Lecture notes on numerical linear algebra

Marc Bonnet

Lecture notes on numerical linear algebra

SIM 203 2020-21[1]

Marc Bonnet
POEMS (CNRS, INRIA, ENSTA), ENSTA, 828 boulevard des Maréchaux, 91120 Palaiseau, France
mbonnet@ensta.fr

[1]Version of August 17, 2021. This document is currently evolving; please check regularly for updates

Contents

CHAPTER 1 MOTIVATION AND EXAMPLES. COMPUTATIONAL CONSIDERATIONS

1.1 MOTIVATION AND OUTLINE. This course is intended as a first introduction to the methods of numerical linear algebra. Students are assumed to be already familiar with the basics of linear algebra, although some facts and properties are recalled here in an effort to make this document self-supporting. Solution methods for solving (often very large) systems of linear equations is one of the main workhorses underlying scientific computation in many areas, including model-based simulation of the response of physical systems, optimization, data analysis and statistics. Even though many students will not need to design or implement such methods, understanding their underlying principles and conditions of applications is a necessary part of the background of anyone involved in mathematical modelling and scientific computation.

The topics presented here follow a natural order:

- Some general considerations relevant to matrix computation,
- Direct solution methods for linear systems and least-squares problems,
- Iterative solution methods for linear systems,
- Numerical solution of eigenvalue problems,
- Ill-posed linear systems and least-squares problems.

There is of course a huge literature on the topic of computational linear algebra, which is very partially reflected by the sample of classical references given in the bibliography of this document without any claim of completeness or extensivity. In particular, the book [14] (still in press as of this writing) is recommmended as a very readable and student-oriented introduction to many of the important topics of computational linear algebra (including advanced topics such as direct solution methods for sparse linear systems), which moreover provides comprehensive examples of implementation (and implementation exercises) based on the Julia programming language (which is open-source and freely available).

To help put this course in context, we also mention (and provide references to) courses on related topics taught at ENSTA Paris.

1.2 EXAMPLES INVOLVING LINEAR SYSTEMS. In this section, we provide a motivation through a series of examples. Our background and field of activity being primarily concerned with the mathematical and computational modelling of systems arising in mechanics (deformable solids, fluids) and more generally in physics (e.g. acoustics, electromagnetism), these motivating exemples are certainly to some extent biased. Mechanics and physics usually rely on known mathematical models describing the underlying physics, often based on ODEs or PDEs, and in this context numerical linear algebra is heavily involved in solution methods for those models, based on discretization (finite elements, boundary elements, finite differences, discontinuous Galerkin methods...). Other situations involve models that are at least partially unknown, the missing knowledge on them being sought from experimental data: this in particular includes *inverse problems* and in many forms (e.g. data assimilation for weather forecast, seismic inversion). Finally, many other applications (e.g. image processing, statistical and data analysis) do not rely on *a priori* chosen mathematical models, but rather attempt to find *ad hoc* models, often without direct physical meaning, that correlate, explain, or allow predictions

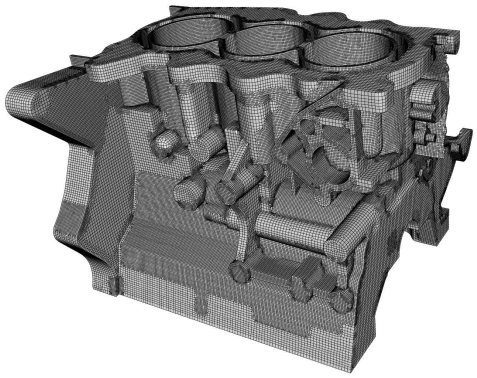


Figure 1.1: Finite element mesh of automotive engine part (from the website of INRIA Gamma 3 team).

from, available data. All those tasks, and many more, rely on computation, and numerical linear algebra is among the key methodological ingredients allowing to achieve those varied goals. Data sets or physical unknowns often have very high dimension, and tasks such as solving large linear systems of equations often occur repeatedly in the overall process, hence the importance of computational efficiency.

1.2.1 Finite element analysis of mechanical structures. This example is both quite classical and very relevant to many sectors of industry (e.g. aerospace, automotive, energy production, civil engineering, biomechanics) where very large models may be solved. Upon finite element (FE) approximation [11, 3, 5] of the elastic equilibrium equations in weak form by means of the Galerkin method, and assuming usual small-strain linearly-elastic constitutive behavior, the unknown kinematic degrees of freedom gathered in a vector U solve the linear system

$$KU = F \tag{1.1}$$

where K is the *stiffness matrix* K while F is the vector of generalized loads obtained by evaluating the virtual work of the applied loads for each basis (test) function of the finite element approximation space.

- Here, the governing matrix K is (i) symmetric, (ii) positive definite[1] and (iii) banded. Therefore K is in particular *sparse*: for a very large FE model, the nonzero entries of K represent only a small fraction of its population.
- Well-known energy principles of mechanics show that (1.1) alternatively expresses that U minimizes the (FE-discretized) potential energy

  $$E_{\text{pot}}(U) := \tfrac{1}{2}U^{\mathrm{T}}KU - U^{\mathrm{T}}F, \tag{1.2}$$

  a remark that is strongly connected to certain *iterative solvers* applicable to the linear system (1.1).

1.2.2 Structural dynamics and modal analysis. Eigenvalue problems are another frequent component of engineering mechanics (in particular). For example, the free vibrations of an undamped elastic solid verify (using a finite element approximation)

$$KU - (2\pi f)^2 MU = 0$$

(where M is the *mass matrix* associated with kinetic energy), which is the *generalized eigenvalue problem* of finding frequencies f and nonzero displacements U (called vibration modes) solving the

[1] Assuming the problem, or the chosen FE approximation, prevents rigid-body motions, failing which K is positive but not invertible and additional precautions must be taken to fix the rigid-body motion contribution to the solution.

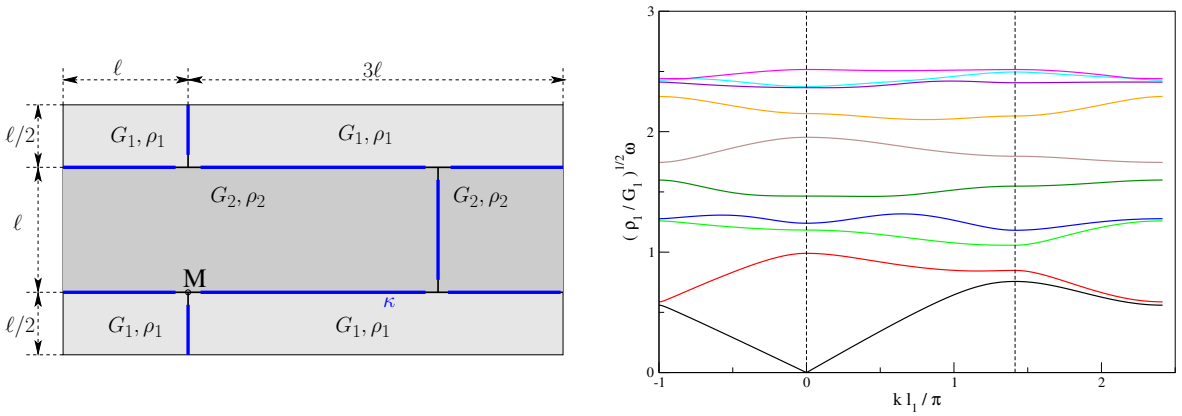


Figure 1.2: Left: Periodicity cell Y for medium made of staggered bricks separated by compliant interfaces (in blue). The bricks are bonded by ligaments near the "triple points" such as M. Right: (wavenumber - frequency) computed dispersion diagram for that cell. Each line plot made of 601 points in wavenumber-frequency space, eachj point requires solving a separate eigenvalue problem. The FE model has 5690 fourth-order triangular finite elements (46426 nodal unknowns). Computations done in connection with research work presented in [21].

above equation. Those modes correspond to potential resonances (so need to be avoided under service loads); moreover they provide convenient basis functions for obtaining low-dimensional models of dynamic response (modal projections).

- The mass matrix is a symmetric positive definite band matrix.

The computation of eigenvalues and eigenvectors can more generally have many motivations:

(i) Physical motivations: investigate the stability of dynamical responses under small perturbations (of systems or excitations), find the dispersion properties of complex materials under wave propagation (i.e. determine the wavenumber / frequency pairs that allow the propagation of free waves), predict potential structural instabilities such as buckling.

(ii) Computational and algorithmic motivations: evaluation of singular value decomposition of arbitrary matrices, of their spectral norm or condition number, estimation of convergence rates of iterative solvers.

To further illustrate (i), the question of whether a certain kind of free wave called *Bloch wave*, characterized by a vector-valued wavenumber $\boldsymbol{k} \in \mathbb{R}^3$, can propagate in a given complex (e.g. periodic) medium is answered by solving a generalized eigenvalue problem of the form (1.2) with the stiffness K replaced by a modified form $\hat{K}(\boldsymbol{k})$ that depends on the wavenumber. Then, solving $\hat{K}(\boldsymbol{k})U - (2\pi f)^2 MU = 0$ for all wavenumbers in a certain region of the $\boldsymbol{k}$-space (producing eigenfrequencies $f_1(\boldsymbol{k}), f_2(\boldsymbol{k}), \ldots$) yield *dispersion diagrams* (which plot $f_i(\boldsymbol{k})$ as functions of $\boldsymbol{k}$) that play an important role in the physics of wave propagation in complex media. See Fig. 1.2 for an illustration.

1.2.3 Boundary element solution of wave scattering. Wave propagation simulation problems often assume the idealization of an unbounded propagation medium exterior to the object that radiates or scatters (e.g. acoustic, electromagnetic or elastic) waves. Applications include noise generation or furtivity studies in the aerospace industry. Such computations are often performed on the bssis of a *boundary element method* (BEM) resulting from the discretization of a boundary integral equation (BIE) governing the (acoustic, electromagnetic, elastic) wave propagation problem. The latter has the typical form

$$\int_S G(\boldsymbol{x},\boldsymbol{y}) u(\boldsymbol{y})\, \mathrm{d}S(\boldsymbol{y}) = f(\boldsymbol{x}) \tag{1.3}$$

where u is the main unknown on the surface S (e.g. the acoustic pressure field), f is the data (e.g. related to the normal velocity of a pulsating surface) and $G(\boldsymbol{x}, \boldsymbol{y})$ is the (known) acoustic Green's function. We refer to e.g. the course [2] for a detailed exposition of BIE theory and BEMs.

- Upon discretization, the BIE (1.3) yields a linear system of the form $GU = F$ where $U \in \mathbb{C}^n$ (unknown), $F \in \mathbb{C}^n$ (data) and $G \in \mathbb{C}^{n\times n}$ (BEM influence matrix). The matrix G is *dense*, and is amenable to block-wise low-rank approximation (for example using the hierarchical matrix method, as in [9])

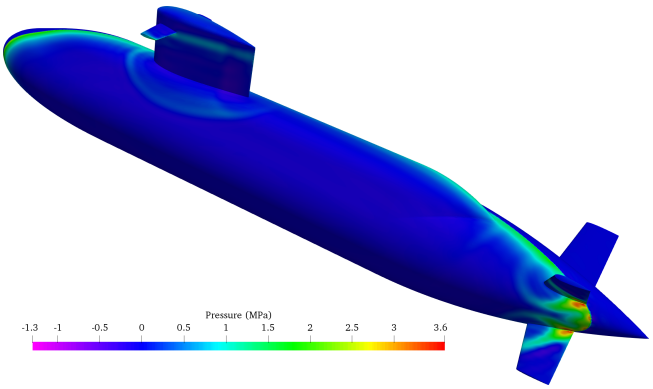


Figure 1.3: Scattering of fluid pressure wave by a rigid motionless submarine: total pressure field on the surface, 5.4 milliseconds after the incident wave first hits the submarine. The boundary element mesh has about $3\,10^6$ pressure unknowns. From [25]

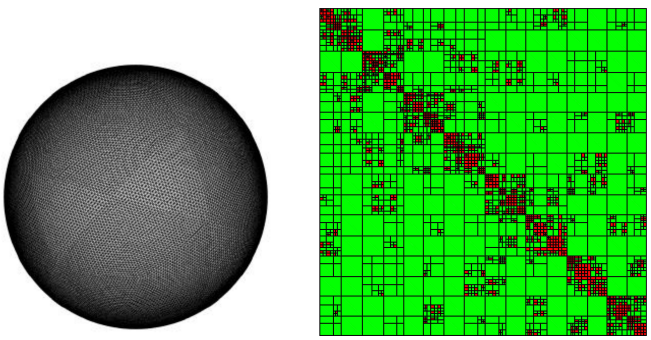


Figure 1.4: Hierarchical matrix blockwise approximation of a BEM matrix. From [9]

1.2.4 Solution of non-linear equations. Many models in mechanics and physics require (after a finite-dimensional approximation process) to solve non-linear equations of the general form:

$$\text{Find } U \in \mathbb{R}^n, \qquad \mathcal{F}(U) = 0$$

where (usually) $\mathcal{F}(U)$ is a $\mathbb{R}^n \to \mathbb{R}^n$ mapping[2]. If $\mathcal{F}$ is differentiable, many solution algorithms follow the iterative Newton-Raphson approach whereby a first-order Taylor expansion of $\mathcal{F}$ about the current iterate U_k allows to define the next iterate U_{k+1} from setting to zero the linearized approximation of $\mathcal{F}(U_{k+1})$:

$$\mathcal{F}(U_{k+1}) \approx \mathcal{F}(U_k) + K_k(U_{k+1} - U_k) \quad \Longrightarrow \quad \text{find } U_{k+1},\ K_k U_{k+1} = K_k U_k - \mathcal{F}(U_k). \tag{1.4}$$

[2]For example, we simply have $\mathcal{F}(U) = KU - F$ in the linear elastic FE case.

where $K_k := \nabla\mathcal{F}(U_k)$ is often called the *tangent matrix* (or, in solid mechanics, the *tangent stiffness*). The above process is started from some initial guess U_0. Each iteration entails solving a linear system of equations governed by the tangent matrix K_k. For instance, with reference to (1.2), nonlinearly-elastic materials may be described by a non-quadratic potentiel energy of the form $E_{\rm pot}(U) = \Phi(U) - U^{\rm T}F$, whose stationarity equation for a given load level E has the form $\mathcal{F}(U) := \nabla_U\Phi(U) - F = 0$ and is no longer linear in U.

- The structure and main characteristics of the tangent matrix K_k are problem-dependent. For many non-linear models arising from mechanical FE models, K_k can be considered as being the stiffness of a fictitious heterogeneous linear-elastic material whose local material parameters depend on the current solution iterate, so shares many of the properties (SPD mand matrix) of standard stiffness matrices.

Nonlinear solution algorithms based on Newton-Raphson iterations of the general form (1.4) are widely implemented in industry engineering mechanics FE codes (such as Abaqus or `code_aster`) for performing a wide variety of non-linear analyses involving material plasticity, frictional contact[3], buckling and many other phenomena.

1.2.5 Identification, inverse problems. Inverse problems are "indirect measurement" situations where quantitative information about some hidden physical variable is sought by exploiting measurements of other, related, physical variables. Classical examples include imaging subterranean media from surface accelerometry data and tumor detection by elastography.

By way of illustration, we briefly consider temperature reconstruction problems that are relevant in space industry: knowing the distribution of temperature on a vehicle after landing, infer temperatures of outer skin during atmospheric reentry phase. This is an instance of the *backwards heat conduction* problem (BHCP): knowing the temperature distribution $x \mapsto \theta(x,T)$ in a conducting body at time $T > 0$, can we infer the distribution $x \mapsto \theta(x,0)$ at an earlier time (say $t = 0$)? The measurement $\theta(\cdot,T)$ and the unknown $\theta(\cdot,0)$ can be shown to obey a linear relationship of the form

$$\theta(\cdot,T) = \mathcal{A}(T)\theta(\cdot,0) \quad\Longrightarrow\quad \text{(discretization) } \Theta(T) = A(T)\Theta(0) \tag{1.5}$$

where the linear operator $\mathcal{A}(T)$ results from the chosen linear heat diffusion physical model. Upon discretization, one obtains a linear system, where the matrix $A(T)$ is dense. The BHCP is known to be ill-posed [8], making its discretized counterpart ill-conditioned: temperature reconstructions with acceptable accuracy are very hard to obtain using "naive" linear algebra methods as soon as the data is even slightly noisy, as illustrated in Fig. 1.5 on a simple spatially 1D configuration. Less-naive approaches designed for ill-conditioned linear systems such as (1.5) will be presented in Chap. 6.

1.2.6 Image processing. A frequent issue in image processing is to "restore", or otherwise improve, images of insufficient quality. For instance, Earth-based telescopes take astronomical images of celestial objects which may be blurred by atmospheric turbulence (through which light propagates before reaching the telescope mirror) or local light pollution. Such problems can often be mathematically formulated as *deconvolution* problems, the imperfect image $\hat{f}$ being related to the perfect image f by

$$\hat{f}(x) = \int_Y k(x-y)f(y)\,\mathrm{d}y \qquad x \in Y$$

(Y being the set of all pixels and the function $y \mapsto f(y)$ representing for example a gray or color level at pixel y), where the *convolution kernel* k models measurement imperfections (due to e.g. the apparatus or environmental conditions). For instance, Gaussian kernels $k(z) = A\exp(-z^2/2b)$ are used to model atmospheric blurring (b is tuned according to the blur severity, and C is a normalization constant). It is well known that k converges (in the distributional limit sense) to the Dirac mass δ as $b \to 0$ (the no-blur limit), in which case the above convolution yields $\hat{f} = f$.

[3] where complications appear due to the non-smooth character of unilateral contact conditions.

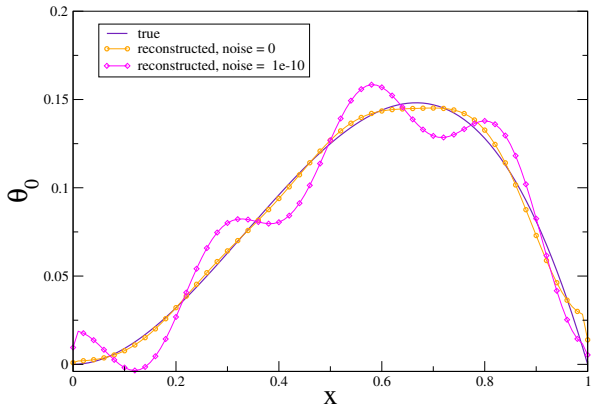


Figure 1.5: 1D reconstruction of initial temperature (at time $t = 0$) from noise-free or slightly noisy data. Observe the quite noticeable degradation in identification caused on the reconstruction by a very slight (simulated) data noise. Observation equation (1.5) solving by (unsuitably) naive methods. Data is temperature at times $t = O(5\,10^3)$s assuming material diffusivity is that of steel.

Figure 1.6: Example of image deblurring: image before degradation (left), degraded image (middle), deblurred image (right)

Upon discretization, deblurring a measured image translates into solving the system

$$\hat{F} = KF \qquad F = \{f(y_1), \ldots, f(y_n)\}^{\mathrm{T}}, \quad \hat{F} = \{\hat{f}(y_1), \ldots, \hat{f}(y_n)\}^{\mathrm{T}}, \quad K_{ij} = \Delta S_j k(x_i - y_j)$$

where ΔS_j is the (small) surface area of the j-th pixel. It turns out that such linear systems are highly sensitive to small perturbations in the data $\hat{F}$, which makes image deblurring a difficult computational problem (and a subject of intense current research). See Figure 1.6 for an illustrative example.

- The matrix K is fully-populated but has a low numerical rank (it can be well approximated by a low-rank matrix). It is strictly speaking invertible, but very ill-conditioned. This topic will be revisited in Chap. 6

1.2.7 Correlation analysis. Consider a data set $D = \left\{y_i, x_{i1}, \ldots, x_{in}\right\}_{i=1}^{m}$ (where each of the m individuals, numbered $i = 1$ to $i = m$, is described by a set of explanatory variables, or *predictors*, $x_1, \ldots, x_n$ and a dependent variable y). Assume that it is desired to find a simple (linear) mathematical relation allowing to predict, on the basis of this data, the value of the dependent variable y for any other individual knowing its predictors $x_1, \ldots, x_n$. Hence, we want to find the "best" model of the form

$$\hat{y} = a^{\mathrm{T}} x + b,$$

which requires finding optimal values for the parameters $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$ of the linear approximation. A common approach is to seek a, b such that the model prediction $\hat{y}$ is closest to y in an average sense

for the data set D, with the average taken as the quadratic mean. In other words:

$$\text{Find } a \in \mathbb{R}^n,\ b \in \mathbb{R} \quad \text{such that} \quad \sum_{i=1}^{n} |y_i - x_i^{\mathrm{T}} a - b|^2 \to \text{minimum}$$

Setting $X = [x_{ij}]$ $(1 \le i \le m,\ 1 \le j \le n)$, this can be recast in matrix terms as a *least squares problem:*

$$\text{Find } a \in \mathbb{R}^n,\ b \in \mathbb{R} \quad \text{such that} \quad \|Xa - (y - b)\|_2^2 \to \text{minimum}$$

Solving this type of problem also rests on numerical linear algebra methods, as we will see.

1.2.8 Learning. Learning methods aim in particular at defining *learning functions* $\mathcal{F}(x)$ that evaluate a classification $y = \mathcal{F}(x)$ associated with some "raw data" x. For example, x is a numerical vector encoding an image, and y is a classification output (e.g. $y = \mathcal{F}(x)$ encodes whether the image x shows a house, a boat, something else?). Such learning functions are for example constructed by composing affine maps $Lx := x \mapsto Ax + b$ and nonlinear functions N, with those steps applied several times (known as *layers*): $\mathcal{F}(x) = N\big(L\big(N(L(\dots Lx))\big)\big)$. Part of the available data (e.g. a set of images) is used as *training data*: A, b (and possibly other parameters) are tuned using optimization methods so that $\mathcal{F}$ computes on the training data the correct (known) classification output; this is the "learning" part. Then, $\mathcal{F}$ can be applied to new data to do actual classification.

Numerical linear algebra is involved in both the definition of the layers of the learning function and the computational methods (in particular optimization methods, which themselves rely heavily on linear algebra) used for tuning weights such as A, b. See [33] (and the references therein) for an introductory exposition together with the accompanying background on linear algebra, optimization, statistics.

1.3 FINITE-PRECISION COMPUTATION.

Despite its paramount importance, we will only briefly address the topic of finite-precision computation, as it is treated in much more detail in the companion course [24]. Numerical computing is based on the well-known *floating-point representation* of numbers, of the form

$$x = s\, m\, b^e$$

where s is the sign of x, m its mantissa, b the basis (normally $b = 2$) and e the exponent. Double-precision real numbers occupy 64 bits (1 for the sign of x, 52 for the mantissa, 11 for $|e|$). The mantissa is such that $1 \le m < 2$ (it has a leading implicit bit set to 1). A few observations immediately come to mind:

- Numbers are in general subject to *roundoff error*;
- Roundoff errors are relative to orders of magnitude; in particular, the numbers described by the floating point representation are not spaced evenly.
- For this reason, estimating *relative* errors on evaluations and algorithms makes more sense than estimating *absolute* errors.

The widely-used IEEE 754 norm for representing floating point numbers and computing with them ensures that

$$\text{for all } x \in \mathbb{R}, \text{there exists } \ \varepsilon,\ |\varepsilon| < \varepsilon_{\text{mach}} \quad \text{such that } \ \mathrm{fl}(x) = x(1+\varepsilon)$$

where $x \mapsto \mathrm{fl}(x)$ is the operator converting a number to its floating point representation and $\varepsilon_{\text{mach}}$, often called the "machine epsilon", is the maximum modulus of relative round-off error ($\varepsilon_{\text{mach}} \approx 10^{-16}$ for double-precision floating-point reals). It also ensures that

$$\text{for all } x \in \mathbb{F}, x \circledast y = \mathrm{fl}(x \star y),$$

where $\star$ is one of the operations $+, -, \times, /, \sqrt{}$, $\circledast$ is the floating-point implementation of that operation and $\mathbb{F}$ is the set of all floating-point numbers, and

$$\text{for all } x \in \mathbb{F}, \text{there exists } \ \varepsilon,\ |\varepsilon| < \varepsilon_{\text{mach}} \quad \text{such that } \ x \circledast y = (x \star y)(1+\varepsilon)$$

Error analysis of computational algorithms (regarding round-off errors) basically consists in inserting

1.4 VECTORS, MATRICES, NORMS.

1.4.1 Matrices.

Vector space. Matrices are arrays of (real or complex) numbers that allow to express linear relationships between elements of finite-dimensional vector spaces. We begin by briefly recalling that vector spaces are sets of objects (called *vectors*) for which certain operations are defined, namely the addition of vectors and the multiplication of a vector by a scalar (i.e. a number). For all usual applications of numerical linear algebra, those scalars are either real or complex numbers, and we will use generically the notation $\mathbb{K}$ to refer to the set of scalars (with either $\mathbb{K}=\mathbb{R}$ or $\mathbb{K}=\mathbb{C}$). A set E is a vector space provided the following axioms are verified:

1. Addition of vectors is commutative:	$xy = y + x$
2. Addition of vectors is associative:	$x + (y + z) = (x + y) + z$
3. There exists a zero vector:	$x + 0 = x$
4. Each vector has an opposite vector:	$x + (-x) = 0$
5. Multiplication of scalars and with a vector are compatible:	$(\alpha\beta)x = \alpha(\beta x)$
6. Scalar multiplication has a unit element:	$1x = x$
7. Scalar multiplication is distributive w.r.t. scalar addition:	$(\alpha+\beta)x = \alpha x + \beta x$
8. Scalar multiplication is distributive w.r.t. vector addition:	$\alpha(x + y) = \alpha x + \alpha y$

To repeat, in this course we will only consider finite-dimensional vector spaces with $\mathbb{K}=\mathbb{R}$ or $\mathbb{K}=\mathbb{C}$. We will usually denote a n-dimensional vector space E on the scalars $\mathbb{K}$ by $\mathbb{K}^n$; this means that a basis of E is (perhaps implicitly) chosen, so that any vector x of E is given as n-uple of numbers: $x = (x_1, \ldots, x_n) \in \mathbb{K}^n$. In this representation, each basis vector e_k is the n-uple $e_k = (0, \ldots, 0, 1, 0, \ldots, 0)$ (with the unit coordinate at the k-th location), and any vector of E is given by

$$x = x_1 e_1 + \ldots + x_n e_n, \qquad (x_1, \ldots, x_n) \in \mathbb{K}^n$$

Matrices. A *matrix* $A \in \mathbb{K}^{m\times n}$ is a rectangular array of numbers a_{ij} $(1 \le i \le m,\ 1 \le j \le n)$, which in particular serves to represent the action of a linear mapping $\mathcal{A} : \mathbb{K}^n \to \mathbb{K}^m$ (after having chosen a basis in both spaces). Letting $x = (x_1, \ldots, x_n) \in \mathbb{K}^n$, its image $\mathcal{A}x = y = (y_1, \ldots, y_m) \in \mathbb{K}^m$ under $\mathcal{A}$ is given by

$$y_i = \sum_{i=1}^{n} a_{ij} x_j, \qquad \text{or} \qquad \begin{Bmatrix} y_1 \\ \vdots \\ y_m \end{Bmatrix} = \begin{bmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \ldots & a_{mn} \end{bmatrix} \begin{Bmatrix} x_1 \\ \vdots \\ x_n \end{Bmatrix} \quad \text{i.e.} \quad \boxed{y = Ax} \text{ in matrix notation} \tag{1.6}$$

Some terminology and notation. We recall and list a the definition of some important types of matrices, and the associated terminology:

- Matrices $A \in \mathbb{K}^{n\times n}$ having the same number n of rows and columns are said to be *square*. Non-square matrices $A \in \mathbb{K}^{m\times n}$ $(m \neq n)$ are said to be *rectangular*.
- A matrix $A \in \mathbb{K}^{m\times n}$ with most of its entries equal to zero is said to be *sparse*[4]. A non-sparse matrix is called *dense*.
- The *transpose* A^{T} of a square matrix $A \in \mathbb{K}^{n\times n}$ is such that $(A^{\mathrm{T}})_{ij} = A_{ji}$.
- The *conjugate transpose* A^{H} of a complex square matrix $A \in \mathbb{C}^{n\times n}$ is such that $(A^{\mathrm{H}})_{ij} = \overline{A_{ji}}$, that is, $A^{\mathrm{H}} = \overline{A^{\mathrm{T}}}$. If A is real, $A^{\mathrm{T}} = A^{\mathrm{H}}$, of course.
- Square real matrices $A \in \mathbb{R}^{n\times n}$ that are equal to their transpose, i.e. verify $A^{\mathrm{T}} = A$, $a_{ji} = a_{ij}$, are said to be *symmetric*.

[4]There is no precise definition of "most". A frequent expectation is that the number of nonzero entries is at most of the same order as the number of rows or columns.

- Square complex matrices $A \in \mathbb{C}^{n\times n}$ that are equal to their conjugate transpose, i.e. verify $A^{\mathrm{H}} = A$, $a_{ji} = \overline{a_{ij}}$, are said to be *Hermitian*. Of course, Hermitian real matrices are simply symmetric matrices.
- Square matrices $A \in \mathbb{K}^{n\times n}$ are *symmetric positive definite* (SPD) if (i) $A^{\mathrm{H}} = A$, (ii) $x^{\mathrm{H}}Ax > 0$ for all $x \in \mathbb{K}^n$ (Hermitian symmetry implies that $x^{\mathrm{H}}Ax \in \mathbb{R}$ for any $x \in \mathbb{K}^n$).

We also list here a few notation conventions that will be used throughout without much further warning:

- We adopt the usual convention that vectors are column vectors, their (conjugate) transpose being row vectors; this is consistent with the usual rules of matrix algebra such as the matrix-vector product $y = Ax$ appearing in (1.6), and also with conventions used in e.g. MATLAB. Scalar products, for example, are hence written as

$$(x, y) = x^{\mathrm{T}}y \quad \text{(for real vectors)}, \qquad (x, y) = x^{\mathrm{H}}y \quad \text{(for complex vectors)}.$$

- Vectors are denoted with lowercase letters, e.g. x, and (in context) x_i is the i-th entry of x.
- Matrices are denoted with uppercase letters, e.g. A, and (in context) a_{ij} is the (i, j)-th entry of A.
- A MATLAB-like colon "`:`" serves to define submatrices by their index ranges. For example, we write

$$\begin{aligned} A_{k:\ell,p:q} &:= [a_{ij}]_{k\le i\le \ell,\, p\le j\le q} && \text{(rectangular submatrix of } A\text{)}, \\ A_{k:\ell,p} &:= [a_{ip}]_{k\le i\le \ell} && \text{(part of the } p\text{-th row of } A\text{)} \end{aligned}$$

1.4.2 Vector norms.. In numerical linear algebra, as in many other areas, it is essential to measure the "magnitude" (and "smallness", "largeness"...) of objects such as vectors or matrices (for instance, the convergence of an algorithm may be defined in terms of errors on the solution becoming "increasingly small"). Such magnitudes are measured using (vector, matrix) norms. Regarding (for now) vectors, any vector norm $\|\cdot\|$ must satisfy the usual defining properties of a norm, namely:

$$\begin{aligned} &\text{zero norm:} && \|x\| = 0 \ \text{ if and only if } \ x = 0, \\ &\text{positive homogeneity:} && \|\lambda x\| = |\lambda|\|x\| \quad \text{for any } \lambda \in \mathbb{K} \qquad \text{for all } x \in \mathbb{K}^n \\ &\text{triangle inequality:} && \|x + y\| \le \|x\| + \|y\| \end{aligned} \tag{1.7}$$

Common vector norms include

$$\|x\|_1 := \sum_{i=1}^{n} |x_i|, \qquad \|x\|_2 := \Big(\sum_{i=1}^{n} |x_i|^2 \Big)^{1/2}, \qquad \|x\|_p := \Big(\sum_{i=1}^{n} |x_i|^p \Big)^{1/p}, \qquad \|x\|_\infty := \max_{1\le i\le n} |x_i|$$

in particular, $\|\cdot\|_2$ is the usual Euclidean norm. All vector norms are equivalent: for any pair $\|\cdot\|_\alpha, \|\cdot\|_\beta$ of norms equipping $\mathbb{K}^n$, there exist constants $C_1 \le C_2$ (depending only on the choice of norms and on the dimension n) such that

$$C_1\|x\|_\alpha \le \|x\|_\beta \le C_2\|x\|_\alpha \quad \text{for all } x \in \mathbb{K}^n, \tag{1.8}$$

and, in particular, we have

$$\begin{aligned} &\|x\|_2 \le \|x\|_1 \le \sqrt{n}\|x\|_2 \\ &\|x\|_\infty \le \|x\|_2 \le \sqrt{n}\|x\|_\infty \qquad \text{for all } x \in \mathbb{K}^n \\ &\|x\|_\infty \le \|x\|_1 \le n\|x\|_\infty \end{aligned}$$

(notice that the upper equivalence constant blows up in all three cases as $n \to \infty$, showing that equivalence no longer holds in the infinite-dimensional limit of e.g. linear spaces of sequences).

Norms are often used to evaluate estimates, i.e. upper bounds, e.g. to show that some vector is "small" or remains bounded under some external changes. Such uses often rely on classical inequalityes, chiefly the Cauchy-Schwarz inequality

$$|x^{\mathrm{H}}y| \le \|x\|_2\|y\|_2,$$

which is a special case of the more-general Hölder inequality

$$|x^{\mathrm{H}}y| \le \|x\|_p\|y\|_q \quad \text{with} \quad \frac{1}{p}+\frac{1}{q}=1,\ 1\le p,q\le\infty.$$

1.4.3 Matrix norms. Matrix norms may be defined as norms induced by vector norms[5]: for any $p>0$, the p norm $\|A\|_p$ of $A\in\mathbb{K}^{m\times n}$ is defined by

$$\|A\|_p := \sup_{x\in\mathbb{K}^n, x\neq 0} \frac{\|Ax\|_p}{\|x\|_p} = \max_{\|x\|_p=1} \|Ax\|_p, \tag{1.9}$$

and thus provides an upper bound on matrix-vector products: for any matrix A and vector x, we have

$$\|Ax\|_p \le \|A\|_p\|x\|_p,$$

with at least one vector x achieving equality in the above inequality[6]. In fact, one may equally well define the q-norm of a matrix induced by the vector p-norm:

$$\|A\|_{p,q} := \sup_{x\in\mathbb{K}^n, x\neq 0} \frac{\|Ax\|_q}{\|x\|_p} = \max_{\|x\|_p=1} \|Ax\|_q,$$

Another frequently used norm is the Frobenius norm $\|\cdot\|_{\mathrm{F}}$ defined by

$$\|A\|_{\mathrm{F}} := \Big(\sum_{i,j}|a_{ij}|^2\Big)^{1/2} = \sqrt{\mathrm{Tr}(AA^{\mathrm{H}})} \tag{1.10}$$

(where A^{H} is the Hermitian transpose of A). The Frobenius norm extends to matrices the Euclidean norm of vectors[7]; it is not an induced norm.

