambda| \cdot \|v\| \leq \|A\| \cdot \|v\|$, and, since $v \neq 0$, this implies $\rho(A) = |\lambda| \leq \|A\|$. □

The spectral radius is a relevant notion in matrix analysis and applied linear algebra. Among its many properties, we recall the following one, that will be used later.

LEMMA 1.4. *If* $\|A\| < 1$ *then the matrix* $I + A$ *is invertible and*

$$\|(I + A)^{-1}\| \leq \frac{1}{1 - \|A\|}.$$

Proof. By contradiction, if $I + A$ is not invertible then 0 is an eigenvalue of $I + A$, so there is a nonzero vector $v \in \mathbb{C}^n$ such that $Av = -v$. But this implies that $\lambda = -1$ is an eigenvalue of A, in contradiction with the hypothesis that $\rho(A) \leq \|A\| < 1$.

To prove the second claim, notice that, for each $N \geq 0$,

$$(I + A) \cdot (I - A + A^2 + \cdots + (-1)^N A^N) = I + (-1)^N A^{N+1},$$

so

$$(I + A)^{-1}(I + (-1)^N A^{N+1}) = \sum_{k=0}^{N} (-1)^k A^k.$$

From this we get

$$(I + A)^{-1} = (-1)^{N+1}(I + A)^{-1} A^{N+1} + \sum_{k=0}^{N} (-1)^k A^k.$$

Taking norms, and using properties (ii)–(iv) in Definition 1.1, we arrive at

$$\|(I + A)^{-1}\| \leq \|(I + A)^{-1}\| \cdot \|A\|^{N+1} + \sum_{k=0}^{N} \|A\|^k.$$

When $N \to \infty$, $\|A\|^N \to 0$, because $\|A\| < 1$, so

$$\|(I + A)^{-1}\| \leq \sum_{k=0}^{\infty} \|A\|^k = \frac{1}{1 - \|A\|},$$

and this concludes the proof. □

1.1.2. Basic notions on matrices. In Lesson 2 we will use the following notion.

DEFINITION 1.5. (Triangular matrix). *A matrix $A = [a_{ij}] \in \mathbb{F}^{n \times n}$ is said to be:*

- upper triangular *if $a_{ij} = 0$ for $i > j$,*
- lower triangular *if $a_{ij} = 0$ for $i < j$,*
- triangular *if it is either upper triangular or lower triangular.*

We will use the following elementary properties on the inverses of triangular matrices.

LEMMA 1.6. *If A is an invertible lower (respectively, upper) triangular matrix, then A^{-1} is also a lower (resp., upper) triangular matrix. Moreover, if all diagonal entries of A are equal to 1, then all diagonal entries of A^{-1} are equal to 1.*

EXERCISE 2. *Prove Lemma 1.6.*

Permutations matrices are used in Lesson 3.

DEFINITION 1.7. (Permutation matrix). *A* permutation matrix *is an $n \times n$ matrix which is obtained by permuting the rows of the $n \times n$ identity matrix.*

It is interesting to notice that a permutation matrix P is obtained after permuting some rows of I_n, but also by permuting some columns of I_n. Actually, the column permutation and the row permutation of I_n are the same. Related to this, if P is a

$n \times n$ permutation matrix, then PI_n is obtained from I_n by permuting some rows, and I_nP is obtained from I_n by permuting exactly the same columns, since $PI_n = I_nP$.

Some elementary properties of permutation matrices that will be used in this course are listed in the following lemma, whose proof is straightforward.

LEMMA 1.8. *Let P, P_1, P_2 be $n \times n$ permutation matrices and X an arbitrary $n \times n$ matrix. Then:*

(i) *PX is obtained from X by permuting the same rows as in PI_n.*
(ii) *XP is obtained from X by permuting the same columns as I_nP.*
(iii) $P^{-1} = P^\top$.
(iv) $\det P = \pm 1$.
(v) *P_1P_2 is also a permutation matrix.*

1.2. Some indications about the content. This course includes some exercises, which are either theoretical or numerical (or both). Those requiring numerical computations with a computer are indicated with the symbol  .

1.3. Some comments on the bibliography. For Lesson 2 we have mainly followed [3] and [6]. Lesson 3 is mainly based on [3], and some parts are also taken from [6]. As for Lesson 4, we have essentially followed [4] and [6]. The remaining references are excellent basic textbooks on numerical linear algebra and matrix computations.

2. Lesson 1: Conditioning, floating point arithmetic, and stability**.** Though some of the notions introduced in this lesson are valid also for complex number, we will focus on the set of real numbers (namely, $\mathbb{F} = \mathbb{R}$). In particular, in Section 2.2, we only look at real arithmetic. Of course, all computers are able to deal with complex numbers, but this in the end is based on real arithmetic. Though the analysis for complex arithmetic does not require too much additional effort, we omit it for two reasons. The first one is a time restriction, and the second one is that it is not frequent in the literature to find an explicit numerical analysis on complex arithmetic for backward error and conditioning of elementary algorithms and problems in NLA.

2.1. Conditioning. When working with computers, we can not be sure that the original data are the exact ones. When introducing our (hopefully exact) data in the computer, these data are affected by rounding errors (see Section 2.2).

We can think of a **problem** as a function from a set of data (X) to a set of solutions (Y), where each set is endowed with a norm.

Problem f			
$f:$	X	$\longrightarrow$	Y
	x (data)	$\mapsto$	$f(x)$ (solution)

We will refer to the elements in X as "points", even though X is not necessarily an affine space. Hence, you can replace sentences like "a given point $x \in X$" by "a given data $x \in X$". Moreover, for simplicity we will write $\|\cdot\|$ for both norms in X and Y, even though they are not necessarily the same.

In this setting, x denotes the exact data, which are affected by some perturbation, δx. These perturbations may come not only from rounding errors, but also from uncertainty in the measurements, or from any other source of error. Then, the error in the solution is given by

$$\delta f := f(x + \delta x) - f(x) \tag{2.1}$$

☞ We want to compare δf with δx, where δf is as in (2.1).

Definition 2.1. (Normwise absolute condition number) *The* condition number *of the problem f at a given point $x \in X$ is the non-negative number*

$$\widehat{\kappa}_f(x) := \lim_{\delta \to 0} \sup_{\|\delta x\| \leq \delta} \frac{\|\delta f\|}{\|\delta x\|}, \tag{2.2}$$

with δf as in (2.1).

Remark 1. *Some observations about Definition* 2.1 *are in order:*

(a) The condition number $\widehat{\kappa}_f(x)$ compares absolute changes in the solutions with absolute changes in the data.

(b) In general, to get an acceptable measure of the condition number $\widehat{\kappa}_f(x)$ it is not necessary to take the limit, but just to get

$$\widehat{\kappa}_f(x) \approx \sup_{\|\delta x\| \leq \delta} \frac{\|\delta f\|}{\|\delta x\|},$$

for δ small enough.

(c) If f is a differentiable function of the data $x \in X$, then

$$\widehat{\kappa}_f(x) = \|J_f(x)\|,$$

where J_f is the Jacobian matrix of f (namely, the linear operator associated to the differential).

If we look at relative variations of the data, and look, accordingly, for relative variations in the solution, then we arrive at the following notion.

DEFINITION 2.2. (Normwise relative condition number) *The* relative condition number *of the problem f at the point $x \in X$ is the nonnegative number:*

$$\kappa_f(x) := \lim_{\delta \to 0} \sup_{\frac{\|\delta x\|}{\|x\|} \leq \delta} \frac{\|\delta f\|/\|f\|}{\|\delta x\|/\|x\|}. \tag{2.3}$$

REMARK 2. *Again, if f is a differentiable function of the data $x \in X$, then we have*

$$\kappa_f(x) = \frac{\|x\|}{\|f(x)\|} \cdot \|J_f(x)\|.$$

☞ One of the central questions in NLA is to measure **condition numbers**. We will analyze the condition number for some of the main four problems of NLA mentioned in Section 1.

We will frequently say that a given problem is *well-conditioned* or *ill-conditioned*, depending on how big is the condition number. There is no a fixed quantity that establishes the difference between these two categories. In general, whether a problem is well-conditioned or ill-conditioned depends on the context. Is it a relative condition number of 10^3 enough to say that a problem is ill-conditioned? There is no a satisfactory answer to this question. What the relative condition number tells us in this case is that a relative error of 1 unit in the data could cause a relative error of 10^3 units in the solutions or, in other words, that the error in the solutions of the problem can be 10^3 times bigger than the error in the data.

REMARK 3. (Choice of the norm). *Condition numbers depend on the norms in X and Y. This is not, in general a problem, and it should be clear from the very beginning which is the norm we are using. In this course, both X and Y will be either $\mathbb{F}^n$ or $\mathbb{F}^{n\times n}$, and the standard norms in these spaces are the norms in Example* 1.

EXAMPLE 2. *Let f be the function (problem) assigning to each nonzero real value its inverse:*

$$\begin{array}{rccc} f: & \mathbb{R}\setminus\{0\} & \longrightarrow & \mathbb{R} \\ & x & \mapsto & f(x) = \dfrac{1}{x}. \end{array}$$

This function is differentiable, so

$$\widehat{\kappa}_f(x) = \frac{1}{|x|^2}.$$

Note that when $x \to 0$, the condition number $\widehat{\kappa}_f(x)$ goes to infinity, so inverting a number close to zero is ill-conditioned in absolute terms..

However, in relative terms

$$\kappa_f(x) = \frac{|x|}{|f(x)|}|f'(x)| = 1,$$

so the problem of inverting nonzero real numbers is well conditioned in relative terms.

EXAMPLE 3. *Let us consider the following particular values in Example 2. Set:* $x = 10^{-6}$ *and* $\widetilde{x} = 10^{-6} + 10^{-10}$. *Then the absolute error in the inversion of* x *is given by:*

$$\frac{\left|\frac{1}{x} - \frac{1}{\widetilde{x}}\right|}{|x - \widetilde{x}|} = \frac{1}{10^{-12} + 10^{-16}} \approx 10^{12},$$

which is a big number. However, the relative error is

$$\frac{\left|\frac{1}{x} - \frac{1}{\widetilde{x}}\right| \div \left|\frac{1}{x}\right|}{\frac{|x-\widetilde{x}|}{|x|}} = \frac{|x|}{|\widetilde{x}|} \approx 1.$$

The reason for such a difference is that

$$\left|\frac{1}{x} - \frac{1}{\widetilde{x}}\right| = 10^6 - \frac{10^6}{1 + 10^{-4}} \approx 10^2,$$

which is a big quantity, compared just with the original data, which are of order 10^{-6}. *However, it is a small error compared to the magnitude of* $1/x$.

In the following, for the sake of simplicity, and since the function f is clear by the context, we will replace the subscript f in the notation for condition numbers by a reference to the norm. Then, for instance, $\widehat{\kappa}_\infty$ is the absolute condition number in the ∞-norm, and κ_1 is the relative condition number in the 1-norm.

EXAMPLE 4. *Let us now consider the function assigning to each pair of real numbers their difference:*

$$\begin{array}{rccc} f: & \mathbb{R}^2 & \longrightarrow & \mathbb{R} \\ & \begin{bmatrix} x \\ y \end{bmatrix} & \mapsto & f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = x - y. \end{array}$$

Again, this function is differentiable, and

$$\widehat{\kappa}_\infty\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \left\|J\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)\right\|_\infty = \left\|\begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix}\right\|_\infty = \|\begin{bmatrix} 1 & -1 \end{bmatrix}\|_\infty = 2,$$
$$\widehat{\kappa}_1\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = 1.$$

However

$$\kappa_1\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \frac{\left\|\begin{bmatrix} x \\ y \end{bmatrix}\right\|_1}{|x - y|}\left\|J\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)\right\|_1 = \frac{|x| + |y|}{|x - y|}.$$

Therefore, $\kappa_1\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)$ *can be arbitrarily large if* $x \approx y$ *and* x, y *are not close to zero.*

Example 4 highlights the well-know fact that subtracting nearby numbers can be problematic. But note that this just gives problems in relative terms, since the absolute condition number is not big. Summarizing:

☞ The problem of subtracting two numbers is **relatively ill–conditioned** when these two numbers are close to each other (and not close to zero), though it is absolutely well-conditioned. The effect produced by ill-conditioning when subtracting two nearby numbers is known as *cancellation.*

EXERCISE 3. *Let* $x = 99, y = 100, x+\delta x = 99, y+\delta y = 101$*, and set* $f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = x - y$.

(a) Compute $\widehat{\kappa}_1\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)$.

(b) Compute $\kappa_1\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)$.

(c) Compute the absolute error: $\dfrac{\left\| f\left(\begin{bmatrix} x+\delta x \\ y+\delta y \end{bmatrix}\right) - f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)\right\|_1}{\left\|\begin{bmatrix} \delta x \\ \delta y \end{bmatrix}\right\|_1}$.

(d) Compute the relative error: $\dfrac{\left\| f\left(\begin{bmatrix} x+\delta x \\ y+\delta y \end{bmatrix}\right) - f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)\right\|_1 \Big/ \left\| f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)\right\|_1}{\left\|\begin{bmatrix} \delta x \\ \delta y \end{bmatrix}\right\|_1 \Big/ \left\|\begin{bmatrix} x \\ y \end{bmatrix}\right\|_1}$.

(e) Compare the results obtained in (c) and (d).

In the previous examples we have included the absolute condition number just for the sake of comparison. However, it is not illustrative, because it does not take into account the magnitude of the original data or the solution. In the following, we will only focus on the relative condition number. Nonetheless, the next example shows that this is not the whole story, and that the normwise relative condition number is not always the best way to get an idea on how much sensible is the problem to realistic perturbations.

EXAMPLE 5. *Let* f *be the function*

$$\begin{array}{rccl} f: & \mathbb{R}^2 \setminus \left\{\begin{bmatrix} x \\ 0 \end{bmatrix}\right\} & \longrightarrow & \mathbb{R} \\ & \begin{bmatrix} x \\ y \end{bmatrix} & \mapsto & f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \dfrac{x}{y}\,. \end{array}$$

This is again a differentiable function, so

$$\begin{aligned} \kappa_2\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) &= \frac{\left\|\begin{bmatrix} x \\ y \end{bmatrix}\right\|_2}{|x/y|}\left\| J\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)\right\|_2 = \frac{\sqrt{x^2+y^2}}{|x/y|}\left\|\begin{bmatrix} \frac{1}{y} & -\frac{x}{y^2} \end{bmatrix}\right\|_2 \\ &= \frac{x^2+y^2}{|xy|} = \left|\frac{x}{y}\right| + \left|\frac{y}{x}\right|. \end{aligned}$$

Example 5 shows that the division of two real numbers x, y is ill-conditioned when either $|x/y|$ or $|y/x|$ is large, namely, when x, y are quite different to each

other. However, this is not in accordance with the experience in practice. The reason to explain the results from Example 5 is that in this example we are considering all perturbations of x, y, and this is **not realistic**. To be more precise, the usual perturbations in practice are small perturbations of x and y independently. However, in Example 5 we are allowing all kind of perturbations and this, when x, y are very different form each other, may produce unrealistic perturbations. An example of a non-realistic perturbation is the following.

EXAMPLE 5. (Continued). *Set $x = 1, y = 10^{-16}$, and $x + \delta x = 1, y + \delta y = 10^{-10}$. Note that the perturbation in the variable y is huge in relative terms if we just look at y, but compared to the norm of the whole pair (x, y) it is a small perturbation:*

$$\frac{\left\| \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} \right\|_\infty}{\left\| \begin{bmatrix} x \\ y \end{bmatrix} \right\|_\infty} = 10^{-10} - 10^{-16}.$$

However, it produces a big (relative) change in the function f, namely:

$$\frac{\left\| f\left(\begin{bmatrix} x + \delta x \\ y + \delta y \end{bmatrix} \right) - f\left(\begin{bmatrix} x \\ y \end{bmatrix} \right) \right\|_\infty \Big/ \left\| f\left(\begin{bmatrix} x \\ y \end{bmatrix} \right) \right\|_\infty}{\left\| \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} \right\|_\infty \Big/ \left\| \begin{bmatrix} x \\ y \end{bmatrix} \right\|_\infty} = \frac{(10^{16} - 10^{10})/10^{16}}{10^{-10} - 10^{-16}} \approx 10^{10}.$$

EXERCISE 4. *We have seen in Example 2 that inverting a nonzero number is a well-conditioned problem in relative terms. However, in Example 5 we have seen that, even when if we fix the numerator to be 1, the condition number of the division between two real numbers x, y can be arbitrarily large. Which is the reason for this?*

A realistic perturbation of the pair (x, y) in Example 5 would be a small relative perturbation in both x and y, namely, a perturbation of the form $(\delta x, \delta y)$ with both $|\delta x|/|x|$ and $|\delta y|/|y|$ being small.

The normwise relative condition number, however, does not take into account this issue. For this, we need to look at what happens in each coordinate of the data. This leads to the notion of *componentwise relative condition number*, which we are not going to define formally because is beyond the scope of this course but, following Definition 2.3, it can be informally defined as

$$\kappa_f^{(comp)}(x) := \lim_{\delta \to 0} \sup_{E_{rc}(x) \leq \delta} \frac{\|\delta f\|/\|f\|}{E_{rc}(x)}, \tag{2.4}$$

where $E_{rc}(x)$ is some measure of the *componentwise relative error* in the data x. In the case of Example 5, this componentwise relative error is:

$$E_{rc} := E_{rc}\left(\begin{bmatrix} x \\ y \end{bmatrix} \right) = \max\left\{ \frac{|\delta x|}{|x|}, \frac{|\delta y|}{|y|} \right\}.$$

Then, with some elementary manipulations, we get

$$\left| \delta f\left(\begin{bmatrix} x \\ y \end{bmatrix} \right) \right| = \left| \frac{x + \delta x}{y + \delta y} - \frac{x}{y} \right| = \left| \frac{x}{y} \right| \left| \frac{\frac{\delta x}{x} - \frac{\delta y}{y}}{1 + \frac{\delta y}{y}} \right| \leq \left| \frac{x}{y} \right| \frac{2E_{rc}}{1 - E_{rc}},$$

so

$$\frac{\left|\delta f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)\right|}{\left|f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)\right|} \leq 2 \cdot \frac{E_{rc}}{1 - E_{rc}} \approx 2E_{rc}\,.$$

As a consequence, $\kappa_f^{(comp)}\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = 2$, which means that the problem of dividing two real numbers x, y, with $y \neq 0$, is well-conditioned componentwise.