The Frobenius norm (1.10) and all induced matrix p-norms (1.9) are *sub-multiplicative*: they all satisfy

$$\|AB\| \le \|A\|\,\|B\| \tag{1.11}$$

The terminology "matrix norm" is sometimes restricted to norms of matrices that do verify the sub-multiplicativity requirement (1.11). While not part of the basic defining properties (1.7) of a norm, sub-multiplicativity is very useful as it permits to derive estimates (upper bounds) of quantities involving matrix multiplications.

Matrix norms are all equivalent, i.e. obey inequalities of the form (1.8); in particular we have[8]

$$\begin{aligned}
\tfrac{1}{\sqrt{m}}\|A\|_1 \le \|A\|_2 \le \sqrt{n}\|A\|_1 \\
\tfrac{1}{\sqrt{n}}\|A\|_\infty \le \|A\|_2 \le \sqrt{m}\|A\|_\infty \qquad \text{for all } A\in\mathbb{K}^{m\times n} \\
\tfrac{1}{\sqrt{\min(m,n)}}\|A\|_{\mathrm{F}} \le \|A\|_2 \le n\|A\|_{\mathrm{F}}
\end{aligned}$$

Notation convention for norms. The generic norm symbol $\|\cdot\|$ will stand for any unspecified vector norm, and indicate the *induced* matrix norm if evaluated on a matrix (e.g. $\|Ax\| \le \|A\|\,\|x\|$); in particular, a Frobenius norm will always be indicated explicitly (e.g. $\|A\|_{\mathrm{F}}$).

[5] In infinite-dimensional linear spaces, induced norms are usually called *operator norms*, see e.g. [7]

[6] This is true because the range of A, as a finite-dimensional subspace of $\mathbb{K}^m$, is closed, implying the existence of x maximizing $\|Ax\|_p$. If the matrix A is *normal*, i.e. satisfies $AA^{\mathrm{H}} = A^{\mathrm{H}}A$, such x is in fact any eigenvector for the eigenvalue of A having largest modulus, see Sec. 5.

[7] Its infinite-dimensional counterpart in Hilbert spaces is known as the Hilbert-Schmidt norm

[8] noticing again the loss of equivalence in the infinite-dimensional limit

1.5 ACCURACY AND STABILITY. Generally speaking, many scientific computing tasks may be viewed as applying a function $\mathcal{F}$ on a given datum $x \in \mathcal{X}$ to obtained a desired result $y = \mathcal{F}(x)$, $y \in \mathcal{Y}$. The data space $\mathcal{Y}$ and the solution space $\mathcal{Y}$ may be very diverse and complex (for instance, function spaces for which some partial differential equation is well-posed, see [7, 6]). The function $\mathcal{F}$, which we may call the solution operator, may itself be complicated (with nonlinear components, tests...) and defined only implicitly as a sequence of simpler operations (an algorithm).

In this course, we limit ourselves to the case where the data and solution spaces are finite-dimensional, and $\mathcal{F}$ may symbolize diverse computational tasks such as solving the linear system $Ay = b$ (A, b being the data and y the sought solution) or diagonalizing a symmetric matrix A (A being the data, and $x = (\lambda_i, x_i)$ the eigenvalue-eigenvector pairs). We will see for example (Sec. 5) that the latter problem cannot be solved by applying a predefined function to the data, and instead requires an iterative algorithm.

We assume that there is an exact problem to be solved (for example, find the solution of $Ay = b$ with A, b known exactly, i.e. without erors induced by e.g. finite machine precision). Exactness here does *not* pertain to a possible prior approximation methodology leading to the finite-dimensional problem being solved (for example, we treat the finite element approximation of a PDE as exact if the stiffness matrix A and load vector b are exact by virtue of prior computation not suffering from any numerical errors), and $y = \mathcal{F}(x)$ symbolizes this exact problem, where an "exact" solution y is found by applying an "exact" method $\mathcal{F}$ to an exact datum x. In practice, only an imperfect version $\widetilde{\mathcal{F}}$ of the solution operator $\mathcal{F}$ can be achieved[9]. To assess the reliability of the algorithm $\widetilde{\mathcal{F}}$, it is natural to consider questions such as

$$\text{How close to } y = \mathcal{F}(x) \text{ is the approximation } \widetilde{y} := \widetilde{\mathcal{F}}(x)\,?$$

Accuracy. Since a good algorithm must approximate the exact problem well, one may for example consider the relative solution accuracy

$$e_{\text{rel}} = \frac{\|\widetilde{\mathcal{F}}(x) - \mathcal{F}(x)\|}{\|\mathcal{F}(x)\|}. \tag{1.12}$$

Under optimal conditions, the best conceivably achievable accuracy is $e_{\text{rel}} = O(\varepsilon_{\text{mach}})$. Individual floating-point arithmetic operations do meet this goal.

Stability. To achieve $e_{\text{rel}} \approx \varepsilon_{\text{mach}}$ in (1.12) is however unrealistically demanding for most algorithms due to factors like large-scale computations or ill-conditioned problems. For these reasons, it is more appropriate to aim for *stability*: the algorithm $\widetilde{\mathcal{F}}$ for a problem $\mathcal{F}$ is deemed stable if

$$\text{for each } x \in \mathcal{X},\ \frac{\|\widetilde{\mathcal{F}}(x) - \mathcal{F}(\widetilde{x})\|}{\|\mathcal{F}(\widetilde{x})\|} = O(\varepsilon_{\text{mach}}) \qquad \text{for some } \widetilde{x} \text{ such that } \frac{\|x - \widetilde{x}\|}{\|x\|} = O(\varepsilon_{\text{mach}}). \tag{1.13}$$

This can be stated informally as: a stable algorithm must yield nearly the right answer if given a nearly right data (with "nearly" meant in the relative sense). It turns out that many algorithms of numerical linear algebra satisfy a stability condition that is stronger than (1.13), known as *backward stability*:

$$\text{for each } x \in \mathcal{X},\ \widetilde{\mathcal{F}}(x) = \mathcal{F}(\widetilde{x}) \qquad \text{for some } \widetilde{x} \text{ such that } \frac{\|x - \widetilde{x}\|}{\|x\|} = O(\varepsilon_{\text{mach}}). \tag{1.14}$$

Definition (1.14) is stronger than (1.13) in that the first $O(\varepsilon_{\text{mach}})$ is replaced by zero. Stated informally, (1.14) says that a backward stable algorithm must yield exactly the right answer if given a nearly right data. As we will see, algorithms of numerical linear algebra are often assessed in terms of their backward stability.

[9] For instance, $\widetilde{\mathcal{F}}$ is a finite-precision method for solving $Ay = b$, yielding an approximate solution y even for exact data A, b.

The notation $O(\varepsilon_{\text{mach}})$ in (1.13) and (1.14) (and similarly in the remainder of these course notes) means that the quantity on the left-hand side is less than $C\varepsilon_{\text{mach}}$, for some fixed constant $C > 0$, whenever $\varepsilon_{\text{mach}}$ is small enough. A given machine has a fixed (small) value for $\varepsilon_{\text{mach}}$ while the $O(\cdot)$ notation is mathematically defined with reference to a limiting process. The two aspects can be reconciled by envisioning a (virtual) family of computers with smaller and smaller $\varepsilon_{\text{mach}}$.

1.6 CONDITIONING, CONDITION NUMBER. The forward and backward errors may be connected by considering the *relative sensitivity* $\varrho(\mathcal{F}; x, \widetilde{x})$, which is the ratio of forward and backward errors:

$$\varrho(\mathcal{F}; x, \widetilde{x}) := \frac{\|\widetilde{\mathcal{F}}(x) - \mathcal{F}(x)\|}{\|\mathcal{F}(x)\|} \Big(\frac{\|\widetilde{x} - x\|}{\|x\|} \Big)^{-1} = \frac{\|\mathcal{F}(\widetilde{x}) - \mathcal{F}(x)\|}{\|\mathcal{F}(x)\|} \frac{\|x\|}{\|\widetilde{x} - x\|}$$

(note that we have used the defining equality $\widetilde{\mathcal{F}}(x) = \mathcal{F}(\widetilde{x})$ of backward stability, see (1.14)). The *condition number* $\kappa = \kappa(\mathcal{F}; x)$ of $\mathcal{F}$ at $x \in \mathcal{X}$ is then defined as the limiting value of the relative sensitivity $\varrho(\mathcal{F}; x, \widetilde{x})$ in the limit of vanishing data perturbations:

$$\kappa(\mathcal{F}; x) := \lim_{\delta \to 0} \sup_{\|\widetilde{x} - x\| \le \delta} \frac{\|\mathcal{F}(\widetilde{x}) - \mathcal{F}(x)\|}{\|\mathcal{F}(x)\|} \frac{\|x\|}{\|\widetilde{x} - x\|}. \tag{1.15}$$

If the solution operator $\mathcal{F}$ is regular enough, the condition number (1.15) is given by the more-explicit formula

$$\kappa(\mathcal{F}, x) = \frac{\|\mathcal{F}'(x)\| \, \|x\|}{\|\mathcal{F}(x)\|},$$

where $\mathcal{F}'(x)$ is the Fréchet derivative[10] of $\mathcal{F}$ at x (which is assumed to exist). Most often no ambiguity arises as to the solution operator $\mathcal{F}$ and data point x being considered, and we will for short write κ rather than $\kappa(\mathcal{F}, x)$.

Interpretation. As said before, κ is the (limit for small data perturbations of the) ratio between relative solution errors (a.k.a. forward errors) and relative data errors (a.k.a. backward errors). Notice in particular that, by construction, κ is a *dimensionless* number (i.e. it has no physical units, even though the data or solution, or both, may have physical units); this implies that κ is deemed small (resp. large) if $\kappa \ll 1$ (resp. $\kappa \gg 1$). A solution process $\mathcal{F}$ is said to be *well-conditioned* if $\kappa = O(1)$ (relative solution error of the same order as relative data error), and *ill-conditioned* if $\kappa(\mathcal{F}) \gg 1$ (relative solution error much larger than relative data error). To be sound and reliable, computational methods must therefore be well-conditioned.

1.7 CONDITION NUMBER OF LINEAR SYSTEMS. We now derive and illustrate the condition number of the computational task of fundamental interest in this course, namely solving a linear system of equations. We focus on the sensitivity of the solution to data errors, the exposition of solution methods being deferred to Chapter 2 and 4. Accordingly, consider the linear system

$$Ay = b, \tag{1.16}$$

with data $x = (A, b)$ and (unique) solution y, A being assumed to be a square invertible matrix. Consider also a perturbed version

$$(A+E)(y+z) = b+f \tag{1.17}$$

of system (1.16), where E, f are perturbations of the matrix and right-hand side (i.e. the absolute backward errors) and z is the induced perturbation on the solution (i.e. the absolute forward error). To examine how z is linked to (E, f), we use (1.16) and (1.17) and find that z solves the system

$$(A+E)z = f - Ey \qquad \text{with} \quad y = A^{-1}b$$

[10] A function $\mathcal{F} : \mathcal{X} \to \mathbb{R}$ is *differentiable* at $x \in \mathcal{X}$ if there exists a continuous linear functional $\mathcal{F}'(x) \in \mathcal{X}'$, called the Fréchet derivative of $\mathcal{F}$ at x, such that $\mathcal{F}(\widetilde{x}) - \mathcal{F}(x) = \langle \mathcal{F}'(x), \widetilde{x} - x \rangle + o(\|\widetilde{x} - x\|)$ for any $\widetilde{x}$ in a neighborhood of x in $\mathcal{X}$; $\mathcal{X}$ may be any normed vector space (with $\mathcal{X}'$ its topological dual). In the case of present interest where $\mathcal{X}$ is finite-dimensional, the Fréchet derivative can be represented as a vector (the gradient $\nabla\mathcal{F}(x)$ of $\mathcal{F}$ at x) and we have $\mathcal{F}(\widetilde{x}) - \mathcal{F}(x) = [\nabla\mathcal{F}(x)]^{\mathrm{T}}(\widetilde{x} - x) + o(\|\widetilde{x} - x\|)$)

We assume the matrix perturbation to be small enough to have $\|A^{-1}\|\|E\| < 1$; by a Neumann series argument, this guarantees in particular that $A+E$ is invertible, and we moreover have

$$(A+E)^{-1} = (I + EA^{-1})^{-1}A^{-1}, \qquad \|(A+E)^{-1}\| \le \frac{\|A^{-1}\|}{1-\|EA^{-1}\|} \le \frac{\|A^{-1}\|}{1-\|E\|\|A^{-1}\|},$$

so that the solution error may be estimated as

$$\|z\| = \|(A+E)^{-1}(f-Ey)\| \le \frac{\|A^{-1}\|}{1-\|E\|\|A^{-1}\|}\Big(\,\|f\| + \|E\|\|y\|\,\Big).$$

We reformulate the above estimate in terms of the forward relative error $\|z\|/\|y\|$ and the backward relative errors $\|E\|/\|A\|$, $\|f\|/\|b\|$, to obtain

$$\frac{\|z\|}{\|y\|} \le \frac{\|A^{-1}\|\|A\|}{1-\|E\|\|A^{-1}\|}\Big(\,\frac{\|f\|}{\|b\|}\,\frac{\|b\|}{\|A\|\|y\|} + \frac{\|E\|}{\|A\|}\,\Big) \le \frac{\|A^{-1}\|\|A\|}{1-\|E\|\|A^{-1}\|}\Big(\,\frac{\|f\|}{\|b\|} + \frac{\|E\|}{\|A\|}\,\Big),$$

the last inequality resulting from $\|b\| \le \|A\|\|y\|$ being a consequence of $Ay = b$. Upper bounds on the relative sensitivity and its limiting value for small relative data errors $\|E\|/\|A\|$ and $\|f\|/\|b\|$ are therefore obtained as

$$\frac{\|z\|}{\|y\|}\Big(\,\frac{\|f\|}{\|b\|} + \frac{\|E\|}{\|A\|}\,\Big)^{-1} \le \frac{\kappa(A)}{1-\kappa(A)\frac{\|E\|}{\|A\|}} = \kappa(A) + O\Big(\,\frac{\|E\|}{\|A\|} + \frac{\|f\|}{\|b\|}\,\Big), \qquad \text{with} \quad \kappa(A) := \|A^{-1}\|\|A\|.$$

Recalling the definition (1.15) of the condition number, $\kappa(A) = \|A^{-1}\|\|A\|$ is an upper bound of the condition number associated with solving $Ay = b$; it is called the *condition number of the matrix* A. Loosely speaking, $\kappa(A)$ is the multiplicative coefficient which, applied to the relative matrix error or the relative right-hand side error (or both), estimates the induced relative solution error. This concept is, obviously, absolutely essential in numerical linear algebra.

Properties. The following properties of $\kappa(A)$ and remarks in particular must be kept in mind:

- We always have $\kappa(A) \ge 1$ (because $\|A^{-1}\|\|A\| \ge \|\|A^{-1}A\| = \|I\| = 1$ for any induced norm).
- The value of $\kappa(A)$ depends on the choice of (matrix) norm; if defined for the induced p-norm, we sometimes write it $\kappa_p(A)$.
- If the matrix A is normal (i.e. verifies $AA^{\mathrm{H}} = A^{\mathrm{H}}A$), it is unitarily diagonalizable (i.e. $A = Q\Lambda Q^{\mathrm{H}}$ for some unitary matrix Q). In this case, and for the choice $\|\cdot\| = \|\cdot\|_2$ of norm, we have $\|A\|_2 = |\lambda_{\max}|$ and $\|A^{-1}\|_2 = 1/|\lambda_{\min}|$, and hence

  $$\kappa_2(A) = |\lambda_{\max}|/|\lambda_{\min}|,$$

  where $\lambda_{\max}$ and $\lambda_{\min}$ are the eigenvalues of A with largest and smallest modulus (noting that invertibility of A requires $|\lambda_{\min}| > 0$). For instance, for the 2×2 (normal) matrix $A = \mathrm{diag}(1,\varepsilon)$, we have $\kappa(A) = 1/|\varepsilon|$ and A is ill-conditioned for small ε.
- For any orthogonal or unitary matrix Q, we have $\|Qx\|_2 = \|x\|_2$ for all $x \in \mathbb{K}^n$, implying that $\|Q\|_2 = 1$. Similarly, $\|Q^{-1}\|_2 = \|Q^{\mathrm{H}}\|_2 = 1$. Consequently, $\kappa_2(Q) = 1$.
- The condition number $\kappa_2(A)$ of an arbitrary matrix $A \in \mathbb{K}^{m\times n}$ is given in terms of either its *pseudo-inverse* or its *singular values*, see Secs. 3.4, 3.5 and Theorem 3.6.

A simple numerical example. Consider the following matrix A, whose inverse (as given below) is exact:

$$A = \begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix}, \qquad A^{-1} = \begin{bmatrix} 25 & 41 & 10 & -6 \\ -41 & 68 & -17 & 10 \\ 10 & -17 & 5 & -3 \\ -6 & 10 & -3 & 2 \end{bmatrix}$$

We consider perturbations f of b, or E of A, and evaluate the induced perturbations z on the solution y of $Ay = b$:

$$b = \begin{Bmatrix} 32 \\ 23 \\ 33 \\ 31 \end{Bmatrix} \implies y = \begin{Bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{Bmatrix},$$

$$f = \begin{Bmatrix} 0.1 \\ -0.1 \\ 0.1 \\ -0.1 \end{Bmatrix} \implies z = \begin{Bmatrix} 8.2 \\ -13.6 \\ 3.5 \\ -2.1 \end{Bmatrix}, \qquad \begin{matrix} E_{23} = 0.1 \\ E_{ij} = 0 \text{ otherwise} \end{matrix} \implies z \approx \begin{Bmatrix} -5.86 \\ -11.7 \\ -2.43 \\ -3.43 \end{Bmatrix}$$

In fact, the eigenvalues of A, numerically computed (see Sec. 5) as listed below in order of decreasing magnitude, give

$$(\lambda_1, \lambda_2, \lambda_3, \lambda_4) \approx (30.29,\ 3.858,\ 0.8431,\ 0.01015),, \qquad \kappa_2(A) \approx 2.98\,10^3$$

This makes A rather ill-conditioned (relative data errors being amplified about 3000-fold), especially given that its size is only 4.

CHAPTER 2 LINEAR SYSTEMS AND DIRECT SOLVERS

As outlined in Chapter 1, solving linear systems $Ax = b$ is a fundamental computational task with myriads of applications. In addition to addressing inherently linear mathematical problems, more-complex situations involving nonlinearity or optimization often rely on iterative procedures producing sequences of linear systems to be solved.

2.1 SOME GENERAL CONSIDERATIONS. First of all, a given linear system may have no solution, exactly one solution, or infinitely many solutions, depending on the data A, b. We will revisit this topic in more detail in Chapter 3. Generally speaking, a given matrix $A \in \mathbb{K}^{m\times n}$ has a null-space $\mathcal{N}(A)$ (the subspace of $\mathbb{K}^n$ containing all vectors x such that $Ax = 0$) and a range $\mathcal{R}(A)$ (the subspace of $\mathbb{K}^m$ containing all vectors of the form Ax, $x \in \mathbb{K}^n$). The system $Ax = b$ is uniquely solvable if (i) $\mathcal{N}(A) = \{0\}$ and (ii) $b \in \mathcal{R}(A)$, which is in particular true when $m = n$ and A is invertible.

Classical linear algebra provides general formulas for the solution(s) of $Ax = b$, if any, namely the Cramer formulas. For example, if A is square invertible of size n, we have

$$x_i = \det(A_i)/\det(A), \quad \text{with} \quad A_i := [x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n].$$

The Cramer formulas are however impractical except for very small sizes, as the computation of determinants becomes both too expensive and unreliable as the size increases.

2.1.1 Systems with triangular matrices. Triangular matrices play a very important role in solving linear systems, because a linear system whose matrix is triangular can be solved by a very simple method based on successive substitutions. First, let us solve $Ax = b$ with A a lower triangular matrix (i.e. $a_{ij} = 0$ if $j > i$). Taking advantage of the structure of A, we proceed stepwise and find x_1, then x_2, then x_3... until x_n:

$$x_1 = b_1/a_{11}, \quad x_2 = (b_2 - a_{21}x_1)/a_{22}, \ldots \qquad x_i = \Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j\Big)/a_{ii} \quad (i \le n) \tag{2.1}$$

The process (2.1) is called a *forward substitution*. If any of the diagonal coefficients of (lower triangular) A vanishes, A is not invertible and the system A has either no solution, or infinitely many solutions.

If, instead, A is upper triangular, the approach is similar, with the stepwise process (called backward substitution) beginning with the last unknown x_n and proceeding backwards:

$$x_n = b_n/a_{nn}, \quad x_{n-1} = (b_{n-1} - a_{n-1,n}x_n)/a_{n-1,n-1}, \ldots \qquad x_i = \Big(b_i - \sum_{j=i+1}^{n} a_{ij}x_j\Big)/a_{ii} \quad (i \ge 1) \tag{2.2}$$

The substitution processes (2.1) and (2.2), while being very simple, are essential as most direct solution methods consist in reducing general linear systems to a format where the solution is obtainable by solving triangular systems. See Algorithms 2.1 and 2.2 for basic pseudocode versions of (2.1) and (2.2). Moreover, both processes obey the following stability result [15, Chap. 1]:

Theorem 2.1 *Let $A \in \mathbb{K}^{n\times n}$ be either lower- or upper-triangular. Then, the solution $\tilde{x}$ of the linear system $Ax = b$ computed by (forward or backward) substitution under finite-precision conditions*

Algorithm 2.1 Forward substitution

$A \in \mathbb{K}^{n\times n}$	(data – only lower triangular part need be stored)
$x_1 = b_1/a_{11}$	(start at upper left corner)
for $k = 2$ to n **do**	
$\quad x_k = \big(b_k - a_{k,1:k-1}^{\mathrm{T}} x_{1:k-1}\big)/a_{kk}$	(solve forward, one unknown at a time)
end for	

Algorithm 2.2 Backward substitution

$A \in \mathbb{K}^{n\times n}$	(data – only upper triangular part need be stored)
$x_n = b_n/a_{nn}$	(start at lower right corner)
for $k = n-1$ to 1 **do**	
$\quad x_k = \big(b_k - a_{k,(k+1):n}^{\mathrm{T}} x_{(k+1):n}\big)/a_{kk}$	(solve backwards, one unknown at a time)
end for	

satisfies $(A+E)\tilde{x} = b$*, where the matrix perturbation* E *is lower- or upper-triangular and satisfies* $|E| \le n\varepsilon_{mach}|A|$ *(where, for any* $X \in \mathbb{K}^{n\times n}$*,* $|X|$ *is the real positive matrix with entries* $|x_{ij}|$ *and the inequality is meant componentwise).*

Theorem 2.1 means that forward and backward substitution are backward-stable algorithms (see Sec. 1.5), as it states that the solution $\tilde{x}$ computed by the finite-precision substitution method coincides with that of a nearby linear system that is solved exactly (i.e. without roundoff errors). In other words, the data A, b (to which the actual algorithm is applied) and $A+E, b$ (to which the ideal algorithm is applied) are close to each other and yield (under these conditions) the same solution $\tilde{x}$.

Exercise 2.1 *Prove Theorem 2.1, by retracing the sequence of arithmetic operations in* (2.1) *and* (2.2)*, introducing roundoff errors wherever appropriate, and interpreting the resulting relations as defining entries of* E*. For example, the very first step of* (2.1) *gives* $\tilde{x}_1 = (b_1/a_{11})(1+\delta) = b_1/(a_{11}+e_{11})$ *with* $e_{11} = a_{11}(1-\delta)$ *(due to finite precision arithmetic) with* $|\delta| \le \varepsilon_{mach}$*.*

2.1.2 Direct solvers: general idea.

The solution(s) of a linear system, if any (see the discussion of Sec. 3.2 on solvability), can be computed by means of *direct* solution methods.

- A method of solution is called *direct* if it allows to compute the exact solution of a problem within a finite number of operations (assuming idealized conditions of exact arithmetic). For example, forward and backward substitution are direct solution methods.
- By contrast, an *iterative* method typically computes a *sequence* of successive approximations whose limit is the exact solution, implying that the limit is (usually) not exactly reached within a finite number of iterations.

The general idea behind many direct solution algorithms for linear systems is to exploit a multiplicative decomposition of A that involves triangular matrices, and possibly some other kinds of "simple" matrices (e.g. diagonal, orthogonal). Archetypal direct solution methods use the LU decomposition or the Cholesky decomposition, and we will now focus mostly on these two methods.

2.2 LU FACTORIZATION.

Any square invertible matrix admits a factorization

$$A = LU \tag{2.3}$$

where L is lower triangular and U is upper triangular, known as the LU factorization. Both factors L and U are invertible. Once the factorization (2.3) is computed (see method shortly thereafter), a linear system $Ax = b$ is solved as follows in two steps:

$$\text{step (i): find } y \text{ solving } Ly = b; \qquad \text{step (ii): find } x \text{ solving } Ux = y \tag{2.4}$$

Both steps entail solving a *triangular* system, using forward substitution (2.1) for step (i) and backward substitution (2.2) for step (ii).

2.2.1 Computing the LU factorization: basic approach. The LU decomposition uses the well-known idea of *Gaussian elimination*, which consists in applying simple transformations from the left that introduce zeros under the main diagonal of A. The first step of Gaussian elimination can be explained as follows: set A in the form

$$A = \begin{bmatrix} a_{11} & u_1 \\ \ell_1 & A_2 \end{bmatrix}, \tag{2.5}$$

where the column $n-1$-vector ℓ_1 holds the first column of the lower triangular part of A (diagonal excluded), the row $n-1$-vector u_1 likewise holds the first row of the upper triangular part of A, and $A_2 \in \mathbb{K}^{(n-1)\times(n-1)}$ is the remaining part of A. Now, let

$$G_1(z) := \begin{bmatrix} 1 & 0 \\ z & I_{n-1} \end{bmatrix}. \tag{2.6}$$

for some column $(n-1)$-vector z (so $G_1(z)$ is formed by placing z in the first column of the identity matrix I_n under its diagonal). It is then easy to verify that

$$G_1(-\ell_1/a_{11})A = \begin{bmatrix} 1 & 0 \\ -\ell_1/a_{11} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{11} & u_1 \\ \ell_1 & A_2 \end{bmatrix} = \begin{bmatrix} a_{11} & u_1 \\ 0 & A_2' \end{bmatrix}, \tag{2.7}$$

zeroing out the first column of A under the diagonal. Observe that the remaining part A_2 is modified in the process (in fact $A_2' = A_2 - \ell_1 u_1/a_{11}$, $\ell_1 u_1$ being a rank-one matrix of size $n-1$). To continue this process, we introduce the notion of "elimination matrix" that generalizes G_1 as defined by (2.6):

Definition 2.1 (elimination matrix) *Let $k \le n$. We call "elimination matrix" any matrix $G_k(z) \in \mathbb{K}^{n\times n}$ of the form*

$$G_k(z) = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & g_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & g_{n,k} & & 1 \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & z_{k+1} & 1 & \\ & & \vdots & & \ddots \\ & & z_n & & 1 \end{bmatrix}$$

where $z \in \mathbb{K}^{n-k}$ is a given vector. In the sequel, the entries of the elimination vector *z placed in the k-th column under the diagonal of $G_k(z)$ are numbered as $\{z_{k+1}, \ldots, z_n\}$ for consistency with the numbering of the entries of G_k.*

Going back to the elimination process, we left-multiply (2.7) by $G_2(-\ell_2/a_{22})$ (with of course G_2 as in Definition 2.1), to obtain

$$G_2(-\ell_2'/a_{22}')G_1(-\ell_1/a_{11})A = \begin{bmatrix} a_{11} & & u_1 \\ 0 & a_{22}' & u_2' \\ 0 & 0 & A_3' \end{bmatrix}, \qquad \text{where} \quad A_2' = \begin{bmatrix} a_{22}' & u_2' \\ \ell_2' & A_3' \end{bmatrix}$$

and so on, until we reach

$$G_{n-1}(-\ell_{n-1}'/a_{n-1,n-1}') \ldots G_2(-\ell_2'/a_{22}')\, G_1(-\ell_1/a_{11})A = U \tag{2.8}$$

where U is upper triangular. We next observe that any matrix of the form $G_k(z)$ verifies $G_k^{-1}(z) = G_k(-z)$ (this is easily checked by inspection), so that the above equality becomes

$$A = G_1(\ell_1/a_{11})\, G_2(\ell_2/a_{22}) \ldots G_{n-1}(\ell_{n-1}/a_{n-1,n-1})U = LU, \tag{2.9}$$

with L as defined above found to be lower triangular[1]. The just-described steps therefore produce a decomposition $A = LU$ where L and U are lower- and upper-triangular, respectively. Notice that the process will break down if a zero appears in the top left entry of any of the $A'_2, A'_3 \ldots$ generated submatrices.

We illustrate the foregoing process on a simple 4×4 example taken from [36]. Let

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

Zeros are introduced at the desired places in three steps:

$$G_1 A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -4 & 0 & 1 & 0 \\ -3 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 5 & 5 \\ 0 & 4 & 6 & 8 \end{bmatrix}$$

$$G_2 G_1 A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -3 & 1 & 0 \\ 0 & -4 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 5 & 5 \\ 0 & 4 & 6 & 8 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 4 \end{bmatrix}$$

$$G_3 G_2 G_1 A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 2 \end{bmatrix} = U$$

Then:

$$(G_3 G_2 G_1)^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 4 & 3 & 1 & 0 \\ 3 & 4 & 1 & 1 \end{bmatrix} = U \qquad \text{and} \quad A = (G_3 G_2 G_1)^{-1} (G_3 G_2 G_1 A) = LU$$

2.2.2 Implementation. Explaining the LU factorization in terms of matrix operations (such as matrix-matrix multiplications) is convenient for the explaining the method and proving its properties. In practice, the factorization is not implemented as explained: the elimination matrices G_k are not explicitly formed, and matrix products such as those appearing in e.g. (2.7) or (2.9) are not computed as such. The LU factorization algorithm in particular stores all information about the factors L, U *in place*: the memory initially occupied by A is rewritten, with L and U stored in the strict lower triangular, and upper triangular, regions of the original matrix A. Algorithm 2.3 shows a "bare-bones" formulation of the basic LU factorization (observe how entries of L and U are directly stored into A).

Algorithm 2.3 Basic LU factorization

$A \in \mathbb{K}^{n \times n}$	(data)
for $k = 1$ to $n-1$ **do**	
for $j = k+1$ to n **do**	
$a_{jk} = a_{jk}/a_{kk}$	(compute jk-th entry of L)
$a_{j,k:n} = a_{j,(k+1):n} - a_{jk} a_{k,(k+1):n}$	(update entries $U_{j,(k+1):n}$ of U)
end for	
end for	

[1] This results from the (easily verified) fact that, for any column vectors z_k, z_m of appropriate length, $G_k(z_k) G_m(z_m) = G_{km}(z_k, z_m)$, where $G_{km}(z_k, z_m)$ is formed by placing z_k and z_m in columns k and m of the identity matrix I_n under its diagonal. The rest follows by induction in a similar manner.

2.2.3 Shortcomings of basic LU factorization method. The previously-described method, which we will refer to as the "basic" LU factorization, cannot however be used as is, as it presents major shortcomings:

- The method may fail on some "nice" (invertible, well-conditioned) matrices by zeroing a diagonal entry of one of the triangular factors. For example, consider

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 3 & 2 & 2 \\ 1 & 2 & 1 & 2 \\ 2 & 5 & 4 & 5 \end{bmatrix}$$

  which are both invertible and well-conditioned (with $\kappa_2(A) = (3+\sqrt{5})/2$ and $\kappa_2(B) \approx 57.6$). LU factorization fails on A due to $A_{11} = 0$ (preventing Gaussian elimination), and on B because Gaussian elimination replaces b_{33} with a zero at some stage. In fact (see e.g. [19, Thm. 3.2.1]):

  Theorem 2.2 *A matrix A has a basic LU factorization if $\det\big(A_{1:k,1:k} \neq 0\big)$ for $1 \leq k \leq n-1$. This condition on determinants of submatrices is* not *satisfied by all nonsingular matrices. If the LU factorization exists and A is nonsingular, then the factorization is unique.*

- The method may produce ill-conditioned factors L or U even though A itself is well-conditioned. For example, consider

$$A_\varepsilon = \begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix} = L_\varepsilon U_\varepsilon \quad \text{with} \quad L_\varepsilon = \begin{bmatrix} 1 & 0 \\ -\varepsilon^{-1} & 1 \end{bmatrix}, \quad U_\varepsilon = \begin{bmatrix} \varepsilon & 1 \\ 0 & 1-\varepsilon^{-1} \end{bmatrix}$$

  for small values of ε. We have $\kappa_2(A_\varepsilon) = (3+\sqrt{5})/2 + O(\varepsilon)$ and $\kappa_2(L_\varepsilon) = 1 + O(\varepsilon)$ but $\kappa_2(U_\varepsilon) = O(\varepsilon^{-2})$, making step (ii) of (2.4) an ill-conditioned linear system.

The latter example shows that one source of potential instability of the basic LU factorization is that the process may encounter small pivots a_{kk}. We saw that "zeroing out" the k-th column of A below the diagonal consists in a left multipication by the matrix $G_k(\ell_k/a_{kk})$ defined in terms of the current column $\ell_k = A_{k:n,k}$ of the lower-triangular part of A, where this whole column is divided by the current pivot a_{kk}. On encountering a pivot of too-small modulus, the process contributes large entries into L and this degrades the conditioning of L. In fact, the following estimate is available [19, Thm. 3.3.1]:

Theorem 2.3 (roundoff error associated with LU factorization) *Assume, as a best-case situation, that $A \in \mathbb{K}^{n\times n}$ contains floating-point numbers (so has no roundoff errors arising from finite machine precision). Assume that the basic LU factorization encounters no zero pivot. The computed triangular factors $\widetilde{L}, \widetilde{U}$ then satisfy*

$$\widetilde{L}\widetilde{U} = A + E, \qquad |E| \leq 3(n-1)\varepsilon_{mach}\big(|A| + |L||U|\big)$$

where $|X|$ denotes the matrix with entries $|x_{ij}|$ (in particular $|X|$ is not a norm of X!).

- As an illustration, consider the simple example

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = LU \qquad \text{with} \quad L = \begin{bmatrix} 1 & 0 \\ c/a & 1 \end{bmatrix}, \quad U = \begin{bmatrix} a & b \\ 0 & d - bc/a \end{bmatrix}$$

Exercise 2.2 *Prove Theorem 2.2.*

Exercise 2.3 *Prove Theorem 2.3 by induction on n.*

Those shortcomings are remedied by a modification of the LU factorization method that incorporates *pivoting*, i.e. judicious permutations of lines or columns.

2.2.4 Computing the LU factorization: pivoting. Theorem 2.3 indicates that the roundoff error on the LU factorization will remain small if both computed factors L, U remain small compared to A. A way to avoid large factors L, U is to avoid divisions by too-small pivots in the zeroing-out process (i.e. to avoid forming vectors ℓ_k/a_{kk} that grow due to a_{kk} having a small modulus). This is accomplished by *pivoting*, i.e. effecting permutations of rows, columns or both so that the pivot location after permutation contains a large (in modulus) entry, whereby the vector ℓ_k/a_{kk} used in subsequent eliminations in column k has small entries. Pivoting is also expected to circumvent the failures implicit to Theorem 2.2 by making all relevant submatrices invertible.

The ideal pivoting consists in one row permutation and one column permutation chosen such that the *largest* entry in the part of A not yet triangularized gets placed at the pivot position. For example, consider the starting situation (2.5) (repeated for convenience):

$$A = \begin{bmatrix} a_{11} & u_1 \\ \ell_1 & A_2 \end{bmatrix},$$

and suppose that a_{11} is small relative to the entries in ℓ_1 (so that $-\ell_1/a_{11}$ effecting elimination via row combinations is undesirably large). Suppose besides that (say) a_{53} is the entry of A with largest modulus. If we then permute rows 1 and 5, and columns 1 and 3 (obtaining a permuted version $\hat{A}$ of A), the former entry a_{53} gets moved to the a_{11} location (i.e. $\hat{a}_{11} = a_{53}$). Since $\hat{a}_{11}$ is largest, the vector $-\hat{\ell}_1/\hat{a}_{11}$ has all its entries of modulus less than 1. After this elimination step (on the first column), we can seek again the best pivot in the $(n-1)\times(n-1)$ submatrix $\hat{A}'_2$ (see (2.7)), and so on.

Pivoting can be done in several ways:

- *Full pivoting:* this is the approach just outlined (find the largest entry in the remaining $(n-k+1)\times(n-k+1)$ submatrix $\hat{A}'_k$. This find the best pivot at each step but at the cost of an expensive search.
- *Row pivoting:* at step k, the largest (in modulus) entry is sought only in the column vector ℓ'_k, where
$$A'_k = \begin{bmatrix} a'_{kk} & u'_k \\ \ell'_k & A'_k \end{bmatrix}$$
Say the largest entry found is in row m; in that case, rows k and m are permuted so that the pivot moves at the location of a_{kk} and the elimination vector obtained after permutation has all entries with modulus ≤ 1. This approach is obviously more economical regarding the pivot search.
- *Rook pivoting:* this is an improvement on row pivoting that achieves imperfect full pivoting. Basically, the search alternates between row and column searches until a pivot that is largest in both its row and its column is found.

Now, there is the question whether pivoting still decomposes A as $A = LU$ with L lower-triangular and U upper-triangular. We will only examine in some detail the row-permutation approach. The permutation of two rows of a matrix X can be formally represented via a left multiplication by a permutation matrix[2] Π, so that ΠX is the row-permuted version of X. Assuming row pivoting to be done at each step, we reach

$$G_{n-1}\Pi_{n-1}\ldots G_2\Pi_2 G_1\Pi_1 A = U \tag{2.10}$$

by analogy with (2.8) (G_k being elimination matrices and Π_k being permutation matrices), where U is upper triangular. Now, we rewrite the left-hand side by using that $\Pi\Pi^{\mathrm{T}} = I$ for any permutation matrix and judiciously inserting such combinations. To explain the process, assume that A is square 4×4, so that

$$G_3\Pi_3 G_2\Pi_2 G_1\Pi_1 A = U.$$

[2]Permutation matrices encode permutation of row or columns of matrices via matrix multiplication. They are the identity matrix with either its rows or its columns permuted depending on the (column or row) permutation to encode. For example, letting $\Pi = \left[\begin{smallmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{smallmatrix}\right]$ and $A \in \mathbb{K}^{m\times 3}$, $A\Pi$ is A with its second and third columns permuted. Permutation matrices are real orthogonal (so $\Pi^{-1} = \Pi^{\mathrm{T}}$).

We then rewrite the above equality as

$$G_3(\Pi_3 G_2 \Pi_3^{\mathrm{T}})(\Pi_3\Pi_2 G_1 \Pi_2^{\mathrm{T}}\Pi_3^{\mathrm{T}})\Pi_3\Pi_2\Pi_1 A = U. \tag{2.11}$$

and observe that $\hat{G}_2 := \Pi_3 G_2 \Pi_3^{\mathrm{T}}$ and $\hat{G}_1 := (\Pi_3\Pi_2 G_1 \Pi_2^{\mathrm{T}}\Pi_3^{\mathrm{T}})$ are elimination matrix with the same structure as G_2 and G_1 (see Exercise 2.4) while $\hat{\Pi} := \Pi_3\Pi_2\Pi_1$ is a permutation matrix. The equality (2.11) thus takes the form

$$G_3\hat{G}_2\hat{G}_1\hat{\Pi}A = U.$$

We can generalize this process to any matrix size n, so that (2.10) takes the form

$$G_{n-1}\hat{G}_{n-2}\hat{G}_{n-3}\dots\hat{G}_1\hat{\Pi}A = U$$

where $\hat{G}_k$ are elimination matrices and $\hat{\Pi}_{n-1}$ is a permutation matrix (so we managed to put the net permutatiuon "on the right" of the whole product). As seen previously,

$$\left[G_{n-1}\hat{G}_{n-2}\dots\hat{G}_2\hat{G}_1\right]^{-1} = \hat{G}_1^{-1}\hat{G}_2^{-1}\dots\hat{G}_{n-2}^{-1}\hat{G}_{n-1}^{-1} =: L$$

is a lower triangular matrix, and we finally find

$$\hat{\Pi}A = LU,$$

that is, $A = LU$ up to a row permutation defined by $\hat{\Pi}$. Hence, the LU-factorized form of the original system $Ax = b$ is

$$LUx = \hat{\Pi}^{\mathrm{T}}b \tag{2.12}$$

so that one just needs to apply the permutation $\hat{\Pi}^{\mathrm{T}}$ to the right-hand side b and apply the usual solution method.

Exercise 2.4 *Let $G_k(u)$ be an elimination matrix with associated $(n-k)$-vector u. Let Π_k a permutation matrix acting on two rows k_1, k_2, with $k_1 > k$ and $k_2 > k$. Show that $\Pi_k^T G_k(u)\Pi_k = G_k(\hat{u})$, where the column vector $\hat{u}$ (with entries numbered $k+1$ to n) is u with entries k_1 and k_2 permuted (i.e. $\hat{u}_{k_1} = u_{k_2}$ and $\hat{u}_{k_2} = u_{k_1}$).*

Exercise 2.5 *Explain in more detail how the decomposition form* (2.12) *is reached.*

2.2.5 Computational complexity of LU factorization. It is relatively easy to estimate the computational work incurred by the basic LU factorization of a dense matrix:

- Step k of the elimination process (i.e. zeroing out the k-th column under the diagonal) requires a total of $(n-k)(n-k+1)$ multiplications and as many additions.
- Summing over k, we estimate

$$\text{operation count of LU} = \sum_{k=1}^{n-1} 2(n-k)(n-k+1) = \tfrac{2}{3}n(n^2-1) \sim \tfrac{2}{3}n^3 \tag{2.13}$$

  where the symbol "$\sim$" (equivalent to) indicates the asymptotic value for large n (i.e. "op. count $\sim \frac{2}{3}n^3$" means "op. count $= \frac{2}{3}n^3(1+o(1))$ as $n \to \infty$").

Then, row (or column) pivoting adds $o(n^3)$ supplementary computational work. For large matrices, we are mainly interested by the leading estimate, lower-degree (in n) contributions becoming of marginal relevance. Estimates such as (2.13) are often called *computational complexity* estimates[3], and this terminology will be used in the sequel.

Exercise 2.6 *Explain in more detail the above-outlined operation count, by inventorying the arithmetic operations entailed in step 1 (i.e. the matrix-matrix product $G_1(-\ell_1/a_{11})A$ in* (2.7)*), generalizing the reasoning to the subsequent steps, and using well-known formulas for sums of powers of consecutive integers.*

[3]Sometimes such estimates are found by counting elementary operations that combine one multiplication and one addition, i.e. scalar evaluations of the form $y = ax+b$. The corresponding asymptotic complexity for the LU factorization would be $\frac{1}{3}n^3$ operations.

2.2.6 Band matrices. A matrix A is said to be a *band matrix* (with lower bandwidth b_{L} and upper bandwidth b_{U}) if $a_{ij} = 0$ whenever $i > j+b_{\mathrm{L}}$ or $j > i+b_{\mathrm{U}}$. Revisiting the steps of the LU factorization method, one can show that the LU factorization takes full advantage of the band structure, see [15, Prop. 2.3]:

Theorem 2.4 (LU factorization of band matrices) *Applying the basic LU factorization to a band matrix A, we find $A = LU$ where L has lower bandwidth b_L and U has upper bandwidth b_U. When the bandwidths b_L and b_U are both small relative to n, computing the LU factorization asymptotically requires $2nb_Lb_U$ arithmetic operations, the asymptotic memory space needed for storing the factorization is $n(b_L+b_U+1)$, and solving the triangular systems by substitution asymptotically costs $2n(b_L+b_U)$ operations.*

If instead partial pivoting is used in the LU factorization, L still has lower bandwidth b_L while U has upper bandwidth b_U+b_L.

2.3 CHOLESKY AND LDL$^{\mathrm{T}}$ FACTORIZATIONS. The LU factorization (with partial pivoting) is suitable for solving any linear system whose matrix A is square invertible. Many applications involve systems where A has the stronger property of being *symmetric positive definite* (abbreviated thereafter as SPD). **(list few examples)**

Definition 2.2 (SPD matrices) *A square matrix $A \in \mathbb{K}^{n\times n}$ is symmetric positive definite (SPD) if (i) $A^H = A$, (ii) $x^HAx > 0$ for all $x \in \mathbb{K}^n$ (notice that Hermitian symmetry implies that the quadratic form $x^HAx > 0$ is necessarily real-valued).*

2.3.1 Computing symmetric factorizations of SPD matrices.. We will now show that a symmetric factorization of A can be derived by a direct adaptation of the Gaussian elimination approach used for the LU factorization without pivoting. We begin by writing

$$A = \begin{bmatrix} a_{11} & \ell_1^{\mathrm{H}} \\ \ell_1 & A_2 \end{bmatrix},$$

where the column $n-1$-vector ℓ_1 holds the first column of the lower triangular part of A (diagonal excluded) and $A_2 \in \mathbb{K}^{(n-1)\times(n-1)}$, the remaining part of A, is symmetric (i.e. $A_2^{\mathrm{H}} = A_2$). Recalling definition (2.6) of Gaussian elimination matrices G_1 and taking advantage of the Hermitian symmetry of A, we find

$$G_1(-\ell_1/a_{11})AG_1^{\mathrm{H}}(-\ell_1/a_{11}) = \begin{bmatrix} a_{11} & 0 \\ 0 & A_2 - \ell_1\ell_1^{\mathrm{H}}/a_{11} \end{bmatrix}, \qquad A_2' := A_2 - \ell_1\ell_1^{\mathrm{H}}/a_{11}$$

and we observe at this point that for any SPD matrix A and any invertible matrix $B \in \mathbb{K}^{n\times n}$, $B^{\mathrm{H}}AB$ is SPD (which is easily proved and left to the reader). Hence, the matrix in the right-hand side above is SPD. Moreover, by the previously-mentioned invertibility property of any $G_k(z)$, we have

$$A = G_1(\ell_1/a_{11}) \begin{bmatrix} a_{11} & 0 \\ 0 & A_2' \end{bmatrix} G_1^{\mathrm{H}}(\ell_1/a_{11}),$$

where the $(n-1)\times(n-1)$ block $A_2' = A_2 - \ell_1\ell_1^{\mathrm{H}}/a_{11}$ is SPD.

The just-described symmetric Gaussian elimination process can then be applied to A_2', and so on recursively. This yields

$$A = LDL^{\mathrm{H}} \qquad \text{with } D \text{ diagonal SPD, } L = G_{n-1}(\ell'_{n-1}/a'_{n-1,n-1}) \dots G_2(\ell_2'/a_{22}')\, G_1(\ell_1/a_{11}). \quad (2.14)$$

For the reasons already given in the case of LU factorization, D as defined above is lower-triangular with unit diagonal. Since, as mentioned before, the "middle" matrices appearing in the recursive elimination remain SPD, D is SPD, which means (since it is diagonal) that its diagonal entries are real and strictly positive.

The factorization (2.14) is usually known as the LDL$^{\mathrm{T}}$ factorization of a SPD matrix A. The closely-linked Cholesky factorization of A is given by

$$A = GG^{\mathrm{H}} \qquad \text{with} \quad G := LD^{1/2} \tag{2.15}$$

(the square root $D^{1/2}$ being defined simply as the diagonal matrix such that $(D^{1/2})_{ii} = \sqrt{D_{ii}}$, $1 \le i \le n$). The matrix G in the Cholesky factorization (2.15), often called the *Cholesky factor* of A, is (as easily checked) lower-triangular.

- The Cholesky factorization method can be found by another method, namely writing the sought factorization $A = GG^{\mathrm{H}}$ (with G upper triangular) in component notation ($a_{ij} = \sum_{k=1}^{i} g_{ik}\bar{g}_{jk}$ for $i \le j$, with sum limits accounting from the assumed lower-triangular nature of G) and finding the entries of G by solving these equations columnwise ($j = 1$ yields g_{11}, then $j = 2$ yields g_{21}, g_{22} and so on).
- The Cholesky factorization can be considered as defining a square root $A^{1/2} := G^{\mathrm{H}}$ of a SPD matrix A, in the sense that $A = A^{\mathrm{H}/2}A^{1/2}$ with this definition.

The Cholesky factorization is shown in concise form in Algorithm 2.4, where entries a_{ij} $(i \le j)$ of the upper triangular part of A are replaced with those of the upper triangular Cholesky factor G^{H}. Contrarily to the basic LU factorization (without pivoting), the Cholesky factor cannot grow large[4], for the following reasons: (i) all diagonal entries a_{ii} of A verify $a_{ii} > 0$ (by virtue of A being SPD), and (ii) evaluating a_{ii} from $A = GG^{\mathrm{H}}$ yields $a_{ii} = \sum_{k=1}^{i} g_{ik}\bar{g}_{ik} = \sum_{k=1}^{i} |g_{ik}|^2$; in particular, $|g_{ik}| \le \sqrt{a_{ii}}$ $(1 \le k \le i)$. In fact, the Ckolesky factorization is proved to be backward stable [36, Chap. 23].

Theorem 2.5 (backward stability of Cholesky factorization) *Let $A \in \mathbb{K}^{m\times n}$ with $m \ge n$. Let $\widetilde{G}$ be its Cholesky factor computed (in finite precision) using Algorithm 2.4. Then there exists a matrix $E \in \mathbb{K}^{m\times n}$ such that*

$$A + E = \widetilde{G}\widetilde{G}^{H} \qquad \textit{with} \quad \|E\|/\|A\| = O(\varepsilon_{mach})$$

Algorithm 2.4 Cholesky factorization

```
1: A ∈ K^{n×n}                          (data – only upper triangular part need be stored)
2: for k = 1 to n do
3:    for j = k+1 to n do
4:       a_{j,j:n} = a_{j,j:n} − a_{k,j:n} a_{kj}/a_{kk}     (update j-th row of upper Cholesky factor G^H)
5:    end for
6:    a_{k,k:n} = a_{k,k:n}/√a_{kk}                          (Normalize k-th row of upper Cholesky factor G^H)
7: end for
```

Exercise 2.7 *Derive the Cholesky factorization from the equations $a_{ij} = \sum_{k=i}^{j} \bar{g}_{ki}g_{kj}$, and show how A being assumed to be SPD ensures completion of the algorithm.*

2.3.2 Solution of SPD linear systems. Once the factorization (2.15) is computed, solving a SPD linear system $Ax = b$ again follows two steps:

$$\text{step (i): find } y \text{ solving } Gy = b; \qquad \text{step (ii): find } x \text{ solving } G^{\mathrm{H}}x = y$$

Both steps entail solving a *triangular* system, using forward substitution (2.1) for step (i) and backward substitution (2.2) for step (ii); they are identical to (2.4) except for the fact that here the same triangular factor G is used for both steps.

[4] Alternatively, using the singular value decomposition $G = U\Lambda V^{\mathrm{H}}$ (see Sec. 3.4), one has $A = (U\Lambda V^{\mathrm{H}})(U\Lambda V^{\mathrm{H}})^{\mathrm{H}} = U\Lambda^2 U^{\mathrm{H}}$, which implies $\|A\|_2 = \|G\|_2^2$.

2.3.3 Computational complexity and stability of Cholesky factorization. Similarly to the LU factorization, arithmetic operations incurred by the Cholesky factorization of a dense matrix can be counted, for example from Algorithm 2.4. For given j,k, instruction 4 of Algorithm 2.4 effects $n-j+1$ multiplications and as many additions, whereas instruction 6 effects $n-k+1$ divisions for each k. Neglecting the work entailed by evaluating the n square roots $\sqrt{a_{kk}}$, we find

$$\text{operation count of Cholesky} = \sum_{k=1}^{n} \left\{ n-k+1 + \sum_{j=k+1}^{n} 2(n-j+1) \right\} = \tfrac{1}{3}n(n+1)(2n+1) \sim \tfrac{1}{3}n^3$$

2.3.4 Band matrices. A SPD matrix A is said to be a *band SPD matrix* (with bandwidth b) if $a_{ij}=0$ whenever $|i-j|>b$. On revisiting its steps as given in Algorithm 2.4, the Cholesky factorization is, like the LU factorization, found to take full advantage of the band structure:

- The Cholesky factor G is a band matrix with lower bandwidth b;
- When b is small relativee to n, the factorization is found with a computational work of (asymptotically) nb^2 arithmetic operations.

Exercise 2.8 *Prove the properties (preservation of band structure and width, asymptotic operation count) of the Cholesky factorization of band matrices.*

2.4 SPARSE MATRICES. Classical direct solution methods such as the LU or Cholesky factorizations are initially designed with dense matrices in mind. More to the point, if A is sparse, factors L,U or G normally do not obey the same sparsity pattern: nonzero entries appear in the factors at locations (i,j) where initially $a_{ij}=0$, a phenomenon usually called *fill-in* which can reduce (or even negate in some cases) the computational advantage afforded by the sparsity of the original problem matrix A.

The design of direct solvers that optimally exploit matrix sparsity is a recent research area, with resulting algorithm making frequent use of graph theory and other concepts from discrete mathematics; for an introduction, see for example the last chapter of [14]. They are somewhat easier to formulate for SPD matrices, in view of the following considerations:

- Some applications naturally produce *banded* SPD matrices, for which there is an integer $p<n-1$ such that $|i-j|>p \implies a_{ij}=0$ (A then has *bandwidth* $2p+1$). For example, diagonal (resp. tridiagonal symmetric) matrices are banded with bandwidth 1 (resp. 3). The finite element method produces symmetric banded stiffness and mass matrices [5, 11, 3].

  The Cholesky factorization applied to a banded SPD matrix produces a Cholesky factor G that is also banded (i.e. $i-j>p \implies \ell_{ij}=0$). However, zeros in the original band of A may be filled in by the Cholesky factorization. In the finite element method, memory occupied by A and its Cholesky factor L is optimized by using sparse matrix storage methods for the respective bands. Similarly, for SPD matrices stored in *skyline* fashion (also a common occurence in finite element methods), the skyline profile of A is preserved by the Cholesky factor G.
- More generally, the symmetric form of the factorization $A=GG^{\text{H}}$ introduce simplifications in the graph or tree structures involved in the resulting algorithms, see e.g. [14]

CHAPTER 3 LEAST-SQUARES PROBLEMS

In Chapter 2 we have only considered linear systems involving a square invertible matrix. Many areas in science and engineering (such as the quantitative interpretation of experimental data, and data analysis more generally) give rise to linear systems that are not square, let alone invertible. Accordingly, we consider in this section the more-general case of linear systems $Ax = b$ where $A \in \mathbb{K}^{m\times n}$ and $b \in \mathbb{K}^m$ without making any *a priori* assumptions on A; in particular, depending on the situation being analysed, we may have $m > n$ (overdetermined system), $m = n$, or $m < n$ (underdetermined system) and want to make the best use of the available information A, b. As before, decomposition methods applied to A will be instrumental for analysing the problem at hand and formulating solution algorithms. LU and Cholesky factorizations are not applicable to arbitrary matrices, Instead our main tools will consist of two decomposition methods that apply to arbitrary matrices and have widespread uses, namely the QR factorization (Sec. 3.1) and the singular value decomposition (Sec. 3.4).

It is important to emphasize that both the QR factorization and the singular value decomposition (SVD) have widespread uses that go well beyond solving least-squares problems. Both methods allow to "dissect" a given linear system and gain insight into attributes like solvability, rank deficiency, conditioning and sensitivity to data errors. Moreover, many other matrix-based computational tasks use either method; for example, efficient algorithms for computing eigenvalues of matrices use the QR factorization (see Sec. 5), while the SVD is extremely useful for understanding and solving ill-conditioned systems of linear equations (see Chap. 6).

Norms in this chapter. Since least-squares problems are intrinsically linked to Euclidean norms (i.e. 2-norms) of residuals and rely a lot on orthogonality, norms $\|\cdot\|$ in this chapter implicitly refer to the (Euclidean) 2-norm and its induced (spectral) matrix norm.

3.1 QR FACTORIZATION. Like the LU or Cholesky factorizations, the QR factorization aims at at a multiplicative decomposition of A involving a triangular factor, and proceeds by zeroing out entries of A below its diagonal. The QR factorization does not perform "zeroing-out" steps using matrices of the previously-used type (2.6), for reasons that will be explained later, and instead resorts to left-multiplications of A by *Householder reflections*. A Householder reflection matrix is a matrix $F(v) \in \mathbb{K}^{m\times m}$ of the form

$$F(v) = I_m - 2\frac{v\,v^{\mathrm{H}}}{\|v\|^2} \tag{3.1}$$

for some (arbitrary) column vector $v \in \mathbb{K}^m$. As can be readily checked, any such $F(v)$ is unitary, i.e.

$$F(v)F(v)^{\mathrm{H}} = F(v)^{\mathrm{H}}F(v) = I_m$$

Geometrically speaking, $F(v)$ represents a reflection whose invariant subspace is the hyperplane orthogonal to v: we have $F(v)v = -v$ and $F(v)v^\perp = v^\perp$; see Fig. 3.1 for a geometrical interpretation. In the QR method, v is chosen so that a left multiplication by $F(v)$ introduces zero entries at the appropriate locations of A.

We examine in detail the most-common case where $A \in \mathbb{K}^{m\times n}$ with $m \geq n$ (overdetermined systems of equations). Consider the first step, whose aim is to introduce zeros in all the first column

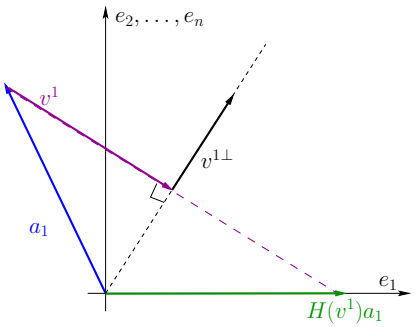


Figure 3.1: Householder reflector $F(v^1)$: geometrical interpretation (case $\text{sign}(a_{11}) = -1$)

$a_1 = A_{1:m,1}$ (with entries a_{1j}, $1 \le j \le m$) except a_{11} (i.e. to replace $A_{2:m,1}$ by zeros). Choosing v^1 given by

$$v^1 = a_1 + \text{sign}(a_{11})\|a_1\|e_1, \qquad \text{with} \;\; \text{sign}(x) = x/|x|$$

(where $\text{sign}(x)$ is the sign of x if $x \in \mathbb{R}$ and a complex number with unit modulus otherwise, this factor beingh introduced to avoid stability issues when a_1 is such that $\big|a_{11} + \|a_1\|\big| \ll \|a_1\|$), we indeed have

$$F(v^1)a_1 = -\text{sign}(a_{11})a_{11}e_1$$

and hence

$$F(v^1)A = F(v^1)\begin{bmatrix} a_{11} & A_{1,2:n} \\ A_{2:m,1} & A_{2:m,2:n} \end{bmatrix} = \begin{bmatrix} -\text{sign}(a_{11})a_{11} & u_1' \\ 0 & A_2' \end{bmatrix} \tag{3.2}$$

The next step consists in applying the same treatment to the second column of $F(v^1)A$, leaving its first row and first column undisturned. This is achieved by left-multiplying $F(v^1)A$ by the Householder reflector $F(v^2)$ with v^2 set to

$$v^2 = \begin{Bmatrix} 0 \\ a_2' + \text{sign}((A_2')_{11})\|a_2\|e_2 \end{Bmatrix}.$$

with $a_2' := (A_2')_{1:m-1}$. This process is then recursively carried out a total of n times[1], the k-th Householder reflector $F(v^k)$ introducing zeros at the $A_{k:m,k:n}$ positions of A being such that $v_i^k = 0$ for $i < k$. Successively applying the reflectors $F(v^1)$ to $F(v^n)$ on A, we reach

$$F(v^n)\,F(v^{n-1})\dots F(v^2)\,F(v^1)A = R, \qquad R = \begin{bmatrix} r_{11} & \dots & r_{1n} \\ 0 & \ddots & \vdots \\ \vdots & & r_{nn} \\ 0 & \dots & 0 \end{bmatrix}$$

Moreover, $F(v^n)\,F(v^{n-1})\dots F(v^2)\,F(v^1)$ is unitary (being the product of unitary matrices), and we hence obtain the desired QR factorization

$$A = QR, \qquad Q := F(v^1)^{\text{H}}\,F(v^2)^{\text{H}}\,\dots F(v^{n-1})^{\text{H}}\,F(v^n)^{\text{H}}, \tag{3.3}$$

where $Q \in \mathbb{K}^{m\times m}$ is unitary and $R \in \mathbb{K}^{m\times n}$ upper triangular.

The QR factorization applies as well to other situations, in particular where $A \in \mathbb{K}^{m\times n}$ with $m \ge n$ but does not have full column rank, or $A \in \mathbb{K}^{m\times n}$ with $m < n$ (underdetermined systems, for which A cannot have full column rank). We do not go into the details of how the QR factorizations are established in those more-complicated cases (refer e.g. to [19, Chap. 5]). The general result on QR factorization is as follows:

Theorem 3.1 (QR factorization) *For any matrix $A \in \mathbb{K}^{m\times n}$ with $m \ge n$, there exist $Q \in \mathbb{K}^{m\times m}$ unitary, $R \in \mathbb{K}^{m\times n}$ upper triangular, and $\Pi \in \mathbb{R}^{n\times n}$ a permutation matrix, such that the QR factorization identity holds:*

$$A\Pi = QR.$$

More precisely, letting $r \le min(m, n)$ denote the rank of A, we have:

[1] If $m = n$, the first $n-1$ reflectors suffice, as in this case we simply have $F(v^n) = I_n$.

(a) *If* $r = n$,

$$A = QR, \qquad R = \begin{bmatrix} R_n \\ 0 \end{bmatrix},$$

where $R_n \in \mathbb{K}^{n\times n}$ *is upper triangular and invertible.*

(b) *If* $r < n$,

$$A = QR\Pi^T, \qquad R = \begin{bmatrix} R_r & R_{n-r} \\ 0 & 0 \end{bmatrix},$$

where $R_r \in \mathbb{K}^{r\times r}$ *is upper triangular and invertible and* $R_{n-r} \in \mathbb{K}^{r\times(n-r)}$.

The Householder QR factorization algorithm is proved to be backward stable [36, Chap. 16]:

Theorem 3.2 (backward stability of Householder QR factorization) *Let* $A \in \mathbb{K}^{m\times n}$ *with* $m \geq n$. *Let* $\widetilde{Q}$ *and* $\widetilde{R}$ *be the factors of the Householder QR factorization of* A *computed (in finite precision) using Algorithm 3.1. Then there exists a matrix* $E \in \mathbb{K}^{m\times n}$ *such that*

$$A + E = \widetilde{Q}\widetilde{R} \qquad \text{with} \quad \|E\|/\|A\| = O(\varepsilon_{mach})$$

The Householder QR factorization algorithm is shown in compact form in Algorithm 3.1, some of whose steps require a bit of explanation:

- Step 5: the vector v^k of the Householder reflector $F(v^k)$ to be applied to the k-th column of A is formed for rows k to m (the other entries implicitly being zero); it is normalized so that $v^k = 1$.
- Step 6: the normalization factor $2/\|v^k\|^2$ of $F(v^k)$ is evaluated and stored in a separate linear array β;
- Step 8: actual application of $F(v^k)$ to A is only needed for the block $[a_{ij}]_{k\leq i\leq m,\, k+1\leq j\leq n}$ of A. Notice the appearance of the normalization factor.
- Step 9: v^k is stored in the available (implicitly zeroed-out) part of the k-th column of A. The value $v^k = 1$ is implicit thereafter; likewise, $v^k_i = 0$ for $i < k$. Thus, only entries v^k_i for $i > k$ need be stored, hence the form of this step.

Algorithm 3.1 Householder QR factorization

1:	$A \in \mathbb{K}^{m\times n}$	(data)
2:	**for** $k = 1$ to n **do**	
3:	$x = A_{k:m,k}$	(part of k-th column below diagonal (included))
4:	$\gamma = \text{sign}(x_1)\|x\|$	
5:	$v = (x + \gamma e_1)/(x_1 + \gamma)$	(Householder vector v^k, normalized so that $v^k = 1$)
6:	$\beta_k = 2/\|v\|^2$	(normalization factor for $F(v^k)$)
7:	$a_{kk} = -\gamma$	(Apply $F(v^k)$ to column k of A)
8:	$A_{k:m,(k+1):n} = A_{k:m,(k+1):n} - \beta_k v(v^H A_{k:m,(k+1):n})$	(Apply $F(v^k)$ to block of A hanging on the right of a_{kk})
9:	$A_{(k+1):m,k} = v_{2:(m-k+1)}$	(store v^k minus leading entry in column k of A)
10:	**end for**	

In particular, we emphasize that the unitary factor Q *must not* in practice be evaluated using formula (3.3), which would make for a very inefficient approach in terms of both computational work and memory requirement. As we will see shortly, least-squares problems require evaluations of $Q^H b$ for some vector $b \in \mathbb{K}^m$ and this task can be efficiently performed using the information on the reflection vectors as obtained and stored by Algorithm 3.1.

Main comments about the QR factorization and its usage. The QR factorization is applicable to any real or complex matrix, regardless of its format or structure. This general applicability makes it one of the major tools of numerical linear algebra. We will shortly see the usefulness of the QR factorization for solving least squares problems (see Sec. 3.3). Moreover, the QR factorization is also a key component of algorithms computing the complete set of eigenvalues of a given matrix, see Chap. 5.

The following remarks emphasize other useful facts on the QR factorization:

- Computing the QR factorization of $A \in \mathbb{K}^{m\times n}$ by means of Householder reflectors asymptotically requires $\frac{2}{3}n^2(3m-n)$ floating-point operations if $m \geq n$. For $m \approx n$, that is about $\frac{4}{3}n^3$ operations, i.e. twice the asymptotic cost of a LU factorization.
- We emphasize again that the QR factorization is applicable to *any* matrix whatsoever.
- By contrast, since invertibility of A is neither assumed, nor in fact meaningful if A is not square, using the matrices G_k introduced for the LU factorization may break down (due to the possible appearance of zero pivots).
- Square invertible matrices can always be decomposed using the QR factorization instead of the LU factorization. However, as said above, the former method is about twice more expensive than the latter.
- The QR factorization is *rank-revealing*: the rank r of A is equal to the number of nonzero diagonal entries of R, i.e. the size of R_r.

3.2 SOLVABILITY OF LINEAR SYSTEMS. Let us consider a linear system $Ax = b$ with $A \in \mathbb{K}^{m\times n}$. Applying the QR factorization to A and assuming the most general situation (case (b) in Theorem 3.1), we have

$$Ax = b \quad \Longrightarrow \quad Rx' = Q^{\mathrm{H}}b, \quad \text{i.e.} \quad \begin{cases} R_r x'_{1:r} + R_{n-r}x'_{r+1:n} & = (Q^{\mathrm{H}}b)_{1:r} \\ 0 & = (Q^{\mathrm{H}}b)_{r+1:m} \end{cases} \qquad (\text{with } x' := \Pi^{\mathrm{T}}x),$$

from which we infer:

- Solvability of the linear system $Ax = b$ requires $(Q^{\mathrm{H}}b)_{r+1:m} = 0$ (which ensures that $b \in \mathcal{R}(A)$);
- Then:

  (i) If $r = n$ (in which case the block R_{n-r} disappears, $R_r = R$ and $\Pi = I_n$), we have $Rx = (Q^{\mathrm{H}}b)_{1:n}$, yielding the unique solution x.

  (ii) If $r < n$, we have $R_r x'_{1:r} = (Q^{\mathrm{H}}b)_{1:r} - R_{n-r}x'_{r+1:n}$; we can choose $x'_{r+1:n}$ arbitrarily, and then find the unique value of $x'_{1:r}$ (which depends on $x'_{r+1:n}$); finally $x = \Pi x$.

In case (ii) above, the solution set is an affine space of dimension $n-r$.

3.3 LEAST SQUARES PROBLEMS. Since the generic linear system $Ax = b$ is unsolvable whenever $b \notin \mathcal{R}(A)$, an alternative option is to seek a *least squares solution*:

$$\min_{x\in\mathbb{K}^n} \|Ax - b\|^2, \tag{3.4}$$

i.e. to find x that achieves the best (or least bad) fit with the data b. Of course, if some x achieves $\|Ax-b\|^2 = 0$, then it solves the original system. Introducing the QR factorization in the least squares problem (3.4), we have (following Sec. 3.2)

$$\begin{aligned} \|Ax - b\|^2 = \|QRx' - b\|^2 = \|Q(Rx' - Q^{\mathrm{H}}b)\|^2 = \|(Rx - Q^{\mathrm{H}}b)\|^2 \\ = \|R_r x'_{1:r} + R_{n-r}x'_{r+1:n} - (Q^{\mathrm{H}}b)_{1:r}\|^2 + \|(Q^{\mathrm{H}}b)_{r+1:m}\|^2, \end{aligned}$$

where the second and third equalities take advantage of the unitary character of Q. Since $\|(Q^{\mathrm{H}}b)_{r+1:m}\|^2$ does not depend on x', the minimization acts only on the first residual $\|R_r x'_{1:r} + R_{n-r}x'_{r+1:n} - (Q^{\mathrm{H}}b)_{1:r}\|^2$. In fact, revisiting the previous solvability discussion shows that this first residual can be set to zero, which is obviously the best possible outcome for the least-squares problem (3.4). Hence:

(i) If $r = n$, we have $Rx = (Q^{\mathrm{H}}b)_{1:n}$, yielding the unique solution x to the least-squares problem (3.4).

(ii) If $r < n$, we solve $R_r x'_{1:r} = (Q^{\mathrm{H}}b)_{1:r} - R_{n-r}x'_{r+1:n}$; by choosing $x'_{r+1:n}$ arbitrarily, find the resulting unique value of $x'_{1:r}$, and finally $x = \Pi x$.