Though componentwise conditioning is relevant in NLA, we will not pay much attention to it, due to time restrictions.

EXERCISE 5.  . *In order to check the effect of cancellation, we are going to compute the value of* e^{-30} *using the Taylor polynomial of the exponential:*

$$e^x \approx \sum_{k=0}^{N} \frac{x^k}{k!}. \tag{2.5}$$

In particular, we look for an approximation to e^{-30} *with absolute error less than* 10^{-10}.

(a) *Use the bound for the residual of the* N*th Taylor expansion for the function* $f(x)$ *around* x_0*:*

$$R_N(x) \leq \frac{|f^{(N+1)}(c)|}{(N+1)!}|x - x_0|^{N+1}, \quad \text{for some } c \text{ with } |x - c| \leq |x - x_0|,$$

to determine which is the smallest value of N *in* (2.5) *that guarantees an approximation (in absolute terms) with error at most* 10^{-10}.

(b) *Write a* `MATLAB` *code to compute* e^{-30} *using the sum* (2.5) *for the value of* N *obtained in* (a)*, and compute it using this routine.*

(c) *Compute the value of* e^{-30} *just typing* `exp(-30)` *in* `MATLAB`*, and compare with the result obtained in* (b)*. Explain what happens and why happens.*

(d) *Compute* e^{-30} *by first computing* e^{30} *using again* (2.5) *with the appropriate value of* N*, and then inverting the result.*

(e) *Compare the result in* (d) *with the ones in* (b)–(c)*. Is this in accordance with your explanations in* (c)*?*

HIGHLIGHTS:

☞ The conditioning of a problem measures the sensitivity of the solution of this problem to (small) changes in the data.

☞ The word "conditioning" makes reference to a problem, and not to an algorithm. Moreover, it is a mathematical property, and it has nothing to do with the computer.

☞ The **normwise condition number** measures the sensitivity of the solution with respect to changes in norm of the set of data. In some cases, it is more appropriate to look at the **componentwise condition number**, which measures the sensitivity of the solution with respect to changes in each of the components of the data, independently.

☞ If the problem, f, is differentiable, then the condition number at a certain data, x, is the **norm of the Jacobian** of f at x.

☞ **Subtracting** two nearby numbers is **ill-conditioned**.

2.2. Floating point arithmetic. Since any machine (computer) has a finite memory, it is only allowed to represent a finite subset of the set of real (or complex) numbers. In this section we provide an overview on some of the basic features of the number system in a computer: How many numbers can a computer store?, how large is the distance between two consecutive numbers?, how does the computer operate with numbers?

The standard number system in the current computers is the **floating point system**. The underlying idea of such a system is that the relative separation between two consecutive numbers in the system is essentially constant through all the system range.

DEFINITION 2.3. (Floating point system). *A* floating point system *(FPS) is a subset $F \subseteq \mathbb{R}$ defined as*

$$F := \{\pm 0.d_1 d_2 \cdots d_t \cdot \beta^e : \ 0 \le d_i \le \beta - 1; \ d_1 \ge 1, \ e_{\min} \le e \le e_{\max}\} \cup \{0\}, \tag{2.6}$$

where

- $\beta \in \mathbb{N}$, $\beta \ge 2$ *is called the* base *of the system;*
- $t \in \mathbb{N}$, $t \ge 1$ *is called the* precision *of the system;*
- $e, e_{\min}, e_{\max} \in \mathbb{Z}$.

Numbers in F are usually called *floating point numbers*, or *machine numbers*.

REMARK 4. *The decimal notation in* (2.6) *stands for the following number:*

$$0.d_1 d_2 \cdots d_t := \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_t}{\beta^t}\,.$$

REMARK 5. *The condition $d_1 \ge 1$ in* (2.6) *guarantees the uniqueness of the representation of any number in the system* (2.6).

EXERCISE 6. (Integer representation of F). *Prove that any number $x \in F$, with F as in* (2.6)*, can be written in the form*

$$x = \pm m\beta^{e-t}, \qquad \text{with} \ \ \beta^{t-1} \le m \le \beta^t - 1. \tag{2.7}$$

Note that the number m in (2.7) is a whole number, and that any whole number in the indicated range provides a number in the system (2.6). Then, Equation (2.7) provides an equivalent expression for numbers in (2.6).

Let us summarize some basic facts about floating point systems.

- The standard floating point systems in most computers are the *IEEE single precision* and *IEEE double precision* systems. Actually, most computers have the double system by default, though the simple can be also chosen (in `MATLAB` just type `single`(a), where a is the number you want to convert to single precision). The basic ingredients of each system are indicated in Table 2.1.
- The floating point system has an additional position for the zero number.
- Any floating point system contains the number 1. It is the first number with exponent $e = 1$.
- The distance between 1 and the next floating point number is β^{1-t}. This distance β^{1-t} is known as the *machine epsilon* (machep).

system	base (β)	precision (t)	$e_{\min}$	$e_{\max}$	machep (β^{1-t})	unit roundoff (β^{-t})
IEEE single	2	24	-126	127	$1.2 \cdot 10^{-7}$	$6 \cdot 10^{-8}$
IEEE double	2	53	-1022	1023	$2.2 \cdot 10^{-16}$	$1.1 \cdot 10^{-16}$

TABLE 2.1
Basic ingredients of IEEE systems

When shifting from base-10 to base-2 representation (or vice versa) it may happen that a number with only a finite amount of nonzero decimals becomes a periodic number. This is, surprisingly, what happens with 0.1.

EXERCISE 7. *Find the expression for the decimal number* 0.1 *in base* 2.

2.2.1. Rounding numbers: the `float` function. When the user inserts a number $x \in \mathbb{R}$ in a computer, the computer automatically rounds it to a number in its floating point system F (as long as x lies within the range of F). We use the standard notation:

$$\texttt{fl}(x) := \text{number in the system (2.6) which is closest to } x.$$

In the case where there are two closest numbers to x, there are several rules to choose one of them, but we are not going to get into these details.

The first natural question to ask is: how far is away the number $\texttt{fl}(x)$ from x? The next result provides a bound for this distance.

THEOREM 2.4. *Let $x \in \mathbb{R}$ be within the range of F. Then*

(a) $\texttt{fl}(x) = x(1+\delta)$, *with* $|\delta| \leq \frac{1}{2}\beta^{1-t}$,

(b) $\texttt{fl}(x) = \dfrac{x}{1+\alpha}$, *with* $|\alpha| \leq \frac{1}{2}\beta^{1-t}$.

Proof. If $x \in \left[\beta^{e-1}, \beta^{e}\right]$, for some $e_{\min} \leq e \leq e_{\max}$, then $|\texttt{fl}(x) - x| \leq \frac{\beta^{e-t}}{2}$, by (2.7). Now:

(a) Let $\delta = \dfrac{\texttt{fl}(x) - x}{x}$, then $|\delta| \leq \dfrac{\beta^{e-t}/2}{\beta^{e-1}} = \dfrac{1}{2}\beta^{1-t}$.

(b) Let $\alpha = \dfrac{x - \texttt{fl}(x)}{\texttt{fl}(x)}$, then $|\alpha| \leq \dfrac{\beta^{e-t}/2}{\beta^{e-1}} = \dfrac{1}{2}\beta^{1-t}$. ∎

The number

$$u := \frac{1}{2}\beta^{1-t},$$

appearing in Theorem 2.4 is known as the **unit roundoff** of the system F. If $\beta = 2$ it is just $u = \beta^{-t}$.

REMARK 6. *Some remarks are in order:*

(a) *The zero number is always rounded to zero.*

(b) *If the number $x \in \mathbb{R}$ does not lie within the range of F, then it is either because it is less (in absolute value) than the smallest nonzero number in the system, or because it is larger than the largest one in F. In the first case,* $\texttt{fl}(x) = 0$ *(this in known as* underflow*), whereas in the second case, the computer does not round it and produces an error message (this is known as* overflow*).*

2.2.2. Absolute and relative separation of numbers in floating point systems. We restrict ourselves in this section to positive numbers, for simplicity. We want to know the absolute and relative distance between two consecutive numbers in F, $x_n < x_{n+1}$. We first analyze the absolute distance.

Let us fix an exponent $e_{\min} \leq e \leq e_{\max}$. Looking at (2.7), we see that the difference between two consecutive numbers with exponent e in (2.6) is

$$x_{n+1} - x_n = \beta^{e-t}.$$

Indeed, the numbers in F with exponent e are of the form

$$\beta^{-1}\beta^e,\ (\beta^{-1} + \beta^{-t})\beta^e,\ (\beta^{-1} + 2\beta^{-t})\beta^e, \ldots, (1 - \beta^{-t})\beta^e.$$

Now, the first number in F with exponent $e+1$ is $\beta^{-1}\beta^{e+1} = \beta^e$, so the difference between this number and the precedent one is

$$x_{n+1} - x_n = \beta^e - (1 - \beta^{-t})\beta^e = \beta^{e-t}.$$

As a consequence, any two consecutive numbers in F between β^{e-1} and β^e are at the same distance, namely $\boxed{\beta^{e-t}}$. This means that, each time we change the exponent by adding up one unit, the absolute separation between floating point numbers increases in β.

Figure 2.1 displays all positive numbers in a floating point system (2.6) with $\beta = 2, t = 3, e_{\min} = -4, e_{\max} = 3$. As can be seen, the more to the right are the numbers, the more separated they are.

Now, let us look at the relative distance. Let us assume that $\beta^{e-1} \leq x_n < x_{n+1} \leq \beta^e$. Then, the relative separation, s_r, between x_n and x_{n+1} satisfies

$$\beta^{-t} = \frac{\beta^{e-t}}{\beta^e} \leq s_r = \frac{x_{n+1} - x_n}{x_n} = \frac{\beta^{e-t}}{x_n} \leq \frac{\beta^{e-t}}{\beta^{e-1}} = \beta^{1-t}.$$

Moreover, the maximum of the relative distance, β^{1-t}, is attained for $x = \beta^{e-1}$ (the smallest number with exponent e), and it decreases until the minimum, β^{-t}, which is attained at $x = (1 - \beta^{-t})\beta^e$ (the largest number with exponent e). This is illustrated in Figure 2.2 for a binary system with precision $t = 24$.

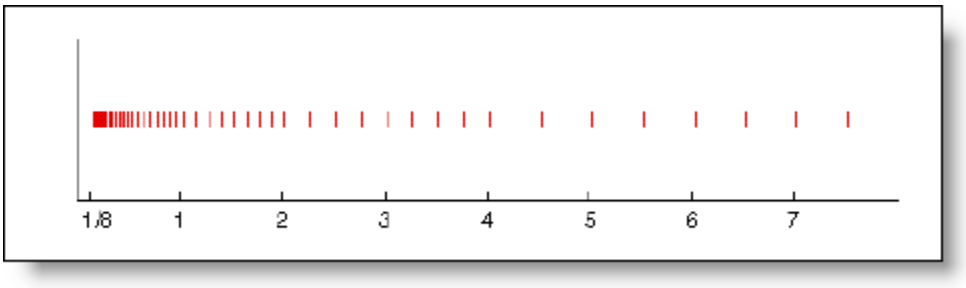

FIG. 2.1. $\beta = 2, t = 3, e_{\min} = -4, e_{\max} = 3$

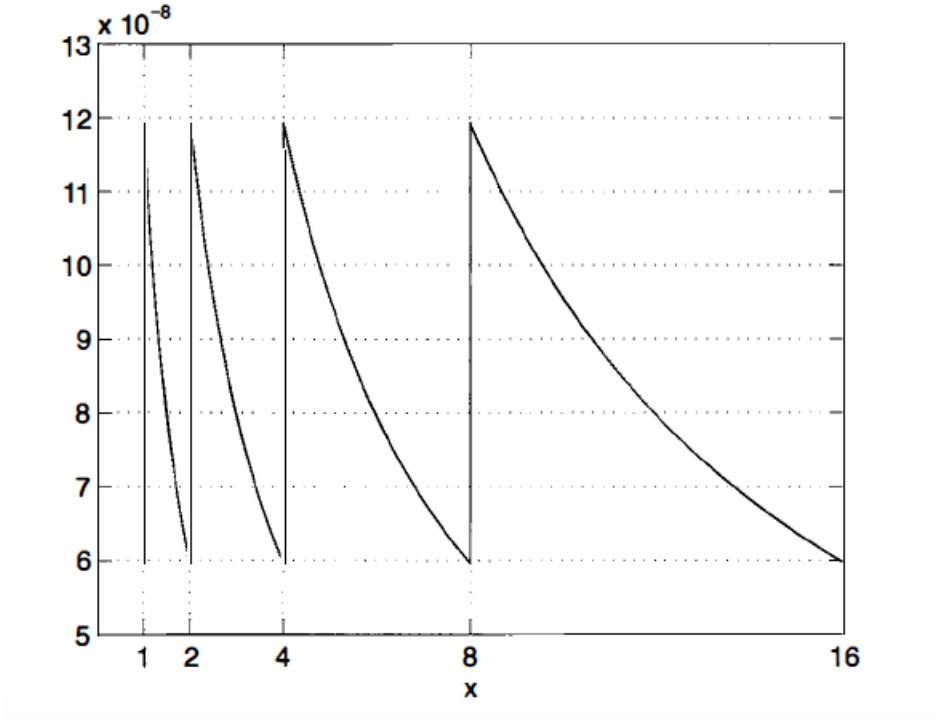

FIG. 2.2. *Relative distance from x to the next larger machine number ($\beta = 2, t = 24$). Taken from* [3, p. 40]

.

2.2.3. Floating point operations. by "floating point operations" we understand the following arithmetic operations: $+$ (sum), $-$ (subtraction), $\times$ (multiplication), $\div$ (division), and $\sqrt{\cdot}$ (square root). Clearly, the system F in (2.6) is not closed under such arithmetic operations. Then, the question that arises is: what does the computer do when the result of operating two numbers in F is not a number in F? The answer is not easy and, essentially, depends on the computer. However, it is possible to model what most computers do as follows.

☞ In the following, we denote by $*$ any of the arithmetic operations mentioned above, and by $\circledast$ its floating counterpart, that is: if $x, y \in F$, then $x \circledast y$ is the output provided

by the computer as $x * y$.

Floating point arithmetic model

Let $x, y \in F$. Then

$$x \circledast y = (x * y)(1 + \delta), \qquad \text{with } |\delta| \leq u, \tag{2.8}$$

where u is the unit roundoff.

Some comments on the floating point arithmetic model are in order:

- Comparing with the natural assumption $x \circledast y = \texttt{fl}(x * y)$, the assumption (2.8) is more pessimistic, because in the case $x * y \in F$, the model does not require $\delta = 0$.
- However, and having in mind the precedent comment, we will sometimes write $\texttt{fl}(x * y)$ instead of $x \circledast y$. This will simplify the developments.
- The model does not tell us which is exactly $x \circledast y$. It only provides a bound on it with respect to the exact value $x * y$. This allows us to deal easily with rounding errors, at the expense of working with unknown quantities (δ).
- Note that the model does not guarantee that the standard arithmetic laws (associative, distributive) are still true in floating point arithmetic. For instance, in general $x \otimes (y \oplus z) \neq (x \otimes y) \oplus (y \otimes z)$, and $x \otimes (y \otimes z) \neq (x \otimes y) \otimes z$.
- In general, we start with two real numbers, $x, y \in \mathbb{R}$ not necessarily in F. In this case, the operation performed by the computer would be $\texttt{fl}(x) \circledast \texttt{fl}(y)$.

EXAMPLE 6. *Let $x, y \in \mathbb{R}^+$. We are going to analyze the error when computing $x + y$ in a computer with unit roundoff u.*

As mentioned before, the computer performs:

$$\texttt{fl}(x) \oplus \texttt{fl}(y) \equiv \texttt{fl}(x + y) = x(1 + u_1) \oplus y(1 + u_2) = [x(1 + u_1) + y(1 + u_2)]\,(1 + u_3),$$

with $|u_1|, |u_2|, |u_3| \leq u$, by (2.8)*. Then*

$$\texttt{fl}(x + y) = x(1 + u_1)(1 + u_3) + y(1 + u_2)(1 + u_3) = x(1 + \delta_1) + y(1 + \delta_2), \tag{2.9}$$

with $|\delta_1|, |\delta_2| \leq 2u + u^2$. As a consequence, $\texttt{fl}(x + y) - (x + y) = x\delta_1 + y\delta_2$, so

$$|\texttt{fl}(x + y) - (x + y)| = |x||\delta_1| + |y||\delta_2| \leq (|x| + |y|)(2u + u^2) = |x + y|(2u + u^2),$$

where in the last equality we have used that x, y are both positive. Equivalently

$$\frac{|\texttt{fl}(x + y) - (x + y)|}{|x + y|} \leq 2u + u^2. \tag{2.10}$$

Equation (2.10) *provides a bound on the relative error when computing the sum of two positive numbers in floating point arithmetic.*

2.3. Computational cost. Another relevant feature of numerical algorithms is their *computational cost* (or computational complexity). This is measured as the number of elementary arithmetic operations. Each floating point operation counts one operation in the computer, termed as *flop*. Then, the computational cost is measured in number of flops. The computational cost of competitive algorithms must be a polynomial in the size of the problem. In the four basic NLA problems mentioned at

HIGHLIGHTS:

☞ A floating point system aims the relative distance between two consecutive numbers to be essentially constant for all numbers in the system.

☞ The relative distance between two consecutive numbers are bounded between the machine epsilon, β^{1-t}, and the unit roundoff, β^{-t}.