In case (ii) above, the solution set of problem (3.4) is again an affine space of dimension $n-r$.

The solution procedure defines the solution(s) in the same way for the linear-system and least-squares cases, the essential difference being that the *existence* of a solution is guaranteed for any data $b \in \mathbb{K}^m$ in the latter case but *not* in the former case.

Evaluation of $Q^{\text{H}}b$. Applying the QR factorization method for solving a least-squares problem (or a linear system) entails the evaluation of $Q^{\text{H}}b$, i.e. the application of Q^{H} to the right-hand side b of the system. Recalling the defining expression (3.3) of Q, we have

$$Q^{\text{H}}b = F(v^n)\,F(v^{n-1})\,\ldots F(v^2)\,F(v^1)b \tag{3.5}$$

The reflector $F(v^k)$, as computed by Algorithm 3.1, is given by

$$F(v^k) = I_m - \beta_k v^k (v^k)^{\text{H}} \qquad v^k = e_k + \widetilde{v}^k$$

where the vector $\widetilde{v}^k$ is such that $\widetilde{v}^k_i = 0$ for $i \le k$ and has its nontrivial part $\widetilde{v}^k_{(k+1):m}$ stored by Algorithm 3.1 in $A_{(k+1):m,k}$. The evaluation of $F(v^k)$ on a given vector $y \in \mathbb{K}^m$ thus takes the form

$$F(v^k)y = y - \beta_k\big(y_k + (\widetilde{v}^k)^{\text{H}} y_{(k+1):m}\big)\big(e_k + \widetilde{v}^k\big)$$

As a result, the evaluation of $Q^{\text{H}}b$ using (3.5) and the above conventions (i.e. exploiting the outcome of Algorithm 3.1) translates into Algorithm 3.2, wherein the result $Q^{\text{H}}b$ is stored in b.

Algorithm 3.2 Evaluation of $Q^{\text{H}}b$ using the outcome of Algorithm 3.1

```
A ∈ K^{m×n}, β ∈ R^n, b ∈ K^m    (A, β as produced by Algorithm 3.1)
for k = 1 to n do
    v = A_{(k+1):m,k}                (Extract the part ṽ^k of v^k stored in A)
    γ = b_k + v^H b_{(k+1):m}
    b_k = b_k − γβ_k                 (update b_k (entries above b_k of b are untouched))
    b_{(k+1):m} = b_{(k+1):m} − γβ_k v   (update b below b_k)
end for
```

3.4 SINGULAR VALUE DECOMPOSITION. Another essential tool in numerical linear algebra is the singular value decomposition (SVD). The SVD generalizes to *arbitrary* (even non-square) matrices the diagonalization

$$A = X\Lambda X^{-1}$$

of $A \in \mathbb{K}^{n\times n}$ (where $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$ holds the eigenvalues of A and X is invertible), which applies only if A is square and even so only conditionally[2].

Let then $A \in \mathbb{K}^{m\times n}$ be any matrix (where m may be larger than, equal to, or smaller than, n). We introduce the two eigenvalue problems

$$A^{\text{H}}Av = \lambda v, \qquad AA^{\text{H}}u = \mu u.$$

Both matrices $A^{\text{H}}A \in \mathbb{K}^{n\times n}$ and $AA^{\text{H}} \in \mathbb{K}^{m\times m}$ are Hermitian and positive. The two eigenvalue problems are well-defined, with real positive eigenvalues $\lambda_1 \ge \lambda_2 \ge \ldots \lambda_n \ge 0$, $\mu_1 \ge \mu_2 \ge \ldots \mu_m \ge 0$ and unitary matrices of corresponding eigenvectors $V := [v_1, \ldots, v_n] \in \mathbb{K}^{n\times n}$, $U := [u_1, \ldots, u_m] \in \mathbb{K}^{m\times m}$. Moreover, for any nonzero eigenvalue λ of $A^{\text{H}}A$, we observe that

$$A^{\text{H}}Av = \lambda v \implies AA^{\text{H}}Av = (AA^{\text{H}})Av = \lambda Av,$$

i.e. if (λ, v) is an eigenpair of $A^{\text{H}}A$, then (λ, Av) is an eigenpair of AA^{H}. By the same argument, if (λ, u) is an eigenpair of AA^{H}, then $(\lambda, A^{\text{H}}u)$ is an eigenpair of $A^{\text{H}}A$. Hence, both matrices share the same nonzero eigenvalues, it is not difficult to show that the multiplicity of each nonzero eigenvalue is the same for $A^{\text{H}}A$ and AA^{H}, and normalized eigenvectors associated with nonzero eigenvalues verify pairwise interrelations

$$\sigma u = Av, \qquad \sigma v = A^{\text{H}}u \qquad \text{with } \sigma := \sqrt{\lambda},$$

the eigenvalue square roots σ being called the *singular values* of A. Collecting all the previous information, we arrive at the main result on SVD:

[2]Some non-symmetric square matrices, called *defective*, are not diagonalizable.

Theorem 3.3 (Singular value decomposition) *Any matrix $A \in \mathbb{K}^{m\times n}$ admits the decomposition*

$$A = USV^H, \tag{3.6}$$

where $U \in \mathbb{K}^{m\times m}$ and $V \in \mathbb{K}^{n\times n}$ are unitary square matrices and $S \in \mathbb{R}^{m\times n}$ contains the $\min(m,n)$ singular values $\sigma_1, \dots \sigma_{\min(m,n)}$ of A on its main diagonal and is otherwise zero (i.e. $S_{ii} = \sigma_i$, $S_{ij} = 0$ if $i \neq j$). The singular values are positive real numbers, conventionally arranged by decreasing order of magnitude (i.e. $\sigma_1 \geq \sigma_2 \geq \dots \sigma_{\min(m,n)} \geq 0$). The vectors $u_1, \dots, u_m$ and $v_1, \dots, v_n$ such that $U = [u_n, \dots, u_m]$ and $V = [v_1, \dots, v_n]$ are called the left and right singular vectors, respectively.

The SVD provides a "dissection" method, applicable to any real or complex matrix regardless of its format or structure, and in particular to matrices that are not diagonalizable. Among other things, the SVD provides an explicit expression for the 2-norm of any matrix:

Theorem 3.4 (2-norm of a matrix) *The 2-norm $\|A\|_2$ of any matrix $A \in \mathbb{K}^{m\times n}$ is given by $\|A\|_2 = \sigma_1$.*

The generality of the SVD places it among most important tools of numerical linear algebra. We will shortly see that the SVD provides key insight into the solvability of linear systems and least squares problems, while being one of the main computational tools for the latter. Moreover, the SVD also plays an essential role in methods aimed at solving ill-conditioned linear systems or setting up low-rank approximations of numerically rank-deficient matrices that often occur in image or data processing, see Chap. 6.

The following remarks emphasize other useful facts on the SVD:

- The SVD is *rank-revealing*: the rank r of A is equal to the number of nonzero singular values (counting multiplicities); in particular $r = \min(m,n)$.
- The nonzero squared singular values $\sigma_1^2, \dots, \sigma_r^2$ are the common nonzero eigenvalues of AA^{H} and $A^{\text{H}}A$, with correct multiplicities (i.e. $\sigma_i^2 = \lambda_i = \mu_i$ for $1 \leq i \leq r$). All remaining eigenvalues of AA^{H} and $A^{\text{H}}A$ are zero (i.e. $\lambda_{r+1} = \dots = \lambda_m = 0$ and $\mu_{r+1} = \dots = \mu_n = 0$).
- The matrix A can equivalently be represented by its *reduced* SVD

$$A = U_r S_r V_r^{\text{H}} \tag{3.7}$$

  where $U = [u_1, \dots, u_r] \in \mathbb{K}^{m\times r}$ and $V = [v_1, \dots, v_r] \in \mathbb{K}^{n\times r}$ contain the r left and right singular vectors associated with the r nonzero singular values $\sigma_1, \dots, \sigma_r$, while $S_r = \text{diag}(\sigma_1, \dots, \sigma_r)$ is now square $r \times r$. The singular vectors $u_{r+}, \dots, u_n$ and $v_{r+}, \dots, v_m$ not used in (3.7) are bases of the null-spaces $N(A)$ and $N(A^{\text{H}})$, respectively.
- The practical computation of a SVD uses adaptations of algorithms for matrix eigenvalue problems (Chap. 5). THe full SVD of $A \in \mathbb{K}^{m\times n}$ requires $O(m^2 n)$ operations [19, Sec. 5.4.5].
- The MATLAB operator `svd` returns either the full SVD (3.6) or the reduced SVD (3.7) of a matrix.

Exercise 3.1 *Prove Theorem 3.4*

Exercise 3.2 *Let $A \in \mathbb{K}^{n\times n}$ be a square diagonalizable matrix. Show that the SVD of A and its diagonalization usually yield different multiplicative decompositions. In which case do SVD and diagonalization produce the same decomposition of A?*

Using the SVD to solve linear systems and least squares problems. The SVD allows to investigate the solvability of an arbitrary linear system $Ax = b$. Using Theorem 3.3 and letting $r \leq \min(m,n)$, we have

$$Ax = b \implies USV^{\text{H}}x = b \implies Sy = z \text{ i.e. } \begin{cases} \sigma_i y_i = z_i & (1 \leq i \leq r) \\ 0 = z_i & (r \leq i \leq m) \end{cases}, \tag{3.8}$$

upon making the changes of variables $y := V^{\text{H}}x$ and $z := U^{\text{H}}b$, i.e. $y_i = v_i^{\text{H}}x$ and $z_i = u_i^{\text{H}}b$ (which express the unknown and data on the bases of right and left singular vectors, respectively).

- The conditions $0 = z_i$ $(r > i \leq m)$ again express the requirement that $b \in \mathcal{R}(A)$ and determine whether the original system $Ax = b$ has a solution.
- If these conditions are met, equations $y_i = z_i$ uniquely determine the first r entries of $y = V^{\text{H}}x$.

- The remaining $n-r$ entries of $y = V^{\mathrm{H}}x$, if any (that is, if A has a column rank deficiency), are then arbitrary, so that all solutions of $Ax = b$ (if any) are given by

$$x = \sum_{i=1}^{r} \frac{z_i}{\sigma_i} v_i + \sum_{i=r+1}^{n} x_i v_i \qquad (x_{r+1}, \ldots, x_n) \in \mathbb{K}^{n-r} \text{ arbitrary} \tag{3.9}$$

- If A has column rank deficiency, setting $x_{r+1} = \ldots = r_n = 0$ in (3.9) yields the least-squares solution with minimum norm. This rule is often used for selecting a solution among all possibilities (3.9) in the absence of definite criteria for choosing otherwise.
- Applying MATLAB's backslash "\" operator to $Ax = b$ (via `x=A\b;`) yields one of the solutions (3.9). In case of non-uniqueness, the user has no control over which possibility is picked (i.e. to which choice of $x_{r+1}, \ldots, x_n$ in (3.9) it corresponds).

The above remarks on solvability are of course equivalent to those found using the QR decomposition, albeit expressed using different ingredients.

Similarly, the SVD can be usefully applied to the least squares problem (3.4), since we have (with y and z defined as in (3.8))

$$\|Ax - b\|^2 = \|USV^{\mathrm{H}}x - b\|^2 = \|U(Sy - z)\|^2 = \|Sy - z\|^2 = \sum_{i=1}^{r} |\sigma_i y_i - z_i|^2 + \sum_{i=r+1}^{m} |z_i|^2 \tag{3.10}$$

Clearly, all solutions given by (3.9), and only those, minimize $\|Ax - b\|^2$. Equation (3.9) thus gives all solutions of the least-squares problem (3.4), without condition on the data. If $b \notin \mathcal{R}(A)$ (incompatible data leading to unsolvability of $Ax = b$), we have

$$\min_{x \in \mathbb{K}^n} \|Ax - b\|^2 = \Big\| \sum_{i=r+1}^{n} x_i v_i \Big\|^2 = \sum_{i=r+1}^{n} |x_i|^2$$

3.5 PSEUDO-INVERSE OF A MATRIX. We just introduced the SVD as a generalization of matrix diagonalization that applies to any matrix whatsoever (including non-square matrices). In the same spirit, one can define the notion of pseudo-inverse of a matrix, in such a way that it makes sense for any matrix.

For any $A \in \mathbb{K}^{m \times n}$, a *pseudo-inverse* (or *generalized inverse*) of A is a matrix $A^+ \in \mathbb{K}^{n \times m}$ satisfying the following four conditions, known as the Moore-Penrose conditions:

$$\begin{array}{llll} \text{(a)} & AA^+A = A, & \text{(b)} & (AA^+)^{\mathrm{H}} = AA^+, \\ \text{(c)} & A^+AA^+ = A^+, & \text{(d)} & (A^+A)^{\mathrm{H}} = A^+A. \end{array} \tag{3.11}$$

The following theorem summarizes the main algebraic properties of the pseudo-inverse:

Theorem 3.5 (algebraic properties of the pseudo-inverse) *Let $A \in \mathbb{K}^{m \times n}$ be any matrix.*
(a) The pseudo-inverse of the pseudo-inverse is the original matrix: $(A^+)^+ = A$.
(b) There is a unique matrix $A^+ \in \mathbb{K}^{n \times m}$, called the Moore-Penrose pseudo-inverse of A, which satisfies all four requirements in (3.11).
(c) When A is invertible, we have $A^+ = A^{-1}$ (as expected!)
(d) When A has full column rank (implying $m \geq n$ and A^HA invertible), we have $A^+ = (A^HA)^{-1}A^H$.
(e) When A has full row rank (implying $m \leq n$ and AA^H invertible), we have $A^+ = A^H(AA^H)^{-1}$.

The following remarks emphasize other useful facts on the pseudo-inverse:

- An explicit formula for A^+ is available from the reduced singular value decomposition (3.7) of A:

$$A^+ = V_r S_r^{-1} U_r^{\mathrm{H}}$$

- The general solution (3.9) of the least-squares problem (3.4) is given in terms of A^+ by

$$x = A^+b + (I - A^+A)w, \qquad w \in \mathbb{K}^n,$$

where $w \in \mathbb{K}^n$ is arbitrary. In particular, setting $w = 0$ in the above formula yields the minimum-norm least-squares solution $x = A^+ b$. Notice also that if A is square invertible (so that $A^+ = A^{-1}$, the above formula becomes $x = A^{-1}b$, which emphasizes further the fact that A^+ generalizes the notion of matrix inverse.

- The pseudo-inverse A^+ does not depend continuously on A. For instance, let A (with rank r) and a perturbed version A_ε (with rank $r+1$) be given in terms of their respective reduced SVDs as

$$A_\varepsilon = U_r S_r V_r^{\mathrm{H}}, \qquad A_\varepsilon = U_r S_r V_r^{\mathrm{H}} + \varepsilon u_{r+1} v_{r+1}^{\mathrm{H}},$$

where $\varepsilon > 0$ is small. Then, we have

$$A_\varepsilon^+ = A^+ + \frac{1}{\varepsilon} v_{r+1} u_{r+1}^{\mathrm{H}} \qquad \text{and} \qquad \frac{\|A_\varepsilon^+ - A^+\|}{\|A^+\|} = \frac{\sigma_r}{\varepsilon} \gg 1$$

- The MATLAB operator `pinv` computes the pseudo-inverse A^+ of an arbitrary matrix A.

Moreover, extending the analysis of Sec. 1.7 to least-squares problems involving arbitrary matrices A, the condition number $\kappa_2(A)$ can be expressed in terms of either its pseudo-inverse or its SVD:

Theorem 3.6 (2-norm condition number of general matrices) *Let $A \in \mathbb{K}^{m\times n}$ be any matrix, and let r be the rank of A. The 2-norm condition number $\kappa_2(A)$ is given by*

$$\kappa_2(A) = \|A\|\|A^+\| = \sigma_1/\sigma_r$$

3.6 THE BACKSLASH OPERATOR. Programming languages like MATLAB or Julia feature a backslash operator for solving linear systems. Its use is deceptively simple: we just type something like

```
x = A\b;      (in MATLAB)
```

to obtain a solution for $Ax = b$. However, this is a quite complex operator, which triggers various algorithms depending on the detected properties of A and always yields a solution (even when none, or many, exist!). So it is important to be aware of what the backslash operator does, and what is the meaning of the returned solution. It turns out that all of the foregoing methods (LU factorization, Cholesky factorization, QR factorization) are involved, see Fig. 3.2.

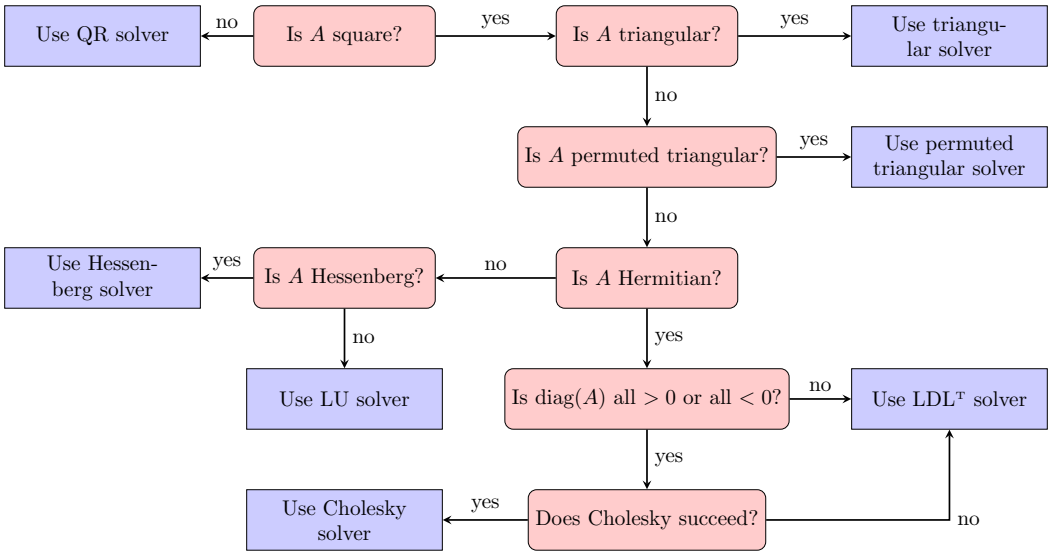


Figure 3.2: Flowchart of the backslash operator "\" of MATLAB

CHAPTER 4 ITERATIVE SOLVERS

4.1 MOTIVATION. Classical direct solution methods such as those described in Chapters 2 and 3 were designed with general (dense) matrices in mind. We have seen that they require $O(n^3)$ arithmetic operations unless applied to matrices with some special structure that can be taken advantage of (e.g. band matrices). Likewise, standard linear solvers require that the whole matrix A (or one-half of it in case of symmetry) be stored beforehand. This makes them ill-suited for many applications, due to any of the following reasons:

- Many applications feature linear systems $Ax = b$ involving matrices that are very large.
- If large and dense, the whole matrix cannot be held in memory, while the $O(n^3)$ computational work is too-expensive.
- On the other hand, large and sparse matrices are frequently encountered (notably from problems resulting from ODE or PDE discretization by e.g. finite elements or finite differences). In this case, factorization methods fill initially-zero entries with non-zero values, while the factorization involves many useless arithmetic operations (multiplications involving zero entries).
- Other situations produce matrices that are dense but have sufficiently accurate low-rank approximations. This is for instance the case in the fast-multipole or hierarchical-matrix treatments of large boundary element models, see e.g. [2]. Other situations involve dense ill-conditioned matrices whose *numerical rank* is much lower than their size, see Chap. 6.

The above (and other) considerations led to the development of alternative solution methods that are *iterative*. In very general terms, iterative solution methods define *approximating sequences* $\{x_0, x_1, x_2, \ldots\}$ that converge to the solution x of $Ax = b$ but (in general) reach x only in the limit:

$$x = \lim_{k\to\infty} x_k.$$

The solution algorithm then consists in iterating a procedure that defines the next solution iterate x_{k+1} from the current iterate x_k and the problem data A, b. As a matter of semantics, the distinction between *direct* and *iterative* characters of solution algorithms is as follows:

- Direct methods yield the exact result within finitely many operations (under ideal conditions of exact arithmetic without roundoff errors). Factorization-based methods (LU, Cholesky, QR...) fall within this category. Factorization algorithms may repeat a certain basic step (e.g. column-wise elimination), but at most finitely many times.
- Iterative methods yield the exact result only as the limiting value of an approximating sequence, so that solving a problem exactly entails an infinite amount of computational work.

In practice, and for a lot of reasons, one is not after the exact solution x (which is anyway out of practical reach due to finite-prtecision arithmetic) and will be happy to stop at the K-th iteration that yields a "close enough" approximation x_K. Ideally, we would like to achieve

$$\|x_K - x\|/\|x\| \leq \varepsilon$$

for some (relative) accuracy tolerance ε, but of course we do not know x beforehand such rule is normally impractical. Instead, we seek for example an approximation x_K such that the linear system is satisfied "well enough":

$$\|b - Ax_K\|/\|b\| \leq \varepsilon$$

Moreover, many iterative algorithm work on the assumption that the matrix A is not available (due to being too large for storage), instead exploiting the ability to evaluate *matrix-vector products* $x_k \mapsto Ax_k$ for given x_k without A being stored[1]. This dramatically expands the applicability of iterative solvers by allowing them to function with $O(n)$ storage (instead of $O(n^2)$ for usual direct solvers).

Here we will present three kinds of iterative solution methods. Classical methods based on matrix splitting are briefly described first (Sec. 4.2), before focusing on two archetypal, and essential, iterative solvers based on matrix-vector products, namely the conjugate gradient method for SPD linear systems (Sec. 4.3) and the generalized minimum residuals (GMRES) method for general square systems (Sec. 4.5). The latter methods, and many other related methods that we do not discuss here, are rooted in the concept of *Krylov spaces*, introduced in Sec. 4.4, from which in particular important insight is gained into the conjugate gradient method (revisited in this light in Sec. 4.6).

4.2 CLASSICAL SPLITTING (FIXED-POINT) METHODS. Early iterative solution methods are based on using *additive splittings* of the governing matrix A. For example, write $A = A_1 - A_2$ where A_1 is chosen so as to be "easily invertible" (and of course $A_2 = A_1 - A$), and define iterations by

$$A_1 x_{k+1} = b + A_2 x_k, \quad k = 0, 1, 2, \ldots \quad (x_0\text{: user-chosen initial guess}). \tag{4.1}$$

If the above iteration do converge to a limit x (which is not guaranteed at all in general), then the limit satisfies $A_1 x = b + A_2 x$, i.e. $Ax = b$. Since A_1 must be invertible for (4.1) to make any practical sense, we can rewrite (4.1) as

$$x_{k+1} = A_1^{-1}\big(b + A_2 x_k\big) = A_1^{-1}\big(b + A_2 A_1^{-1}(b + A_2 x_{k-1})\big) \ldots = A_1^{-1}\Big\{ \sum_{i=0}^{k} (A_2 A_1^{-1})^i b \Big\}$$

and the limit as $k \to \infty$ of x_{k+1} exists provided the bracketed sums are partial sums of a convergent series. On using the triangle inequality and the properties of matrix norms, we have

$$\Big\| \sum_{i=0}^{k} (A_2 A_1^{-1})^i b \Big\| \le \Big\{ \sum_{i=0}^{k} \|A_2 A_1^{-1}\|^i \Big\} \|b\|,$$

so that a sufficient condition for convergence of the partial sums is $\|A_2 A_1^{-1}\| < 1$. In fact:

Theorem 4.1 *Iterations* (4.1) *converge for any initial guess* x_0 *if and only if* $\rho(A_2 A_1^{-1}) < 1$, *where* $\rho(X)$ *is the* spectral radius *of a matrix* X *(equal to the largest modulus of eigenvalues of* X*).*

Iterations (4.1) can, alternatively, be seen as fixed-point iterations

$$x_{k+1} = F(x_k), \qquad F(x) := A_1^{-1}\big(b + A_2 x\big) \tag{4.2}$$

whose limit is a fixed point (if such exists) of $x \mapsto F(x)$. In this light, the condition $\|A_2 A_1^{-1}\| < 1$ guarantees that the $\mathbb{K}^n \to \mathbb{K}^n$ mapping $z \mapsto F(z)$ is a *contraction*, thereby ensuring (i) existence and uniqueness of the fixed point and (ii) convergence of the iterations (4.2) to the fixed point x of F.

Several well-known iterative methods follow the above general idea. To present then concisely and with unified notation, we begin by splitting A according to

$$A = D - L - U$$

where $D, -L, -U$ are the diagonal, strict lower triangular, and strict upper triangular parts of A respectively[2]. Then, various possibilities for splittings of the general form (4.1) are defined in terms of D, L, U:

- *Jacobi iterations* They are produced by iterations (4.1) with $A_1 = D$, $A_2 = L + U$.
- *Gauss-Seidel iterations* They are produced by iterations (4.1) with $A_1 = D - L$, $A_2 = U$.

[1] For example, in a finite element mechanical analysis where nodal displacements $U \in \mathbb{R}^n$ satisfy the SPD system $KU = F$ with the *stiffness matrix* $K \in \mathbb{R}^{n\times n}$ a SPD band matrix, this means that the finite element assembly procedure is used at each iteration to evaluate directly KU_k (or the *residual* $F - KU_k$) for a given solution approximation U_k instead of assembling K.

[2] So the parts L, U are *not* the factors generated by the LU factorization!

- *Successive over-relaxation (SOR) iterations* They are produced by iterations (4.1) with $A_1 = D - \eta L$, $A_2 = \eta U + (\eta - 1)D$ for some $\eta \in \mathbb{R}$, $\eta \neq 0$, and so take the form

$$\big[D - \eta L\big] x_{k+1} = \big[\eta U + (1-\eta)D\big] x_k + \eta b \qquad (x_0\text{: user-chosen initial guess}).$$

  In particular, they contain the Gauss-Seidel iterations as a special case (for $\eta = 1$). The basic idea is to define an iteration method that depends on a tunable parameter (here η, called the *relaxation parameter*), whose choice may either ensure convergence or enhance convergence rates. Still, SOR iterations do not converge for all matrix A, even with the best choice for η.

Some important properties of splitting-based iterations are summarized in the next theorem:

Theorem 4.2 (properties of Jacobi, Gauss-Seidel and SOR iterations)
(a) If A is diagonally dominant *(i.e. $|a_{ii}| > \sum_{j\neq i} |a_{ij}|$ for each $1 \leq i \leq n$), Jacobi iterations converge for any initial guess x_0.*
(b) If A is real SPD, Gauss-Seidel iterations converge for any initial guess x_0.
(c) A necessary condition for the convergence of SOR iterations is $0 < \eta < 2$ (if not, there exists an initial guess x_0 for which SOR iterations diverge).
(d) If A is real SPD, SOR iterations converge for any η such that $0 < \eta < 2$ and any initial guess x_0.

The convergence results (b) and (d) are a consequence of the Householder-John theorem: if both A and $A - B - B^{\mathrm{T}}$ are real SPD matrices, then $(A-B)^{-1}B$ has spectral radius less than one.

Outlook. The splitting-based fixed-point methods are very useful for certain classes of problems to which they fit well, in which case a proof of convergence is often available. However, their convergence is not guaranteed for general matrices A, and some of them rely on solving triangular systems, so are not well suited to large dense systems for that reason. By contrast, we will now consider other methods (a) whose convergence is guaranteed and (b) which do not rely on $O(n^2)$ storage requirement.

Exercise 4.1 *Construct simple examples (e.g. using 2×2 matrices) of systems for which Jacobi, Gauss-Seidel or SOR methods converge, and other examples for which they do not converge.*

4.3 CONJUGATE GRADIENT METHOD.

We have seen in Chapter 3 that linear systems of equations can be solved in the least-squares sense (unique solvability of $Ax = b$ corresponding to unique solvability of the least-squares problem together with a zero minimum value of the residual). More generally, solving linear systems can in many situations be addressed in relation to optimization problems [39].

4.3.1 SPD systems and quadratic minimization problems.

In this section, we consider the very important case of systems governed by real SPD matrices (which are in particular uniquely solvable). Let $A \in \mathbb{R}^{n\times n}$ be SPD and $b \in \mathbb{R}^n$. The generic system

$$Ax = b$$

can in particular considered as expressing the stationarity of a quadratic cost functional $J : \mathbb{R}^n \to \mathbb{R}$:

$$J(x) := \tfrac{1}{2} x^{\mathrm{T}} A x - x^{\mathrm{T}} b + c, \qquad \nabla J(x) = 0 \;\Leftrightarrow\; Ax = b \tag{4.3}$$

(where the value of $c \in \mathbb{R}$ plays no important role). Since A is SPD (by assumption), $J(x)$ is strictly convex and has an unique minimizer $x^\star$, at which ∇J vanishes:

$$x^\star := \arg\min_{x\in\mathbb{R}^n} J(x), \qquad \text{verifying } \nabla J(x^\star) = 0, \text{ i.e. } Ax^\star = b$$

Being SPD, A has n real positive eigenvalues $\lambda_1 \geq \lambda_2 \ldots \geq \lambda_n > 0$; its condition mumber being $\kappa_2(A) = \lambda_1/\lambda_n$. Moreover, for the same reason, A allows to define the *energy norm*

$$\|x\|_A^2 := x^{\mathrm{T}} A x, \qquad \lambda_n \|x\|_2 \leq \|x\|_A \leq \lambda_1 \|x\|_2 \tag{4.4}$$

which is equivalent to the Euclidean vector norm (as shown by the above double inequality) and its associated scalar product

$$(y, x)_A := y^{\mathrm{T}} A x$$

Gradient-based minimization algorithms usually proceed (for unconstrained minimization) as follows:

- Initialization: $x = x_0$
- Iterations $k = 0, 1, 2, \ldots$ until convergence:

  (i) Choose a *descent direction* $p_k \in \mathbb{R}^n$ (such that $p_k^{\mathrm{T}} \nabla J(x_k) < 0$);

  (ii) Perform one-dimensional minimization of the objective function along the descent direction (line-search step), in closed form for the quadratic function J:

$$t_k = \arg\min_{t \geq 0} \left\{ j(t) := J(x_k + t p_k) \right\} = \frac{p_k^{\mathrm{T}} r_k}{p_k^{\mathrm{T}} A p_k} \qquad r_k := b - A x_k \text{: current } \textit{residual} \tag{4.5}$$

  (iii) Update the unknown:

$$x_{k+1} := x_k + t_k p_k \tag{4.6}$$

- Convergence test based e.g. on *relative residual*: have we reached $\|b - Ax\|/\|b\| \leq \varepsilon$ for some preset small tolerance ε?

A crucial ingredient is therefore the method by which the descent direction is defined for each iteration.

4.3.2 Steepest descent method and its deficiencies.

A very natural idea is to set

$$p_k := -\nabla J(x_k) \tag{4.7}$$

at each iteration, since this gives the steepest descent direction starting from the current iterate x_k. In practice, and somewhat counter-intuitively, this approach turns out to be often inefficient, see e.g. the detailed analysis in [32] where in particular it is shown that

$$e_k \leq \left(\frac{\kappa_2(A) - 1}{\kappa_2(A) + 1} \right)^k e_0, \qquad \text{with} \quad e_k := \|x_k - x^\star\|_A, \tag{4.8}$$

the successive solution errors e_k being measured in terms of the energy norm (4.4). This estimate shows that the convergence becomes very slow for matrices with even moderately large condition numbers. For example, if $\kappa_2(A) = 100$, we have $e_k \leq (99/101)^k e_0$ and converging within a (rather undemanding) relative tolerance of $\varepsilon = 10^{-3}$ may take hundreds of iterations even for a 2×2 matrix.

Example. Consider the system (used as illustrative example in [32])

$$Ax = b \qquad \text{with} \quad A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}, \; b = \begin{bmatrix} -2 \\ 8 \end{bmatrix}, \tag{4.9}$$

whose (SPD) matrix A has eigenvalues $\lambda_1 = 7$, $\lambda_2 = 2$ and (hence) condition number $\kappa_2(A) = 7/2$ (note that this matrix cannot be considered as ill-conditioned). The solution of (4.9) is $x^\star = \{2, \; -2\}^{\mathrm{T}}$. Starting from the initial guess $x_0 = 0$, the steepest-descent method takes 19 (resp. 43) iterations to converge within a 10^{-4} (resp. 10^{-10}) relative residual $\|b - Ax\|/\|b\|$, see Fig. 4.1. The rate of convergence per iteration predicted by (4.8) is $5/9$, suggesting that about 16 (resp. 39) iterations are needed to reduce the solution error by a factor of 10^{-4} (resp. 10^{-10}).

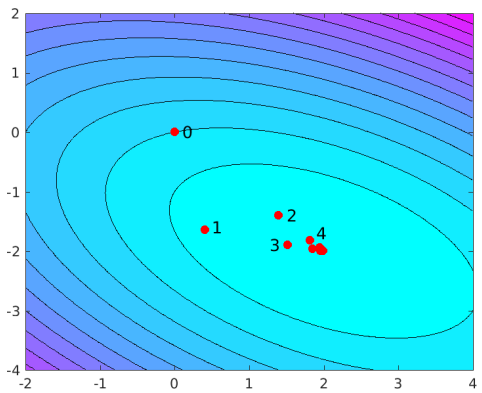


Figure 4.1: Steepest-descent iterates for example (4.9), superimposed on contour lines of the associated quadratic function J given by (4.3) with $c = 0$.

4.3.3 Heuristic derivation of the conjugate gradient algorithm. To remedy the above deficiencies, the conjugate gradient (CG) algorithm is based on a different method for the generation of successive descent directions. The matrix A, being SPD, can be used to define weighted versions $(\cdot,\cdot)_A$ of the scalar product and $\|\cdot\|_A$ of the 2-norm:

$$\big(u,v\big)_A = u^{\mathrm{T}}Av, \qquad \|u\|_A^2 = \big(u,u\big)_A = u^{\mathrm{T}}Au$$

(the norm $\|u\|_A$ being sometimes called the *energy norm* of u because A is often associated with the discretization of an energy, such as the strain energy for deformable elastic solids). The basic heuristic idea behind the conjugate gradient method is to define each new descent direction p_k so that it is *conjugate* to all previous directions, i.e.

$$\big(p_j,p_k\big)_A = 0 \quad \text{for all } j<k. \tag{4.10}$$

In other words, we want p_k to be A-orthogonal to the whole subspace $\mathrm{span}(p_0,\ldots,p_{k-1})$ explored by all previous iterations combined. Then, as before, the new solution iterate is found by solving (in closed form) the one-dimensional minimization along the search direction (line-search step). The modification of the generic unconstrained minimization algorithm is arrived at by seeking a descent direction p_k of the form

$$p_k = -\nabla J(x_k) + \sum_{j=1}^{k} \beta_j p_{j-1}$$

with coefficients β_j to be determined, i.e. a modification of the steepest-descent direction $-\nabla J(x_k)$ that satisfies the conjugacy criterion (4.10). Implementing this idea and reasoning by induction on k, it turns out that the only nonzero coefficient β_j in the above sum is β_k (so $\beta_1 = \ldots = \beta_{k-1} = 0$): the current descent direction is a linear combination of the gradient of J at the current iterate and the descent direction at the previous iteration.

The end result of carrying out in detail the above heuristic reasoning is the *conjugate gradient algorithm*, whose formulation is remarkably simple:

Algorithm 4.1 Conjugate gradient algorithm for SPD problems (explanatory form)

$A \in \mathbb{R}^{n\times n}$ SPD and $b \in \mathbb{R}^n$	(problem data)
$x_0 = 0$, $r_0 = b$, $p_0 = r_0$	(initialization)
for $k = 1, 2, \ldots$ **do**	
$q_{k-1} = Ap_{k-1}$	(matrix-vector product)
$\alpha_k = \dfrac{r_{k-1}^{\mathrm{T}} r_{k-1}}{p_{k-1}^{\mathrm{T}} q_{k-1}}$	(optimal step)
$x_k = x_{k-1} + \alpha_k p_{k-1}$	(solution update)
$r_k = r_{k-1} - \alpha_k q_{k-1}$	(residual update)
$\beta_k = \dfrac{r_k^{\mathrm{T}} r_k}{r_{k-1}^{\mathrm{T}} r_{k-1}}$	(conjugacy coefficient)
$p_k = r_k + \beta_k p_{k-1}$	(descent direction for next iteration)
Stop if convergence, set $x = x_k$	
end for	

The value of the optimal step α_k is determined from the basic and usual requirement that $\alpha \mapsto J\big(x_{k-1}+\alpha p_{k-1}\big)$ be minimal. However, the iterates x_k found by the CG algorithm achieve a much stronger result, which is not matched by steepest-descent method:

Theorem 4.3 (optimality property of CG iterates) *The current residual r_k generated by the CG method is orthogonal to all search directions $p_0,\ldots,p_{k-1}$ generated so far by the algorithm:*

$$p_j^{\mathrm{T}} r_k = 0, \quad 0 \le j < k. \tag{4.11}$$

Consequently:

- *x_k minimizes $J(x)$ over the k-th dimensional subspace span$\big(p_0, \ldots, p_{k-1}\big)$ generated by all search directions $p_0, \ldots, p_{k-1}$ so far, not just over the current search direction.*
- *The CG method must converge in at most n iterations (since at that point span$\big(p_0, \ldots, p_{n-1}\big) = \mathbb{R}^n$)*

Exercise 4.2 (proof of Theorem 4.11) *Prove the orthogonality property* (4.11) *(use conjugacy requirement* (4.10) *and relations implemented by Algorithm 4.1). Then, show* (4.11) *implies that $\nabla J(x_k)$ has a zero projection on span$\big(p_0, \ldots, p_{k-1}\big)$.*

- That the CG method must converge after at most n iterations, while reassuring (the steepest descent does not have this property and may for example take much more iterations), is insufficient in practical aplications involving large systems. Then, the practical goal is to converge within much fewer than n iterations. This is achievable by a combination of factors:
  (a) Theorem 4.3 states a theoretical result about reaching exactly the solution $x = x^\star$ of $Ax = b$. In practice, we set $x^\star \approx x_K$, where x_K is the first iterate that is "close enough" according to some tolerance, for example such that $\|b - Ax_K\|/\|b\| \leq \varepsilon$ with ε a preset small tolerance.
  (b) The original system $Ax = b$ may be replaced by a "better" (equivalent) version obtained by *preconditioning*. See Section 4.7, which has a faster rate of convergence, i.e. needs fewer iterations to reach an iterate within the desired tolerance.

Other important remarks can be made regarding the conjugate gradient method:

- Each iteration requires one matrix-vector product. For large SPD systems, this is the main computational task, as all other operations act only on vectors (linear combinations, scalar products).
- Each iteration uses the same value of the matrix-vector product twice (see lines), which is why it is computed (once per iteration) and stored in a vector q_k rather than evaluated "on the fly".
- We follow the common usage of emphasizing the *residual* $r := b - Ax$ of the linear system rather than the gradient $\nabla J(x)$ of the associated quadratic functional, keeping in mind that $r(x) = -\nabla J(x)$. In particular, $r(x)$ gives the direction of steepest descent starting at x.
- Vectors and coefficients, such as $p_k, r_k, \alpha_k \ldots$, carry a subscript k or $k-1$ mostly for explanatory purposes, as there is no need to store (for example) the complete sequence of search directions $p_0, p_1, \ldots$ and the vectors are rewritten at each iteration. The economical (hence practical) form of the CG algorithm 4.1 is given next in Algorithm 4.2, wherein the equal sign "=" indicates that the right-hand side is evaluated and the result stored in the left-hand side[3] (either defining it or rewriting its content if already defined).
- There is no loss of generality in setting $x_0 = 0$ for the initial guess in Algorithm 4.1. Using a different initial guess just requires to either (i) modify the initialization phase in Algorithm 4.1, or (ii) make the requisite translation of the variable x in the definition (4.3) of $J(x)$.

Algorithm 4.2 Conjugate gradient algorithm for SPD problems (practical form)

$A \in \mathbb{R}^{n\times n}$ SPD and $b \in \mathbb{R}^n$	(problem data)
$x = 0$, $r = b$, $p = r$	(initialization)
for $k = 1, 2, \ldots$ **do**	
$q = Ap$	(matrix-vector product)
$\gamma = p^{\mathrm{T}} q$	(auxiliary coefficient)
$\alpha = (r^{\mathrm{T}} r)/\gamma$	(optimal step)
$x = x + \alpha p$	(solution update)
$r = r - \alpha q$	(residual update)
$\beta = (r^{\mathrm{T}} r)/\alpha\gamma$	(conjugacy coefficient)
$p = r + \beta p$	(descent direction for next iteration)
Stop if convergence, return solution x	
end for	

[3] This common coding convention is used by MATLAB, for instance, and is often used in explanatory pseudo-code.

We will discuss the convergence and other important properties of the CG algorithm later (see Sec. 4.6), after having introduced additional notions, in particular the concept of Krylov subspaces. This pertains to a wider class of iterative solution algorithms, to which the CG method actually belongs.

4.4 KRYLOV SPACES. We just presented a heuristic derivation of the CG algorithm, an iterative algorithm that is of major importance and widespread usefulness. At this point, it is time to introduce a concept that plays a primary role in the formulation and analysis of iterative solution methods for linear systems, namely that of Krylov spaces. We will then (in Sec. 4.5) describe GMRES, the other major iterative solution algorithm presented in this course, whose definition is based on Krylov spaces, before revisiting (in Sec. 4.6) the CG method and analyse it as another Krylov-based algorithm. Before all that, we begin by defining and motivating in simple terms the notion of Krylov spaces.

Going back to the splitting idea used in early iterative methods, we may consider the splitting $A = I + (A - I)$ and formulate iterations for solving $Ax = b$ on the basis of

$$x_{k+1} + (A-I)x_k = b \implies x_{k+1} = b + x_k - Ax_k, \qquad x_1 = b. \tag{4.12}$$

Actual convergence of the above scheme requires $\rho(A - I) < 1$, a rather restrictive condition on A. This is too restrictive for recommending the above iterations for general use, but here we only want to make the following observation: solution iterates generated by (4.12) are $x_1 = b$, $x_2 = 2b - Ab$, $x_3 = 3b - 3Ab + A^2b \ldots$ and a simple induction indicates that

$$x_k \in \text{span}\big(b, Ab, A^2b, \ldots, A^{k-1}b\big), \quad k = 1, 2, \ldots$$

Definition 4.1 (Krylov subspaces) *Let $A \in \mathbb{K}^{n\times n}$ be a square matrix, and let $b \in \mathbb{K}^n$. For any $k \leq n$, the* Krylov subspace $\mathcal{K}_k = \mathcal{K}_k(A, b)$ *associated with A, B is defined as*

$$\mathcal{K}_k(A, b) := \mathit{span}\big(b, Ab, A^2b, \ldots, A^{k-1}b\big).$$

Importantly, Krylov subspaces are nested*: we always have $k < \ell \implies \mathcal{K}_k \subset \mathcal{K}_\ell$.*

The notion of Krylov subspace emphasizes two important aspects of many iterative solution methods: (i) Krylov basis vectors are generated by computing matrix-vector products involving A (since e.g. $A^k b = A(A^{k-1}b)$), and (ii) vectors belonging to Krylov subspaces are representable by means of polynomials evaluated on A. Indeed, introducing the set $\mathcal{P}_k$ of all polynomials $p(X)$ of degree k such that $p(0) = 1$ (the zeroth-degree coefficient is set to 1 while all others are free, which makes $\mathcal{P}_k$ an affine space of dimension k), we have

$$x \in \mathcal{K}_k(A, b) \;\Leftrightarrow\; x = p(A)b \text{ for some polynomial } p \in \mathcal{P}_{k-1}.$$

The following well-known definition is important in the present context as it emphasizes additional links between matrices and polynomials:

Definition 4.2 (characteristic polynomial, minimal polynomial) *The* characteristic polynomial p_A *of a square matrix $A \in \mathbb{K}^{n\times n}$ is defined by $p_A(\lambda = det(A - \lambda I)$. The degree of p_A is n.*

- *For any matrix $A \in \mathbb{K}^{n\times n}$, we have $P_A(A) = 0$ (this is the Cayley-Hamilton theorem).*
- *The polynomial q_A of smallest degree such that $q_A(A) = 0$ is called the* minimial polynomial *of A; it divides p_A.*

Since the minimal polynomial of A can be of degree less than n, sequences of Krylov subspaces may stagnate:

Definition 4.3 (grade of Krylov subspaces) *It may happen that $\mathcal{K}_\ell(A, b) = \mathcal{K}_{\ell+1}(A, b)$ for some $\ell < n$. In such a case, the sequence of Krylov subspaces becomes stationary: we have $\mathcal{K}_\ell(A, b) = \mathcal{K}_{\ell+1}(A, b) = \mathcal{K}_{el_{+2}}(A, b) = \ldots = \mathcal{K}_n(A, b)$. The smallest value of ℓ such that $\mathcal{K}_\ell(A, b) = \mathcal{K}_{\ell+1}(A, b)$ (i.e. the degree of the minimal polynomial of b with respect to A) is called the* grade *of b with respect to A. The grade ℓ is at most equal to the degree of q_A.*

With definition 4.1, we see that successive iterates produced by (4.12) are such that $x_k \in \mathcal{K}_k(A, b)$. As it turns out, many of the major iterative solution methods for linear systems, including the CG method (already discussed in Sec. 4.3 and to be revisited in this light in Sec. 4.6) and GMRES (to be presented next) are rooted in the concept of Krylov subspaces.

4.5 GMRES. The general approach of solving a linear system $Ax = b$ via the minimization of its residual $b - Ax$ translates, for general square systems whose matrix $A \in \mathbb{K}^{n\times n}$ is only required to be invertible (i.e. no symmetry or sign requirement), into the celebrated GMRES algorithm [31, 29]. GMRES stands for *Generalized Minimal RESiduals*.

4.5.1 Basic idea. The basic idea is simple: the iterate x_k is sought as the element of $\mathcal{K}_k(A, b)$ that minimizes the norm of the residual:

$$x_k = \operatorname*{arg\,min}_{x\in\mathcal{K}_k(A,b)} \|b - Ax\|_2^2 \tag{4.13}$$

A naive (and insufficiently efficient) implementation of this idea would go as follows. The formulation (4.13) implies that x_k is sought as

$$x_k = y_0 b + y_1 Ab + \ldots + y_{k-1}A^{k-1}b,$$

(notice the polynomial-in-A form of x_k) with the unknown coefficients $y_0, \ldots, y_{k-1}$ to be determined by solving the least-squares problem in (4.13). This can be more compactly formulated by introducing the Krylov matrix $K_k := \begin{bmatrix} b, & Ab, & \ldots A^{k-1}b \end{bmatrix}$ and observing that the least-squares problem (4.13) can be reformulated directly in terms of $y_k := \{y_0, \ldots, y_{k-1}\}^{\mathrm{H}}$ as

$$y_k = \operatorname*{arg\,min}_{y\in\mathbb{K}^k} \|b - AK_k y_k\|_2^2 \tag{4.14}$$

where the matrix AK_k has size $n \times k$, while A is square $n \times n$. Equations (4.13) and (4.14) define a procedure that in principle is valid but presents significant practical isues:

- Any sequence of Krylov vectors $b, Ab, A^2b \ldots$ converges to a non-normalized eigenvector associated with the largest (in modulus) eigenvalue of A (see Sec. 5.2), making them increasingly parallel as the dimension of $\mathcal{K}_k(A, b)$ increases. This makes GMRES iterations potentially ill-conditioned.
- Problem (4.14) is conceivably solvable by classical least-squares methods using e.g. the QR factorization or the SVD, see Sec. 3.3, but this would be rather inefficient as (for example) computing the requisite QR factorization at iteration k would require $O(nk^2)$ floating-point multiplications.

4.5.2 The GMRES algorithm. We now explain how to modify the basic approach (4.13) and (4.14) so as to address the above remarks and obtain a stable and efficient version of GMRES. The increasingly-collinear nature of the Krylov vectors is remedied by successive orthogonalization (i.e. computing a sequence of orthonormal vectors $q_1, q_2, \ldots$ such that $\mathcal{K}_k(A, b) = \operatorname{span}(q_1, \ldots, q_k)$ for each k), while the computational cost of solving (4.14) is reduced by a suitable decomposition fo A that reduces it to a $(k+1)\times k$ least squares problem in recursive fashion.

Arnoldi iteration. As it happens, both orthogonalization and format reduction are very neatly solved by a recursive process, called Arnoldi iteration, that is quite simple to explain and implement. Arnoldi iterations seek a sequence of orthonormal n-vectors $q_1, q_2, \ldots \in \mathbb{K}^n$ such that for each $k = 1, 2, \ldots$ we have

$$AQ_k = Q_{k+1}H_k, \quad \text{that is, } A[q_1 \ \ldots \ q_k] = [q_1 \ \ldots \ q_k, q_{k+1}]H_k, \tag{4.15}$$

where $H_k \in \mathbb{K}^{(k+1)\times k}$ is of the form

$$H_k = \begin{bmatrix} h_{11} & & \ldots & h_{1k} \\ h_{21} & h_{22} & & \\ & \ddots & \ddots & \vdots \\ & & h_{k,k-1} & h_{kk} \\ 0 & & & h_{k+1,k} \end{bmatrix}$$

with entries h_{ij}, $i-j \geq 1$ to be determined together with the q_j (all entries below the first sub-diagonal are zeros); such a matrix H_k is known as a *upper Hessenberg matrix*. The vectors q_i and entries h_{ij} are found recursively by successively enforcing equality (4.15) for $k = 1, 2, \ldots$, as follows:

- Initialization: choose an arbitrary vector $b \in \mathbb{K}^n$, set $q_1 = b/\|b\|$.
- First iteration: seek $q_2 \in \mathbb{K}^n$ and h_{11}, h_{21} satisfying (4.15) for $k=1$, i.e.:

$$Aq_1 = h_{11}q_1 + h_{21}q_2, \qquad \text{with (a) } q_1^{\mathrm{H}}q_2 = 0, \quad \text{(b) } \|q_2\| = 1$$

  Condition (a) gives $h_{11} = q_1^{\mathrm{H}}Aq_1$, then condition (b) yields $h_{21} = \|Aq_1 - h_{11}q_1\|$. The resulting vector $q_2 = (Aq_1 - h_{11}q_1))/h_{21}$ satisfies (a) and (b).
- Running (k-th) iteration: proceed similarly, seeking q_{k+1} and $h_{1k}, \ldots, h_{(k+1),k}$ such that the k-th columns in both sides of (4.15) are equal (equality being already true for the leading $k-1$ columns as a result of the previous iterations):

$$Aq_k = h_{1k}q_1 + h_{2k}q_2 + \ldots h_{k+1,k}q_{k+1}, \qquad \text{with (a) } q_j^{\mathrm{H}}q_{k+1} = 0 \ (1 \le j \le k), \quad \text{(b) } \|q_{k+1}\| = 1$$

  Conditions (a) yield $h_{jk} = q_j^{\mathrm{H}}Aq_k$, then (b) gives $h_{k+1,k} = \|Aq_k - h_{1k}q_1 - \ldots h_{kk}q_k\|$. The resulting vector $q_{k+1} = \big(Aq_k - h_{1k}q_1 - \ldots h_{kk}q_k\big)/h_{k+1,k}$ satisfies conditions (a), (b).

This process has many applications in numerical linear algebra, and is in particular involved in algorithms for eigenvalue problems. To emphasize the importance of Arnoldi iterations, they are shown in basic pseudocode form:

Algorithm 4.3 Arnoldi iterations

```
1:  A ∈ K^{n×n} and b ∈ K^n                (problem data)
2:  q_1 = b/‖b‖                            (initialization)
3:  for k = 1, 2, ... do
4:     q_{k+1} = A q_k                        (new Krylov vector: initialize q_{k+1})
5:     for j = 1 to k do
6:        h_{jk} = q_j^H q_{k+1}              (entry in k-th column of H_k)
7:        q_{k+1} = q_{k+1} − h_{jk} q_j      (update (not yet normalized) vector q_{k+1})
8:     end for
9:     h_{k+1,k} = ‖q_{k+1}‖                  (last entry in k-th column of H_k)
10:    q_{k+1} = q_{k+1}/‖h_{k+1,k}‖          (normalize q_{k+1})
11: end for
```

Exercise 4.3 *Prove by induction that Arnoldi iterations produce vectors $q_1, q_2, \ldots$ satisfying span$(q_1, \ldots, q_k) = \mathcal{K}_k(A, b)$ for each $k = 1, 2, \ldots$.*

Application of Arnoldi iteration to GMRES. It is easy to see that the sequence $q_1, q_2 \ldots$ produced by the above Arnoldi iterations are such that $\mathrm{span}(q_1, \ldots, q_k) = \mathcal{K}_k(A, b)$ for each k. Therefore, going back to the defining minimization problem (4.13), we can seek $x_k \in \mathcal{K}_k(A, b)$ by expanding x_k on the orthonormal vectors q_j (instead of the Krylov vectors $A^{j-1}b$), setting $x_k = Q_k y_k$. By virtue of (4.15), the minimization (4.13) becomes

$$\begin{aligned} y_k = \underset{y \in \mathbb{K}^k}{\arg\min} \|b - Q_{k+1}H_k y\|_2^2 &= \underset{y \in \mathbb{K}^k}{\arg\min} \|Q_{k+1}^{\mathrm{H}}b - H_k y\|_2^2 \\ &= \underset{y \in \mathbb{K}^k}{\arg\min} \|\beta e_1 - H_k y\|_2^2 \qquad (\beta = \|b\|) \end{aligned} \tag{4.16}$$

instead of (4.14). The matrix H_k in the above least-squares problem is of size $(k+1) \times k$. The last equality stems from the fact that $b = \|b\|q_1 = \beta q_1$, see line 2 of Algorithm 4.3. Moreover, the transition from step $k-1$ to step k of GMRES only requires one new vector q_{k+1} and augmentation of the existing Hessenberg matrix H_{k-k} by one new column with entries $h_{k1}, \ldots h_{k+1,k}$; in other words, each GMRES iteration requires one Arnoldi iteration. A simple version of the GMRES algorithm is shown in pseudocode form in Algorithm 4.4.

Algorithm 4.4 GMRES algorithm with Arnoldi iterations

1: $A \in \mathbb{K}^{n\times n}$ and $b \in \mathbb{K}^n$ (problem data)
2: $x = 0$, $\beta = \|b\|$, $q_1 = b/\beta$ (initialization)
3: **for** $k = 1, 2, \ldots$ **do**
4: Step k of Arnoldi iteration (see Algorithm 4.3)
5: Find $y \in \mathbb{K}^k$, $\|\beta e_1 - H_k y\|_2^2 \to \min$ (least squares problem (4.16))
6: $x = Q_k y$ (current solution)
7: **Stop** if convergence, return x
8: **end for**

Restarted GMRES. In theory, we could let GMRES run for up to n iterations (we will see shortly that GMRES must in theory converge within at most n iterations). However, the least-squares problems become increasingly large and costly as k increases. A frequent approach thus consists in using *restarted GMRES*, where GMRES runs for a preset number $p \ll n$ iterations (unless convergence is reached earlier) and is stopped before the convergence criterion is reached. The solution x_p achieved at this stage is then used as initial guess for a new sequence $k = 1, 2, \ldots$ of GMRES iterations. This approach is often denoted `gmres(p)` for short. The *restart parameter* p is often set to $p = 50$–100.

Complete Arnoldi iterations. Normally GMRES does not require to run Arnoldi iterations until their completion (i.e. a total of n Arnoldi steps), since (i) satisfactory convergence may occur much earlier, or (ii) if not, GMRES is restarted after $m \ll n$ iterations, making the last Arnoldi iteration unimportant in this context. For completeness, we however observe that the last Arnoldi iteration ($k = n$) cannot have the same form as the running iteration for $2 \le k \le n-1$ (as defined e.g. in lines 4–10 of Algorithm 4.3), since it is not possible to define a new vector q_{n+1} that is orthogonal to $q_1, \ldots, q_n$. Instead, the last Arnoldi iteration simply consists in writing

$$Aq_n = h_{1n}q_1 + h_{2k}q_2 + \ldots h_{nn}q_n, \quad \Longrightarrow \quad h_{in} = q_i^{\mathrm{H}} A q_n \ (1 \le i \le n).$$

The last row of H_n (as a $(n+1)\times n$ matrix) is omitted, and $H_n \in \mathbb{K}^{n\times n}$ is a square upper Hessenberg matrix, i.e. has the form

$$H = \begin{bmatrix} h_{11} & & \cdots & h_{1n} \\ h_{21} & h_{22} & & \\ & \ddots & \ddots & \vdots \\ 0 & & h_{n,n-1} & h_{nn} \end{bmatrix}$$

Theorem 4.4 (complete Arnoldi iteration) *Let Q_n and H_n be the matrices reached at the end of the n steps entailed by running the Arnoldi iteration process to completion. The decomposition*

$$AQ = QH, \quad \textit{i.e.} \quad A = Q^{\mathrm{H}} H Q \qquad (Q \in \mathbb{K}^{n\times n} \textit{ unitary, } H \in \mathbb{K}^{n\times n} \textit{ square upper Hessenberg})$$

is in fact obtained, where $Q := Q_n$ and $H := H_n$.

The matrix Q_k generated by $k-1$ Arnoldi steps is the orthogonal factor of the reduced QR decomposition of the Krylov matrix K_k (see (4.14)): the matrix $R_k := Q_k^{\mathrm{H}} K_k \in \mathbb{K}^{n\times k}$, defined so that $K_k = Q_k R_k$, is upper triangular.

4.5.3 Polynomial approximation problem. Convergence properties of GMRES.

The following observation plays a key role for analyzing the properties of the GMRES algorithm: any element $x_k \in \mathcal{K}_k(A, b)$ is of the form

$$x = q(A)b$$

for some polynomial $q(X)$ of degree $k-1$ (the k coefficients of q being the coordinates of x in the Krylov basis $b, Ab, \ldots, A^{k-1}b$). Then, for the residual $r_k := b - Ax_k$, we have

$$r_k = b - Ax_k = b - Aq(A)b = (1 - Aq(A))b.$$

The polynomial $p(A) := 1 - Aq(A)$ is of degree k and verifies $p(0) = 1$, and thus belongs to the affine space $\mathcal{P}_k$ introduced in Sec. 4.4. This implies that the defining least-squares minimization problem (4.13) can be equivalently recast as the following minimization problem over polynomials:

$$\text{Find } p_k \in \mathcal{P}_k, \qquad \|p_k(A)b\|_2^2 \to \text{minimum} \qquad \Big(\text{i.e.} \quad p_k = \arg\min_{p\in\mathcal{P}_k} \|p(A)b\|_2^2\Big)$$

This characterization of the k-th iteration of GMRES implies:

Theorem 4.5 (convergence of GMRES iterates)

- *Since $p_A(A) = 0$, the GMRES iterations must converge after at most n iterations.*
- *If ℓ is the grade of b relative to A (see Def. 4.3), the GMRES iterations converge after ℓ iterations to a solution with zero residual.*

For the minimizing polynomial p_k, we have

$$\|r_k\| = \|p_k(A)b\| \le \|p_k(A)\|\|b\|, \tag{4.17}$$

for any induced matrix norm $\|\cdot\|$, so that, for given b, the size of the residual r_k at iteration k mainly depends on $\|p_k(A)\|$. As one wants $\|r_k\|$ to be as small as possible, a key factor in the efficiency of GMRES iterations is: kow small can $\|p_k(A)\|_2$ be for a given matrix A?

If we restrict our attention to matrices A that are diagonalizable (recall that some non-symmetric matrices A, known as defective, are not diagonalizable, see Secs. 3.4 and 7.2), the latter question turns out to be amenable to a relatively simple analysis. Assume that $A = X\Lambda X^{-1}$, where X is square invertible and $\Lambda = \text{diag}\big(\lambda_1, \ldots, \lambda_n\big)$ holds the eigenvalues of A, conventionally arranged by decreasing moduli as usual. Then, we have $P(A) = Xp(\Lambda)X^{-1}$ and can therefore write:

$$\|p(A)\| = \|Xp(\Lambda)X^{-1}\| \le \|X\|\|X^{-1}\| \sup_{\lambda=\lambda_1,\ldots,\lambda_n} |p(\lambda)| = \kappa(X) \sup_{\lambda=\lambda_1,\ldots,\lambda_n} |p(\lambda)|.$$

from the above and (4.17), we deduce:

$$\frac{\|r_k\|}{\|b\|} \le \kappa(X) \sup_{\lambda=\lambda_1,\ldots,\lambda_n} |p(\lambda)|,$$

which shows that

- Fast convergence can be achieved if polynomials $p_k \in \mathcal{P}_k$ can be found such that their size on the eigenvalues of A decreases quickly with the degree k.

Eigenvalue clustering. The above can be achieved when the eigenvalues of A are clustered away from the origin. For instance, assume that all eigenvalues of A are evenly distributed in the disk $D(z_0, r)$ of radius r centered at $z_0 \in \mathbb{C}$ in the complex plane, with $|z_0| > r$. In this case, $p_k(\lambda)$ can be approximately minimized over $D(z_0, r)$ by setting

$$p_k(z) = \Big(\frac{z_0 - z}{z_0}\Big)^k$$

which satisfies the normalization constraint $p(0) = 1$. This gives

$$\sup_{z\in D(z_0,r)} p_k(z) = \big(r/|z_0|\big)^k,$$

i.e. a residual reduction by a factor $r/|z_0| < 1$ at each iteration (the smaller $r/|z_0| < 1$, the faster the convergence). By contrast, we see that situations where eigenvalues are clustered *around* $z = 0$ in $\mathbb{C}$ can be expected to have unfavorable convergence properties. This suggests the pssibility to transform the original system $Ax = b$ into another, equivalent, system whose matrix has eigenvalues clustered away from the origin; this is one possible approach to *preconditioning*, see Sec. 4.7.

4.6 REVISITING THE CONJUGATE GRADIENT METHOD: KRYLOV-SPACE INTERPRETATION, CONVERGENCE. It turns out that the Krylov-subspace framework is equally relevant fot the conjugate gradient method. Indeed, from Algorithm 4.1, it is easy to show (by induction on k) that $x_k \in \mathcal{K}_k(A,b)$, which implies that $r_k = b - Ax_k \in \mathcal{K}_{k+1}(A,b)$. Moreover, a simple rearrangemment shows that

$$J(x) = \tfrac{1}{2}x^{\mathrm{T}}Ax - x^{\mathrm{T}}b + c = \tfrac{1}{2}(x - A^{-1}b)^{\mathrm{T}}A(x - A^{-1}b) - \tfrac{1}{2}b^{\mathrm{T}}Ab + c = \tfrac{1}{2}\|x - A^{-1}b\|_A^2 + \text{constant},$$

so that the problems of minimizing $J(x)$ or $\|x - A^{-1}b\|_A^2$ are equivalent (their solution x is the same), the additive constant being irrelevant. Now, since $x^\star := A^{-1}b$ is the sought solution of $Ax = b$, $e := x - A^{-1}b$ is the solution error and

$$\|x - A^{-1}b\|_A^2 = \|x - x^\star\|_A^2 = \|e(x)\|_A^2.$$

In other words, the original minimization problem for $J(x)$ is equivalenty to minimizing the solution error in energy norm. Moreover:

$$x_k \in \mathcal{K}_k(A,b) \subset \mathcal{K}_{k+1}(A, A^{-1}b)) = \mathcal{K}_{k+1}(A, e(0))) \implies e(x_k) \in \mathcal{K}_{k+1}(A, e(0)));$$

(since $e(x_0) = e(0) = A^{-1}b$) in other words, for each iterate k, there exists a polynomial $p_k \in \mathcal{P}_k$ (of degree k) such that $e(x_k) = p_k(A)e(0)$ and the error minimization problem takes the form

$$\min_{p\in\mathcal{P}_k} \|p(A)e(0)\|_A^2$$

Here, using the diagonalization $A = Q\Lambda Q^{\mathrm{T}}$ of A ($Q \in \mathbb{R}^{n\times n}$ being orthogonal) and expanding $e(0)$ on the eigenvector basis as $e(0) = \varepsilon_1 q_1 + \ldots \varepsilon_n q_n$, we have

$$\|p(A)e(0)\|_A^2 = e(0)^{\mathrm{T}}\big(Qp(\Lambda)Q^{\mathrm{T}}\big)Q\Lambda Q^{\mathrm{T}}\big(Qp(\Lambda)Q^{\mathrm{T}}\big)e(0) = \sum_{i=1}^{n} \varepsilon_i^2 \lambda_i \big[p(\lambda_i)\big]^2,$$

implying

$$\frac{\|p(A)e(0)\|_A^2}{\|e(0)\|_A^2} \le \lambda_1 \max_{\lambda_1,\ldots,\lambda_n} \big[p(\lambda_i)\big]^2$$

To estimate the rate of convergence of the CGM, we are thus led to solve the polynomial minimization problem

$$\min_{p\in\mathcal{P}_k} \Big\{ \max_{\lambda_1,\ldots,\lambda_n} |p(\lambda_i)| \Big\},$$

with the additional information that the eigenvalues of A lie in the positive real interval $[\lambda_n, \lambda_1]$. Assuming no other information to be available on the eigenvalue distribution, it is relevant to consider instead the polynomial minimization problem

$$\min_{p\in\mathcal{P}_k} \Big\{ \max_{\lambda\in[\lambda_1,\lambda_n]} |p(\lambda)| \Big\},$$

which happens to have a known solution: the minimizing polynomial is given by

$$p_k(\lambda) = \frac{1}{T_k(\gamma)} T_k\Big(\gamma - \frac{2\lambda}{\lambda_1 - \lambda_n}\Big), \quad \max_{\lambda\in[\lambda_1,\lambda_n]} |p(\lambda)| = \frac{1}{T_k(\gamma)} \qquad \text{with } \gamma := \frac{\kappa+1}{\kappa-1}$$

(recalling that $\kappa = \kappa_2(A) = \lambda_1/\lambda_n$ is the spectral condition number of A) where T_k is the Chebyshev polynomial[4] of degree k. In the above result, $\lambda \in [\lambda_1, \lambda_n]$ implies that the argument of T_k ranges in $[-1,1]$, and it is known that $\max_{t\in[-1,1]} |T_k(t)| = 1$. Moreover, it is also known that, for any t such that $|t| > 1$, we have $T_k(t) = (z^k + z^{-k})/2$, having set $t = (z + z^{-1})/2$. Here, we have $\gamma = (z + z^{-1})/2$ with $z = (\sqrt{\kappa}+1)/(\sqrt{\kappa}-1)$, and hence

$$T_k(\gamma) = \frac{1}{2}\Big[\Big(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\Big)^k + \Big(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\Big)^{-k}\Big] \ge \frac{1}{2}\Big(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\Big)^k.$$

[4]The Chebyshev polynomials T_k are degree-k polynomials whose defining property is that $T_k(\cos\theta) = \cos(k\theta)$ for any $\theta \in [0, 2\pi]$, implying that $T_k([-1,1]) = [-1,1]$.

Theorem 4.6 (convergence rate of the conjugate gradient method) *Let the CG iterations (Algorithm 4.1) be applied to a SPD linear system $Ax = b$, the SPD matrix A having a spectral condition number κ. Then, the energy norms of the successive solution errors satisfy*

$$\frac{\|e(x_k)\|_A}{\|e(x_0)\|_A} \leq 2\Big(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\Big)^k$$

Exercise 4.4 *Prove that the k-th iterate x_k of the conjugate gradient method verifies $r_k \in \mathcal{K}_{k+1}(A, b)$.*

4.7 PRECONDITIONING. As we have seen, the rate of convergence of iterative solution methods such as the conjugate gradient or GMRES strongly depends on characteristics of the matrix A such as the distribution of its eigenvalues. Speeding up the convergence (i.e. reducing the number of iterations needed to reach a preset tolerance) is essential for efficiently solving large systems. To this aim, a frequently employed approach consists in *preconditioning* the linear system. This means that the original system $Ax = b$ is equivalently reformulated as either

$$\begin{aligned} (M_1A)x &= M_1b && \text{(left preconditioning)},\\ (AM_2)z &= b, \quad M_2z = x && \text{(right preconditioning)},\\ (M_1AM_2)z &= M_1b, \quad M_2z = x && \text{(symmetric preconditioning)}. \end{aligned} \tag{4.18}$$

The heuristic idea is to find matrices M_1 and/or M_2, called *preconditioners*, that are invertible and produce equivalent systems with better convergence properties. For the left or right preconditioning, the theoretically ideal choice is of course $M_1 = A^{-1}$ or $M_2 = A^{-1}$, but this has no practical value as computing A^{-1} beforehand would (a) yield directly the solution without any need for iterating, and (b) at a cost greater than just solving one system[5] The practical idea then usually consists in finding preconditioners that "approximate" the inverse of A while being "easy" (i.e. comparatively inexpensive) to apply.

The various options for preconditioning are sometimes expressed by introducing preconditioners in inverse form, e.g.

$$\begin{aligned} (M_1^{-1}A)x &= M_1^{-1}b && \text{(left preconditioning)},\\ (AM_2^{-1})z &= b, \quad M_2x = z && \text{(right preconditioning)}. \end{aligned} \tag{4.19}$$

instead of (4.18). This is of course a matter of convention and meaning in context. In (4.19), preconditioners M_1, M_2 are intended as "rough approximations" of A rather than A^{-1}.

Freequently-used ideas for preconditioning linear systems include the following:

- The simplest idea is the diagonal preconditioner (called Jacobi preconditioner), e.g. $M_1 = \mathrm{diag}(a_{11}^{-1}, \ldots, a_{nn}^{-1})$ for the left preconditioner.
- Sometimes the problem naturally suggests block preconditioners, where M_1 or M_2 is block diagonal.
- An important algebraic approach to preconditioning uses incomplete factorizations (e.g. incomplete LU or Cholesky factorizations) where the triangular factors are sparse or easy to invert for some other reason.
- Yet another algebraic approach to preconditioning consists in seeking a sparse matrix M such that (e.g. for left preconditioning) $\|I - MA\| \to \min$; such M is called a *sparse approximate inverse* of A.
- Preconditioners may in some cases be defined by adopting less-precise discretizations for the same problem (by means of e.g. multigrid methods, truncated Fourier or wavelet expansions) or otherwise ignoring small-scale components in multiscale models.
- Physical systems being modelled sometimes feature several components, or even many objects (such as inclusions in complex media or particles carried by fluids), which suggests preconditioners based on the idea of solving the same problem for one isolated component (this is to some extent related to the idea of block-diagonal preconditioners).