☞ The absolute separation increases as the numbers increase in absolute value. It increases by a factor or β as the exponent of the numbers increase by one unit.

☞ The floating point arithmetic model assumes that the relative error when rounding an elementary arithmetic operation of machine numbers is at most the unit roundoff of the computer.

the beginning of this notes, the size of the problem refers to the size of the matrix A and, when it is square, this value is n. We say that an algorithm is, for instance, of order n^3, and denote it by $O(n^3)$, when the amount of operations involved in the algorithm is a cubic polynomial in n. Sometimes we also indicate the coefficient of the leading term of this polynomial and say, for instance, that an algorithm is $O(\frac{2}{3}n^3)$.

2.4. Order of approximation. The "big-O" notation will be also used in this course with a different meaning to the one in the previous section, related to small quantities, instead of big quantities. This notation is relevant to measure the accuracy of the algorithms.

By $O(u)$ we denote a function of u such that, when u approaches 0, there is a quantity C, independent on u, such that $O(u) \leq Cu$. However, the quantity C may depend on other quantities (like the size of the problem).

2.5. Stability of numerical algorithms. Following the developments in Section 2.1, we represent both a problem, f, and an algorithm to solve this problem, $\widetilde{f}$, as functions from a set of data (X) to a set of solutions (Y):

Problem f			
$f:$	X	$\longrightarrow$	Y
	x (data)	$\mapsto$	$f(x)$ (exact solution)

Algorithm $\widetilde{f}$			
$\widetilde{f}:$	X	$\longrightarrow$	Y
	x (data)	$\mapsto$	$\widetilde{f}(x)$ (computed solution)

For a given data $x \in X$, the algorithm computes an approximate solution $\widetilde{f}(x)$ to the exact solution $f(x)$.

☞ The **GOAL** of this section is to compare $f(x)$ and $\widetilde{f}(x)$ in a relative way, that is, to measure the relative error

$$\frac{\|\widetilde{f}(x) - f(x)\|}{\|f(x)\|}. \tag{2.11}$$

The first attempt to introduce a notion of a good algorithm in this respect is the following definition.

DEFINITION 2.5. *The algorithm $\widetilde{f}$ for the problem f is* accurate *if, for any $x \in X$,*

$$\frac{\|\widetilde{f}(x) - f(x)\|}{\|f(x)\|} = O(u). \tag{2.12}$$

However, condition (2.12) is too ambitious, since it is a condition on the algorithm that disregards the conditioning of the problem. More precisely, even in the ideal situation where

$$\widetilde{f}(x) = f(\texttt{fl}(x)), \tag{2.13}$$

then the relative error (2.11) can be large if the problem is ill-conditioned (and this last property has nothing to do with the algorithm). We also want to note that (2.13) means that the only error committed by the computer is the roundoff error, which is an unavoidable assumption.

Tho following definition is more realistic, since it takes into account the conditioning of the problem.

DEFINITION 2.6. *The algorithm $\widetilde{f}$ for the problem f is* stable *if for any $x \in X$ there is some $\widetilde{x} \in X$ satisfying*

$$\frac{\|\widetilde{f}(x) - f(\widetilde{x})\|}{\|f(x)\|} = O(u), \quad \textit{and} \quad \frac{\|x - \widetilde{x}\|}{\|x\|} = O(u).$$

In other words, a stable algorithm produces an approximate solution for an approximate data. We can go even further in this direction.

DEFINITION 2.7. *The algorithm $\widetilde{f}$ for the problem f is* backward stable *if for any $x \in X$ there is some $\widetilde{x} \in X$ satisfying*

$$\widetilde{f}(x) = f(\widetilde{x}), \quad \textit{and} \quad \frac{\|x - \widetilde{x}\|}{\|x\|} = O(u).$$

In other words, a backward stable algorithm provides the **exact** solution for a nearby data. This is the most ambitious requirement for an algorithm, since data are always subject to rounding errors.

Definitions 2.5, 2.6, and 2.7 depend on two objects: (a) the norm in the sets of data and solutions, and (b) the quantity C such that $O(u) \leq Cu$ when u approaches 0. Regarding the norm, in all problems of this course, X and Y are finite dimensional vector spaces. Since all norms in finite dimensional spaces are equivalent, the notions of accuracy, stability, and backward stability do not depend on the norm. As for the quantity C, it must be independent on u, but it will usually depend on the dimension of the spaces X, Y. However, in order for the quantity $O(u)$ not being too large, this dependence is expected to be, at most, polynomial.

A natural question regarding Definitions 2.5, 2.6, and 2.7 is about the relationships between them. For instance: is an accurate algorithm necessarily backward stable, or vice versa? It is immediate that a backward stable algorithm is stable, and that an accurate algorithm is stable. However, is not so easy to see whether the other

implications are true or not. Actually, none of them is true. In particular, an accurate algorithm is not necessarily backward stable.

EXAMPLE 7. *Equation* (2.10) *tells us that the sum of two real numbers of the same sign in floating point arithmetic is an accurate algorithm, and Equation* (2.9) *implies that it is backward stable as well.*

In terms of the goal mentioned at the beginning of this section, one may wonder which is the connection between the stability (or the backward stability) and the relative error (2.11). The following results provides an answer to this question.

THEOREM 2.8. *Let $\widetilde{f}$ be a stable algorithm for the problem $f : X \to Y$, whose relative condition number at the data $x \in X$ is denoted by $\kappa(x)$. Then*

$$\frac{\|\widetilde{f}(x) - f(x)\|}{\|f(x)\|} = O(u\kappa(x)).$$

The proof of Theorem 2.8 for $\widetilde{f}$ being backward stable is immediate (see [6, Th. 15.1]). For f being just stable is more involved, and is out of the scope of this course.

What Theorem 2.8 tells us is that when using a stable algorithm for solving a problem, the forward error of the algorithm depends on the conditioning. In particular, this can be summarized as:

$$\boxed{\widetilde{f} \text{ stable} + f \text{ well-conditioned} \Rightarrow \widetilde{f} \text{ accurate}}$$

HIGHLIGHTS:

☞ Stability is a property of an algorithm used to solve a given problem.

☞ The best property one can expect from an algorithm is to be backward stable, because accuracy is too ambitious due to the presence of rounding errors.

☞ A stable algorithm applied to solve a well-conditioned problem produces an accurate solution.

3. LESSON 2: SOLUTION OF SYSTEMS OF LINEAR EQUATIONS. The main object of this Lesson is a (real) *system of linear equations*, namely:

$$Ax = b, \qquad A \in \mathbb{F}^{n\times n}, \quad b \in \mathbb{F}^n. \tag{3.1}$$

Moreover, we will assume that the system (3.1) has a **unique solution**, so the coefficient matrix A is **invertible**.

The goals of this Lesson are:

1. To review the standard method for solving (3.1).
2. To analyze the conditioning of the problem of solving (3.1).
3. To analyze the backward error of the standard algorithm to solve (3.1).

3.1. Conditioning. In this section, we analyze both the relative normwise and relative componentwise conditioning of the general problem

$$\begin{array}{rccc} f: & \mathbb{F}^{n\times n}\times\mathbb{F}^n & \longrightarrow & \mathbb{F}^n \\ & (A,b) & \mapsto & x, \end{array} \tag{3.2}$$

where $x \in \mathbb{R}^n$ is the solution of (3.1). In other words, the problem f associates to each pair $(A, b) \in \mathbb{R}^{n\times n} \times \mathbb{R}^n$ (data) the solution of the corresponding system (3.1).

3.1.1. Normwise conditioning. We begin with the following result.

THEOREM 3.1. *Let $A \in \mathbb{F}^{n\times n}$ be an invertible matrix and $b, \delta b \in \mathbb{F}^n$. Let $\delta A \in \mathbb{F}^{n\times n}$ be such that $\|A^{-1}\| \cdot \|\delta A\| < 1$, for some matrix norm $\|\cdot\|$. Let x be a solution of* (3.1) *and δx be such that*

$$(A + \delta A)(x + \delta x) = b + \delta b. \tag{3.3}$$

Then

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\|A^{-1}\| \cdot \|A\|}{1 - \|A^{-1} \cdot \delta A\|} \cdot \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right). \tag{3.4}$$

Proof. We first prove that the condition $\|A^{-1}\| \cdot \|\delta A\| < 1$ implies that the matrix $A + \delta A$ is invertible. For this, note that $A + \delta A$ is invertible if and only if $I + A^{-1} \cdot \delta A$ is invertible. From Lemma 1.3 and property (iv) in Definition 1.1 we get $\rho(A^{-1} \cdot \delta A) \leq \|A^{-1} \cdot \delta\| \leq \|A^{-1}\| \cdot \|\delta A\| < 1$, and then Lemma 1.4 implies that $I + A^{-1} \cdot \delta A$ is invertible.

Now, from (3.3) we immediately get $\delta A \cdot x + (A + \delta A)\delta x = \delta b$, which implies $\delta x = (A + \delta A)^{-1}(\delta b - \delta A \cdot x)$, and taking norms and applying the second claim in the statement of Lemma 1.4 we arrive at

$$\|\delta x\| \leq \frac{\|A^{-1}\|}{1 - \|A^{-1} \cdot \delta A\|} \left(\|\delta A\| \cdot \|x\| + \|\delta b\| \right).$$

Now, dividing by $\|x\|$ and using that $\|b\| \leq \|A\| \cdot \|x\|$ in the rightmost summand, we arrive at (3.4). □

DEFINITION 3.2. *The number*

$$\kappa_{\|\cdot\|}(A) = \|A^{-1}\| \cdot \|A\| \tag{3.5}$$

is the condition number for inversion *of the matrix A in the norm $\|\cdot\|$.*

Note that, using (3.5) and $\|A^{-1}\cdot\delta A\|\le\|A^{-1}\|\cdot\|\delta A\|$, inequality (3.4) leads to

$$\frac{\|\delta x\|}{\|x\|}\le\frac{\kappa_{\|\cdot\|}(A)}{1-\kappa_{\|\cdot\|}(A)\cdot\frac{\|\delta A\|}{\|A\|}}\cdot\left(\frac{\|\delta A\|}{\|A\|}+\frac{\|\delta b\|}{\|b\|}\right). \tag{3.6}$$

When $\|\delta A\|/\|A\|$ is small (which should be the case if the perturbation δA is small enough), then

$$\frac{\kappa_{\|\cdot\|}(A)}{1-\kappa_{\|\cdot\|}(A)\cdot\frac{\|\delta A\|}{\|A\|}}\approx\kappa_{\|\cdot\|}(A)$$

so (3.6) is approximately

$$\frac{\|\delta x\|}{\|x\|}\le\kappa_{\|\cdot\|}(A)\cdot\left(\frac{\|\delta A\|}{\|A\|}+\frac{\|\delta b\|}{\|b\|}\right). \tag{3.7}$$

Bound (3.7) indicates that the condition number for inversion of A, $\kappa_{\|\cdot\|}(A)$, is a measure of the condition number for the solution of the general SLE (3.2). If the right-hand side of (3.7) is considered as a normwise measure of the relative error in the data (A,b), then the relative error in the solution is not magnified more than $\kappa_{\|\cdot\|}(A)$ times.

EXERCISE 8. *Set*

$$A=\begin{bmatrix}1 & 1\\ 1+10^{-n} & 1\end{bmatrix},\qquad b=\begin{bmatrix}1+10^{n}\\ 2+10^{n}\end{bmatrix}.$$

(a) Give an explicit expression, depending on n, for the solution, x, of (3.1)*, with A,b as above.*
(b) Compute $\kappa_1(A)$, depending on n.
(c) Use `MATLAB` *to compute the solution of* (3.1)*, with A,b as above, for $n=8,10,12,14$.*
(d) Are your results in accordance with (3.7)*?*
(In order to answer (d) you should know that the unit roundoff by default in `MATLAB`*, which works in IEEE with double precision by default, is $u=10^{-16}$, see Table* 2.1*. This means that the rightmost factor in the right-hand side of* (3.7) *is approximately 10^{-16}).*

3.1.2. Componentwise conditioning. In the following, a matrix inequality $A\le B$, with $A,B\in\mathbb{F}^{n\times n}$ is understood entrywise, that is $a_{ij}\le b_{ij}$, for any entries a_{ij},b_{ij} of A and B, respectively.

Let us assume that the matrix δA satisfies

$$|\delta A|\le\varepsilon|A|, \tag{3.8}$$

for some $\varepsilon>0$, and, for simplicity, we assume $\delta b=0$ (so the right-hand side of (3.1) is not perturbed).

Then

$$A\cdot\delta x=-\delta A(x+\delta x)\Rightarrow\delta x=A^{-1}(-\delta A(x+\delta x))\Rightarrow|\delta x|\le|A^{-1}|(|\delta A|\cdot|x+\delta x|).$$

For the last implication we have used that $|AB| \leq |A| \cdot |B|$, for any pair of matrices $A \in \mathbb{C}^{m\times n}, B \in \mathbb{C}^{n\times p}$. Now, from (3.8) we get

$$|\delta x| \leq \varepsilon |A^{-1}| \cdot |A| \cdot |x + \delta x|.$$

Then, if $\|\cdot\|$ is any norm with the property

$$|A| \leq |B| \Rightarrow \|A\| \leq \|B\|$$

(like, for instance, any absolute norm, as well as the 2-norm), this implies

$$\|\delta x\| \leq \varepsilon \||A^{-1}| \cdot |A|\| \cdot \|x + \delta x\|,$$

which in turn implies

$$\frac{\|\delta x\|}{\|x + \delta x\|} \leq \varepsilon \||A^{-1}| \cdot |A|\|. \tag{3.9}$$

The factor $\kappa_{CR}(A) := \||A^{-1}| \cdot |A|\|$ in the right-hand side of (3.9) is know as the *componentwise condition number*, or *Bauer-Skeel* condition number.

Inequality (3.9) provides also an estimation of the componentwise relative conditioning of the problem (3.1). This estimation is, precisely, the Bauer-Skeel condition number. The advantage of inequality (3.9) against (3.7) is that the Bauer-Skeel condition number $\kappa_{CR}(A)$ is smaller than the condition number for inversion $\kappa(A)$, and in some cases it is much smaller.

☞ Or course, both (3.4) and (3.9) are bounds, which means that the right-hand side could be much greater than the left-hand side. However, these bounds are attainable, which means that, for any value of n, there are data (A, b) for which these inequalities become equalities.

3.2. Gaussian elimination and LU factorization. Gaussian elimination is still the standard algorithm for solving systems of linear equations. The goal of this section is to analyze this method from the point of view of the computational cost and backward error. We will also see that this method is equivalent to factorizing a matrix as a product of a lower (L) and an upper (U) triangular matrix, which leads to the LU factorization.

3.2.1. Triangular systems. A general strategy to solve a SLE is to transform the original system (3.1) into an equivalent one (i.e., one having exactly the same solutions),

$$Tx = b', \tag{3.10}$$

whose solution can be obtained easily. The standard choice for such a system (3.10) is one whose coefficient matrix T is a **triangular** matrix. Both upper triangular and lower triangular are equivalent alternatives, but here we concentrate on **upper triangular** matrices. The arguments and developments for lower triangular matrices are completely analogous.

The standard algorithm to solve an upper triangular system (3.10) is as follows. We denote $T = [t_{ij}]$ and, for simplicity, $b' = [b_i]$. Then the system (3.10) becomes

$$\begin{array}{rcl} t_{11}x_1 + t_{12}x_2 + t_{13}x_3 + \cdots + t_{1n}x_n &=& b_1 \\ t_{22}x_2 + t_{23}x_3 + \cdots + t_{2n}x_n &=& b_2 \\ \vdots \quad \vdots \quad \vdots & & \\ t_{n-1,n-1}x_{n-1} + t_{n-1,n}x_n &=& b_{n-1} \\ t_{nn}x_n &=& b_n \end{array} \quad . \tag{3.11}$$

The following algorithm solves (3.11) through the standard procedure of backward substitution.

`Algorithm 1`: Algorithm to solve an upper triangular system (3.11) by backward substitution.

$\widehat{x}_n = b_n \oslash t_{nn}$
for $i = n-1 : -1 : 1$
    $\widehat{x}_i = (b_i \ominus (t_{i,i+1} \otimes \widehat{x}_{i+1}) \ominus (t_{i,i+2} \otimes \widehat{x}_{i+2}) \ominus \cdots \ominus (t_{in} \otimes \widehat{x}_n)) \oslash t_{ii}$
end

In `Algorithm 1`, and in what follows, we will use a hat, $\widehat{(\cdot)}$, for the computed (approximate) quantities. Note that in `Algorithm 1` we are assuming that the data, (T, b) are the exact values, so they are machine numbers. This will be a general assumption in this course, and it is only relevant for the backward error analysis.

For lower triangular systems `Algorithm 1` works in a similar way with forward substitution instead.

Computational cost.

Note that the first step of `Algorithm 1` involves just one flop (a division). At each new step we add 2 new flops, namely, one subtraction and one multiplication. Hence, the total cost of `Algorithm 1` is

$$1 + 3 + 5 + \cdots + 2n - 1 = \boxed{n^2}.$$

This cost is exact, and not an approximation.

Backward error analysis.

The following result shows that `Algorithm 1` is backward stable.

THEOREM 3.3. *Let us assume that* `Algorithm 1` *computes a solution* $\widehat{x}$ *of the system* $Tx = b$*, with* T *upper triangular. Then there is an upper triangular matrix* ΔT *such that*

$$(T + \Delta T)\widehat{x} = b, \tag{3.12}$$

with

$$|\Delta T| \leq (nu + O(u^2))|T|,$$

and therefore $\|\Delta T\|_{1,\infty,F} \leq (nu + O(u^2))\|T\|_{1,\infty,F}$.

Proof. From the third line in `Algorithm 1` we get

$$t_{i,i+k} \otimes \widehat{x}_{i+k} = t_{i,i+k}\widehat{x}_{i+k}(1 + \varepsilon_{i+k}),$$

with $|\varepsilon_{i+k}| \leq u$ and $1 \leq k \leq n - i$. We also have

$$b_i \ominus (t_{i,i+1} \otimes \widehat{x}_{i+1}) = (b_i - t_{i,i+1}\widehat{x}_{i+1}(1 + \varepsilon_{i+1}))(1 + \delta_1)$$

and, iterating

$$\begin{gathered}(b_i \ominus (t_{i,i+1} \otimes \widehat{x}_{i+1} \ominus \cdots \ominus (t_{i,i+k-1} \otimes \widehat{x}_{i+k-1})) \ominus (t_{i,i+k} \otimes \widehat{x}_{i+k}) = \\ (b_i \ominus (t_{i,i+1} \otimes \widehat{x}_{i+1} \ominus \cdots \ominus (t_{i,i+k-1} \otimes \widehat{x}_{i+k-1}) - t_{i,i+k} \otimes \widehat{x}_{i+k})(1 + \delta_k),\end{gathered}$$

with $|\delta_k| \leq u$, and $1 \leq k \leq n-i$. Then we arrive at

$$\widehat{x}_i = \frac{b_1(1+\delta_1)\cdots(1+\delta_{n-i}) - \sum_{k=1}^{n-1} \widehat{x}_{i+k} t_{i,i+k}(1+\varepsilon_{i+k})(1+\delta_k)\cdots(1+\delta_{n-i})}{t_{ii}(1+\alpha_i)},$$

with $|\alpha_i| \leq u$. Operating we get

$$\frac{t_{ii}(1+\alpha_i)}{(1+\delta_1)\cdots(1+\delta_{n-i})} \cdot \widehat{x}_i = b_i - \sum_{k=1}^{n-i} \frac{t_{i,i+k}(1+\varepsilon_{i+k})}{(1+\delta_1)\cdots(1+\delta_{k-1})} \cdot \widehat{x}_{i+k}.$$

Now, setting

$$\begin{aligned} t_{ii} + \Delta t_{ii} &= \frac{t_{ii}(1+\alpha_i)}{(1+\delta_1)\cdots(1+\delta_{n-i})} = t_{ii}(1+\theta_{ii}), \\ t_{i,i+k} + \Delta t_{i,i+k} &= \frac{t_{i,i+k}(1+\varepsilon_{i+k})}{(1+\delta_1)\cdots(1+\delta_{k-1})} = t_{i,i+k}(1+\theta_{i,i+k}), \end{aligned}$$

with $\theta_{ii} \leq (n-i+1)u + O(u^2) \leq nu + O(u^2)$, and $\theta_{i,i+k} \leq ku + O(u^2) \leq nu + O(u^2)$, the result follows. □

Some relevant remarks about Theorem 3.3:

- Theorem 3.3 guarantees that the backward substitution algorithm for upper triangular systems is backward stable, not only normwise, but also componentwise.
- It also says that the computed solution is a solution of a nearby system which has the same right-hand side as the original system, namely b.
- Moreover, this nearby system is also upper triangular.

EXERCISE 9. *With the notation from Theorem* 3.3*, prove that*

$$\frac{\|x - \widehat{x}\|_{1,\infty,F}}{\|\widehat{x}\|_{1,\infty,F}} \leq (nu + O(u^2)) \||T^{-1}| \cdot |T|\|_{1,\infty,F}. \tag{3.13}$$

Equation (3.13) says that the Bauer-Skeel condition number of T is the condition number of solving the triangular system (3.10). For a more general result in this direction see [3, p. 155].

3.2.2. General systems. Gaussian elimination is a numerical method to get an upper triangular matrix from a given matrix $A \in \mathbb{F}^{n\times n}$ using elementary row operations. It performs at most n steps, where each single step can be depicted as follows:

$$A = \left[\begin{array}{c|c} a & b^\top \\ \hline c & A_{22} \end{array}\right] \longrightarrow \widetilde{A} = \left[\begin{array}{c|c} a & b^\top \\ \hline 0 & \widetilde{A}_{22} \end{array}\right], \qquad \widetilde{A}_{22} = A_{22} - \frac{1}{a} c \cdot b^\top, \tag{3.14}$$

where $a \neq 0$ and $b, c \in \mathbb{F}^{n-1}$.

Let us review the *Gaussian elimination method* with an elementary example. We want to solve the SLE:

$$\begin{aligned} 2x_1 + 4x_2 + 4x_3 &= 2, \\ x_1 - x_2 + 2x_3 &= 2, \\ 3x_1 + x_2 + 2x_3 &= 0. \end{aligned}$$

This system can be written in the form (3.1), with

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

The Gaussian elimination method (or, the *Gauss method*, or *Gaussian elimination*, for short) proceeds by choosing the $(1,1)$ entry of the coefficient matrix A (which is the *pivot entry* of this step) and performs elementary row operations to make zero all the entries below the pivot entry. Then the process is iterated with the $(2,2)$ entry, then the $(3,3)$ entry, and so on. In the case of the previous system this goes as follows:

$$\left[\begin{array}{ccc|c} 2 & 4 & 4 & 2 \\ 1 & -1 & 2 & 2 \\ 3 & 1 & 2 & 0 \end{array}\right] \xrightarrow[F_{31}(-\frac{3}{2})]{F_{21}(-\frac{1}{2})} \left[\begin{array}{ccc|c} 2 & 4 & 4 & 2 \\ 0 & -3 & 0 & 1 \\ 0 & -5 & -4 & -3 \end{array}\right] \xrightarrow{F_{32}(-\frac{5}{3})} \left[\begin{array}{ccc|c} 2 & 4 & 4 & 2 \\ 0 & -3 & 0 & 1 \\ 0 & 0 & -4 & -\frac{14}{3} \end{array}\right],$$

where $F_{ij}(\alpha)$ is the elementary row operation that consists of replacing the ith row by the ith row plus α times the jth row, that is: $F_{ij}(\alpha) \equiv \mathrm{Row}_i + \alpha \mathrm{Row}_j \mapsto \mathrm{Row}_i$. Let us just look at the coefficient matrix, forgetting for a moment about the right-hand side term b. If we recall that elementary row operations are equivalent to multiply on the left by the corresponding elementary matrix, then the previous transformations on the matrix A can be written as matrix multiplications in the form:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{5}{3} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ -\frac{3}{2} & 0 & 1 \end{bmatrix} A = \begin{bmatrix} 2 & 4 & 4 \\ 0 & -3 & 0 \\ 0 & 0 & -4 \end{bmatrix}. \tag{3.15}$$

Multiplying by the inverses of the two lower triangular matrices in the left-hand side of (3.15), we arrive at

$$A = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{3}{2} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{5}{3} & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 4 \\ 0 & -3 & 0 \\ 0 & 0 & -4 \end{bmatrix}. \tag{3.16}$$

Now, notice that we can multiply the first two matrices in the right-hand side of (3.16) in a fast way. Just replace the first column in the identity matrix by the first column of the first matrix, and the second column of the identity matrix by the second column of the second matrix. In other words:

$$A = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{3}{2} & \frac{5}{3} & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 4 \\ 0 & -3 & 0 \\ 0 & 0 & -4 \end{bmatrix}. \tag{3.17}$$

Now, if we denote

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{3}{2} & \frac{5}{3} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 2 & 4 & 4 \\ 0 & -3 & 0 \\ 0 & 0 & -4 \end{bmatrix},$$

then Equation (3.17) gives

$$A = LU, \quad \text{with} \quad \begin{cases} L\text{: lower triangular with 1's in the diagonal,} \\ U\text{: upper triangular.} \end{cases} \tag{3.18}$$

☞ The matrix U is the upper triangular matrix obtained after applying Gaussian elimination on A, and the matrix L encodes in each column the **multipliers** used at each step to make zeroes below the corresponding pivot (with the signs shifted) .

The previous example illustrates the LU factorization and its connection with Gaussian elimination in a particular case. However, it is immediate to realize about problems with this procedure for a general matrix A. As we will see, not every $n \times n$ matrix can be decomposed in the form (3.18). It may happen that at some step of the Gaussian elimination, say the ith step, the (i, i) entry of the updated matrix is zero, so it can not act as a pivot to make zeroes below it. Nonetheless, it is also easy to find a solution for this problem: just look at some nonzero entry in the ith column and below the ith row, (namely, at the $(i + j, i)$ position), and permute the ith row with the $(i + j)$th row. Note that we must permute with a row which is below the ith one, since otherwise that would place a nonzero entry below and to the left of the (i, i) entry, and we would loose the upper triangular property of the resulting matrix. A row permutation is equivalent to multiply on the left by a permutation matrix, by Lemma 1.8. Then, it is natural to replace (3.18) by

$$A = PLU, \quad \text{with} \quad \begin{cases} P\text{: permutation matrix,} \\ L\text{: lower triangular with 1's in the diagonal,} \\ U\text{: upper triangular.} \end{cases} \tag{3.19}$$

If a given $n \times n$ matrix A can be written in the form (3.19), then the following algorithm computes the solution of (3.1) by solving only triangular systems.

`Algorithm 2`: Solves (3.1) using Gaussian elimination.

1. Factorize A as in (3.19).
2. Solve $PLy = b$ as $Ly = P^\top b$ and using forward substitution.
3. Solve $Ux = y$ by backward substitution.

`Algorithm 2` provides a method to solve linear systems (3.1) provided that the coefficient matrix can be factorized in the form (3.19). The natural question now is: can every $n \times n$ invertible matrix be decomposed in the form (3.19)? To answer this question is the goal of the following section.

3.2.3. Existence and uniqueness of LU and PLU factorization for invertible matrices. By an LU factorization of a matrix we mean a factorization like in (3.18), and a PLU factorization is a factorization like (3.19).

THEOREM 3.4. *Let $A \in \mathbb{F}^{n\times n}$ be invertible. If A has a an LU factorization, then it is unique.*

Proof. Let $A = LU = \widetilde{L}\widetilde{U}$ be two LU factorizations of A. Then, since A is invertible, both factors are invertible and $L^{-1}\widetilde{L} = U\widetilde{U}^{-1}$. Since $L^{-1}\widetilde{L}$ is lower triangular and $U\widetilde{U}^{-1}$ is upper triangular, it must be $L^{-1}\widetilde{L} = U\widetilde{U}^{-1} = D$, with D diagonal. But, since the main diagonal of $L^{-1}\widetilde{L}$ is all 1's, it must be $D = I_n$, and this implies $L = \widetilde{L}, U = \widetilde{U}$. □

The following result characterizes the set of matrices having an LU factorization.

THEOREM 3.5. *An invertible matrix $A \in \mathbb{F}^{n\times n}$ has an LU factorization if and only if the matrix $A(1 : j, 1 : j)$ is invertible, for all $j = 1, \ldots, n$.*

Proof. $\boxed{\Rightarrow}$ Let us assume that A has an LU factorization, and, for a fixed $1 \leq j \leq n$, let us partition it as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}, \tag{3.20}$$

with $A_{11} = A(1:j, 1:j)$. Then (3.20) implies that $A_{11} = L_{11}U_{11}$. Since A is invertible, both L_{11} and U_{11} must be invertible, so A_{11} must be invertible as well.

$\boxed{\Leftarrow}$ Let us proceed by induction on n. For $n = 1$ the result is trivial. Let us assume the result true for $n-1$, and let us partition A as

$$A = \begin{bmatrix} B & b \\ c^\top & a_{nn} \end{bmatrix}$$

with $B = A(1:n-1, 1:n-1)$. Since, by hypothesis, B is invertible, we can apply the induction hypothesis to B, so, it has an LU decomposition, namely $B = L_B U_B$. We are going to see that there is an LU factorization of A of the form

$$A = \begin{bmatrix} L_B & 0 \\ \ell^\top & 1 \end{bmatrix} \begin{bmatrix} U_B & u \\ 0 & u_{nn} \end{bmatrix} = \begin{bmatrix} L_B U_B & L_B u \\ \ell^\top U_B & \ell^\top u + u_{nn} \end{bmatrix},$$

for some $\ell, u \in \mathbb{C}^n$. For this, just equate in the previous identity to get

$$\begin{array}{rcl} L_B u = b & \Rightarrow & u = L_B^{-1} b, \\ \ell^\top U_B = c^\top & \Rightarrow & \ell^\top = c^\top U_B^{-1}, \\ \ell^\top u + u_{nn} = a_{nn} & \Rightarrow & u_{nn} = a_{nn} - \ell^\top u, \end{array}$$

where in the first two equations we have used that L_B and U_B are invertible, because B is invertible. As a consequence, A has an LU factorization. ∎

Theorem 3.5 is in accordance with the intuition that matrices like $A = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ do not have an LU factorization since it is not possible to perform Gaussian elimination (without permutation of rows). However, the following result guarantees that, if we allow for such row permutations, then we can always get a PLU factorization.

THEOREM 3.6. *Let $A \in \mathbb{F}^{n\times n}$ be an invertible matrix. Then A has a PLU factorization.*

Proof. Let us prove it by induction on n. The proof is constructive and actually provides the standard algorithm to compute the LU factorization of A.

For $n = 1$, we just take $P = L = 1$, $U = A \in \mathbb{F}$. Let us assume the result true for matrices of order $n-1$.

Let A be of size $n \times n$ and let P_1 be a permutation matrix such that $P_1 A(1,1) = a_{11}$ is nonzero. Such a P_1 exists because A is invertible. Now we solve the block matrix equation

$$P_1 A = \begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ L_{21} & I_{n-1} \end{bmatrix} \begin{bmatrix} u_{11} & U_{12} \\ 0 & \widetilde{A}_{22} \end{bmatrix} = \begin{bmatrix} u_{11} & U_{12} \\ L_{21}u_{11} & L_{21}U_{12} + \widetilde{A}_{22} \end{bmatrix},$$

where $A_{21}, A_{12}^\top \in \mathbb{F}^{(n-1)\times 1}$, for the unknowns $L_{21} \in \mathbb{F}^{(n-1)\times 1}, u_{11} \in \mathbb{F}, U_{12} \in \mathbb{F}^{1\times(n-1)}$, and $\widetilde{A}_{22} \in \mathbb{F}^{(n-1)\times(n-1)}$ (note that $\widetilde{A}_{22}$ is not necessarily upper triangular). Equating above gives

$$\left\{ \begin{array}{rcl} u_{11} = a_{11} \neq 0, & & \\ U_{12} = A_{12}, & & \\ L_{21}u_{11} = A_{21} & \Rightarrow & L_{21} = \dfrac{1}{a_{11}} \cdot A_{21}, \\ \widetilde{A}_{22} = A_{22} - L_{21}U_{12} & = & A_{22} - \dfrac{1}{a_{11}} \cdot A_{21}A_{12}. \end{array} \right. \tag{3.21}$$

Since A is invertible,

$$0 \neq \det(P_1 A) = u_{11} \det \widetilde{A}_{22} \Rightarrow \det \widetilde{A}_{22} \neq 0,$$

so $\widetilde{A}_{22}$ is invertible as well. Since it has size $(n-1) \times (n-1)$ we can apply the induction hypothesis, which guarantees that there is some $\widetilde{P}$ such that

$$\widetilde{P}\widetilde{A}_{22} = \widetilde{L}\widetilde{U}. \tag{3.22}$$

As a consequence,

$$\begin{aligned} P_1 A &= \begin{bmatrix} 1 & 0 \\ L_{21} & I_{n-1} \end{bmatrix} \begin{bmatrix} u_{11} & U_{12} \\ 0 & \widetilde{A}_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ L_{21} & I_{n-1} \end{bmatrix} \begin{bmatrix} u_{11} & U_{12} \\ 0 & \widetilde{P}^\top \widetilde{L}\widetilde{U} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ L_{21} & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \widetilde{P}^\top \widetilde{L} \end{bmatrix} \begin{bmatrix} u_{11} & U_{12} \\ 0 & \widetilde{U} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & \widetilde{P}^\top \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \widetilde{P}L_{21} & \widetilde{L} \end{bmatrix} \begin{bmatrix} u_{11} & U_{12} \\ 0 & \widetilde{U} \end{bmatrix}. \end{aligned}$$

Finally, adding up the first permutation we arrive at the PLU decomposition of A:

$$\underbrace{\left(\begin{bmatrix} 1 & 0 \\ 0 & \widetilde{P} \end{bmatrix} P_1 \right)}_{P} A = \underbrace{\begin{bmatrix} 1 & 0 \\ \widetilde{P}L_{21} & \widetilde{L} \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} u_{11} & U_{12} \\ 0 & \widetilde{U} \end{bmatrix}}_{U}, \tag{3.23}$$

so $A = P^\top LU$. □

EXERCISE 10. *Adapt the proof of Theorem* 3.6 *to prove that it is also true for any $A \in \mathbb{F}^{n \times n}$, invertible or not. What do L and U in the singular case look like?*

REMARK 7. *Since the only operations used in the proof of Theorem* 3.6 *are sums, subtractions, multiplications, and divisions, the LU is valid over any field $\mathbb{F}$.*

REMARK 8. *Note that a similar result to Theorem* 3.6 *is true for a factorization of the form $AP = LU$, that is, multiplication on the right by a permutation (equivalently, performing column operations instead of row operations). Moreover, we can get a decomposition $A = P_1 L U P_2$, with P_1, P_2 being permutation matrices. This allows us to choose the pivot at the ith step not only from the ith column, but from the rectangular submatrix with indices $(i:n, i:n)$.*

3.2.4. The LU algorithm using Gaussian elimination. The following practical comments on the proof of Theorem 3.5 are key to get the algorithm for computing the PLU factorization:

- Instead of $A = PLU$ we apply the permutations on A, to get $PA = LU$, so we end up with an LU factorization of some matrix obtained from A by row permutation.
- At the ith step, the permutation (if any) only permutes rows from i to n. Moreover, we see from (3.23) that the successive permutations can be stored independently and applied on the whole matrix A at the end of the algorithm.
- At the ith step, once we have computed $\widetilde{A}_{22}$ using (3.21), the first row of U and the first column of L remain fixed until the end of the process, so they are the final row of U and the final column of L. This is iterated at each step, so at the ith step we get the ith row of U and the ith column of L.