[5] Computing A^{-1} is equivalent to solving all systems $Az_k = e_k$ $(1 \leq k \leq n)$ and then setting $A^{-1} = [z_1\ z_2\ \ldots z_n]$.

Preconditioning an iterative solution method is to some extent an "art", as finding an efficient preconditioning method strongly depends on the characteristics of the problem at hand. Sometimes purely algebraic approaches such as incomplete factorizations or sparse approximate inverses work well. Some other times the underlying physics strongly suggest a more-approximate way to solve the problem at a lesser cost, a viewpoint that can also be translated into employing coarser discretizations. For this reason, it is not possible to state a general preconditioning methodology applicable to all linear systems (or to wide classes, e.g. all SPD systems), and each application must be considered via its specific characteristics.

4.7.1 Example: left-preconditioned GMRES algorithm. Let us consider a left-preconditioned system of the form

$$M^{-1}Ax = M^{-1}b$$

where (as discussed before) M "approximates" A and is such that systems $My = f$ are significantly cheaper to solve than the original system $Ax = b$. The left-preconditioned GMRES algorithm is a simple modification of Algorithm 4.4 where each new Krylov vector in the Arnoldi iteration is computed by solving $Mq_{k+1} = Aq_k$ instead of $Mq_{k+1} = Aq_k$ (see Algorithm 4.3). A pseudocode version of the left-preconditioned GMRES algorithm is shown in Algorithm 4.5, with the Arnoldi iteration incorporated into it; the one substantial modification with respect to the non-preconditioned version is the modified evaluation of new Krylov vectors (line 4 of Algorithm 4.5).

Algorithm 4.5 Left-preconditioned GMRES algorithm

1:	$A \in \mathbb{K}^{n\times n}$ and $b \in \mathbb{K}^n$	(problem data)
2:	$x = 0$, $\beta = \|b\|$, $q_1 = b/\beta$	(initialization)
3:	**for** $k = 1, 2, \ldots$ **do**	
4:	solve $Mq_{k+1} = Aq_k$ for q_{k+1}	(new Krylov vector: initialize q_{k+1})
5:	**for** $j = 1$ to k **do**	
6:	$h_{jk} = q_j^{\mathrm{H}} q_{k+1}$	(entry in k-th column of H_k)
7:	$q_{k+1} = q_{k+1} - h_{jk} q_j$	(update (not yet normalized) vector q_{k+1})
8:	**end for**	
9:	$h_{k+1,k} = \|q_{k+1}\|$	(last entry in k-th column of H_k)
10:	$q_{k+1} = q_{k+1}/\|h_{k+1,k}\|$	(normalize q_{k+1})
11:	Find $y \in \mathbb{K}^k$, $\|be_1 - H_k y\|_2^2 \to \min$	(least squares problem (4.16))
12:	$x = Q_k y$	(current solution)
13:	**Stop** if convergence, return x	
14:	**end for**	

4.7.2 Illustration: preconditioned boundary element method for scattering problems. We illustrate the effect and usefulness of preconditioning on the large-scale boundary element method (BEM) applied to wave scattering problems, with an example borrowed from [13], and refer to the brief description of the BEM given in Sec. 1.2 and the references therein. The geometrical configuration consists in a spherical shell with an open cavity (a resonator), whose surface is meshed with boundary elements, located in an infinite 3D acoustic medium (see Fig. 4.2). The computational problem consists in determining the acoustic fields arising when a known incident acoustic wave is perturbed by the resonator. The open cavity, in particular, makes the acoustic domain trapping (part of the wave energy being trapped inside the cavity) and this results in slower convergence of iterative solvers such as GMRES used for solving the linear system $Gu = f$ arising from the BEM discretization of a boundary integral equation similar to (1.3). A preconditioning method that is specific to BEM acoustic problems, based on the concept of *on-surface radiation condition* (OSRC), is proposed and validated in the cited work [13], to which we refer for any details on the methodology. Figure 4.3 shows how the distribution of the eigenvalues of G in the complex plane is modified by preconditioning;

notice in particular the clustering effect away from the origin brought by the preconditioning method. Figure 4.4 demonstrates the dramatic reduction of GMRES iterations brought by preconditioning.

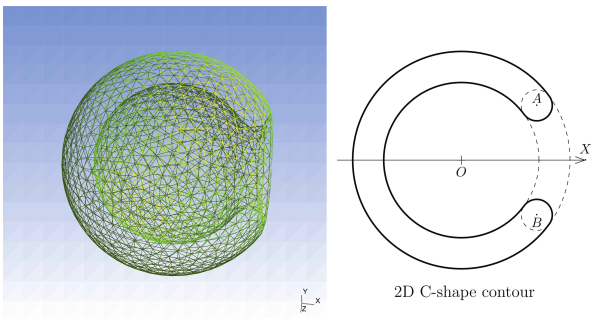


Fig. 4. Sphere with cavity and its 2D generator.

Figure 4.2: BEM analysis of sphere with cavity: geometrical configuration. From [13].

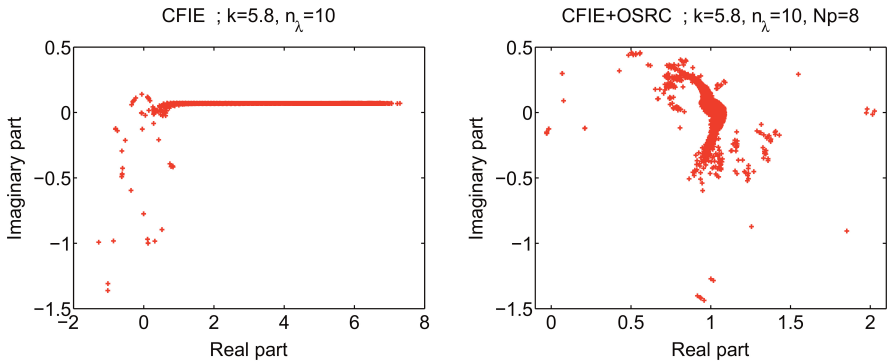
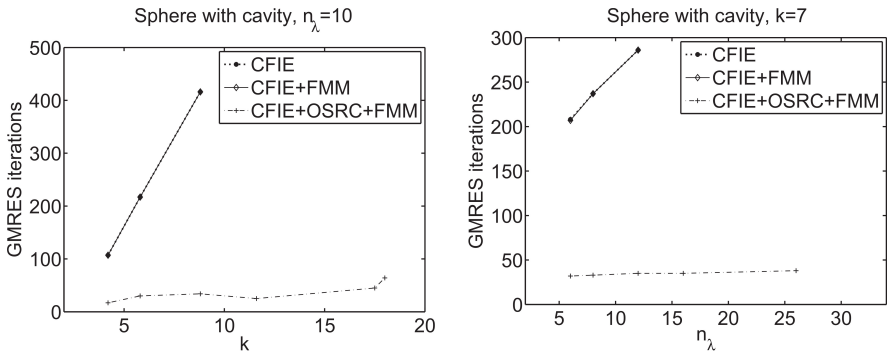


Fig. 14. Sphere with cavity: distribution of the eigenvalues, $k = 5.8,\ n_\lambda = 10$.

Figure 4.3: BEM analysis of sphere with cavity: eigenvalue distribution for the non-preconditioned (left) and preconditioned (right) formulations. From [13].



(a) GMRES iterations vs k (b) GMRES iterations vs n_λ

Figure 4.4: BEM analysis of sphere with cavity: GMRES iteration count with or without OSRC preconditioning, plotted against frequency (left) or discretization density per wavelength (right). From [13].

CHAPTER 5 MATRIX EIGENVALUE PROBLEMS

5.1 OVERVIEW. Eigenvalues and eigenvectors are useful for a very diverse array of reasons.

- The stability in time of many mechanical and physical systems is determined by whether certain eigenvalues satisfy (or not) certain bounds.
- Likewise, eigenvalues reveal conditions at which a system may undergo resonance.
- Eigenvectors are often used (e.g. in structural dynamics) as a convenient and physically meaningful way to approximate the system response as combinations of a moderate number of eigenvectors.

Moreover, such physical arguments often directly translate into algorithmic concepts or improvements. Other uses are more closely related to algorithms. For example:

- Computing the SVD of a matrix requires eigenvalues and eigenvectors of symmetric positive matrices. Then, the singular values and vectors often provide crucial information about e.g. the information content of a linear system and its amenability to compression.
- Understanding the behavior of matrix eigenvalues is essential in the design and assessment of *preconditioning* methods that enhance iterative solution algorithms.

This chapter presents methods for the computation of matrix eigenvalue and eigenvectors, which all evolve from the basic idea of power iterations. We focus on the symmetric (Hermitian) eigenvalue problem, which involve matrices that are always diagonalizable and whose eigenvalues are always real. The more-complex unsymmetric eigenvalue problems are only briefly discussed at the end of the chapter. We begin with methods designed for finding isolated eigenvalues and eigenvectors, and then show how their extension to methods allowing to compute complete matrix spectra.

5.1.1 Summary of basic facts on matrix eigenvalues. For any square matrix $A \in \mathbb{K}^{n\times n}$, there exist n numbers $\lambda_1, \ldots, \lambda_n$ (called eigenvalues) and associated vectors $x_1, \ldots, x_n$ (called eigenvectors) such that

$$Ax_i = \lambda_i x_i \qquad 1 \le i \le n.$$

The n eigenvalues are not necessarily distinct: they are the n roots of the *characteristic polynomial*

$$p_A(\lambda) := \det(A - \lambda I),$$

and as such can have multiplicities. The eigenvalues and eigenvectors of real matrices may be complex.

- If all eigenvectors x_i are linearly independent (i.e. if $\mathbb{K}^n = \mathrm{span}(x_1, \ldots, x_n)$), A is *diagonalizable*: setting $\Lambda := \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ and $X := [x_1, \ldots, x_n]$, we have $A = X\Lambda X^{-1}$.
- If in fact $A = Q\Lambda Q^{-1} = Q\Lambda Q^{\mathrm{H}}$ with Q unitary, A is said to be *unitarily diagonalizable*. Unitary diagonalization of A is possible if and only if A is *normal*, i.e. satisfies $AA^{\mathrm{H}} = A^{\mathrm{H}}A$.
- Symmetric (Hermitian if $\mathbb{K} = \mathbb{C}$) matrices, which verify $A = A^{\mathrm{H}}$, are in particular normal, and hence unitarily diagonalizable. Moreover, all its eigenvalues are real. If A is real symmetric, Q is real orthogonal and $A = Q\Lambda Q^{\mathrm{T}}$.
- If A is not diagonalizable, it is called *defective*. For example, $A = \left[\begin{smallmatrix} a & b \\ 0 & a \end{smallmatrix}\right]$ is defective, having a double eigenvalue $\lambda = a$ whose eigenspace $E_\lambda = \mathrm{span}(x)$ with $x = \{1, 0\}^{\mathrm{T}}$ has dimension 1[1].

[1] Generally speaking, each eigenvalue λ has an *algebraic multiplicity* as a root of $p_A(\lambda)$, and a *geometric multiplicity* given by the dimension $\dim(E_\lambda)$ of its eigenspace. The former is always at least as great as the latter. A matrix is defective when at least one eigenvalue is of algebraic multiplicity strictly greater than its geometric multiplicity.

- **Gershgorin theorem:** All the eigenvalues of $A \in \mathbb{K}^{n\times n}$ lie in one of the n *Gershgorin disks* $\mathcal{D}_i(A)$ of A, where $\mathcal{D}_i(A) := \big\{ z \in \mathbb{C} : |z - a_{ii}| \le \sum_{j\neq i} |a_{ij}| \big\}$.

5.1.2 Iterative methods. We start with a very important fact about the numerical solution of eigenvalue problems: *no direct general solution method can exist for the computation of matrix eigenvalues.*

Indeed, the eigenvalues of $A \in \mathbb{K}^{n\times n}$ are the roots of the characteristic polynomial $p_A(\lambda) = \det(A - \lambda I)$. Any direct eigenvalue solver would therefore also be a direct algorithm for finding the zeros of $p_A(\lambda)$. However, the groundbreaking analysis of Galois (made at age 18!) established that no general method for the closed-form solution of polynomial equations of degree greater than 4 exists, precluding direct eigenvalue computation methods. We moreover observe that the tasks of finding eigenvalues of a matrix or zeros of a polynomial are in fact equivalent: given an arbitrary polynomial $P(X) = a_0 + a_1 X + \ldots + a_{n-1}X^{n-1} + X^n$ (assumed to be monic without loss of generality), there exists a matrix $A = A_P \in \mathbb{K}^{n\times n}$ whose characteristic polynomial is P, called the *companion matrix* of P:

$$A_P = \begin{bmatrix} 0 & 0 & \ldots & 0 & -a_0 \\ 1 & 0 & \ldots & 0 & -a_1 \\ 0 & 1 & \ldots & 0 & -a_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & -a_{n-1} \end{bmatrix}.$$

Consequently: *any general method for computing matrix eigenvalues must be iterative.*

5.1.3 Setting for this chapter. With the exception of the closing Section 5.5.1, the matrix $A \in \mathbb{K}^{n\times n}$ whose eigenvalues and (possibly) eigenvectors are sought is assumed throughout this chapter to be Hermitian (and hence unitarily diagonalizable with real eigenvalues). We will denote by $A = Q\Lambda Q^{\mathrm{H}}$ the diagonalized form of A, where $\Lambda = \operatorname{diag}\big(\lambda_1, \ldots, \lambda_n\big)$ holds the eigenvalues conventionally ordered according to $|\lambda_1| \ge |\lambda_2| \ge \ldots \ge |\lambda_n|$ and the unitary matrix $Q = [q_1, \ldots, q_n] \in \mathbb{K}^{n\times n}$ collects corresponding orthonormal eigenvectors. All norms appearing in this chapter are vector 2-norms and the corresponding induced matrix norm, so that $\|\cdot\| = \|\cdot\|_2$ throughout.

5.2 COMPUTATION OF ISOLATED EIGENVALUES. We begin by examining conceptually simple methods that provide partial information (typically one or a few eigenvalues and associated eigenvectors). In addition to being useful on their own, they constitute a first step towards defining and understanding more-advanced algorithms such as the QR iteration algorithm described in Sec. 5.3.3.

5.2.1 Rayleigh quotient. Let $A \in \mathbb{K}^{n\times n}$ Hermitian, and pick a vector $x \in \mathbb{K}^n$. The Rayleigh quotient $r(x)$ is the real scalar

$$r(x) := \frac{x^{\mathrm{H}} A x}{x^{\mathrm{H}} x}$$

We first observe, via a straightforward derivation, that

$$\frac{\partial r(x)}{\partial x_i} = \frac{2}{x^{\mathrm{H}} x}\big(Ax - r(x)x\big)_i, \qquad \text{i.e.} \qquad \nabla r(x) = \frac{2}{x^{\mathrm{H}} x}\big(Ax - r(x)x\big)$$

which indicates that

- The Rayleigh quotient is stationary (has a vanishing gradient) if x is an eigenvector, and the value of $r(x)$ at such point is an eigenvalue of A.
- In particular, the smallest and largest eigenvalue of A minimize and maximize, respectively, the Rayleigh quotient over $x \in \mathbb{K}^n \setminus \{0\}$.

The above remarks have a generalization to the abstract setting of self-adjoint operators on Hilbert spaces, known as the *Courant-Fischer min-max principle* [6, Chap. 3].

5.2.2 Power iterations. These facts on the Rayleigh quotient suggest the very simple idea of iterating applications of A to some starting vector $x_0 \in \mathbb{K}^n$ and evaluating Rayleigh quotients along the way, in the expectation that repeated evaluations will converge to an eigenvector corresponding to the largest eigenvalue of A:

Algorithm 5.1 Power iteration

$A \in \mathbb{K}^{n\times n}$ Hermitian (input), $x^{(0)} \in \mathbb{K}^n$ with $\|x^{(0)}\| = 1$ (initialization)
for $k = 0, 1, 2, \ldots$ **do**
  $v = Ax^{(k)}$ (apply A to current normalized iterate)
  $\lambda^{(k)} = v^{\mathrm{H}} x^{(k)}$ (Rayleigh quotient)
  $x^{(k+1)} = v/\|v\|$ (next normalized iterate)
  Stop if convergence, set $\lambda_1 = \lambda^{(k)}$, $q_1 = x^{(k)}$
end for

The effect of repeated applications of A is to "promote" the eigenvalue λ_1 of A whose modulus is largest: expanding $x^{(0)}$ as a linear combination of the orthonormal eigenvectors, we have

$$\begin{aligned} x^{(0)} &= y_1 q_1 + y_2 q_2 + \cdots + y_n q_n \\ x^{(k)} &= c_k A^k x^{(0)} = c_k \lambda_1^k \big[y_1 q_1 + y_2 (\lambda_2/\lambda_1)^k q_2 + \cdots + y_n (\lambda_n/\lambda_1)^k q_n \big] \end{aligned} \tag{5.1}$$

for some normalization constant c_k and assuming $|\lambda_2/\lambda_1| < 1$. This allows one to deduce

Theorem 5.1 (convergence of power iterations) *Assume that $q_1^H x^{(0)} \neq 0$ and $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$ (i.e. that the largest eigenvalue has unit multiplicity). Then:*

$$|\lambda^{(k)} - \lambda_1| = O(|\lambda_2/\lambda_1|^{2k}), \qquad \| \pm x^{(k)} - q_1\| = O(|\lambda_2/\lambda_1|^k)$$

as $k \to \infty$. The sign $\pm$ indicates that for each step k a sign choice must be made for the above bound on the eigenvector to hold.

We observe in particular that

- The convergence of λ^k to λ_1 is linear (the error being reduced by a constant factor at each iteration), and the same is true for that of $x^{(k)}$ to q_1.
- The error reduction factor depends on the closeness of $|\lambda_1|$ and $|\lambda_2|$ (slow convergence if the eigenvalues are close in magnitude).
- The just-described "raw" power iteration only allows to evaluate λ_1 and an associated eigenvector.
- The main computational cost is incurred by that of one matrix-vector product per power iteration.

5.2.3 Inverse iterations. Let A be invertible, in which case we can consider applying power iterations using A^{-1}. This generates a sequence of vectors satisfying $x^{(k)} = A^{-1}x^{(k-1)}$, i.e. $Ax^{(k)} = x^{(k-1)}$ (so that each new iterate requires solving a linear system), which is expected to produce an eigenvector of A for the eigenvalue λ_n with *smallest* modulus.

Importantly, this observation implies that the eigenvalue of A closest to some reference value μ can be found by applying power iterations to $(A-\mu I)^{-1}$, the inverse of the *shifted* matrix $A-\mu I$. Assume that μ is close to an eigenvalue λ_j of A with eigenvector x_j (but not equal to that eigenvalue, so that $A-\mu I$ is invertible). Then, $\sigma_j := (\lambda_j - \mu)^{-1}$ is an eigenvalue of $(A-\mu I)^{-1}$ with the same eigenvector q_j; moreover:

$$|\mu - \lambda_j| < |\mu - \lambda_i| \ (i \neq j) \qquad \Longrightarrow \ |\sigma_j| > |\sigma_i| \ (i \neq j)$$

so that power iterations applied to $(A-\mu I)^{-1}$ will reveal σ_j and the eigenvector q_j, from which we deduce $\lambda_j = \sigma_j^{-1} + \mu$. This version of the power iteration, called *inverse iteration*, work as shown in Algorithm 5.2. Transposing Theorem 5.1, we have (with λ_ℓ the second-closest eigenvalue to λ_j) that

$$|\lambda_j^{(k)} - \lambda_j| = O\Big(\frac{|\lambda_j - \mu|}{|\lambda_\ell - \mu|} \Big)^{2k}, \qquad \| \pm x_k - q_j\| = O\Big(\frac{|\lambda_j - \mu|}{|\lambda_\ell - \mu|} \Big)^{k}$$

Algorithm 5.2 Inverse iteration

$A \in \mathbb{K}^{n\times n}$ Hermitian (input), $x^{(0)} \in \mathbb{K}^n$ with $\|x^{(0)}\| = 1$ (initialization), $\mu \in \mathbb{R}$ close to λ_j
for $k = 1, 2, \ldots$ **do**
  solve $(A-\mu I)x^{(k)} = x^{(k-1)}$ (apply $(A-\mu I)^{-1}$ to $x^{(k-1)}$)
  $x^{(k)} = x^{(k)}/\|x^{(k)}\|$ (next normalized iterate)
  $\lambda_j^{(k)} = (x^{(k)})^{\mathrm{H}} A x^{(k)}$ (Rayleigh quotient)
  Stop if convergence, set $\lambda_j = \lambda_j^{(k)}$, $q = x^{(k)}$
end for

5.2.4 Rayleigh quotient iterations. The idea of power iterations can be refined further by noticing that direct power iterations naturally produce an eigenvector whereas inverse iterations help in obtaining eigenvalues. The idea is to combine both algorithms, basically by setting μ in the inverse iteration to the current eigenvalue estimate yielded by one direct power iteration. This idea translates into the following (Rayleigh quotient) iteration algorithm:

Algorithm 5.3 Rayleigh quotient iteration

$A \in \mathbb{K}^{n\times n}$ Hermitian (input), $x^{(0)} \in \mathbb{K}^n$ with $\|x_0\| = 1$ (initialization)
$\lambda^{(0)} = (x^{(0)})^{\mathrm{H}} A x^{(0)}$ (Initialize Rayleign quotient)
for $k = 1, 2, \ldots$ **do**
  solve $(A-\lambda^{(k-1)} I)x^{(k)} = x^{(k-1)}$ (apply $(A-\lambda^{(k-1)}I)^{-1}$ to $x^{(k-1)}$)
  $x^{(k)} = x^{(k)}/\|x^{(k)}\|$ (next normalized iterate)
  $\lambda^{(k)} = (x^{(k)})^{\mathrm{H}} A x^{(k)}$ (Rayleigh quotient)
  Stop if convergence, set $\lambda = \lambda^{(k)}$, $q = x^{(k)}$
end for

The main properties of this algorithm can be summarized as follows:

Theorem 5.2 (convergence of Rayleigh quotient iterations) *Rayleigh iterations converge for almost all starting vectors $x^{(0)}$. Assume that $x^{(0)}$ (normalized to $\|x^{(0)}\| = 1$) is close to the eigenvector q_j. Then:*

$$|\lambda^{(k+1)} - \lambda_j| = O(|\lambda^{(k)} - \lambda_j|^3), \qquad \| \pm x^{(k+1)} - q_j\| = O(\| \pm x^{(k)} - q_j\|^3)$$

as $k \to \infty$. The signs $\pm$ (not necessarily the same on the two sides) indicate that for each step k a sign choice must be made for the above bounds to hold.

5.3 COMPUTATION OF MATRIX SPECTRA: ORTHOGONAL AND QR ITERATIONS. Power iterations methods and their improvements allow to compute samples of eigenvalues or eigenfunctions, but not the complete set of eigenvalues and eigenvectors (spectrum) of a given (Hermitian) matrix A. We now describe methods aimed at finding the complete spectrum. They rely on *invariance of eigenvalues under similarity transformations*:

- Let $A \in \mathbb{K}^{n\times n}$, and let $X \in \mathbb{K}^{n\times n}$ be invertible. Then, A and $B := XAX^{-1}$ have the same characteristic polynomial ($P_A(\lambda) = P_B(\lambda)$), and hence the same eigenvalues with same multiplicities.

Eigenvalue computation algorithms therefore revolve around finding a similarity decomposition $A = XTX^{-1}$ such that the eigenvalues of T are "easy" to compute. For example, if T is triangular (or, of course, diagonal), its diagonal holds the eigenvalues.

Factorizations introduced for direct solution methods, such as the LU, QR or Cholesky factorizations, do not work in this respect. For instance, the factorization $A = LU$ reveals the eigenvalues of L and U (both matrices being diagonal) but then there is no simple way to connect the eigenvalues of A to those. In fact, this must be the case since otherwise we would have a direct method for computing eigenvalues, which we have seen cannot be true in general. On the other hand:

- Any square matrix $A \in \mathbb{K}^{n\times n}$ admits a *Schur decomposition*: there exists $Q \in \mathbb{K}^{n\times n}$ unitary and $T \in \mathbb{K}^{n\times n}$ upper triangular such that
$$A = QTQ^{\text{H}} \tag{5.2}$$
The Schur decomposition (5.2) always exists, and may be non-unique. It is a similarity transformation of A, implying that $P_A(\lambda) = P_T(\lambda)$. Since T is triangular, its diagonal holds the eigenvalues of A.
- Even when A is real, Q and T may be complex. This is to be expected, since a real matrix may have complex eigenvalues.
- If A is Hermitian, its Schur factor T is in fact real and diagonal.

We can then consider the idea of computing matrix eigenvalues by finding a Schur decomposition of A, and this approach indeed leads to a general algorithm. As with earlier factorizations underlying direct solution methods for linear systems, finding a decomposition (5.2) basically consists in introducing zeros in the off-diagonal lower-triangular region of A (which was already the guiding idea behind the LU, QR and Cholesky factorizations). The following considerations must be taken into consideration in order to define a workable approach:

(a) Assume a transformation (such as a Householder reflector, see Sec. 3.1) is applied on the left for zeroing out (say) the whole first column under the diagonal; this can be encoded by a left multiplication by a suitable unitary matrix $F(u)$, see (3.1). However, achieving a similarity form then requires to also apply $F(u)^{\text{H}}$ from the right. It is easy to verify that the latter operation will reintroduce nonzero entries in the initially zeroed-out column. For example, let $F(u_1)$ be designed like in (3.2), so that $F(u_1)A$ now has the $A_{2:n,1}$ entries replaced with zeros. However, if we now compute $(F(u_1)A)F(u_1)^{\text{H}}$, we find that in general the entries located at $A_{2:n,1}$ become non-zero again.

In other words, applying (conjugate-transposed) Householder reflectors from the left *and* the right does *not* produce the upper-triangular factor sought for the Schur decomposition.

(b) By contrast, if we design Householder reflectors $H(u)$ that do *not* zero out the entry immediately below the diagonal, the symmetric right-multiplication by $H(u)^{\text{H}}$ *preserves* the introduced zeros.

(c) Obtaining a pure Schur decomposition (5.2) may entail using complex arithmetic even for a real original matrix A (since T is in general expected to be complex), whereas we would prefer working with real arithmetic for $A \in \mathbb{R}^{n\times n}$.

5.3.1 Orthogonal iterations. A natural idea consists in applying power iterations for finding the full spectrum of a matrix, or a significant portion of it. For instance, assume that the eigenvalues of A verify $|\lambda_1| > |\lambda_2| > \dots |\lambda_n|$. Letting $X \in \mathbb{K}^{n\times p} = [x_1, \dots, x_p]$ denote a set of p linearly independent vectors, we apply power iterations on X:
$$X^{(0)} = X, \quad X^{(k)} = AX^{(k-1)}$$
A straightforward extension of (5.1) and its implications shows that the sequence of subspaces $E_p^{(k)} := \text{span}\big(x_1, \dots, x_p\big)$ converges to the subspace $E_p := \text{span}\big(q_1, \dots, q_p\big)$ spanned by the p eigenvectors associated with the p largest eigenvalues $\lambda_1, \dots, \lambda_p$. The first p eigenvalues and eigenvectors of A can then conceivably be found from computing a finite number k of power iterations and diagonalizing the smaller matrix $A_p^{(k)} := (X^{(k)})^{\text{H}} A X^{(k)}$ obtained by projecting A on $X^{(k)}$. However, each iterated vector $Ax_i^{(k)}$ converges (upon normalization) to q_1 (as it undergoes a power iteration in the sense of Algorithm 5.1), which makes the set of vectors $X^{(k)}$ increasingly collinear, and the projected matrix $A_p^{(k)}$ increasingly ill-conditioned.

Like with the sequences of Krylov vectors underlying GMRES, where a similar issue arises, (see Section 4.5), the remedy consists in frequent orthogonalization. This suggests the following version

of power iterations, called orthogonal iterations, which orthogonalize the set of p vectors at each iteration: successive iterates $X_p^{(k)}$ are sets of p orthonormal vectors satisfying

$$X_p^{(k)} R^{(k)} = A X_p^{(k-1)} \tag{5.3}$$

with $R^{(k)}$ upper triangular. Each iteration evaluates $AX_p^{(k-1)}$, then computes its reduced QR decomposition (see Section 3.1), ensuring that the p columns of each iterate $X_p^{(k)}$ are orthonormal vectors. The resulting orthogonal iteration algorithm is very simple:

Algorithm 5.4 Orthogonal iterations

1: $A \in \mathbb{K}^{n\times n}$ Hermitian, $X_p^{(0)} = [x_1^{(0)}, \dots, x_p^{(0)}] \in \mathbb{K}^{n\times p}$ with orthonormal columns (initialization)
2: **for** $k = 1, 2, \dots$ **do**
3: $\quad Z^{(k)} = AX_p^{(k-1)}$ (apply A)
4: $\quad X_p^{(k)} R_p^{(k)} = Z^{(k)}$ (compute reduced QR decomposition of $Z^{(k)} \in \mathbb{K}^{n\times p}$)
5: $\quad$ **Stop** if convergence, set $\lambda_i = (x_i^{(k)})^{\mathrm{H}} A q_i^{(k)}$, $q_i = x_i^{(k)}$
6: **end for**

5.3.2 How and why orthogonal iterations work. To explain how and why orthogonal iterations find eigenvalues and eigenvectors of A, we focus on their action on a set of $p = 2$ vectors. In this case, $X_p^{(k)} \in \mathbb{K}^{n\times 2}$ and $R_p^{(k)}$ is a 2×2 matrix of the form $R_p^{(k)} = \begin{bmatrix} r_{11}^{(k)} & r_{12}^{(k)} \\ 0 & r_{22}^{(k)} \end{bmatrix}$, so that steps 3,4 of Algorithm 5.4 become

$$\text{(a)}\ \ r_{11}^{(k)} x_1^{(k)} = A x_1^{(k-1)}, \qquad \text{(b)}\ \ r_{12}^{(k)} x_1^{(k)} + r_{22}^{(k)} x_2^{(k)} = A x_2^{(k-1)}. \tag{5.4}$$

The equality (a) above shows that the sequence $x_1^{(k)}$ is generated by power iterations (Algorithm 5.1) applied to $x_1^{(0)}$, so that $x_1^{(k)}$ is expected to converge to the eigenvector q_1 associated with the eigenvalue λ_1 with largest modulus (possibly up to a sign change, in which case we redefine q_1 so that $x_1^{(k)} \to q_1$).

We now exploit this observation to work out the behavior of the sequence $x_2^{(k)}$. Since $x_1^{(k)} \to q_1$, we write $x_1^{(k)} = q_1 + \varepsilon_1^{(k)}$, with the eigenvector error verifying $\|\varepsilon_1^{(k)}\| \to 0$. Now we introduce the Hermitian matrix $\widehat{A} \in \mathbb{K}^{n\times n}$ defined by

$$\widehat{A} = (I - q_1 q_1^{\mathrm{H}})^{\mathrm{H}} A (I - q_1 q_1^{\mathrm{H}}) = A - \lambda_1 q_1 q_1^{\mathrm{H}},$$

with the second equality obtained by expanding the formula and using $Aq_1 = \lambda_1 q_1$ and (A being Hermitian) $q_1^{\mathrm{H}} A = \lambda_1 q_1^{\mathrm{H}}$. By the orthonormality of the eigenvectors q_i, $\widehat{A}$ verifies $\widehat{A} q_1 = 0$ and $\widehat{A} q_i = A q_i$ ($i \geq 2$); in other words, $\widehat{A}$ has the same eigenvectors as A, associated with the same eigenvalues except q_1 whose eigenvalue is now 0 instead of λ_1. Applying $\widehat{A}$ to $x_2^{(k-1)}$, we then find

$$\widehat{A} x_2^{(k-1)} = A x_2^{(k-1)} - \lambda_1 (q_1^{\mathrm{H}} x_2^{(k-1)}) q_1 = r_{12}^{(k)} x_1^{(k)} + r_{22}^{(k)} x_2^{(k)} - \lambda_1 (q_1^{\mathrm{H}} x_2^{(k-1)}) q_1. \tag{5.5}$$

Moreover, taking the scalar product of (5.4b) by $x_1^{(k)}$ and using the orthonormality of $\left(x_1^{(k)}, x_2^{(k)}\right)$ allows to evaluate $r_{12}^{(k)}$ as

$$r_{12}^{(k)} = (x_1^{(k)})^{\mathrm{H}} A x_2^{(k-1)} = \left(q_1 + \varepsilon_1^{(k)}\right)^{\mathrm{H}} A x_2^{(k-1)} = \lambda_1 \left(q_1^{\mathrm{H}} x_2^{(k-1)}\right) + \left(\varepsilon_1^{(k)}\right)^{\mathrm{H}} A x_2^{(k-1)}$$

and then use this expression in (5.5), to obtain

$$\widehat{A} x_2^{(k-1)} = r_{22}^{(k)} x_2^{(k)} + \left(q_1^{\mathrm{H}} x_2^{(k-1)}\right) \varepsilon_1^{(k)} + \left(\left(\varepsilon_1^{(k)}\right)^{\mathrm{H}} A x_2^{(k-1)}\right) x_1^{(k)} = r_{22}^{(k)} x_2^{(k)} + O\left(\varepsilon_1^{(k)}\right)$$

(since $x_1^{(k)} - q_1 = \varepsilon_1^{(k)}$). The above equality shows that, as $x_1^{(k)}$ converges to q_1 (i.e. $\varepsilon_1^{(k)} \to 0$), the iterates $x_2^{(k)}$ are updated in a manner increasingly close to a power iteration for the matrix $\widehat{A}$, whose eigenvalue of largest modulus is λ_2 with eigenvector q_2. Concluding, orthogonal iterations applied to a set of $p = 2$ vectors produce as $k \to \infty$ the eigenvalues λ_1, λ_2 and eigenvectors q_1, q_2 provided $|\lambda_1| \neq |\lambda_2|$. This analysis can be recursively extended to any p, so that orthogonal iterations verify:

Algorithm 5.5 Basic QR iterations

1: $A \in \mathbb{K}^{n\times n}$ Hermitian, $T^{(0)} = A$ (initialization)
2: **for** $k = 1, 2, \ldots$ **do**
3: $\quad Q^{(k)}R^{(k)} = T^{(k-1)}$ (compute QR decomposition of $T^{(k-1)} \in \mathbb{K}^{n\times p}$)
4: $\quad T^{(k)} = R^{(k)}Q^{(k)}$ (update $T^{(k)}$)
5: $\quad$ **Stop** if convergence, $\mathrm{diag}(T^{(k)})$ contains the eigenvalues of A
6: **end for**

Theorem 5.3 *Let $A \in \mathbb{K}^{n\times n}$ be Hermitian, and assume that its p largest eigenvalues verify $|\lambda_1| > |\lambda_2| > \ldots |\lambda_p|$. Assume in addition that all leading submatrices $(Q_p^H X_p^{(0)})_{1:q,1:q}$ $(1 \le q \le p)$ of $Q_p^H X_p^{(0)} \in \mathbb{K}^{p\times p}$ are nonsingular. Let $X_p^{(k)} = [x_1^{(0)}, \ldots, x_p^{(0)}]$ be the set of orthonormal vectors produced after k iterations of Algorithm 5.4. Then:*

$$\|x_i^{(k)} \pm q_i\| = O(C^k)$$

where $C = \max_{1\le j\le p} |\lambda_{j+1}|/|\lambda_j| < 1$ and the $\pm$ signs indicate that a sign choice must be made for the estimate to hold.