- Up to permutations, we see from (3.23) that the first column of L and the first row of U at each step are computed like in (3.21), which is just one step of Gaussian elimination (3.14). More precisely, the rows of matrix U are the rows of the upper triangular matrix obtained by Gaussian elimination, whereas the ith column of L encodes the multipliers used to make zeroes below the pivot at the ith step.
- Similarly, the columns of A with indices from 1 to $i-1$ are not used anymore from the ith step on, so we can replace them by the columns of L obtained in steps from the 1st to the $(i-1)$th, in order to save storage requirements.

With all these observations in mind we are in the position of stating the algorithm to compute the LU factorization of $A \in \mathbb{F}^{n\times n}$.

`Algorithm 3`: LU factorization with Gaussian elimination.

% INPUT: $A \in \mathbb{F}^{n\times n}$ invertible.
% OUTPUT: A permutation matrix P and a matrix $\widetilde{A}$ such that:
% $U = \texttt{triu}(\widetilde{A}), L = I_n + \texttt{tril}(\widetilde{A}, -1)$ and $PA = LU$

$P = I_n$
for $i = 1 : n-1$
    Permute the ith and the jth row of A, with $j > i$, such that $A(i,i) \neq 0$
    Exchange the ith and jth row in P
    $A(i+1:n, i) = A(i+1:n, i)/A(i,i)$
    $A(i+1:n, i+1:n) = A(i+1:n, i+1:n) - A(i+1:n, i) \cdot A(i, i+1:n)$
end

Though in this course we do not pay special attention to the storage requirements, `Algorithm 3` only requires $2n^2$ storage, since at each step it produces a couple of $n \times n$ matrices, namely P and $\widetilde{A}$.

☞ `Algorithm 3` is a very simple algorithm. Note that, forgetting about permutations, it just contains **two lines** of code!!

3.2.5. Computational cost of solving (3.1) **using the** LU **factorization.** We want to estimate the computational cost of `Algorithm 2`. This algorithm involves `Algorithm 3` for computing the PLU factorization of A. Then, it consists of three steps:

Step1. Compute P, L, U as in (3.19) using `Algorithm 3`.
Step2. Solve $PLy = b$.
Step3. Solve $Ux = y$.

In the estimation of the computational complexity we will not take into account the permutations, and just focus on counting the number of flops.

Let us start counting the number of flops of `Algorithm 3`. The first line, at the ith step involves $n-i$ divisions. The second line involves $(n-i)^2$ products and $(n-i)^2$ subtractions. Then, the cost of `Algorithm 3` is:

Cost of `Algorithm 3`:

$$\begin{aligned}\sum_{i=1}^{n-1} \left(2(n-i)^2 + (n-i)\right) &= 2\sum_{j=1}^{n-1} j^2 + \sum_{j=1}^{n-1} j = 2\frac{(n-1)n(2n-1)}{6} + \frac{(n-1)n}{2} \\ &= \frac{(n-1)n(4n+1)}{6} = \frac{2}{3}n^3 + O(n^2).\end{aligned}$$

The cost of **Step 2** is $n^2 - n$ (exact), and the cost of **Step 3** is n^2 (exact) (see the cost of `Algorithm 1`).

Then, adding up:

☞ The cost of `Algorithm 2` (solving (3.1) with Gaussian elimination / LU factorization) is $\boxed{\frac{2}{3}\,n^3 + O(n^2)}$

3.2.6. Backward error analysis of Gaussian elimination. Let us start with the following result, whose proof is beyond the scope of this course.

THEOREM 3.7. *Let $A \in \mathbb{F}^{n\times n}$ be invertible and $\widehat{L}, \widehat{U}$ be the factors computed through* `Algorithm 3` *in a machine with unit roundoff u. Then there is some matrix E such that*

$$A + E = \widehat{L}\widehat{U}, \quad \textit{and} \quad |E| \leq (nu + O(u^2))\,|\widehat{L}| \cdot |\widehat{U}|. \tag{3.24}$$

As a consequence of Theorem 3.7 we get the following result for the solution of (3.1) using `Algorithm 2`.

THEOREM 3.8. *Let $A \in \mathbb{F}^{n\times n}$ invertible and $b \in \mathbb{F}^n$. Let us assume that the SLE* (3.1) *is solved using* `Algorithm 2` *in a computer with unit roundoff u. Then the computed solution $\widehat{x}$ satisfies*

$$(A + \Delta A)\widehat{x} = b, \quad \textit{with} \quad |\Delta A| \leq (3nu + O(u^2))\,|\widehat{L}| \cdot |\widehat{U}|, \tag{3.25}$$

with $\widehat{L}, \widehat{U}$ being the computed factors of the LU factorization of A from `Algorithm 3`.

EXERCISE 11. *Prove Theorem* 3.8 *using Theorem* 3.7.

Some relevant observations on Theorems 3.7 and 3.8 are:

- Taking absolute norms $\|\cdot\|$ in both (3.24) and (3.25) we arrive at

  $$\|E\| \leq (nu + O(u^2))\|\widehat{L}\| \cdot \|\widehat{U}\|,$$

  in (3.24), and

  $$\|\Delta A\| \leq (3nu + O(u^2))\|\widehat{L}\| \cdot \|\widehat{U}\| \tag{3.26}$$

  from (3.25). This gives us a bound for the backward error of `Algorithm 3` and `Algorithm 2`, respectively.
- The previous bounds do not guarantee that computing the LU factorization using `Algorithm 3`, is backward stable. For that, we should guarantee that the computed factors $\widehat{L}, \widehat{U}$ are the LU factors of a nearby matrix, that is, of some ΔA such that

  $$\begin{array}{ll} \|\Delta A\| \leq O(u)\|A\| & \text{(normwise), or} \\ |\Delta A| \leq O(u)|A| & \text{(componentwise).} \end{array}$$

Then, Theorem (3.24) does not guarantee that `Algorithm 3` is backward stable. In section 3.2.8 we will analyze this issue more in depth. First, we are going to motivate it by pointing at the place where it may fail in order to be backward stable.

3.2.7. The need for an appropriate pivoting strategy. Let us consider the following example. Set

$$A = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix}.$$

The exact LU factorization of A is

$$L = \begin{bmatrix} 1 & 0 \\ 10^{20} & 1 \end{bmatrix}, \qquad U = \begin{bmatrix} 10^{-20} & 1 \\ 0 & 1 - 10^{20} \end{bmatrix}.$$

However, if we apply `Algorithm 3` **without row permutations** (since there is no need for that) then we get

$$\begin{aligned} \widehat{L} &= \begin{bmatrix} 1 & 0 \\ \mathtt{fl}(10^{20}) & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 10^{20} & 1 \end{bmatrix} \\ \widehat{U} &= \begin{bmatrix} 10^{-20} & 1 \\ 0 & \mathtt{fl}(1 - 10^{20}) \end{bmatrix} = \begin{bmatrix} 10^{-20} & 1 \\ 0 & -10^{20} \end{bmatrix}. \end{aligned}$$

You can check in `MATLAB` (with double precision) that $\mathtt{fl}(10^{20}) = 10^{20}$ and $\mathtt{fl}(1 - 10^{20}) = -10^{20}$. Then this would give:

$$\widehat{L}\widehat{U} = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 0 \end{bmatrix}.$$

As a consequence,

$$\Delta A = A - \widehat{L}\widehat{U} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix},$$

so the backward error in the ∞-norm of this computed LU factorization is

$$\frac{\|\Delta A\|_\infty}{\|A\|_\infty} = \frac{1}{2} \neq O(u).$$

This is a huge backward error which indicates that `Algorithm 3` is not backward stable.

Furthermore, let us apply this LU factorization to solve a particular SLE using `Algorithm 2`. Let us consider, for instance

$$Ax = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

with A as above. The exact solution is

$$x = \frac{1}{1 - 10^{-20}} \begin{bmatrix} -1 \\ 1 \end{bmatrix}. \tag{3.27}$$

However, if we follow `Algorithm 2` with the computed factors $\widehat{L}, \widehat{U}$, then we first solve

$$\widehat{L}\widehat{y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \Rightarrow \widehat{y} = \begin{bmatrix} 1 \\ -10^{20} \end{bmatrix},$$

and then we solve

$$\widehat{U}\widehat{x} = \widehat{y} \Rightarrow \widehat{x} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{3.28}$$

Comparing x and $\widehat{x}$ in (3.27) and (3.28) we realize that there is a huge error, since the computed solution has nothing to do with the exact one (they are in completely different directions!).

Let us explain what has happened in this case. The problem has arised when approximating $\texttt{fl}(1-10^{20}) \equiv \texttt{fl}(a_{22}-10^{20}) = -10^{20}$ (actually we would have the same problem with any value of a_{22} for which $\texttt{fl}(a_{22}-10^{20}) = -10^{20}$).

EXERCISE 12. *Describe the set of real numbers* $0 < a < 10^{20}$ *such that* $\texttt{fl}(a - 10^{20}) = -10^{20}$ *in a floating point system with unit roundoff* $u = 10^{-16}$.

This problem arises because there are some entries of L and/or U which are quite big compared to the entries of A. Then, when rounding these numbers, we loose all information about some of the entries of A, which are completely different to the exact ones when multiplying the computed factors $\widehat{L}, \widehat{U}$.

How to avoid this problem? The idea is to keep control on the size of the entries of L and U against the size of the elements of A. But the question is: how can we do this?

Looking at `Algorithm 3` we see that there are different choices for the permutation P. In the precedent example, we have not performed any permutation since the $(1,1)$ entry of A is nonzero. This has produced a huge number in the $(2,1)$ position of L. But this can be avoided if we exchange the first and second row, namely, starting with

$$PA = \begin{bmatrix} 1 & 1 \\ 10^{-20} & 1 \end{bmatrix}.$$

Now, the exact LU factorization of PA is

$$PA = \underbrace{\begin{bmatrix} 1 & 0 \\ 10^{-20} & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} 1 & 1 \\ 0 & 1-10^{-20} \end{bmatrix}}_{U},$$

and the computed factors would be

$$\widehat{L} = \begin{bmatrix} 1 & 0 \\ 10^{-20} & 1 \end{bmatrix}, \qquad \widehat{U} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

Now we have

$$A - \widehat{L}\widehat{U} = \Delta A = \begin{bmatrix} 0 & 0 \\ 0 & 10^{-20} \end{bmatrix},$$

so the backward error of the LU factorization is

$$\frac{\|\Delta A\|_\infty}{\|A\|_\infty} = \frac{10^{-20}}{2} = O(u).$$

The underlying problem here is that in `Algorithm 3` it is not specified how to permute the rows, at each step, if there is more than one nonzero entry in the ith column. This should be specified in the algorithm.

The way to choose how the rows are permuted in `Algorithm 3` is known as a *pivoting strategy*. There are several options, but here we are only going to consider the following one:

Partial pivoting strategy: At each step of `Algorithm 3`, chose a permutation, P_i such that $P_iA(i,i)$ is the entry with largest modulus in $A(i:n,i)$.

When we use the partial pivoting strategy in `Algorithm 3` we end up with what is known as **Gaussian elimination with partial pivoting**.

3.2.8. Is `Algorithm 3` with partial pivoting backward stable?. `Algorithm 3` is currently the standard algorithm (with some improvements) to compute the LU factorization of a given matrix $A \in \mathbb{F}^{n\times n}$. Even though it is widely used and works well in practice, it is still an open question to determine whether it is backward stable or not. Though this answers the question in the title of this section, we are going to explain the natural attempts to analyze the backward stability of `Algorithm 3`.

Looking at (3.26), `Algorithm 2` would be normwise backward stable if

$$\|\widehat{L}\| \cdot \|\widehat{U}\| = O(\|A\|). \tag{3.29}$$

Gaussian elimination with partial pivoting ensures that $|\widehat{L}(i,j)| \leq 1$, for $i \geq j$, which implies $\|\widehat{L}\|_\infty \leq n$, and (3.25) gives (in the ∞-norm):

$$\|\Delta A\|_\infty \leq (3n^2u + O(u^2))\|\widehat{U}\|_\infty. \tag{3.30}$$

Therefore, the problem reduces to analyze what happens with $\|\widehat{U}\|_\infty$. In order to achieve (3.29), and following (3.30), we need to analyze the ratio between $\|A\|_\infty$ and $\|\widehat{U}\|_\infty$. This leads to the following definition.

Definition 3.9. (Growth factor). *Let $A \in \mathbb{F}^{n\times n}$ be invertible and let $\widehat{U}$ be its LU factor computed using Gaussian elimination with partial pivoting. Then the* growth factor *of A is*

$$g_{pp}(A) := \frac{\max_{i,j} |\widehat{U}(i,j)|}{\max_{i,j} |A(i,j)|}.$$

Note that $\frac{\|\widehat{U}\|_\infty}{n} \leq \max_{i,j} |\widehat{U}(i,j)| \leq (\max_{i,j} |A(i,j)|) \cdot g_{pp}(A) \leq g_{pp}(A)\|A\|_\infty$. As a consequence, using (3.26) and Theorem 3.8, Gaussian elimination with partial pivoting provides a solution of (3.1), $\widehat{x}$, such that

$$(A + \Delta A)\widehat{x} = b, \quad \text{with} \quad \|\Delta A\|_\infty \leq (3n^2u + O(u^2))g_{pp}(A)\|A\|_\infty.$$

Then, in order to guarantee backward stability of Gaussian elimination with partial pivoting, we should guarantee that the growth factor $g_{pp}(A)$ is not quite big. The following result establishes a bound on the growth factor.

Proposition 3.10. *If $g_{pp}(A)$ is the growth factor of the Gaussian elimination with partial pivoting applied to a matrix $A \in \mathbb{F}^{n\times n}$, then*

$$g_{pp}(A) \leq 2^{n-1}.$$

Exercise 13. *Prove Proposition* 3.10.

Unfortunately, the bound in Proposition 3.10 is sharp, since there are $n \times n$ matrices for which this bound is attainable.

EXERCISE 14. *Find a matrix M_n, with size $n \times n$ such that $g_{pp}(M_n) = 2^{n-1}$, and prove that, indeed, $g_{pp}(M_n) = 2^{n-1}$. (*Warning: *Do not try to find M_n by yourself! Look at the bibliography.)*

From Theorems 3.7 and 3.8, Equation (3.30), and using Proposition 3.10, we can conclude the following.

THEOREM 3.11. *Let $A \in \mathbb{F}^{n\times n}$ be invertible and $\widehat{L}, \widehat{U}$ be the factors computed through* `Algorithm 3` *with partial pivoting in a machine with unit roundoff u. Then there is some matrix E such that*

$$A + E = \widehat{L}\widehat{U}, \quad \textit{and} \quad \|E\|_\infty \leq (n^2 2^{n-1} u + O(u^2)) \, \|A\|_\infty \, .$$

As a consequence of Theorem 3.7 we get the following result for the solution of (3.1) using `Algorithm 2`.

THEOREM 3.12. *Let $A \in \mathbb{F}^{n\times n}$ be invertible and $b \in \mathbb{F}^n$. Let us assume that the SLE* (3.1) *is solved using* `Algorithm 2` *in a computer with unit roundoff u. Then the computed solution $\widehat{x}$ satisfies*

$$(A + \Delta A)\widehat{x} = b, \quad \textit{with} \quad \|\Delta A\|_\infty \leq (3n^2 2^{n-1} u + O(u^2)) \, \|A\|_\infty, \tag{3.31}$$

with $\widehat{L}, \widehat{U}$ being the computed factors of the LU factorization of A from `Algorithm 3`.

Theorems 3.11 and 3.12 say, in some sense, that the *LU* factorization and Gaussian elimination with partial pivoting are backward stable, since the factors $n^2 2^{n-1}$ and $3n^2 2^{n-1}$ depend only on the size of the problem, n. However, they are not polynomials in n, but exponential, so they are huge for large values of n. Therefore, for n large enough, these bounds could not guarantee any single digit of accuracy in the computations. To show this is the goal of the next exercise.

EXERCISE 15. *Let $n = 100$ and assume that we apply Gaussian elimination (*`Algorithm 2`*) with partial pivoting to solve* (3.1) *in a machine with IEEE double (unit roundoff $u = 10^{-16}$).*

(a) *Estimate the bound in* (3.31).

(b) *Theorem* 2.8 *indicates that*

$$\frac{\|x - \widehat{x}\|}{\|x\|} \leq (3n^2 2^{n-1} u + O(u^2)) \kappa_\infty(A).$$

Assume $\kappa_\infty(A) = 1$ (which is a very optimistic assumption!). Then compute the bound for the relative error in the solution given by the inequality above. How many digits of accuracy in the computed solution $\widehat{x}$ are guaranteed by this bound?

The value $n = 100$ in Exercise 15 is just a moderate value (the standard size of matrices in applications are of the order of thousands). Therefore, the situation can be even much worse than the one in this exercise. However, it is very rare in practice that g_{pp} is so big as 2^{n-1}.

One of the most challenging open problems concerning *LU* factorization is to provide an average estimation of the growth factor. In practice, $g_{pp}(A) \lesssim \sqrt{n}$, and Gaussian elimination with partial pivoting is considered backward stable in practice.

HIGHLIGHTS:

☞ The standard algorithm to solve a SLE (3.1) is Gaussian elimination. This is equivalent to compute the LU factorization of A and then solve two triangular systems.

☞ Every square matrix A has an LU factorization, permuting adequately the rows of A.

☞ The pivot at each step is not unique. There are several strategies to choose it, but the standard one is the one known as *partial pivoting*, which chooses the entry with largest modulus among all possible pivots which can be obtained by row permutations.

☞ The computational complexity of Gaussian elimination is $\frac{2}{3}\,n^3 + O(n^2)$.

☞ Though Gaussian elimination with partial pivoting has not been proved to be backward stable, it is considered backward stable in practice.

3.3. Positive definite matrices and Cholesky factorization. The Gaussian elimination method admits many refinements when applied to matrices enjoying special structures. Among these structures, one of particular relevance is the one of symmetric positive definite matrices. These matrices appear frequently in applications in different areas of science. Besides its algorithmic simplifications, the refined Gaussian elimination method has an important property when applied to symmetric positive definite matrices, namely, it is backward stable. This fact is really surprising, because the algorithm is no more than an adjustment of Gaussian elimination to the particular structure.

3.3.1. Properties of symmetric positive definite and symmetric positive semidefinite matrices. Let us start with the basic definition of these kinds of matrices.

DEFINITION 3.13. (Symmetric positive definite and symmetric positive semidefinite matrix). *A matrix $A \in \mathbb{R}^{n\times n}$ is* symmetric positive definite *(respectively,* symmetric positive semidefinite*) if*

(i) $A = A^\top$

(ii) $x^\top Ax > 0$ *(resp.* $x^\top Ax \geq 0$*), for all $x \in \mathbb{R}^n$ and $x \neq 0$.*

Condition (i) in Definition 3.13 is the *symmetric* condition, and condition (ii) is the *positive definite* (resp. *positive semidefinite*) condition.

In the following, we will use the abbreviations spd (symmetric positive definite) and sps (symmetric positive semidefinite). We will also use the following properties of spd and sps matrices.

LEMMA 3.14.

(i) *If $X \in \mathbb{R}^{n\times n}$ is invertible, then A is spd (resp., sps) if and only if $X^\top AX$ is spd (resp., sps).*

(ii) *If A is spd (resp., sps) and $H := A(i:j, i:j)$ is a principal submatrix of A, then H is spd (resp., sps).*

(iii) *If A is spd (resp., sps) then*

(iii-a) $A(i,i) > 0$ *(resp.,* $A(i,i) \geq 0$*).*

(iii-b) $|A(i,j)| < \sqrt{A(i,i)A(j,j)}$ *(resp.,* $|A(i,j)| \leq \sqrt{A(i,i)A(j,j)}$*) if $i \neq j$.*

(iii-c) $\max_{i,j} |A(i,j)| \leq \max_i A(i,i)$.
(iii-d) $\max_{i \neq j} |A(i,j)| < \max_i A(i,i)$.

EXERCISE 16. *Prove Lemma* 3.14.

3.3.2. Cholesky factorization. The following result is the analogue to the *LU* decomposition for spd and sps matrices.

THEOREM 3.15. (Cholesky factorization). *A matrix $A \in \mathbb{R}^{n \times n}$ is spd (resp., sps) if and only if A can be factorized as*

$$A = LL^\top \tag{3.32}$$

for some lower triangular matrix $L \in \mathbb{R}^{n \times n}$ with positive (resp., nonnegative) elements in the main diagonal. Moreover, if A is spd, then the factorization (3.32) *is unique.*

The factorization (3.32) *is called the* Cholesky *factorization of A, and L is called the* Cholesky factor *of A.*

Proof. $\boxed{\Leftarrow}$ Let $A = LL^\top$. Then $x^\top Ax = x^\top LL^\top x = (L^\top x)^\top L^\top x = \|L^\top x\|_2^2$, which is > 0, for all $x \neq 0$ (if A is spd) and ≥ 0 (if A is sps).

$\boxed{\Rightarrow}$ Let us first prove the existence of the factorization (3.32). We proceed by induction. The case $n = 1$ is trivial, since $A = \sqrt{A}\sqrt{A} \equiv LL^\top$. Let us assume the result true for matrices with size $(n-1) \times (n-1)$.

Let $A \in \mathbb{R}^{n \times n}$ be spd. Using Gaussian elimination (3.21) we can decompose A as

(3.33)
$$\begin{aligned} A = \begin{bmatrix} a_{11} & A_{21}^\top \\ A_{21} & A_{22} \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ \frac{1}{a_{11}} A_{21} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{11} & A_{21}^\top \\ 0 & A_{22} - \frac{1}{a_{11}} A_{21} A_{21}^\top \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ \frac{1}{\sqrt{a_{11}}} A_{21} & I_{n-1} \end{bmatrix} \begin{bmatrix} \sqrt{a_{11}} & A_{21}^\top \\ 0 & A_{22} - \frac{1}{\sqrt{a_{11}}} A_{21} A_{21}^\top \end{bmatrix} \\ &= \begin{bmatrix} \sqrt{a_{11}} & 0 \\ \frac{A_{21}}{\sqrt{a_{11}}} & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & A_{22} - \frac{A_{21} A_{21}^\top}{a_{11}} \end{bmatrix} \begin{bmatrix} \sqrt{a_{11}} & \frac{A_{21}^\top}{\sqrt{a_{11}}} \\ 0 & I_{n-1} \end{bmatrix} \end{aligned}$$

(note that, by Lemma 3.14(iii-a), $a_{11} > 0$). Now, using Lemma 3.14, the submatrix $A_{22} - \frac{A_{21} A_{21}^\top}{a_{11}}$ is spd, so applying the induction hypothesis, there is some $\widetilde{L}$ in the conditions of the statement such that $A_{22} - \frac{A_{21} A_{21}^\top}{a_{11}} = \widetilde{L}\widetilde{L}^\top$. Replacing this in (3.33) we arrive at

$$A = \underbrace{\begin{bmatrix} \sqrt{a_{11}} & 0 \\ \frac{A_{21}}{\sqrt{a_{11}}} & \widetilde{L} \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} \sqrt{a_{11}} & \frac{A_{21}^\top}{\sqrt{a_{11}}} \\ 0 & \widetilde{L}^\top \end{bmatrix}}_{L^\top}, \tag{3.34}$$

which is a decomposition (3.32).

If A is sps, then the arguments are exactly the same ones as for the spd case if $a_{11} \neq 0$. If $a_{11} = 0$, Lemma 3.14(iii-b) implies that $A_{21} = 0$ in (3.33), so $A_{21}^\top = 0$ as well, and then $A = \left[\begin{smallmatrix} 0 & 0 \\ 0 & A_{22} \end{smallmatrix}\right]$, with A_{22} being sps by Lemma 3.14(ii). Then we can apply the induction hypothesis on A_{22}, to get

$$A = \begin{bmatrix} 0 & 0 \\ 0 & A_{22} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & \widetilde{L}\widetilde{L}^\top \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & \widetilde{L} \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & \widetilde{L}^\top \end{bmatrix},$$

which is a decomposition like (3.32).

As for the uniqueness of (3.32), we will proceed by showing that equation (3.32) has a unique solution, provided that L is lower triangular. Therefore, we just need to solve for entries (i,j) with $i \leq j$. Equating the (i,j) entries in (3.32) we arrive at

$$A(i,j) = \sum_{k=1}^{j} L(i,k)(L^\top)(k,j) = \sum_{k=1}^{j} L(i,k)L(j,k). \tag{3.35}$$

Now, for $i = j = 1$ in (3.35) we get $A(1,1) = L(1,1)^2 \Rightarrow L(1,1) = \sqrt{A(1,1)}$, and for $j = 1, i \geq 2$ in (3.35) we get $L(i,1) = \frac{A(i,1)}{L(1,1)}$. This uniquely determines the first column of L. Now, let us assume that we have uniquely determined the first $j-1$ columns of L. Then (3.35) gives

$$A(j,j) = \sum_{k=1}^{j} L(j,k)^2 \Rightarrow L(j,j) = \sqrt{A(j,j) - \sum_{k=1}^{j-1} L(j,k)^2},$$

which is uniquely determined. Moreover, for $i > j$ (3.35) also gives

$$L(i,j)L(j,j) = A(i,j) - \sum_{k=1}^{j-1} L(i,k)L(j,k) \Rightarrow L(i,j) = \frac{A(i,j) - \sum_{k=1}^{j-1} L(i,k)L(j,k)}{L(j,j)},$$

which is again uniquely determined. As a consequence, the jth column of L is also uniquely determined. □

REMARK 9. *The Cholesky factorization* (3.32) *for spd matrices is the analogous result to the LU factorization for general matrices. Note, however, that* (3.32) *requires just one matrix* (L)*, instead of two* (L, U) *in* (3.18)*. This is completely natural and expected due to the symmetry of* A*.*

The uniqueness condition of (3.32) is no longer true for sps matrices. As a counterexample, consider the following two different decompositions (3.32) of the same sps matrix:

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

Equation (3.33) provides the following algorithm to compute the Cholesky factor of A. It is exactly the same as Gaussian elimination (`Algorithm` 3), without permutations and with an additional line for the diagonal entry.

`Algorithm 4`: Cholesky factorization.

% INPUT: $A \in \mathbb{R}^{n\times n}$ spd.
% L such that $A = LL^\top$

for $i = 1 : n$
  $A(i,i) = \sqrt{A(i,i)}$
  $A(i+1:n, i) = A(i+1:n, i)/A(i,i)$
  $\texttt{tril}A(i{+}1:n, i{+}1:n) = \texttt{tril}(A(i{+}1:n, i{+}1:n) - A(i{+}1:n, i)\cdot A(i, i{+}1:n))$
end
$L = \texttt{tril}A$

Computational cost: Since `Algorithm 4` performs half of the operations in `Algorithm 3`, the computational cost of the Cholesky factorization is $\boxed{\frac{1}{3}n^3 + O(n^2) \text{ flops}}$

3.3.3. Backward stability of Cholesky factorization. The most important property of `Algorithm 4` is that it is backward stable. This is in stark contrast with `Algorithm 3` for general $n \times n$ matrices. In order to approach this result, we first state the following theorem, which is just the analogue of Theorem 3.7.

THEOREM 3.16. *Let $A \in \mathbb{R}^{n\times n}$ be a spd matrix, and let us assume that* `Algorithm 4` *produces a matrix $\widehat{L}$ in a machine with unit roundoff u. Then*

$$\widehat{L}\widehat{L}^\top = A + E, \quad \textit{with} \quad |E| \leq ((n+1)u + O(u^2))|\widehat{L}| \cdot |\widehat{L}^\top|\,. \tag{3.36}$$

Similarly, the following result is the analogue of Theorem 3.8.

THEOREM 3.17. *Let $A \in \mathbb{R}^{n\times n}$ be a sdp matrix, and let $\widehat{L}$ be its Cholesky factor computed with* `Algorithm 4` *in a machine with roundoff error u. Let $\widehat{x}$ be the computed solution of the SLE* (3.1)*, obtained using the Cholesky factorization and then solving two triangular systems. Then, there is some matrix ΔA such that*

$$(A + \Delta A)\widehat{x} = b, \quad \textit{with} \quad |\Delta A| \leq ((3n+1)u + O(u^2))|\widehat{L}| \cdot |\widehat{L}^\top|.$$

So far, there is nothing new concerning backward error for linear systems with spd coefficient matrices compared to general systems, as should be expected. However, the following key inequality makes the difference:

$$\||\widehat{L}| \cdot |\widehat{L}^\top|\|_2 \leq \||\widehat{L}|\|_2^2 \leq \||\widehat{L}|\|_F^2 = \|\widehat{L}\|_F^2 \leq n\|\widehat{L}\|_2^2 = n\|A + E\|_2, \tag{3.37}$$

with E as in (3.36). Now, joining (3.36) and (3.37),

$$\begin{aligned} \|E\|_2 &\leq \||E|\|_2 \leq ((n+1)u + O(u^2))\||\widehat{L}| \cdot |\widehat{L}^\top|\|_2 \\ &\leq ((n+1)u + O(u^2))n(\|A\|_2 + \|E\|_2), \end{aligned} \tag{3.38}$$

and solving for $\|E\|_2$ we arrive at

$$\|E\|_2 \leq (n(n+1)u + O(u^2))\|A\|_2. \tag{3.39}$$

EXERCISE 17. *Obtain* (3.39) *from* (3.38).

Replacing (3.39) in (3.36) we obtain the main results in this section, which guarantee backward stability of `Algorithm 4` and the solution of (3.1), with A being spd, using Cholesky factorization.

THEOREM 3.18. (Computing Cholesky with Gaussian elimination is backward stable). *Let $A \in \mathbb{R}^{n\times n}$ be a spd matrix, and let us assume that* `Algorithm 4` *produces an outpout matrix $\widehat{L}$ in a machine with unit roundoff u. Then*

$$\widehat{L}\widehat{L}^\top = A + E, \quad \textit{with} \quad \|E\|_2 \leq (n(n+1)u + O(u^2))\|A\|_2.$$

THEOREM 3.19. (Solving SLE with Cholesky is backward stable). *Let $A \in \mathbb{R}^{n\times n}$ be a sdp matrix, and $\widehat{L}$ its Cholesky factor computed with* `Algorithm 4` *in a machine with roundoff error u. Let $\widehat{x}$ be the computed solution of the SLE* (3.1)*, obtained using the Cholesky factorization and then solving two triangular systems. Then, there is some matrix ΔA such that*

$$(A + \Delta A)\widehat{x} = b, \quad \textit{with} \quad \|\Delta A\|_2 \leq ((3n+1)nu + O(u^2))\|A\|_2.$$

HIGHLIGHTS:

☞ For symmetric positive definite matrices the Cholesky factorization plays the role of LU factorization.

☞ The computational cost to compute the Cholesky factorization is $\frac{1}{3}n^3 + O(n^2)$.

☞ Cholesky algorithm for positive definite matrices is **backward stable**.

4. Lesson 3: The QR factorization.

4.1. The importance of unitary/orthogonal matrices. Another important class of matrices is the following.

Definition 4.1. (Unitary and orthogonal matrix). *A matrix $U \in \mathbb{C}^{n\times n}$ is said to be*

- unitary *if $U^*U = I_n$,*
- orthogonal *if $U^\top U = I_n$.*

Unitary matrices play a relevant role in matrix computations, due essentially to the following properties.

Lemma 4.2. *Let U, V be unitary matrices of appropriate sizes. Then*

(i) $\|U\|_2 = 1$.
(ii) $\|UAV\|_2 = \|A\|_2$, *for any matrix A of appropriate size.*
(iii) UV *is also a unitary matrix.*

Exercise 18. *Prove Lemma* 4.2.

Real orthogonal matrices are the counterpart of (complex) unitary matrices over the real field. In particular, all claims in Lemma 4.2 can be directly translated to real orthogonal matrices replacing $*$ by $\top$. For this reason, orthogonal matrices are specially relevant over the real field. Note, however, that orthogonal matrices can also have complex entries.

The relevance of unitary matrices in matrix computations relies on claim (ii) in Lemma 4.2. Most problems in NLA involve matrix multiplications. Think, for instance, of Gaussian elimination: in order to get the upper triangular matrix U from a given matrix A we need to perform elementary row operations, which consist, as we have seen, of multiplying A on the left by elementary matrices. The problem with matrix multiplications is that they can significantly alter the condition number of the original matrix (consider for instance, the trivial example $A = I_n$ and multiply it by another $n \times n$ matrix with large condition number). However, property (ii) in Lemma 4.2 guarantees that this does not happen with unitary matrices and with the 2-norm. For this reason, and from the point of view of conditioning, to work with unitary (or real orthogonal) matrices is preferable than doing it with general matrices.

4.2. QR factorization. In Lesson 3 we have reviewed the standard way to solve linear systems (3.1) (Gaussian elimination), which consists of transforming the coefficient matrix A into an upper triangular matrix using elementary row operations. These transformations are not unitary (which means that the L matrix in the LU factorization is not unitary). This fact is, precisely, the one that prevents the LU factorization from being backward stable. More precisely, the norm of the matrix U obtained after applying these elementary transformations may be much larger than the one of the original matrix A. Therefore, one may wonder whether it is possible to achieve an upper triangular form using unitary transformations. The answer is yes, and is given by the QR factorization. As in Lesson 3, we focus on the real case (namely, $\mathbb{F} = \mathbb{R}$).

Theorem 4.3. (QR factorization). *Every invertible matrix $A \in \mathbb{R}^{n\times n}$ has a factorization $A = QR$, where $Q \in \mathbb{R}^{n\times n}$ is an orthogonal matrix and $R \in \mathbb{R}^{n\times n}$ is an upper triangular matrix. The factorization is unique if R has positive diagonal elements.*

Proof. The proof is essentially the one in [4, Fact. 3.32]. Let us first prove the existence of a QR factorization. Since A is nonsingular, there is some $x \neq 0$ such that $Ax \neq 0$, so $x^\top A^\top Ax = \|Ax\|_2 > 0$. Since the matrix $M = A^\top A$ is symmetric, the previous inequality implies that it is spd. Then, by Theorem 3.15, there is a lower triangular matrix L with positive diagonal elements such that $M = LL^\top$. Moreover, L is invertible, because M is. Then $M = A^\top A = LL^\top$. Multiplying by $A^{-\top} := (A^{-1})^\top$ on the left gives $A = QR$, with $Q = A^{-\top}L$, and $R = L^\top$, which is upper triangular with positive diagonal entries. It remains to prove that Q defined in this way is orthogonal. For this, let us multiply

$$\begin{aligned} Q^\top Q &= (A^{-\top}L)^\top A^{-\top}L = L^\top A^{-1}A^{-\top}L = L^\top (A^\top A)^{-1}L = L^\top M^{-1}L \\ &= L^\top (LL^\top)^{-1}L = L^\top (L^{-\top}L^{-1})L = (L^\top L^{-\top})(L^{-1}L) = I_n, \end{aligned}$$

so Q is, indeed, orthogonal.

Now, let us prove the uniqueness. Let us assume that $A = QR = \widetilde{Q}\widetilde{R}$ are two factorizations of A as in the statement. Then $S := Q^{-1}\widetilde{Q} = R\widetilde{R}^{-1}$. The matrix S is both orthogonal, by Lemma 4.2(iii), and upper triangular with positive diagonal elements. Now Exercise 19 implies that $S = I_n$, which in turn implies that $Q = \widetilde{Q}$ and $R = \widetilde{R}$. □

EXERCISE 19. *Prove that if $G \in \mathbb{R}^{n\times n}$ is both orthogonal and upper triangular with positive diagonal elements, then it must be $G = I_n$. Why this is not true anymore if we remove the condition of having positive diagonal elements?*

The QR factorization provides an alternative method to `Algorithm 2` for solving the SLE (3.1), namely:

1. Compute the QR factorization of $A = QR$.
2. Compute $\widehat{y} = Q^\top b$.
3. Solve $Rx = \widehat{y}$.

However, in order to decide about whether it is competitive or not with `Algorithm 2` solved using `Algorithm 3` we should first derive an algorithm to compute the QR factorization and analyze it from the point of view of backward stability and computational cost.