5.3.3 QR iterations. We have just seen that, for given $p < n$, orthogonal iterations yield approximations of the p largest (in modulus) eigenvalues of A; in fact, $(X_p^{(k)})^{\mathrm{H}} A X_p^{(k)}$ converges to $\mathrm{diag}(\lambda_1, \ldots, \lambda_p)$. Moreover, the condition $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_p|$ is potentially restrictive. To adapt orthogonal iterations to the problem of computing all n eigenvalues of A from an arbitrary starting set $X^{(0)} \in \mathbb{K}^{n\times n}$ of n orthonormal vectors, we focus on the sequence of matrices $T^{(k)} := (X^{(k)})^{\mathrm{H}} A X^{(k)}$, which converges to the diagonal matrix with the eigenvalues of A. Specifically, we seek to deduce from (5.3) (with $p = n$) an iteration rule that determines $T^{(k)}$ in terms of $T^{(k-1)}$. We have

$$\begin{aligned} T^{(k-1)} &= (X^{(k-1)})^{\mathrm{H}} A X^{(k-1)} = (X^{(k-1)})^{\mathrm{H}} Z^{(k)} \\ &= (X^{(k-1)})^{\mathrm{H}} X^{(k)} R^{(k)} \\ T^{(k)} &= (X^{(k)})^{\mathrm{H}} A X^{(k)} = (X^{(k)})^{\mathrm{H}} A X^{(k-1)} (X^{(k-1)})^{\mathrm{H}} X^{(k)} = (X^{(k)})^{\mathrm{H}} X^{(k)} R^{(k)} (X^{(k-1)})^{\mathrm{H}} X^{(k)} \\ &= R^{(k)} (X^{(k-1)})^{\mathrm{H}} X^{(k)}, \end{aligned}$$

which can be reformulated as

$$\text{(a) } T^{(k-1)} = Q^{(k)} R^{(k)} \quad \text{and} \quad \text{(b) } T^{(k)} = R^{(k)} Q^{(k)}, \tag{5.6}$$

where the matrix $Q^{(k)} := (X^{(k-1)})^{\mathrm{H}} X^{(k)}$ is unitary. We notice that (a) in fact yields the QR decomposition of $T^{(k-1)}$. The relations (5.6) can then be interpreted as: (a) compute the QR decomposition of $T^{(k-1)}$, then (b) compute $T^{(k)}$ by using the QR factors in reverse order. This is the essence of the so-called QR algorithm for eigenvalue problems, which Algorithm 5.5 shows in its basic form.

Remark 1 (QR decomposition vs. QR iteration) *We emphasize in passing that the QR acronym now pertains to two different (but related) methods: (i) the QR decomposition of a matrix (introduced in Sec. 3.1), which is a direct factorization method, and (ii) the QR iteration method for solving matrix eigenvalue problems (introduced here), which is an iterative method.*

5.4 IMPROVED QR ALGORITHM. The QR iteration as formulated in Algorithm 5.5, while workable (and in particular backward stable by virtue of the recourse to orthogonalization and stable QR decompositions), lacks efficiency for a combination of reasons:

- Each QR factorization in (5.6a) requires $O(n^3)$ operations, and computing the eigenvalues with sufficient accuracy is expected to need $O(n)$ iterations; the overall computational effort could then be of order $O(n^4)$.
- Similarly to the orthogonal iteration of Algorithm 5.4 which the basic QR iteration extends, the rate of convergence of the eigenvalues computed by the latter depends on their distribution. Moreover, if two or more eigenvalue have the same magnitude, convergence may fail.

Two major improvements to the QR iteration algorithm allow to adresss the above issues:

- The $O(n^4)$ computing time bottleneck is addressed by first putting A in *tridiagonal form*; as we will see, this step has in general a $O(n^3)$ computational cost. Then, each QR decomposition of a tridiagonal matrix requires $O(n^2)$ operations. This results in a $O(n^3)$ overall computational effort, assuming that at most $O(n)$ QR iterations are needed.
- As with the improved versions of power iterations for a single eigenvalue, convergence is accelerated by using a shifted version of the QR algorithm, even in cases where eigenvalue moduli are not well-separated.

5.4.1 Reduction to tridiagonal form by Lanczos iterations. We have seen in Sec. 4.5 that any matrix $A \in \mathbb{K}^{n\times n}$ can be reduced to upper Hessenberg form $A = QHQ^{\mathrm{H}}$ (with $Q \in \mathbb{K}^{n\times n}$ unitary) by means of the Arnoldi iteration. When A is in addition Hermitian, $H = Q^{\mathrm{H}}AQ$ is also Hermitian, so must be tridiagonal (being both Hermitian and upper Hessenberg). For notational convenience, we set H in the form

$$H = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \dots & 0 \\ \beta_1 & \alpha_2 & & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & & \ddots & \beta_{n-1} \\ 0 & \dots & 0 & \beta_{n-1} & \alpha_n \end{bmatrix}$$

accounting for its expected structure, $\alpha_1, \dots, \alpha_n$ and $\beta_1, \dots, \beta_{n-1}$ being the independent coefficients.

The reduction to tridiagonal form of any Hermitian matrix A can therefore be obtained by applying all n Arnoldi steps to A. In this context, the Arnoldi iteration takes a simpler form, as the expected tridiagonal character of H reduces the running Arnoldi step to a relation between three consecutive orthogonal vectors. The Arnoldi iteration specialized to Hermitian matrices is a very important algorithm in its own right, having many applications in numerical linear algebra; it is known as a *Lanczos iteration*. The vectors q_i and coefficients α_i, β_i are found recursively by the following appropriately-modified Arnoldi iterations (where iteration k aims at verifying the equality $A = QHQ^{\mathrm{H}}$ applied to the k-th column vector q_k of Q):

- Initialization: choose an arbitrary vector $b \in \mathbb{K}^n$, set $q_1 = b/\|b\|$.
- First iteration: seek $q_2 \in \mathbb{K}^n$ and α_1, β_1 satisfying $Aq_1 = QHQ^{\mathrm{H}}q_1$, i.e.:

  $$Aq_1 = \alpha_1 q_1 + \beta_1 q_2, \qquad \text{with (a) } q_1^{\mathrm{H}} q_2 = 0, \text{ (b) } \|q_2\| = 1$$

  Condition (a) gives $\alpha_1 = q_1^{\mathrm{H}} A q_1$, then condition (b) yields $\beta_1 = \|Aq_1 - \alpha_1 q_1\|$. The resulting vector $q_2 = (Aq_1 - \alpha_1 q_1))/\beta_1$ satisfies (a) and (b).
- Running (k-th) iteration: seek q_{k+1} and α_k, β_k such that $Aq_k = QHQ^{\mathrm{H}}q_k$, i.e.:

  $$Aq_k = \beta_{k-1} q_{k-1} + \alpha_k q_k + \beta_k q_{k+1},$$

  with (a) $q_j^{\mathrm{H}} q_{k+1} = 0$ $(1 \le j \le k)$ and (b) $\|q_{k+1}\| = 1$ (β_{k-1} being known from iteration $k-1$).
- Last (n-th) iteration: find α_n such that $Aq_n = QHQ^{\mathrm{H}}q_n$, which yields

  $$Aq_n = \beta_{n-1} q_{n-1} + \alpha_n q_n.$$

Each Lanczos iteration entails one matrix-vector product (costing $O(n^2)$ operations), so that reducing A to tridiagonal form needs $O(n^3)$ operations overall. The Lanczos iteration in pseudocode form reads:

The following remarks show the usefulness of a preliminary reduction of A to tridiagonal form (by means of the Lanczos iteration) before applying QR iterations:

- Applying the Lanczos iteration has a fixed computational cost for given matrix size n (Exercise 5.1), i.e. is a *direct* part of the overall-iterative eigenvalue / eigenvector computation method.

Algorithm 5.6 Lanczos iterations

1:	$A \in \mathbb{K}^{n\times n}$ and $b \in \mathbb{K}^n$	(problem data)
2:	$q_1 = b/\|b\|$, $h_{01} = 0$	(initialization)
3:	**for** $k = 1$ to $n-1$ **do**	
4:	$v = Aq_k$	(new Krylov vector: initialize q_{k+1})
5:	$\alpha_k = q_k^{\mathrm{H}} v$	(diagonal entry α_k of H)
6:	$v = v - \beta_{k-1}q_{k-1} - \alpha_k q_k$	(update (not yet normalized) vector q_{k+1})
7:	$\beta_k = \|v\|$	(off-diagonal entry β_k of H)
8:	$q_{k+1} = v/\beta_k$	(normalize q_{k+1})
9:	**end for**	
10:	$\alpha_n = q_n^{\mathrm{H}} A q_n$	(last diagonal entry α_n of H)

- Subsequent QR iterations preserve the symmetric tridiagonal nature of the iterates (i.e. if $T^{(k-1)}$ is symmetric tridiagonal, then so is $T^{(k)}$), see Exercise 5.2.
- The QR decomposition of a real tridiagonal matrix $T \in \mathbb{R}^{n\times n}$ requires only $O(n)$ operations, by allowing to apply 2×2 *Givens rotations* instead of Householder reflectors for zeroing-out the subdiagonal of $T^{(k)}$.

Exercise 5.1 *Show that the asymptotic computational cost of reducing an arbitrary Hermitian matrix $A \in \mathbb{K}^{n\times n}$ to tridiagonal form using the Lanczos iteration is $\frac{4}{3}n^3$.*

Exercise 5.2 *Let $T \in \mathbb{R}^{n\times n}$ be tridiagonal symmetric, and let $T = QR$ be the QR decomposition of T. Prove that $RQ \in \mathbb{R}^{n\times n}$ is tridiagonal symmetric. This property implies that the QR algorithm preserves the tridiagonal character of the iterates $T^{(k)}$.*

5.4.2 Shifted QR algorithm.

We saw in Section 5.2 that the introduction of shifts considerably improves the convergence rate of iterations based on inverse or Rayleigh quotient iterations. In fact, while based as we saw on (forward) power iterations, the QR algorithm can also be interpreted as performing inverse iterations, and this alternative interpretation explains the (dramatic) performance improvement brought by shifts.

Inverse-iteration interpretation of the QR algorithm. The generic orthogonal iteration at the root of the basic QR algorithm, summarized by (5.3) with $p = n$, implies

$$A = X^{(k)} R^{(k)} X^{(k-1)\mathrm{H}}.$$

If A is nonsingular, the above equality can be inverted, yielding

$$A^{-1} = X^{(k-1)}(R^{(k)})^{-1} X^{(k)\mathrm{H}} = X^{(k)}(R^{(k)})^{-\mathrm{H}} X^{(k-1)\mathrm{H}}, \tag{5.7}$$

with the second equality resulting from the (Hermitian) symmetry of A. Now, let $P \in \mathbb{R}^{n\times n}$ denote the "flipped identity" matrix, i.e. the permutation matrix that reverses the order of the rows or columns of a given matrix:

$$P = \begin{bmatrix} 0 & \cdots & & 1 \\ \vdots & & \iddots & \\ & \iddots & & \vdots \\ 1 & & \cdots & 0 \end{bmatrix}$$

(so that, for example, $X = [x_1, \ldots, x_n]$ gives $PX = [x_n, \ldots, x_1]$). Since $P^2 = I$, we can rewrite (5.7) as

$$A^{-1} = \left(X^{(k)}P\right)\left(P(R^{(k)})^{-\mathrm{H}}P\right)\left(X^{(k-1)}P\right)^{\mathrm{H}},$$

and observe that (i) $X^{(k-1)}P$ and $X^{(k)}P$ are unitary matrices, while (ii) $P(R^{(k)})^{-\mathrm{H}}P$ is upper triangular[2]. Comparing the above equality to (5.3), the orthogonal iteration is seen as being equivalent to an orthogonal iteration for A^{-1} applied to the column-flipped matrix $X^{(k-1)}P = [x_n^{(k-i)}, \ldots, x_1^{(k-i)}]$. In particular, the first column of $X^{(k-1)}P$, i.e. $x_n^{(k-1)}$, undergoes pure power iterations under A^{-1}, i.e. inverse iterations (see Sec. 5.2.3) without shift.

Shifted QR algorithm. The inverse-iteration interpretation of the orthogonal iteration (and hence also of the basic QR algorithm) makes it natural to modify the generic QR iteration by introducing a shift $\mu^{(k)}$, so that its main steps follow

$$\text{(a) } T^{(k-1)} - \mu^{(k)} I = Q^{(k)} R^{(k)} \quad \text{and} \quad \text{(b) } T^{(k)} = R^{(k)} Q^{(k)} + \mu^{(k)} I,$$

instead of (5.6). As with Rayleigh iterations, $\mu^{(k)}$ could be set at each iteration to the value of a Rayleigh quotient. The easiest choice is to set $\mu^{(k)} := t_{nn}^{(k-1)}$ since this is the Rayleigh quotient for $q_n^{(k-1)}$, but this is known to fail on some "nice" matrices (for reasons quite similar as those making the non-pivoted LU factorization fail on certain well-behaved matrices, see Sec. 2.2). A better alternative (for Hermitian matrices) is to use the *Wilkinson shift*, whereby the eigenvalues of the 2×2 bottom rightmost block of $T^{(k-1)}$ are computed and $\mu^{(k)}$ is set to the one closest to $t_{nn}^{(k-1)}$.

The practical (improved) QR algorithm, featuring both the preliminary reduction of A to tridiagonal form and shifts, is shown in pseudocode form in Algorithm 5.7.

Algorithm 5.7 Shifted QR iterations

1: $A \in \mathbb{K}^{n \times n}$ Hermitian (data)
2: $(Q^{(0)})^{\mathrm{H}} T^{(0)} Q^{(0)} = A$ (Tridiagonalization of A, see Algorithm 5.6)
3: **for** $k = 1, 2, \ldots$ **do**
4: Choose $\mu^{(k)}$ (shift value, e.g. use the *Wilkinson shift*)
5: $Q^{(k)} R^{(k)} = T^{(k-1)} - \mu^{(k)} I$ (compute QR decomposition of $T^{(k-1)} - \mu^{(k)} I \in \mathbb{K}^{n \times p}$)
6: $T^{(k)} = R^{(k)} Q^{(k)} - \mu^{(k)} I$ (update $T^{(k)}$)
7: If any off-diagonal entry $t_{j,j+1}^{(k)}$ is sufficiently small,
set $t_{j,j+1}^{(k)} = t_{j+1,j}^{(k)} = 0$ to obtain $T^{(k)} = \begin{bmatrix} T_1^{(k)} & 0 \\ 0 & T_2^{(k)} \end{bmatrix}$.
From now, apply the QR algorithm separately to $T_1^{(k)}$ and $T_2^{(k)}$
8: **Stop** if convergence, $\mathrm{diag}(T^{(k)})$ contains the eigenvalues of A
9: **end for**

Step 7 in Algorithm 5.7 is called a *deflation*: when the tridiagonal matrix $T^{(k)}$ has a small enough off-diagonal entry, setting that entry to zero splits $T^{(k)}$ into two disconnected tridiagonal parts, to which subsequent QR iterations can be applied separately.

5.5 EXTENSIONS.

5.5.1 Generalized symmetric eigenvalue problems.

Many engineering applications involve generalized eigenvalue problems of the form

$$\text{Find } \lambda, x \text{ such that} \quad Kx = \lambda M x. \tag{5.8}$$

In particular, the study of vibrations and forced dynamical motions of mechanical structures and systems involves the eigenvalues and eigenvectors of problems of the form (5.8), where $K \in \mathbb{R}^{n \times n}$ and M are the (real, symmetric, positive) stiffness and mass matrices of the structure (see Sec. 1.2.2). More-general versions of problem (5.8) with unsymmetric complex matrices K appear when damping

[2] Like $(R^{(k)})^{-1}$, $(R^{(k)})^{-1}$ is upper triangular, and hence $(R^{(k)})^{-\mathrm{H}}$ is lower triangular. Applying P on the left then on the right flips the triangle horizontally then vertically, resulting in $P(R^{(k)})^{-\mathrm{H}}P$ being upper triangular.

is taken into account. The book [17] provides a very comprehensive exposition on the modeling and computation of mechanical vibrations and structural dynamics.

Here we briefly examine the common (and simplest) case of conservative (undamped) vibrations of structures with linearly elastic structures, where the stiffness matrix A is real symmetric positive (but has zero eigenvalues unless the boundary conditions prevent rigid body motions) and the mass matrix is real symmetric positive definite. The latter property allows to set the Cholesky decomposition $M = GG^{\mathrm{T}}$ of M, where the Cholesky factor G is real, lower-triangular and invertible, so that we have

$$Kx - \lambda Mx = G\big(A - \lambda I\big)G^{\mathrm{T}}x, \qquad A = G^{-1}KG^{-\mathrm{T}}$$

and the generalized eigenvalue problem (5.8) can be recast as the equivalent symmetric eigenvalue problem

$$\text{Find } \lambda, x \text{ such that} \quad Ay = \lambda y, \quad G^{\mathrm{T}}x = y. \tag{5.9}$$

The methods presented in this chapter can then in principle be applied to problem (5.9). In practice, the matrix $A = G^{-1}KG^{-\mathrm{T}}$ is not evaluated, and the algorithms for standard eigenvalue problems are instead adapted to the specific form of problem (5.8), while the eigenvectors are often normalized according to $x^{\mathrm{T}}Mx = 1$. For example, the inverse iteration algorithm with a given shift μ takes the form shown in Algorithm 5.8.

Algorithm 5.8 Inverse iteration for structural vibrations

$K, M \in \mathbb{R}^{n\times n}$ (stiffness and mass matrices), $x^{(0)} \in \mathbb{K}^n$ with $\|x^{(0)}\| = 1$ (initialization), $\mu \in \mathbb{R}$ close to λ_j
for $k = 1, 2, \ldots$ **do**
  solve $(K - \mu M)x^{(k)} = Mx^{(k-1)}$ (solve forced vibration problem with $Mx^{(k-1)}$ as load)
  $x^{(k)} = x^{(k)}/\sqrt{x^{(k)\mathrm{T}}Mx^{(k)}}$ (next mass-normalized iterate)
  $\lambda_j^{(k)} = (x^{(k)})^{\mathrm{H}}Kx^{(k)}$ (Rayleigh quotient)
  Stop if convergence, set $\lambda_j = \lambda_j^{(k)}$, $q = x^{(k)}$
end for

Due to the fact that FEM models of mechanical structures may be very large in analyses performed by e.g. aerospace, automotive, energy or civil engineering industries, a common computational task is to require the fraction of eigenvalues and eigenfunctions corresponding to the lower part of the frequency sprctrum of the system (which on a given FEM mesh are the ones most accurately evaluated). Eigenfunctions are afterwards often used to defined reduced bases for other dynamical analyses.

5.5.2 Unsymmetric eigenvalue problems. We now consider the case where A is no longer Hermitian. In this case, a finite sequence of suitable unitary transformations (e.g. based on Householder reflectors) applied symmetrically (from the left and the right) allow to obtain a decomposition of the form

$$A = QHQ^{\mathrm{H}}$$

where Q is unitary and H is a *upper Hessenberg* matrix, i.e. a matrix of the form

$$H = \begin{bmatrix} \times & \times & \times & \dots & \times \\ \times & \times & & & \times \\ 0 & \times & \times & & \times \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & \times & \times \end{bmatrix}$$

whose entries below the first subdiagonal are zeros. Importantly, $H \in \mathbb{R}^{n\times n}$ and $Q \in \mathbb{R}^{n\times n}$ if $A \in \mathbb{R}^{n\times n}$.

At this point, the eigenvalues of A and H coincide, and the QR algorithm can be applied to H. In practice, shifted QR iterations may be applied as described by Algorithm 5.7, with step 2 performing a reduction of A to Hessenberg form instead of a tridiagonalization.

- If A is complex, the matrix sequence $T^{(k)}$ in Algorithm 5.7 converges to an upper triangular matrix, whose diagonal holds the eigenvalues of A.
- If A is real and the QR algorithm is performed in real arithmetic, the matrix sequence $T^{(k)}$ in Algorithm 5.7 converges to an "almost upper triangular" matrix with 2×2 real-valued diagonal blocks yielding (conjugated) complex eigenvalues.

CHAPTER 6 ILL-CONDITIONED PROBLEMS, LOW-RANK APPROXIMATIONS

6.1 INTRODUCTION. Some areas of applications give rise to linear systems of equations with "unpleasant" properties, involving matrices which have, strictly speaking, full column rank while in practice the very fast decay of their singular values makes them rank-deficient in a numerical sense. As already indicated in Chapter 1, such situations occur for example in connection with

- Inverse and identification problems (where "hidden" physical properties are to be inferred from indirect measurements)
- Image processing and image restoration
- Data analysis

For example, backward heat conduction problems (see Section 1.2.5) give rise after discretization to matrix equations whose singular values decay extremely fast, in fact faster-than-exponentially. To illustrate this remark, the singular values of the matrix $A(T) \in \mathbb{R}^{202\times 101}$ associated with the initial temperature identification example shown in Fig. 1.5 are plotted in Fig. 6.1. In fact, we have $\sigma_1 0 \approx 5.8\,10^{-16}\sigma_1$ and the singular values σ_i for i larger than about 10 cease to be significant due to round-off errors (hence the plateau in the line plot).

In such cases, we typically have to extract information from a (possibly large and often non-square) set $Ax = b$ of linear equations whose matrix $A \in \mathbb{K}^{m\times n}$ is ill-conditioned while the right-hand side $b \in \mathbb{K}^m$ may be outside the range of A. Approximate solutions to $Ax = b$ are then often sought by treating the linear system in a least squares sense. Moreover, two dual considerations typically arise:

- The data b is imperfect, for example due to measurement errors, and x (approximately) satisfying $Ax = b$ is highly sensitive to data perturbations: if x_1 and x_2 are respective "best" solutions for data b_1 and b_2, we have
$$\|x_1 - x_2\| \gg \|b_1 - b_2\|, \tag{6.1}$$

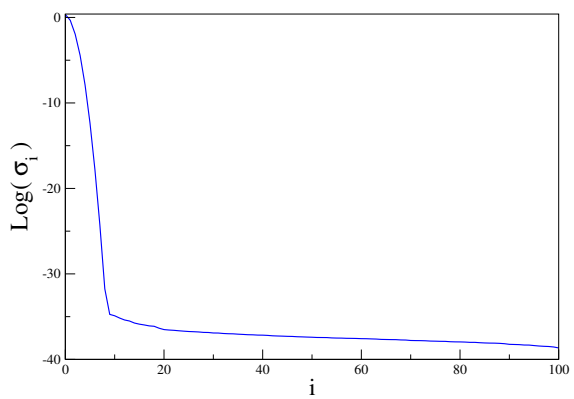


Figure 6.1: Singular values of the matrix $A(T) \in \mathbb{R}^{202\times 101}$ associated with the initial temperature identification example shown in Fig. 1.5. Values σ_i for i larger than about 10 cease to be significant due to round-off errors.

i.e. the solution x does not, in practical terms, depend continuously on the data b. Regularization approaches allow to restore continuity to some extent.

- The matrix A is close to a rank-deficient matrix: there exists a matrix $A_r \in \mathbb{K}^{m\times n}$ such that $r := \mathrm{rank}(A_r) \ll n$ while $\|A - A_r\| \ll \|A\|$. The matrix A_r therefore contains "almost the same" information as A but requires a storage $O(r/n)$ times that of A, i.e. is amenable to substantial compression (relative to A) by means of *low-rank approximations*.

The concept of singular value decomposition (SVD), introduced in Section 3.4, plays a major role in quantifying and analyzing the above types of deficiencies and their computational remedies.

6.1.1 SVD and truncated SVD. We will assume throughout this chapter that $A \in \mathbb{K}^{m\times n}$ with $m \geq n$ (i.e. systems of equations that are either square or potentially overdetermined), which is by far the most frequent practical case. We recall (see Theorem 3.3) that A always admits the singular value decomposition (SVD)

$$A = USV^{\mathrm{H}}, \qquad \begin{cases} U = [u_n, \ldots, u_m] \in \mathbb{K}^{m\times m}, \quad V = [v_1, \ldots, v_n] \in \mathbb{K}^{n\times n}, \\ U^{\mathrm{H}}U = UU^{\mathrm{H}} = I_m, \quad V^{\mathrm{H}}V = VV^{\mathrm{H}} = I_n \end{cases} \tag{6.2}$$

where $S \in \mathbb{R}^{m\times n}$ contains the n *singular values* $\sigma_1 \geq \sigma_2 \ldots \geq \sigma_n \geq 0$ of A on its main diagonal[1] and is otherwise zero. To focus the forthcoming discussion on the *numerical* (as opposed to exact) rank deficiency, A is assumed to have full column rank (with the expectation that many of its singular values σ_i are such that $|\sigma_i| \ll |\sigma_1|$). Of course, there are matrices that are algebraically rank deficient while also suffering from numerical rank deficiency.

6.1.2 Sensitivity of least squares solutions to errors in the data. First, let us assume that $Ax = b$ is to be solved as an ordinary least squares problem

$$\min_{x\in\mathbb{K}^n} \|Ax - b\|^2, \tag{6.3}$$

whose general solution is given by (3.9). Since A is assumed to have full column rank, problem (6.3) has a unique solution, given by

$$x = \sum_{i=1}^{n} \frac{z_i}{\sigma_i} v_i \tag{6.4}$$

(with $z := U^{\mathrm{H}}b$, as in (3.8)). Now, let us consider the case where the available data is *noisy* (e.g. due to measurement errors): instead of the exact data $b \in \mathbb{K}^m$, we have a perturbed version $b_\delta = b + w$ with $\|b - b_\delta\|_2 = \|w\|_2 = \delta$ (i.e. δ is the size of the data error $w \in \mathbb{K}^m$). Using (6.4) and x being linear in b, the perturbed solution x_δ, the solution error $e_\delta := x_\delta - x$ and the solution sensitivity to dana noise are readily found to be given by

$$x_\delta = \sum_{i=1}^{n} \frac{b_\delta^{\mathrm{H}} u_i}{\sigma_i} v_i, \qquad e_\delta = \sum_{i=1}^{n} \frac{w^{\mathrm{H}} u_i}{\sigma_i} v_i, \qquad \frac{\|e_\delta\|_2}{\delta} = \frac{1}{\delta}\Big(\sum_{i=1}^{n} \frac{|w^{\mathrm{H}} u_i|^2}{\sigma_i^2}\Big)^{1/2}.$$

If A is numerically rank deficient (or simply if some of the singular values of A are nonzero but very small relative to σ_1), we may have situations such that

$$\frac{w^{\mathrm{H}} u_1}{\sigma_1} \text{ is small, while } \frac{w^{\mathrm{H}} u_i}{\sigma_i} \text{ is large for some } i,$$

i.e. the projection of the data error on singular vectors u_i associated with very small singular values σ_i induces an error magnification on the solution; recall for example the extremely fast singular value decay for the inverse heat conduction problem (Fig. 6.1). This situation makes the solution sensitivity $\|e_\delta\|_2/\delta$ potentially large even for small data errors δ, i.e we are in the unpleasant situation (6.1). It constitutes a typical difficulty when solving e.g. inverse problems or deconvolution problems for image processing. Very ill-conditioned matrices may have singular values that are exponentially decaying w.r.t. their index, so that $|w^{\mathrm{H}} u_i|/\sigma_i$ may be very large even for small data errors δ.

[1] The assumption $m \geq n$ implies that A has $\min(m, n) = n$ singular values, counting multiplicities.

6.1.3 Low-rank approximations. Let $A \in \mathbb{K}^{m\times n}$ be a given matrix, for which no special property or requirements are assumed (i.e. A may be picked arbitrarily). It is often useful to replace A by a *low-rank approximation* A_r with given column rank $r < n$, where "approximation" means that $\|A - A_r\|$ is deemed small enough in the sense of some matrix norm. To this end, it is natural to introduce the notion of truncated singular value decomposition, upon which low-rank approximations can conveivably be defined:

Definition 6.1 (truncated singular value decomposition) *Let $A \in \mathbb{K}^{m\times n}$ be an arbitrary matrix. Let U, S, V be the elements of the SVD of A as introduced in* (6.2)*. For any given $r \leq n$, the leading rank-r truncated singular value decomposition (TSVD) $\widehat{A}_r$ of A is then defined by*

$$\widehat{A}_r := U_r S_r V_r = \sum_{i=1}^{r} \sigma_i u_i v_i^H,$$

where $U_r = [u_1, \ldots, u_r] \in \mathbb{K}^{m\times r}$, $V_r = [v_1, \ldots, v_r] \in \mathbb{K}^{n\times r}$ and $S_r = diag(\sigma_1, \ldots, \sigma_r) \in \mathbb{R}^{r\times r}$ are defined in terms of the SVD elements U, S, V. In other words, the rank-r TSVD of A is found from the SVD of A given by (6.2) *by setting $\sigma_{r+1} = \ldots = \sigma_n = 0$.*

Indeed, the following classical result states that the best rank-r approximation of A in the sense of either the (spectral) 2-norm or the Frobenius norm (respectively defined by (1.9) with $p = 2$ and (1.10)) is given by the truncated singular value decomposition (TSVD) of A:

Theorem 6.1 (Eckart-Young-Mirsky [16]) *Let $A \in \mathbb{K}^{m\times n}$ be an arbitrary matrix. Let $r \leq n$. The rank-r TSVD $\widehat{A}_r$ of A provides the best rank-r approximation of A for the spectral and Frobenius norms:*

$$\widehat{A}_r = \underset{B\in\mathbb{K}^{m\times n},\, rank(B)=r}{\arg\min} \|A - B\|_2 = \underset{B\in\mathbb{K}^{m\times n},\, rank(B)=r}{\arg\min} \|A - B\|_F.$$

Moreover, the respective truncation errors verify

$$\|A - \widehat{A}_r\|_2 \leq \sigma_{r+1}, \qquad \|A - \widehat{A}_r\|_F^2 \leq \sum_{i=r+1}^{n} \sigma_i^2. \tag{6.5}$$

The above result immediately elicits a few simple remarks:

- The estimates (6.6) translate into corresponding estimates for the relative truncation errors:

$$\frac{\|A - \widehat{A}_r\|_2}{\|A\|_2} \leq \frac{\sigma_{r+1}}{\sigma_1}, \qquad \frac{\|A - \widehat{A}_r\|_F}{\|A\|_F} \leq \frac{\sqrt{\sum_{i=r+1}^{n} \sigma_i^2}}{\sqrt{\sum_{i=1}^{n} \sigma_i^2}}.$$

- They show that knowing the singular values of A allows to determine the smallest rank r such that $\widehat{A}_r$ approximates A within a chosen tolerance.
- On the other hand, the actual computation of $\widehat{A}_r$ for given A is potentially expensive, as it entails evaluating the SVD of A, at a $O(m^2 n)$ cost.

6.2 REGULARIZED LEAST SQUARES. A classical approach for regularizing ill-conditioned linear systems of equations (which often result from the discretization of ill-posed linear equations involving unknown functions [35, 8]) consists in introducing a modified least squares problem with an additional term that expresses some additional prior information on the unknown. In the simplest form of this approach, the modified minimization problem takes the form

$$\min_{x\in\mathbb{K}^n} J_\alpha(x), \qquad J_\alpha(x) := \|Ax - b\|_2^2 + \alpha\|x\|_2^2, \tag{6.6}$$

where $\alpha > 0$ is the *regularization parameter*. The heuristic idea behind introducing the above J_α is to "augment" the least-squares minimization of the residual $b - Ax$ with an additional term $\alpha\|x\|_2^2$, with a small weight α, that "penalizes" potential solutions x whose norm is large. In other words, supplementary *prior information* is brought to the problem by deciding that solutions with smaller norms to be are preferred. This idea, which goes back to the seminal works [27, 34], is often known as (the simplest form of) a *Tikhonov(-Phillips) regularization*.

The simplest way to investigate the properties and solvability of problem (6.7) relies, once more, on the SVD of A. We adapt (3.10) to the objective function J_α by the simple expedient of noticing that $\|x\|_2^2 = \|V^{\mathrm{H}}x\|_2^2$, to find

$$J_\alpha(x) = \sum_{i=1}^{n} \Big\{ |\sigma_i y_i - z_i|^2 + \alpha |y_i|^2 \Big\} + \sum_{i=n+1}^{m} |z_i|^2$$

(with $y := V^{\mathrm{H}}x$ and $z := U^{\mathrm{H}}b$, as in (3.8)). The minimization problem (6.7) thus uncouples into n univariate quadratic minimization problems (one for each projection y_i of the initial unknown x), and its solution x_α is easily found to be given by

$$y_i = \frac{\sigma_i z_i}{\sigma_i^2 + \alpha} \qquad \text{and hence} \quad x_\alpha = \sum_{i=1}^{n} \frac{\sigma_i z_i}{\sigma_i^2 + \alpha} v_i \tag{6.7}$$

The regularized least squares solution x_α has important characteristics:

- Irrespective of the column rank of A, x_α is unique for any $\alpha > 0$ (whereas the pure least squares solution $x = x_0$ is not unique when A is algebraically rank deficient, see (3.9));
- For $\alpha > 0$, x_α does not minimize $\|Ax - b\|_2^2$ (as explained in more detail in Sec. 6.2.1);
- The basic least-squares solution x given by (6.4) is the limit of x_α as $\alpha \to 0$.

An alternative approach based on adapting the normal equation formulation yields

$$x_\alpha = \big(A^{\mathrm{H}}A + \alpha I \big)^{-1} b, \tag{6.8}$$

where the matrix $A^{\mathrm{H}}A + \alpha I$ is easily seen to be SPD (and hence in particular invertible), and the above comments can also be shown to be true on the basis of (6.9).