The proof of Theorem 4.3 provides a construction from the Cholesky factorization of the matrix $M = A^\top A$, and it involves the following three steps: (i) the computation of the product $A^\top A$; (b) the computation of the Cholesky factorization of $A^\top A$; and (c) the computation of the product $A^{-\top}L$ to get the matrix Q. This procedure present several disadvantages from the computational point of view. The most important one is that the computation of $A^\top A$ is subject to rounding errors which may produce a non-symmetric (computed) matrix, and then the Cholesky factorization of M could not be computed. As a consequence, this procedure is not appropriate.

4.2.1. Computation of QR factorization: Householder reflectors. There are several ways to compute the QR factorization of $A \in \mathbb{R}^{n\times n}$. The one we present here uses *Householder reflectors* and goes as follows.

We multiply on the left the matrix A by a sequence of orthogonal transformations, $Q_1, Q_2, \ldots, Q_n$ (Householder reflectors), such that Q_k, for $k = 1, \ldots, n$ makes zeroes below the diagonal in the kth column while preserving the zeroes introduces by the matrices $Q_1, \ldots, Q_{k-1}$. In this sense, it works like Gaussian elimination, but with orthogonal transformations, instead of elementary row transformations. The resulting

matrix $Q_n \cdots Q_2 Q_1 A = R$ is the upper triangular matrix of the QR factorization, while the transpose of the matrix Q is $Q^\top = Q_n \cdots Q_2 Q_1$.

Let us explain how the Householder reflector Q_k works. The matrix Q_k, for $k = 1, \ldots, n$, is of the form

$$Q_k = \begin{bmatrix} I_{k-1} & 0 \\ 0 & H_k \end{bmatrix}, \tag{4.1}$$

where H_k is a unitary matrix. Assume that we have already constructed $k-1$ matrices $Q_1, \ldots, Q_{k-1}$ such that, for $1 \leq i \leq k-1$, the ith column of $(Q_{k-1} \cdots Q_1)A$ has null entries below the ith coordinate. Then, we want to construct an orthogonal matrix Q_k such that the same holds for the first k columns of $(Q_k \cdots Q_1)A$. If Q_k has the form (4.1), this guarantees that multiplying on the left by Q_k does not alter the first $k-1$ columns of $(Q_{k-1} \cdots Q_1)A$. Now, let $v^{(k)}$ be the kth column of $(Q_{k-1} \cdots Q_1)A$ and let us denote

$$x^{(k)} = \begin{bmatrix} x_1 \\ \vdots \\ x_{n-k+1} \end{bmatrix} \in \mathbb{R}^{n-k+1},$$

the vector whose coordinates are the last (bottom) $n-k+1$ coordinates of $v^{(k)}$. In other words, $x^{(k)}$ is the vector obtained from $v^{(k)}$ after removing the first $k-1$ coordinates. Now, we look for an orthogonal transformation, H_k, such that

$$H_k x^{(k)} = \pm \|x^{(k)}\|_2 e_1, \tag{4.2}$$

where e_1 is the first column of I_{n-k+1}. The only nonzero entry of the right-hand side vector in (4.2) is the first one, which means that the kth column of $(Q_k \cdots Q_1)A$ will have null entries below the kth coordinate, as wanted. The factor $\|x^{(k)}\|_2$ must appear because we want H_k to be orthogonal, and then $\|H_k x^{(k)}\|_2 = \|x^{(k)}\|_2$, by Lemma 4.2(ii), taking $U = H_k$, $V = I$, and $A = x^{(k)}$ in this lemma. A natural choice for such H_k is the "reflection" across the hyperplane orthogonal to $\|x^{(k)}\|_2 e_1 - x^{(k)}$. Figure 4.1 depicts this reflection in $\mathbb{R}^2$ (the orthogonal hyperplane is the dashed line H, and x plays the role of $x^{(k)}$).

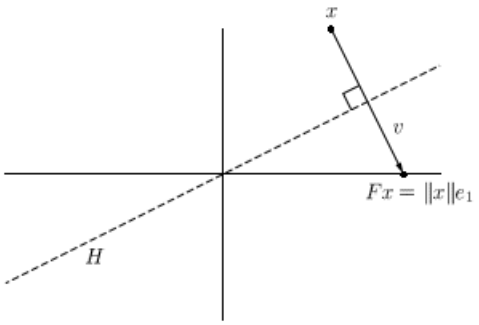

FIG. 4.1. *Householder reflector in* $\mathbb{R}^2$. *Taken from* [6].

However, there is another way to get the same vector, namely, reflecting across the orthogonal hyperplane to $-\|x^{(k)}\|_2 e_1 - x^{(k)}$. This is illustrated in Figure 4.2, where

the (orthogonal) hyperplanes are the dashed lines, and are denoted by H^+ and H^-. Note that, as shown in the picture, x goes to a vector whose first entry is negative if we reflect across H^-.

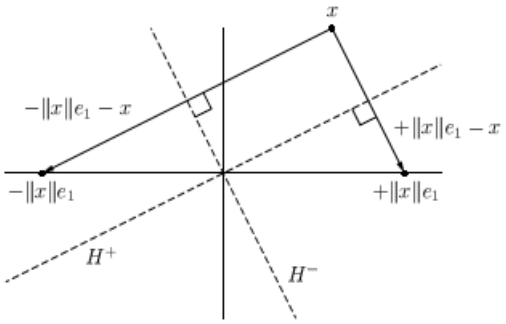

FIG. 4.2. *Two possible Householder reflector in* $\mathbb{R}^2$. *Taken from* [6].

In both cases, H_k is defined as follows:

$$H_k = I_{n-k+1} - 2\,\frac{y^{(k)}(y^{(k)})^\top}{(y^{(k)})^\top y^{(k)}}, \tag{4.3}$$

where $y^{(k)} = \operatorname{sign}(x_1)\|x^{(k)}\|_2 e_1 \pm x^{(k)}$. The right choice is the plus sign (+) in order to keep $x^{(k)}$ as far away as possible from its image to avoid cancellation (see [6]).

Summarizing, by choosing H_k as in (4.3) we guarantee the following:

(P1) The matrix H_k is orthogonal.

(P2) The first k columns of $(Q_k \cdots Q_1)A$ are of the form $\begin{bmatrix} \pm & * & \cdots & * \\ 0 & \pm & \cdots & * \\ 0 & 0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \pm \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$,

where $\pm$ are nonzero entries and the entries marked with $*$ are not relevant.

Note that (4.3), with the (+) sign for $y^{(k)}$, makes the signs ($\pm$) in the diagonal to be all (+). Essentially, what we do is to first reflect onto $-\|x\|e_1$ and then shift the (−) sign with (+).

Property (P2) is an immediate consequence of (4.2). The proof of property (P1) is left as an exercise.

EXERCISE 20. *Prove property* (P1) *above.*

Note that (P1) implies that that Q_k in (4.1) is orthogonal as well. As for (P2), it shows that the product $(Q_n \cdots Q_1)A$ is upper triangular with positive diagonal entries.

The following algorithm constructs the matrix R of the QR transformation using the previous ideas. As for the matrix U in the LU factorization, R is obtained by

replacing the corresponding entries in the matrix A, for storage saving. Note that the algorithm also stores the vectors q_k (for latter use).

```
Algorithm 5: Householder QR factorization.

% INPUT: An n × n matrix A
% OUTPUT: The matrix R of the QR factorization of A

for k = 1 : n
      x = A(k : n, k)
      q_k = sign(x_1)‖x‖_2 e_1 + x
      q_k = q_k/‖q_k‖_2
      A(k : n, k : n) = A(k : n, k : n) − 2q_k(q_k^⊤ A(k : n, k : n))
end
R = A
```

EXERCISE 21.  *Let*

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

(a) *Solve $Ax = b$ using* `Algorithm 2`*. Provide not just the solution, x, but also the LU factorization of A, with 4 decimal digits.*
(b) *Solve $Ax = b$ using* `Algorithm 7`*. Provide not just the solution, x, but also the matrix R in the QR factorization of A, together with the vectors q_k, for $k = 1, \ldots, 5$, with 4 decimal digits.*

4.2.2. ***LU* versus *QR* for solving linear systems.** `Algorithm 5` does not compute the matrix Q of the QR factorization of A. This is because for solving SLE we do not need to compute it. To see this, note that if $A = QR$ is the QR factorization of A, then (3.1) is equivalent to $Rx = Q^\top b$, so we just need to compute R and $Q^\top b$. It is possible to compute $Q^\top b$ without explicitly computing Q. This is the reason for storing the vectors q_k. In particular, from these vectors we can compute $Q^\top b$ using the following algorithm. Again, $Q^\top b$ is replaced by b for saving storage.

```
Algorithm 6: Implicit calculation of the product Q^⊤ b
% INPUT: A vector b and the vectors q_k, for k = 1 : n in Algorithm 5
% OUTPUT: The vector Q^⊤ b, with Q being the Q factor in the QR
% decomposition of A

for k = 1 : n
      b(k : n) = b(k : n) − 2q_k(q_k^⊤ b(k : n))
end
```

Now, joining `Algorithm 5` and `Algorithm 6` we arrive at the following algorithm to solve (3.1) using the QR factorization.

`Algorithm 7`: Solves (3.1) using QR factorization.

1. Factorize $A = QR$ using Householder reflectors.
2. Compute $y = Q^\top b$.
3. Solve $Rx = y$ by backward substitution.

Computational cost: The computational cost of `Algorithm 7` is $\boxed{\frac{4}{3}n^3}$ flops (see, for instance, [6, Lecture 10]).

The relevant difference between `Algorithm 2` and `Algorithm 7` is that `Algorithm 7` **is backward stable**. Moreover, we have a perfect backward stability result.

THEOREM 4.4. (Solving SLE using QR is backward stable). *If* `Algorithm 7` *produces a solution* $\widehat{x}$ *to* (3.1)*, in a machine with unit roundoff* u*, then there is some matrix* ΔA *such that*

$$(A + \Delta A)\widehat{x} = b, \quad \textit{with} \quad \|\Delta A\| \leq O(u)\|A\|. \tag{4.4}$$

For the proof of Theorem 4.4, we refer to the proof of Theorem 16.2 in [6]. We just give some general indications to understand why Theorem 4.4 is true. This is a consequence of the fact that all three steps in `Algorithm 7` are backward stable. We already know that the third step (solution of triangular systems by backward substitution) is backward stable (Theorem 3.3). It is not difficult to prove that the second step is also backward stable. As for the first one (computation of the QR factorization using Householder reflectors), the backward stability strongly relies on the property that Householder reflectors are orthogonal matrices.

REMARK 10. *Note that no special assumptions in the statement of Theorem* 4.4 *are made on the matrix norm* $\|\cdot\|$*. Among the standard norms introduced in Section* 1*, only the* 2*-norm satisfies* $\|UAV\| = \|A\|$*, for any* U, V *unitary (in order words, only the* 2*-norm is* unitarily invariant*). This means that, if* $\|\cdot\|$ *is any of the other norms (the* 1*,* ∞*, or Frobenius norms) then it could happen that, when multiplying by the Householder reflectors, the norm of the resulting matrix increases substantially. However, it can only increase up to some point. In particular, it can not go to infinity as* u *goes to zero, since all norms are equivalent in a finite-dimensional vector space. Note that inequality* (4.4) *is independent on the choice of the norm, as long as they are all equivalent.*

However, `Algorithm 7` is twice more expensive than `Algorithm 2`, and for this reason `Algorithm 2` is the one used in practice.

HIGHLIGHTS:

☞ Any invertible matrix $A \in \mathbb{R}^{n\times n}$ can be written as $A = QR$, with Q being orthogonal and R being upper triangular with positive diagonal entries. This is known as the QR *factorization of* A, and it is unique.

☞ The computational cost to compute the QR factorization using Householder reflections is $\frac{4}{3}n^3 + O(n^2)$, which is twice the cost of the LU factorization.

☞ The computation of the QR factorization using Householder reflections is **backward stable**.

☞ The QR factorization can be used to solve SLE (3.1) through `Algorithm 7`, which is **backward stable**.

☞ Despite the backward stability of `Algorithm 7`, its computational cost, which is twice as much as the computational cost of solving (3.1) using the LU factorization, makes this last procedure preferable in practice.

5. Lesson 4: Least squares problems and the SVD.

5.1. The singular value decomposition (SVD). The singular value decomposition (SVD) of a matrix $A \in \mathbb{C}^{n\times n}$ is another factorization of A that provides very useful information, including:

- The rank of A.
- The 2-norm of A.
- The robustness of $\operatorname{rank} A$ against small perturbations. This tells us how far is A away from a matrix of lower rank. In particular, if A is invertible, the SVD will tell us how far is A away from being singular.

Furthermore, there are many applications of the SVD decomposition. In this course, we will review one of its basic applications to image compressing and restoration (Section 5.4).

Note that in this section we are considering matrices with complex entries and either square or rectangular. This is a more general setting than in precedent sections, where only the real square case was considered.

In the proof of the SVD we will use the following results. We recall that a *Hermitian matrix* is a matrix $H \in \mathbb{C}^{n\times n}$ such that $H^* = H$, and that a matrix A is *unitarily diagonalizable* is there is a unitary matrix U such that U^*AU is diagonal.

Exercise 22. *Prove that* $\operatorname{rank} A = \operatorname{rank} A^*A$*, for any* $A \in \mathbb{C}^{m\times n}$*.*

Exercise 23. *Prove that the eigenvalues of a Hermitian positive definite matrix are real nonnegative numbers.*

Theorem 5.1. *Any Hermitian matrix is unitarily diagonalizable.*

The SVD is stated in the following result, whose proof is taken from [5, p. 31].

Theorem 5.2. (SVD). *Let* $A \in \mathbb{C}^{m\times n}$*. Then, there are two unitary matrices* $U \in \mathbb{C}^{m\times m}, V \in \mathbb{C}^{n\times n}$*, and a diagonal matrix* $\Sigma = \operatorname{diag}(\sigma_1, \dots, \sigma_r, 0, \dots, 0)$*, with* $\sigma_1 \geq \cdots \geq \sigma_r > 0$ *such that*

$$A = U\Sigma V^*. \tag{5.1}$$

The values $\sigma_1, \dots, \sigma_r$ *are uniquely determined and they are known as the* singular values of A*. The columns of* U *(resp.* V*) are known as* left *(resp.* right singular vectors *of* A*.*

If A *has real entries, then all* U, Σ, V *in* (5.1) *can be taken to be real.*

Proof. Let $r := \operatorname{rank} A$. By Exercise 22, $\operatorname{rank} A^*A = r$. The matrix A^*A is Hermitian and positive definite. By Exercise 23, the eigenvalues of A^*A are real and nonnegative. Let $\sigma_1^2 \geq \cdots \geq \sigma_r^2 > 0 = \sigma_{r+1}^2 = \cdots = \sigma_n^2$ be the eigenvalues of A^*A. Let $V = \left[\begin{array}{c|c} V_1 & V_2 \end{array}\right]$ (with $V_1 \in \mathbb{C}^{n\times r}$) be a unitary matrix whose columns are the eigenvectors of A^*A. This matrix exists, by Theorem 5.1. Then

$$V^*(A^*A)V = \begin{bmatrix} \Sigma_+^2 & 0 \\ 0 & 0 \end{bmatrix},$$

where $\Sigma_+ = \operatorname{diag}(\sigma_1, \dots, \sigma_r)$. Then we have

$$V_1^*A^*AV_1 = \Sigma_+^2, \qquad V_2^*A^*AV_2 = 0. \tag{5.2}$$

The second identity is equivalent to $(AV_2)^*(AV_2) = 0$. By Exercise 22, $\operatorname{rank}(AV_2) = \operatorname{rank}(AV_2)^*(AV_2) = 0$, so we conclude that

$$AV_2 = 0. \tag{5.3}$$

Now let

$$U_1 = AV_1\Sigma_+^{-1}. \tag{5.4}$$

Then (5.2) implies that $U_1^*U_1 = I_r$. Let us choose U_2 such that $\left[\begin{array}{c|c} U_1 & U_2 \end{array}\right]$ is unitary. Then, (5.2), (5.3), and (5.4) together imply

$$U^*AV = \begin{bmatrix} U_1^*AV_1 & U_1A^*V_2 \\ U_2^*AV_1 & U_2^*AV_2 \end{bmatrix} = \begin{bmatrix} \Sigma_+ & 0 \\ 0 & 0 \end{bmatrix},$$

and from this it follows that $A = U\Sigma V^*$, since U, V are unitary.

If A is real, note that all arguments are still true replacing * by $\top$ and all matrices involved can be chosen to be real. □

The decomposition (5.1) is known as a *singular value decomposition* (SVD) of A.

One may wonder about the uniqueness of the SVD of A. Theorem 5.2 claims uniqueness of the singular values, but nothing is said about the singular vectors. In the next result we provide an answer to this question, together with some other properties of the SVD.

Lemma 5.3. *Let A be as in the statement of Theorem* 5.2.

(i) *The singular values are the nonnegative square roots of the eigenvalues of AA^* or, equivalently, of A^*A (that's why they are uniquely determined).*
(ii) *The columns of U are eigenvectors of AA^*, and the columns of V are eigenvectors of A^*A, arranged in the same order as the corresponding eigenvalues σ_i^2.*
(iii) *If $m \leq n$ and if AA^* has different eigenvalues, then U is determined up to a right diagonal factor $D = \operatorname{diag}(e^{\mathrm{i}\theta_1}, \ldots, e^{\mathrm{i}\theta_n})$, with all $\theta_i \in \mathbb{R}$, that is, if $A = U_1\Sigma V_1^* = U_2\Sigma V_2^*$ then $U_2 = U_1D$.*
(iv) *If $m < n$ then V is never uniquely determined.*
(v) *If $m = n$ and U is given, then V is uniquely determined .*
(vi) *If $n \leq m$, analogous results hold by considering A^*.*
(vii) $\|A\|_2 = \sigma_1$.

Exercise 24. *Prove Lemma* 5.3.

Moreover, the following properties on U_1 and V_2 hold.

Lemma 5.4. *If U_1 and V_1 are as above, then*

(a) *The columns of U_1 are an orthonormal basis of the column space of A.*
(b) *The columns of V_2 are an orthonormal basis of the null space of A.*

Exercise 25. *Prove Lemma* 5.4.

We are not getting into the problem of computing the SVD of a given matrix. Instead of this, we will review some relevant applications of the SVD.

5.2. Low rank approximations. In many situations it may be convenient to approximate a given matrix A by some other matrix with smaller rank, which is called a *low-rank approximation* of A. It is desirable that this low-rank approximation is the closest to A in some sense. A natural notion of "closeness" is given by the distance, and in particular the distance defined by the 2-norm (we will see more on this in Section 5.3). Then the low-rank approximation problem is as follows:

Low-rank approximation problem:
Given a matrix A, find a matrix A_ρ such that

$$\|A - A_\rho\|_2 = \min\{\|A - B\|_2 : \quad \text{rank } B \leq \rho\}$$

The solution to this problem goes though the SVD of A. To state it, we need to recall that any matrix $M \in \mathbb{C}^{m\times n}$ with $\operatorname{rank} M = \rho$ can be written as a sum of rank-1 matrices

$$M = x_1 y_1^* + \cdots + x_\rho y_\rho^*,$$

for some vectors $x_i, y_i \in \mathbb{C}^n$, for $i = 1, \ldots, \rho$. In particular, the SVD of A in (5.1) allows to decompose A as a sum of r rank-1 matrices as follows:

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^*, \tag{5.5}$$

where u_i, v_i, for $i = 1, \ldots, r$, are the ith columns of U and V, respectively. With these considerations in mind, we can state the solution of the **Low-rank approximation problem**.

THEOREM 5.5. (Optimal low-rank approximation). *Let $A \in \mathbb{C}^{m\times n}$ be as in* (5.5)*, with $\sigma_1 \geq \cdots \geq \sigma_r > 0$ being the nonzero singular values of A, and let $\rho \leq r$. Set*

$$A_\rho := \sum_{i=1}^{\rho} \sigma_i u_i v_i^*. \tag{5.6}$$

Then

$$\|A - A_\rho\|_2 = \min\{\|A - B\|_2 : \quad \operatorname{rank} B \leq \rho\} = \sigma_{\rho+1},$$

where, if $\rho = r$, then $\sigma_{\rho+1} = 0$.

Proof. First note that, if A and A_ρ are as in (5.5) and (5.6), respectively, then $\|A - A_\rho\|_2 = \|\sum_{i=\rho+1}^{r} \sigma_i u_i v_i^*\|_2 = \sigma_{\rho+1}$, by Lemma 5.3(vii). It remains to prove that, for any B with $\operatorname{rank} B \leq \rho$, it must be $\|A - B\|_2 \geq \sigma_{\rho+1}$. We omit this part and refer to [4, Fact 4.13] for a proof. □

As a consequence of Theorem 5.5, the best approximation to A with rank ρ is obtained from the SVD of A by truncating the sum in (5.5) from the ρth singular value on.

In particular, we can determine the distance of an invertible matrix to the set of singular (i. e., non-invertible) matrices.

COROLLARY 1. *Let $A \in \mathbb{C}^{n\times n}$ be invertible and $\sigma_1 \geq \cdots \geq \sigma_n > 0$ be the singular values of A. Then*

$$\sigma_n = \min\{\|E\|_2 : \quad A + E \quad \textit{is singular}\}.$$

Proof. If E is such that $A+E$ is singular, then $\operatorname{rank}(A+E) \leq n-1$, so by Theorem 5.5,

$$\|E\|_2 = \|A-(A+E)\|_2 \leq \sigma_n.$$

□

Corollary 1 tells us about how far is an invertible matrix away from being singular. In other words, it provides the norm of the smallest perturbation that makes an invertible matrix to become singular. Though it does not explicitly show us how to construct this matrix, we can derive a procedure to get it using the SVD. This goes as follows:

1. Compute the SVD (5.1) of A.
2. Set $E = -\sigma_n u_n v_n^*$.

The matrix E constructed above clearly satisfies $\|E\|_2 = \sigma_n$ and $A+E$ is singular. Note, moreover, that E is a rank-1 perturbation.

EXERCISE 26. *Let A be the matrix in Exercise 8.*
(a) *Compute the* SVD *of A for $n = 8, 10, 12, 14$.*
(b) *Indicate the distance to singularity of A for the previous values of n.*
(c) *Construct a perturbation E such that $A+E$ is singular, for each of the previous values of n.*

EXERCISE 27. *Assume that A is invertible with* SVD *as in* (5.1). *Give an expression for a* SVD *of A^{-1}.*

The previous arguments are closely related to the condition number for inversion of A. This number is a measure of the sensitivity (conditioning) of the problem of inverting a matrix. It is expected to be big if the matrix is invertible (otherwise the problem of computing its inverse makes no sense) but close to being singular. In particular, from Lemma 5.3(vii) and Exercise 27, we see that

$$\kappa_2(A) = \|A^{-1}\|_2 \|A\|_2 = \frac{\sigma_1}{\sigma_n}.$$

This shows, again, that if σ_n is small, then A is ill-conditioned by inversion.

5.3. Least squares problems. We are now interested in the case where the linear system (3.1) does not have a solution. This happens when the vector b does not belong to the column space of $A \in \mathbb{C}^{m\times n}$. The typical situation is when $m > n$, so the system has more equations than unknowns, because these systems generically have no solution. In this setting, we are interested in computing a vector $\widehat{x}$ which is the "closest" to a solution. But, for this, we first need to define what we mean by "close". This closeness, as in the previous section, is measured in terms of the distance induced by the 2-norm.

Least squares problem (LSP):

Given $A \in \mathbb{C}^{m\times n}$ and $b \in \mathbb{C}^m$, the *least squares problem* associated to $Ax = b$ consists of finding a vector $\widehat{x} \in \mathbb{C}^n$ such that

$$\|A\widehat{x} - b\|_2 = \min\{\|Ax - b\|_2 : \quad x \in \mathbb{C}^n\}. \tag{5.7}$$

The vector $\widehat{x}$ in (5.7) is a *least squares solution* of the system (3.1), and $A\widehat{x} - b$ is called the *least squares residual*. Then, a LSP aims to find those vectors $\widehat{x} \in \mathbb{C}^n$ which minimize the residual.

Some observations are in order:

- For the sake of completeness, we have stated the LSP for complex rectangular matrices, instead of real (and square) ones.
- The least squares solution is not necessarily unique.

The main question related to the LSP is the following: Does it always have a solution? In other words, Given $A \in \mathbb{C}^{m\times n}$ and $b \in \mathbb{C}^m$ arbitrary, does necessarily exist a vector $\widehat{x}$ satisfying (5.7)? The answer to this problem is YES. Now, we are going to review several different ways to compute it.

5.3.1. Solution of LSP using the SVD. Let us decompose the SVD of A (5.1) as in the proof of Theorem 5.2, namely

$$A = U \begin{bmatrix} \Sigma_+ & 0 \\ 0 & 0 \end{bmatrix} V^*, \quad U = \left[\begin{array}{c|c} U_1 & U_2 \end{array}\right], \quad U = \left[\begin{array}{c|c} V_1 & V_2 \end{array}\right],$$

where $\Sigma_+ = \operatorname{diag}(\sigma_1, \ldots, \sigma_r)$ is $r \times r$, U_1 is $m \times r$, U_2 is $m \times (m-r)$, V_1 is $n \times r$, and V_2 is $n \times (n-r)$.

The following result provides a formula for the solution of the LSP.

THEOREM 5.6. (Solution of the LSP). *Let $A \in \mathbb{C}^{m\times n}$ and $b \in \mathbb{C}^m$. Then the solutions of the* LSP (5.7) *are of the form*

$$\widehat{x} = V_1\Sigma_+^{-1}U_1^*b + V_2z,$$

for any $z \in \mathbb{C}^{n-r}$.

Proof. Let $\widehat{x}$ be a solution of the least squares problems, and let us partition

$$V^*\widehat{x} = \begin{bmatrix} V_1^*\widehat{x} \\ V_2^*\widehat{x} \end{bmatrix} = \begin{bmatrix} y \\ z \end{bmatrix}.$$

Now, we replace the SVD of A into the residual

$$A\widehat{x} - b = U \begin{bmatrix} \Sigma_+ & 0 \\ 0 & 0 \end{bmatrix} V^*\widehat{x} - b = U \begin{bmatrix} \Sigma_+ y - U_1^*b \\ -U_2^*b \end{bmatrix}.$$

Since the 2-norm is unitarily invariant (see Remark 10), applying the square of the 2-norm in the previous identity we get

$$\|A\widehat{x} - b\|_2^2 = \|\Sigma_+ y - U_1^*b\|_2^2 + \|U_2^*b\|_2^2.$$

Since the second summand in the right-hand side is independent on y, z, the residual is minimized when the first summand in the right-hand side is zero. This happens when $y = \Sigma_+^{-1}U_1^*b$. Therefore, the solution to the LSP (5.7) is given by

$$\widehat{x} = V \begin{bmatrix} y \\ z \end{bmatrix} = V_1y + V_2z = V_1\Sigma_+^{-1}U_1^*b + V_2z.$$

Note that, by (5.3), $A(V_2z) = 0$, for any $z \in \mathbb{C}^{n-r}$. Therefore, the vector V_2z does not have any influence on the residual, so z can be any vector. ∎

As a consequence of Theorem 5.6, if $\operatorname{rank} A = r < n$, then the LSP (5.7) has infinitely many solutions, and the essential part of the solution contains the matrix $V_1\Sigma_+^{-1}U_1^*$. This matrix is obtained from the SVD of A just inverting the singular values and exchanging the roles of U and V. Note that, when A is invertible, this is just A^{-1} (as expected). Moreover, when A is not invertible, this matrix shares some properties with the inverse of a matrix. In fact, it is a particular case of what is known as a *pseudoinverse* (or *generalized inverse*).

DEFINITION 5.7. *Let $A \in \mathbb{C}^{m\times n}$ with an* SVD *given by* (5.1)*, and partition $\Sigma = \left[\begin{smallmatrix} \Sigma_+ & 0 \\ 0 & 0 \end{smallmatrix}\right]$, with Σ_+ invertible. Then the matrix*

$$A^\dagger := V \begin{bmatrix} \Sigma_+^{-1} & 0 \\ 0 & 0 \end{bmatrix} U^*$$

is called the Moore-Penrose *inverse of A.*

The following lemma supports the use of the word "inverse" for $A^\dagger$.

LEMMA 5.8. *Let $A \in \mathbb{C}^{m\times n}$.*

- *If A is invertible then $A^\dagger = A^{-1}$.*
- *If $\operatorname{rank} A = n$, then $A^\dagger = (A^*A)^{-1}A^*$. This implies $A^\dagger A = I_n$, so $A^\dagger$ is a* left inverse *of A.*
- *If $\operatorname{rank} A = m$, then $A^\dagger = A^*(AA^*)^{-1}$. This implies $AA^\dagger = I_m$, so $A^\dagger$ is a* right inverse *of A.*

EXERCISE 28. *Prove Lemma* 5.8.

Using the Moore-Penrose inverse we can give a more condensed expression for the least squares solution.

COROLLARY 2. *Let $A \in \mathbb{C}^{m\times n}$, and $b \in \mathbb{C}^m$. The solution of the LSP* (5.7) *is*

$$\widehat{x} = A^\dagger b + y, \quad \text{where } y \in \operatorname{Ker}(A).$$

5.3.2. Solution of LSP using the QR factorization. Another way to solve the LSP (5.7) is by means of the so-called normal equations, as explained in the following result.

THEOREM 5.9. *Let $A \in \mathbb{C}^{m\times n}$ and $b \in \mathbb{C}^m$. Then $\widehat{x}$ is a solution of the LSP* (5.7) *if and only if $\widehat{x}$ is a solution of*

$$A^*A\widehat{x} = A^*b. \tag{5.8}$$

EXERCISE 29. *Prove Theorem* 5.9.

The equations of system (5.8) are known as the *normal equations* associated with the LSP (5.7). They relate the LSP with a SLE, which is nice because it allows us to address a LSP as a SLE. However, the solution of the LSP using normal equations can be numerically unstable, since the conditioning of the problem can be as big as $\kappa_2(A)^2$, instead of $\kappa_2(A)$ (see [4, p. 103]).

In this course we have just defined the QR factorization for real square invertible matrices. However, the QR factorization can be extended to complex arbitrary matrices (square and rectangular). Though we are not getting into the details, the QR factorization for complex rectangular matrices is of the same form as for real invertible

matrices. More precisely, given a matrix $A \in \mathbb{C}^{m\times n}$, there are two matrices $Q \in \mathbb{C}^{m\times n}$ (with orthonormal columns, that is, satisfying $Q^*Q = I_n$) and $R \in \mathbb{C}^{n\times n}$ (upper triangular) such that $A = QR$. Using this decomposition, we can find an expression for the solution of the normal equations (5.8). To be precise, (5.8) is equivalent to

$$R^*R\widehat{x} = R^*Q^*b.$$

If A has full column rank (which means that $m \geq n$ and $\operatorname{rank} A = n$), then R^* is invertible, and this system is equivalent to

$$R\widehat{x} = Q^*b.$$

This is an upper triangular system, that can be solved by backward substitution.

5.4. Image processing. Another application of the SVD is in image processing. In particular, we are going to review the implications of the SVD in the process of compression and restoration of images.

For the sake of simplicity, let us focus on the simplest case of a black and white image. The information of this image is stored in a matrix, whose dimensions correspond to the *resolution* of the image. For instance, an image with resolution 1368×1712 means that the image is decomposed into a grid of 1368×1712 small squares (*pixels*), each one having an independent intensity of black color. This corresponds to a matrix of size 1368×1712, where each entry encodes the intensity of black color in the corresponding pixel. In some cases, it may be preferable to reduce the amount of stored information, at the cost of loosing some definition in the image. This leads to the question on how to get the best compromise between the amount of stored information and the quality of the image. The SVD is helpful in this question, based on Equation (5.5) and Theorem 5.5. More precisely, the decomposition in 5.5 allows to store the information of the matrix $A \in \mathbb{C}^{n\times n}$ through the sum of r rank-1 matrices, together with r real values (the singular values $\sigma_1 \geq \cdots \geq \sigma_r > 0$). This is a total amount of $r(m+n)+r$ numbers, since each rank-1 matrix of size $m \times n$ is stored in 2 vectors, one of size $m \times 1$, and the other one of size $1 \times n$. Of course, if $r = m = n$ (so the matrix is invertible), then this amounts to $2n^2 + n$, which is more information than storing directly the $n \times n$ whole matrix. However, when the matrix is rectangular, and $m << n$ or $n << m$, or even when it is square $n \times n$ but $r << n$, this quantity can be much smaller than mn, which is the storage requirement of the whole matrix. Then, what we can do is to replace the whole matrix A by a good low-rank approximation. The notion of "good" here depends on the particular case, but the idea is to find a low rank approximation that allows to decrease the storage cost without loosing too much definition in the image. Theorem 5.5 tells us that the best rank-ρ approximation to a given matrix (in terms of the distance induced by the 2-norm) is obtained by truncating the sum (5.5), namely removing the last summands, and keeping the first ρ summands, which correspond to the largest singular values.

In general, a rank-100 approximation (that is, using only the largest 100 singular values) is enough to get a good approximation to the original image. However, the amount of information stored in this approximation is much less than the one stored in the original matrix. In particular, for an image with 1368×1712 resolution, the information required to store the rank-100 approximation as a matrix (5.5) with $r = 100$ is $100(1368 + 1712) + 100 = 308100$ bytes. Compared to the size of the whole matrix this gives a compression rate of

$$c_r = \frac{308100}{1368 \cdot 1712} = 0.1315,$$

which means that the information stored in the approximation is only 13.15% of the whole information, which is an important saving.

EXERCISE 30. (Mandatory). *Choose your favorite picture (one not having a very high resolution!) and put it into your working path in* `MATLAB` *(for instance, as a jpg file). Name the picture using the command* name `= imread('image.jpg')`. *Then run the command* `compress` *(that should be as well in your working path in* `MATLAB`*), by choosing* 4 *different values for the rank, which is the variable* p *in the input of the code. Your choices should include: a small rank (around* $1/10$ *of the rank of the whole matrix), a moderate choice (around* 100*), and a large number (like* $1/2$ *of the whole rank). Then display the following information:*

- *The original picture.*
- *The* 4 *pictures obtained after applying the* `compress` *code.*
- *The rate of compression of each picture (indicate the computations carried out to get it).*

HIGHLIGHTS:

☞ The SVD is a matrix factorization which displays relevant information of an arbitrary matrix (like the rank and the singular values).

☞ The singular values measure how far is a matrix of certain rank away from the set of matrices with smaller rank.

☞ The SVD is a useful tool in many applied settings, including low-rank approximations, solution of LSP, and image processing.

☞ A LSP arises naturally from a SEL with no solution, and aims to compute the vector with smallest residual (that is, in some sense, the "closest" to a solution).

REFERENCES

[1] J. W. Demmel. *Applied Numerical Linear Algebra.* SIAM, Philadelphia, 1997.

[2] G. H. Golub, C. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, 1996.

[3] N. J. Higham. *Accuracy and Stability of Numerical Methods.* SIAM, Philadelphia, 1998.

[4] I. C. F. Ipsen. *Numerical Matrix Analysis.* SIAM, Philadelhia, 2009.

[5] G. W. Stewart, J. G. Sun. *Matrix Perturtbation Theory* Academic Press, new York, 1990.

[6] L. N. Threfeten and D. Bau III. *Applied Numerical Linear Algebra.* SIAM, Philadelphia, 1997.