6.2.1 Application to noisy data. Solving $Ax = b$ by means of formulation (6.7) instead of (6.3) is useful when the available data b_δ is noisy (if only due to finite-precision arithmetic in the case of a very ill-conditioned matrix A). The solution $x_{\alpha,\delta}$ of problem (6.7) with data b_δ is given by (6.8) with b replaced with $b_\delta = b + w$, i.e.:

$$x_\alpha = \sum_{i=1}^{n} \frac{\sigma_i (b_\delta^{\mathrm{H}} u_i)}{\sigma_i^2 + \alpha} v_i = x_\alpha + \sum_{i=1}^{n} \frac{\sigma_i (w^{\mathrm{H}} u_i)}{\sigma_i^2 + \alpha} v_i$$

so that the solution error $e_{\alpha,\delta}$ (relative to the pure least squares solution) is

$$e_{\alpha,\delta} := x_{\alpha,\delta} - x = e_{\alpha,\delta}^{\mathrm{reg}} - e_{\alpha,\delta}^{\mathrm{noise}} \tag{6.9}$$

where $e_{\alpha,\delta}^{\mathrm{reg}}$ and $e_{\alpha,\delta}^{\mathrm{noise}}$, given by

$$e_{\alpha,\delta}^{\mathrm{reg}} = \sum_{i=1}^{n} \Big(\frac{\sigma_i}{\sigma_i^2 + \alpha} - \frac{1}{\sigma_i} \Big) z_i v_i = -\alpha \sum_{i=1}^{n} \frac{z_i}{\sigma_i(\sigma_i^2 + \alpha)} v_i, \qquad e_{\alpha,\delta}^{\mathrm{noise}} = \sum_{i=1}^{n} \frac{\sigma_i (w^{\mathrm{H}} u_i)}{\sigma_i^2 + \alpha} v_i$$

are the contributions of regularization and data noise to the solution error (since $e_{\alpha,\delta}^{\mathrm{reg}} = 0$ if $\alpha = 0$ and $e_{\alpha,\delta}^{\mathrm{noise}} = 0$ if $w = 0$).

A natural question is how to optimally choose the regularization parameter α. Formulas (6.10) hint at a complex interplay between α and the data noise w. Much insight is gained into this question by examining how each term of the optimal value $J_\alpha(x_\alpha)$ of J_α depends on α. We then write

$$J_\alpha(x_\alpha) = D(\alpha) + \alpha R(\alpha), \qquad D(\alpha) := \|Ax_\alpha - b\|_2^2, \quad R(\alpha) := \|x_\alpha\|_2^2,$$

where $D(\alpha)$ and $R(\alpha)$ are respectively the linear system residual evaluated at the minimizer x_α and the minimizer squared norm. Using (6.8), we easily obtain the following formulas for $D(\alpha)$ and $R(\alpha)$:

$$D(\alpha) = \sum_{i=1}^{n} \frac{\alpha^2}{(\sigma_i^2 + \alpha)^2} |z_i|^2 + \sum_{i=n+1}^{m} |z_i|^2, \qquad R(\alpha) = \sum_{i=1}^{n} \frac{\sigma_i^2}{(\sigma_i^2 + \alpha)^2} |z_i|^2 \tag{6.10}$$

(notice that the second sum in $D(\alpha)$ is a constant). In turn, the functions $\alpha \mapsto D(\alpha)$ and $\alpha \mapsto R(\alpha)$ given by the above formulas are easily differentiated, to obtain

$$D'(\alpha) = 2\alpha \sum_{i=1}^{n} \frac{\sigma_i^2}{(\sigma_i^2+\alpha)^2}|z_i|^2 > 0, \qquad R'(\alpha) = -2\sum_{i=1}^{n} \frac{\sigma_i^2}{(\sigma_i^2+\alpha)^3}|z_i|^2 < 0; \tag{6.11}$$

in other words, $\alpha \mapsto D(\alpha)$ is increasing while $\alpha \mapsto R(\alpha)$ is decreasing (and, in particular, the linear system residual achieved by x_α is larger than its minimum $\|Ax_0 - b\|$ for any $\alpha > 0$). Next, we can consider the parametric curve C defined in the (D, R) plane by $\alpha \in [0, \infty[\to \big(D(\alpha), R(\alpha)\big) \in \mathbb{R}^2$. The curve C, often called for reasons to be clarified soon the *L-curve* associated with the regularized least squares problem, has the following properties:

Lemma 6.1 (properties of the L-curve)
(a) The curve C is monotonically decreasing, i.e. R is a decreasing function of D;
(b) The curve C is convex;
(c) The extremities $A = \big(D(0), R(0)\big)$ and $A = \big(D(\infty), R(\infty)\big)$ of C are given by

$$D(0) = \|Ax - b\|_2^2, \quad R(0) = \|x\|_2^2, \qquad D(\infty) = \|b\|_2^2, \quad R(\infty) = 0$$

where $x = x_0$ is the basic least-squares solution.

Proof. (a) follows directly from $D'(\alpha) > 0$ and $R'(\alpha) < 0$, while formulas (c) are easily found from (6.11). Property (b) is equivalent (since D, R are smooth enough functions of α) to the curvature $\kappa(\alpha)$ being positive for any α. Classical differential geometry provides $\kappa(\alpha) = (D'R'' - D''R')/(D'^2 + R'^2)^{3/2}$. We find from (6.12) that $D'(\alpha) = -\alpha R'(\alpha)$ for any α, from which we deduce that $(D'R'' - D''R')(\alpha) > 0$. Hence $\kappa(\alpha) > 0$ for any α, i.e. C is a convex curve. The proof of the lemma is complete. □

Now, assume that we are using noisy data b_δ and that we know the data noise level δ (the latter assumption is realistic in some cases, such as mechanical testing experiments where displacement fields in the sample are monitored using digital image correlation). The regularized problem (6.7) is predicated on the assumption that solutions x whose norm is large are discouraged. In keeping with this viewpoint, Lemma 6.1 allows to consider the regularized problem (6.7) in the form

$$\min_{x \in \mathbb{K}^n} \|x\|_2^2, \qquad \text{subject to} \quad \|Ax - b\|_2^2 \leq \delta^2 \tag{6.12}$$

Indeed, let the value $\alpha(\delta)$ of α be selected so that the system (squared) residual $D(\alpha)$ be equal to the (squared) measurement noise δ^2, i.e. as the solution of

$$D(\alpha) = \delta^2, \quad \text{i.e.} \quad \|Ax_\alpha - b_\delta\|_2^2 = \delta^2 \tag{6.13}$$

In view of the monotonicity and convexity properties summarized in Lemma 6.1, the above equation has a unique solution $\alpha(\delta)$ provided $\delta < \|b_\delta\|_2$ (i.e. the noisy data must have a norm larger than that of the data noise, a most reasonable assumption indeed!). Moreover, we have

$$D(\alpha) < D(\alpha(\delta)), \quad R(\alpha) > R(\alpha(\delta)) \qquad \text{for any} \quad \alpha < \alpha(\delta),$$

so that x_α with $\alpha = \alpha(\delta)$ chosen according to (6.14) solves (6.13). The following remarks can be made on the regularized least squares solution method including the parameter choice rule (6.14):

- The selection rule (6.14) for α is known as Morozov's discrepancy principle; it basically consists in finding the parameter α that achieves a satisfactory compromise between minimizing the system residual and preventing the solution norm from becoming troo large. A very important feature of this regularization method, and in fact of many other regularization methods for ill-conditioned linear systems, is that the optimal value of α depends on the data noise. In fact, Fig. 6.2 makes it clear (again as a consequence of the convexity of the curve C) that $\delta \mapsto \alpha(\delta)$ is increasing: the larger the data noise, the larger α has to be chosen.

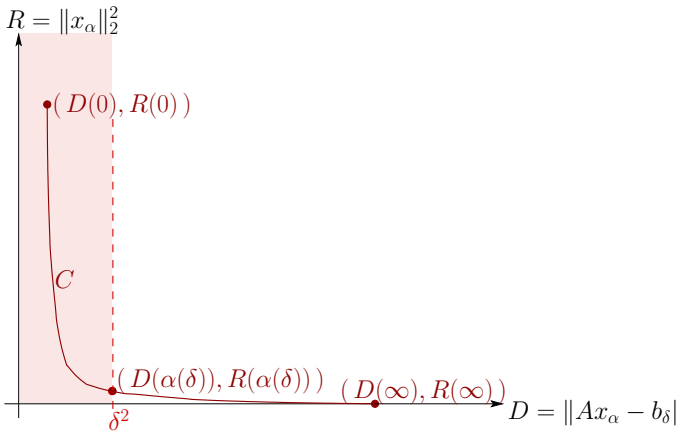


Figure 6.2: L-curve for the regularized least squares problem (6.7), and graphical illustration of the properties given in Lemma 6.1 and of the determination of the optimal regularization parameter $\alpha(\delta)$. The shaded band covers the part of C satisfying the inequality constraint $\|Ax - b\|_2^2 \leq \delta^2$ of (6.13).

- The regularized least squares approach can be adapted, with straightforward modifications, to other types of prior information where the term $\alpha\|x\|_2^2$ in problem (6.7) is replaced with $\alpha\|x–x^{\text{ref}}\|_2^2$ (to emphasize solutions x_α that are as close as possible to a specified reference value x^{ref} suggested by e.g. physical or enginering considerations) or $\alpha\|Gx\|_2^2$ (where G is a discrete differentiation operator, so as to avoid solutions with large gradient norm due to e.g. small-scale oscillations).
- The curve C is convex for regularized least-squares problems, and is still often (roughly) L-shaped in more-complex (in particular non-quadratic) generalizations of problem (6.7); for this reason, it is known as the L-curve associated with the regularized problem.
- The regularization of ill-conditioned discrete problems and ill-posed continuous problems is covered by an abundant literature, see e.g. the monographs [18, 23, 26].
- The regularization parameter α may be chosen on the basis of a variety of other heuristics, e.g. by seeking the corner of the L-curve (if such can be defined) or its point closest to the origin of the (D, R) plane, or alternatively by using the *generalized cross-validation* method.
- Many regularization methods for linear ill-conditioned problems, in particular those based on the L-curve concept, are implemented in the freely-available MATLAB toolbox *regularization tools*[2].
- In particular, another natural approach for regularizing ill-conditioned linear systems consists in reducing the dimension of the linear space in which x is sought. This can be done for example by truncating the SVD of A, and we briefly describe this treatment in the next section.

6.3 REGULARIZATION USING TRUNCATED SVD. Matrices with fast decaying singular values can be well approximated by a truncated SVD, as previously stated in Theorem 6.1. A natural alternative to regularized least squares then consists in seeking the minimum-norm solution of the original least squares problem with the matrix A replaced with its rank-r TSVD approximation $\widehat{A}_r$ (which is by construction rank-deficient except if $r = n$). In particular, one hopes to be able to choose an optimal value $r(\delta$ of the truncation parameter r depending on the data noise level δ.

Upon replacing A by $\widehat{A}_r$ in problem (6.3) (i.e. replacing the singular values $\sigma_{r+1}, \ldots, \sigma_n$ by zero), the minimum-norm solution x_r of

$$\min_{x \in \mathbb{K}^n} \|\widehat{A}_r x - b\|^2,$$

is given by (3.9) with $x_{r+1} = \ldots = x_n = 0$, i.e.

$$x_r = \sum_{i=1}^{r} \frac{z_i}{\sigma_i} v_i.$$

[2]See `http://www.imm.dtu.dk/~pcha/Regutools/+`.

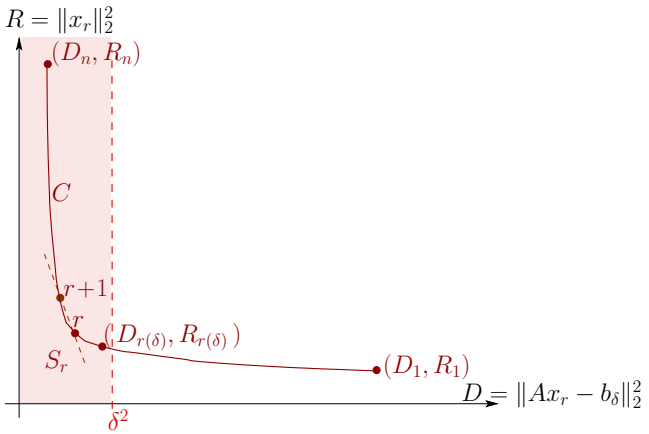


Figure 6.3: L-curve for the TSVD-based regularization, and graphical illustration of the determination of the optimal truncation parameter $r(\delta)$. The shaded band covers the part of C satisfying the inequality constraint $\|Ax - b\|_2^2 \leq \delta^2$ of (6.13).

Reasoning as in the case of regularized least squares, we again introduce the linear system residual evaluated at the minimizer x_r and the minimizer squared norm:

$$D_r := \|\widehat{A}_r x_r - b\|_2^2 = \sum_{i=r+1}^{m} |z_i|^2, \qquad R_r := \|x_r\|_2^2 = \sum_{i=1}^{r} \frac{|z_i|^2}{\sigma_i^2} \tag{6.14}$$

(with D_r again evaluated with the help of (3.10)). Clearly, the (finite) sequence D_r is decreasing while the sequence R_r is increasing. Moreover, we can define the counterpart C_n of the L-curve by introducing the discrete set n of points (D_r, R_r) $(1 \leq r \leq n)$ in the (D, R) plane and join points with consecutive indices by straight lines. From expressions (6.15), we moreover observe that

$$S_r := \frac{R_r - R_{r+1}}{D_r - D_{r+1}} = -\frac{|z_{r+1}|^2}{\sigma_{r+1}^2} \frac{1}{|z_{r+1}|^2} = -\frac{1}{\sigma_{r+1}^2}$$

(S_r being the slope of the segment starting at (D_{r+1}, R_{r+1}) and ending at (D_r, R_r), i.e. following C_n along the direction of increasing abscissa). Given that the singular values are listed along decreasing values, the (negative) slopes decrease as r increases. This remark implies, as graphically illustrated by Fig. 6.3, that the piecewise-linear curve C_n is convex. We thus are in a situation similar to the case of regularized least squares, with the discrete parameter $1/r$ playing the same role as the continuous regularization parameter α. For data b_δ with a known level of noise δ, the optimal value $r(\delta)$ if r is determined by

$$r(\delta) = \min_{1 \leq r \leq n} \quad \text{subject to} \quad D_r \leq \delta^2 \tag{6.15}$$

As a result, the regularization method using truncated SVD and the selection rule (6.16) behaves similarly to the regularized least squares method with the selection rule (6.14).

6.4 REGULARIZATION BY PROMOTION OF SPARSITY. In some areas of applications, it is important to find approximate solutions of linear systems that are sparse, i.e. for which x has many zero entries. This is for instance the case when dealing with image deblurring, where the sought restored image (coded in a vector $x \in \mathbb{R}^n$) is related to the available (blurred) image (coded in a vector $b \in \mathbb{R}^m$) through a relationship of the form

$$Ax = b + w, \qquad A = BW \tag{6.16}$$

where $B \in \mathbb{R}^{m \times m}$ models the blurring (e.g. by atmospheric turbulences for astronomical images) undergone by the image, $W \in \mathbb{R}^{m \times n}$ contains in its columns a wavelet basis and $w \in \mathbb{R}^m$ is an

unknown noise. In this description, the "correct" image is coded in a vector Wx representing an expansion on a basis of wavelets with coefficients given in the vector x. As most images have a sparse wavelet representation, it is desirable to find approximate solutions to (6.17) that are sparse. Using a 2-norm regularizer for the system (6.17), as discussed in Section 6.2, is inefficient for the purpose of promoting sparse solutions x, and it is better to use instead a 1-norm regularizer. In this case, approximate regularized solutions to (6.17) are sought via solving the minimization problem

$$\min_{x\in\mathbb{R}^n} J_\alpha(x), \qquad J_\alpha(x) := \|Ax-b\|_2^2 + \alpha\|x\|_1 \tag{6.17}$$

where α is the regularization parameter. An abundant literature is available on sparsity-promoting solution methods. In what follows we describe one specific algorithn, namely the fast iterative shrinkage-thresholding algorithm (FISTA) proposed in [4], which is relatively simple to explain and implement.

6.4.1 Minimization of functionals with a nonsmooth part. In Chapter 4, we presented solution methods for linear systems based on their interpretation as the stationarity equations for the minimization of a quadratic functional $J(x)$. For instance, recalling (4.6) and (4.7), one step of the steepest gradient descent method consists in updating x through

$$x^{(k+1)} = x^{(k)} - t\nabla J(x^{(k)}), \tag{6.18}$$

where the value $t^{(k)}$ of the step length t is determined by a line-search; this update step is explicit inasmuch it is based on the value of ∇J at the initial point $x^{(k)}$. The minimization problem (6.18) is more complicated in that the objective functional is not quadratic, and in fact is even not differentiable at any vector x having at least one zero entry. Updating steps of the form (6.19) therefore cannot be applied because (i) $\nabla J_\alpha(x^{(k)})$ may not be defined and (ii) the line-search method determining $t^{(k)}$ may trigger values of $x^{(k)} - t^{(k)}\nabla J_\alpha(x^{(k)})$ at which J_α is not differentiable. We now describe a method for generalizing updating steps of the form (6.19) fo the minimization of objective functionals of the general form

$$J(x) = f(x) + g(x) \tag{6.19}$$

where f and g are both convex functions of the vector-valued variable x, f is differentiable but g may be non-smooth; obviously the format (6.20) of J generalizes that of problem (6.18). The idea consists in first considering a version of the update step (6.19) that is explicit for f but implicit for g, by setting

$$\text{(a)}\quad \hat{x}^{(k)} = x^{(k)} - t\nabla f(x^{(k)}), \qquad \text{(b)}\quad x^{(k+1)} = \hat{x}^{(k)} - t\nabla g(x^{(k+1)}). \tag{6.20}$$

For given step length t, the implicit equation (6.21b) has to be solved for $x^{(k+1)}$. This would be trivial for a quadratic function g and otherwise feasible using a Newton method if g is twice-differentiable (in fact, the explicit and explicit-implicit update rules (6.19) and (6.21) are equivalent when both f and g are quadratic in x, see Exercise 6.1). In the present framework, equation (6.21b) needs to be reformulated since g may not be differentiable. If g is differentiable, (6.21b) is equivalent recast as

$$\frac{1}{t}\big(x^{(k+1)} - \hat{x}^{(k)}\big) + \nabla g(x^{(k+1)}) = 0,$$

which (since g is assumed to be convex) is equivalent to

$$x^{(k+1)} = \arg\min_{x\in\mathbb{R}^n}\Big(\frac{1}{2t}\|x-\hat{x}^{(k)}\|_2^2 + g(x)\Big)$$

When g is convex but not differentiable, the above minimization problem still makes sense, the minimizer being unique. In fact, for any lower semicontinuous[3] convex function $h : \mathbb{R}^n \to \mathbb{R}$, let the *proximal operator* $\text{prox}_h : \mathbb{R}^n \to \mathbb{R}$ be defined as

$$\text{prox}_h(y) = \arg\min_{x\in\mathbb{R}^n}\big(\tfrac{1}{2}\|x-y\|_2^2 + g(x)\big). \tag{6.21}$$

[3]This means that

Using this definition in (6.21b), the update rule (6.21) applicable to objective functions (6.20) with a possibly-nonsmooth (convex) term g becomes

$$x^{(k+1)} = \mathrm{prox}_{tg}\big(x^{(k)} - t\nabla f(x^{(k)})\big) \tag{6.22}$$

6.4.2 Minimization of L^2-L^1 functionals. We now specialize the update rule (6.23) to the L^2-L^1 functional J introduced in (6.18). In this case, major simplifications occur due to the fact that the non-smooth part $g(x) = \alpha\|x\|_1 = \alpha\big(|x_1|+\ldots+|x_n|\big)$ is a sum of univariate functions. The minimization problem (6.22) hence uncouples into n univariate minimization problems, and the proximal operator prox_{tg} is obtained componentwise from the proximal operator of the univariate function $y \mapsto \alpha t|y|$, which can be expressed in closed form by means of elementary arguments:

$$\mathrm{prox}_{u\mapsto\alpha t|u|}(y) = \operatorname*{arg\,min}_{x\in\mathbb{R}} \big(\tfrac{1}{2}(x-y)^2 + \alpha t|x|\big) = \begin{cases} 0 & |y_i| \le \alpha t \\ y(1-\alpha t/|y_i|) & |y| \ge \alpha t \end{cases}$$

As a result, the update rule (6.23) for the L^2-L^1 functional J and a given step length t reads

$$\text{(a)}\quad \hat{x}^{(k)} = x^{(k)} - tA^{\mathrm{T}}(Ax^{(k)}-b), \qquad \text{(b)}\quad x_i^{(k+1)} = \begin{cases} 0 & |\hat{x}_i^{(k)}| \le \alpha t \\ \hat{x}_i^{(k)} - \alpha t & \hat{x}_i^{(k)} \ge \alpha t \\ \hat{x}_i^{(k)} + \alpha t & \hat{x}_i^{(k)} \le \alpha t \end{cases}. \tag{6.23}$$

Remark 6.1 *Notice that the value of $prox_{u\mapsto\alpha t|u|}(y)$ is zero whenever $|y|$ is below the threshold αt. This is the essence of the sparsity-promoting mechanism of the L^2-L^1 minimization. Reducing the regularization parameter α will lower that threshold, i.e. weaken the sparsity-promotion effect.*

Remark 6.2 (condition for convergence of proximal iterations [4]) *The step length t must verify $t \le 1/L$, where $L := \|A^{\mathrm{T}}A\|_2$ is the spectral radius of $A^{\mathrm{T}}A$, for the iterations* (6.24) *to converge (the proof of this important fact is left to the reader as Exercise 6.2).*

The update rule (6.24) features a constant step length, which makes for a slow convergence. In differentiable optimization, this is usually remedied my adjusting the step length at each iteration (usually by means of a line-search, i.e. an univariate minimization w.r.t. t such as (4.5)). The performance of the L^2-L^1 minimization is improved by applying convergence acceleration methods. For instance, the FISTA (fast iterative shrinkage-thresholding algorithm) proposed by Beck and Teboulle [4] applied to problem (6.18) produces the following algorithm:

Algorithm 6.1 FISTA iterations for the L^2-L^1 minimization problem [4]

1: $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $\alpha > 0$, $x^{(0)} \in \mathbb{R}^n$ (data and initial guess)
2: $L = \|A^{\mathrm{T}}A\|_2$ (spectral radius of $A^{\mathrm{T}}A$: compute largest eigenvalue of $A^{\mathrm{T}}A$)
3: $t = \alpha/L$ (step length, maximum permissible value)
4: $y^{(1)} = x^{(0)}$, $s^{(1)} = 1$ (first iteration)
5: **for** $k = 1, 2, \ldots$ **do**
6: $\quad x^{(k)} = \mathrm{prox}_{u\mapsto t\|u\|}(y^{(k)})$ (apply proximal operator)
7: $\quad s^{(k+1)} = \frac{1}{2}\big(1+\sqrt{1+4s^{(k)2}}\big)$ (update algorithmic parameter $s^{(k)}$)
8: $\quad y^{(k+1)} = x^{(k)} + \frac{s^{(k)}-1}{s^{(k+1)}}\big(x^{(k)} - x^{(k-1)}\big)$
9: $\quad$ **If convergence test satisfied:** return $x = x^{(k)}$
10: **end for**

Exercise 6.1 *Consider the implicit update step* (6.21) *applied to the quadratic regularized least squares problem* (6.7)*, for which $g(x) = \alpha\|x\|_2^2$. Show that the explicit update rule* (6.19) *and its explicit-implicit variant* (6.21) *yield the same updated solution $x^{(k+1)}$ upon suitably redefining the value of the step length t.*

Exercise 6.2 (step length limitation for the L^2-L^1 functional) *(a) Set the update rule* (6.24) *in the form* $x^{(k+1)}) = x^{(k)} - ty$ *and give the explicit expression of each entry* y_i *of* y.

(b) Using the above update rule for the L^2-L^1 *functional* J_α *introduced in* (6.18)*, prove that*

$$J_\alpha(x^{(k+1)}) - J_\alpha(x^{(k)}) \leq \tfrac{1}{2}t^2 y^T A^T A y - t y^T y$$

(c) Prove that if $t\|A^T A\|_2 \leq 1$*, we have* $J_\alpha(x^{(k+1)}) - J_\alpha(x^{(k)}) \leq 0$*, i.e. the implied step length limitation ensures that each iteration reduces the value of the* L^2-L^1 *functional.*

(d) Conversely, prove that if $t\|A^T A\|_2 > 1$*, the update rule may result in* $J_\alpha(x^{(k+1)}) - J_\alpha(x^{(k)}) > 0$*, i.e. in an (unwanted) increase of the* L^2-L^1 *functional.*

CHAPTER 7 APPENDICES

7.1 SUMMARY OF BASIC DEFINITIONS AND FACTS.

Trace of a matrix. The usual definition of the trace $\mathrm{Tr}(A)$ of a square matrix $A \in \mathbb{K}^{n\times n}$ is

$$\mathrm{Tr}(A) := \sum_{i=1}^{n} a_{ii} \tag{7.1}$$

A more-satisfactory definition is as follows: let $v_1, \ldots, v_n$ be any basis of $\mathbb{K}^n$, and define the dual basis vectors $v^1, \ldots, v^n$ by $(v^k)^{\mathrm{T}} v_\ell = \delta_{k\ell}$ ($\delta_{k\ell}$ being the Kronecker symbol). Then, we set

$$\mathrm{Tr}(A) = \sum_{i=1}^{n} (v^i)^{\mathrm{T}} (A v_i). \tag{7.2}$$

Definition (7.2) agrees with (7.1) if $v_1, \ldots, v_n$ is the standard basis (which coincides with its dual basis). Moreover, $\mathrm{Tr}(A)$ as defined by (7.2) does not depend on the choice of basis.

Matrix transpose and adjoint. Let $A \in \mathbb{R}^{m\times n}$. There exists a unique matrix A^{T}, called the transpose of A, such that

$$y^{\mathrm{T}} A x = x^{\mathrm{T}} (A^{\mathrm{T}} y) \qquad \text{for all } x \in \mathbb{R}^n,\ y \in \mathbb{R}^m.$$

Likewise, for any $A \in \mathbb{C}^{m\times n}$, there exists a unique matrix A^{H}, called the Hermitian transpose of A, such that

$$y^{\mathrm{H}} A x = \overline{x^{\mathrm{H}} (A^{\mathrm{H}} y)} \qquad \text{for all } x \in \mathbb{C}^n,\ y \in \mathbb{C}^m.$$

The (Hermitian) transpose of a matrix is in fact the matrix of the linear map that is the *adjoint* to that represented by A. Its generic entry is given by

$$(A^{\mathrm{T}})_{ij} = a_{ji} \quad (A \in \mathbb{R}^{m\times n}), \qquad (A^{\mathrm{H}})_{ij} = \overline{a_{ji}} \quad (A \in \mathbb{C}^{m\times n}).$$

Determinant.

Orthogonal matrices. A matrix $Q = [q_1, \ldots, q_n] \in \mathbb{K}^{m\times n}$ is *orthogonal* if its columns Q_i are mutually orthogonal:

$$(q_i)^{\mathrm{H}} q_j = \delta_{ij} \quad (1 \le i, j \le n), \qquad \text{i.e.} \quad Q^{\mathrm{H}} Q = I_n.$$

Orthogonal matrices conserve the 2-norm and scalar product:

$$\|Qx\|_2^2 = \|x\|_2^2, \qquad (Qy)^{\mathrm{H}} (Qx) = y^{\mathrm{H}} x \qquad \text{for all } \ x, y \in \mathbb{K}^m$$

Orthogonal square matrices are *unitary matrices* (and in particular are invertible):

$$Q^{-1} = Q^{\mathrm{H}}, \qquad Q^{\mathrm{H}} Q = Q Q^{\mathrm{H}} = I_n, \qquad \det(Q) = \pm 1$$

Spectral radius. The *spectral radius* $\varrho(A)$ of a square matrix $A \in \mathbb{K}^{n\times n}$ is defined by

$$|\lambda_1| = \varrho(A) = \max\big(|\lambda_1|, \ldots, |\lambda_n|\big) = |\lambda_1|$$

where $\lambda_1, \ldots, \lambda_n$ are the n eigenvalues of A, counted with multiplicities and ordered (as usual in this document) so that $|\lambda_1| \geq |\lambda_2| \geq \ldots |\lambda_n|$. For any induced matrix norm $\|\cdot\|$, we have

$$\varrho(A) \leq \|A\|$$

Projectors.

Sherman-Morrison-Woodbury formula. Let $A \in \mathbb{K}^{n\times n}$ be an invertible square matrix. Let the modified matrix $A+UCV$ also be invertible, where $C \in \mathbb{K}^{r\times r}$, $U \in \mathbb{K}^{n\times r}$ and $V \in \mathbb{K}^{r\times n}$ and $r \leq n$. The Sherman-Morrison-Woodbury formula relates $(A+UCV)^{-1}$ and A^{-1}:

$$(A+UCV)^{-1} = A^{-1} - A^{-1}U\big(C^{-1}+VA^{-1}U\big)^{-1}VA^{-1} \tag{7.3}$$

(proof: multiply both sides of (7.3) by $A+UCV$, check that equality $I = I$ is obtained). Obtaining $(A+UCV)^{-1}$ knowing A^{-1} using (7.3) therefore requires the inverse of two $r\times r$ matrices, namely C and $C^{-1}+VA^{-1}U$. This makes (7.3) useful when $r \ll n$, where the computational cost of two $r\times r$ inversions (plus the $2r$ matrix-vector products involved in (7.3)) is much lower than that of computing $(A+UCV)^{-1}$ anew. The case where $r = 1$ (inversion of the rank-one modification of an invertible matrix), where V and U are row and column vectors, respectively, is particularly useful in practice as the "small" auxiliary $r\times r$ matrices reduce to scalars.

7.2 OVERVIEW OF MATRIX DECOMPOSITIONS. Matrix decompositions are essential tools in numerical linear algebra, as many algorithms work by first converting a given matrix into a (usually multiplicative) form involving "simpler" matrices. In this section, we list the definitions of the main matrix decompositions (some already met, others not yet). They are often built upon three fundamental types of matrices:

- Diagonal matrices (and also block-diagonal or multidiagonal matrices);
- Triangular matrices (which can be either lower-triangular or upper-triangular and allow to solve linear systems of equations by forward or backward substitution);
- Orthogonal matrices (whose inverse is immediately known, and which preserve condition numbers, i.e. accuracy).

LU decomposition: any square matrix $A \in \mathbb{K}^{n\times n}$ admits the decomposition $A = LU$, where L and U are lower- and upper-triangular, respectively.

Cholesky decomposition: any SPD square matrix $A \in \mathbb{K}^{n\times n}$ admits the decomposition $A = GG^{\mathrm{H}}$ (where $G \in \mathbb{K}^{n\times n}$ is lower-triangular and has a real positive main diagonal), and also the decomposition $A = LDL^{\mathrm{H}}$ (where $L \in \mathbb{K}^{n\times n}$ is lower-triangular with unit diagonal entries and $D \in \mathbb{R}^{n\times n}$ is diagonal with strictly positive entries).

QR decomposition: any matrix $A \in \mathbb{K}^{m\times n}$ admits the decomposition $A = QR$, where $Q \in \mathbb{K}^{m\times m}$ is unitary and $R \in \mathbb{K}^{m\times n}$ is upper triangular. Moreover, if r is the rank of A, all rows of R below the r-th are zero and the *reduced QR decomposition* $A = Q_rR$ holds, with $Q_r \in \mathbb{K}^{m\times r}$ containing the first r column vectors of Q . Extensions of the QR decomposition are available for $A \in \mathbb{K}^{m\times n}$ with $m < n$.

Schur decomposition: any square matrix $A \in \mathbb{K}^{n\times n}$ admits the decomposition $A = QTQ^{\mathrm{H}}$, where Q is unitary and T is upper triangular. The diagonal of T is made of the eigenvalues of A. If A is real, Q may still be complex.

Reduction to upper Hessenberg form: any matrix $A \in \mathbb{K}^{n\times n}$ admits the decomposition $A = QHQ^{\text{H}}$, where $Q \in \mathbb{K}^{n\times n}$ is unitary and $H \in \mathbb{K}^{n\times n}$ is upper Hessenberg (i.e. $h_{ij} = 0$ if $i-j > 1$: all the entries of H below its first sub-diagonal are zero).

Reduction to tridiagonal form: any Hermitian matrix $A \in \mathbb{K}^{n\times n}$ admits the decomposition $A = QHQ^{\text{H}}$, where $Q \in \mathbb{K}^{n\times n}$ is unitary and $H \in \mathbb{K}^{n\times n}$ is tridiagonal (i.e. $h_{ij} = 0$ if $|i-j| > 1$). This is the reduction to upper Hessenberg form when applied to Hermitian matrices, since Hermitian upper Hessenberg matrices are in fact tridiagonal.

Other decompositions are based on eigenvalues of A or of a matrix associated to A. These include:

Diagonalization any non-defective square matrix $A \in \mathbb{K}^{n\times n}$ admits the decomposition $A = X\Lambda X^{-1}$, where $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$ contains the eigenvalues of A (with multiplicity) and $X = [x_1, \ldots, x_n]$ with $Ax_i = \lambda_i x_i$. Diagonalizability requires invertibility of X; if X is not invertible, A is not diagonalizable and is called *defective*.

Jordan form any square matrix $A \in \mathbb{K}^{n\times n}$ can be given in Jordan form $A = XJX^{-1}$, where J is block-diagonal: $J = \text{diag}(J_1, \ldots, J_\ell)$, $\ell \leq n$, where J_k is a *Jordan block*. Each Jordan block has one eigenvalue of A on its diagonal, and any J_k whose size is greater than 1 is defective.

Singular value decomposition (SVD): any matrix $A \in \mathbb{K}^{m\times n}$ admits the decomposition $A = USV^{\text{H}}$, where U and V are unitary square matrices (of size m and n, respectively) and $S \in \mathbb{R}^{m\times n}$ contains the $\min(m, n)$ *singular values* $\sigma_1, \ldots \sigma_{\min(m,n)}$ of A on its main diagonal and is otherwise zero (i.e. $S_{ii} = \sigma_i$, $S_{ij} = 0$ if $i \neq j$). The singular values are nonnegative real numbers, usually arranged by decreasing order of magnitude (i.e. $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_{\min(m,n)} \geq 0$).

7.3 OVERVIEW OF BASIC PROCESSES.

7.4 MAIN MATLAB OPERATORS FOR NUMERICAL LINEAR ALGEBRA.

- `chol`: computes the Cholesky decomposition of a SPD matrix.
- `eig`: computes eigenvalues and eigenvectors, for both standard and generalized eigenvalue problems.
- `gmres`: solves a square linear system $Ax = b$ iteratively by the GMRES method.
- `hess`: computes the Hessenberg form of an arbitrary matrix A.
- `lu`: computes the LU decomposition of a SPD matrix.
- `mldivide`, abbreviated as "\": black-box linear equation solver.
- `pinv`: computes the pseudo-inverse A^+ of an arbitrary matrix A.
- `pcg`: solves a SPD linear system $Ax = b$ iteratively by the (preconditioned) conjugate gradient method.
- `qr`: computes the QR factorization of an arbitrary matrix A.
- `sparse`: declares that a matrix A is sparse, triggering storage and matrix arithmetic methods appropriate to sparse matrices.
- `svd`: computes the singular value decomposition of an arbitrary matrix A.

Bibliography

[1] Barrett R., Berry M., Chan T.F., Demmel J., Donato J.M., Dongarra J., Eijkhout V., Pozo R., Romine C., Van der Vorst H. *Templates for the solution of linear systems: building blocks for iterative methods*. SIAM (1994).

[2] Bécache E., Chaillat S. Solution of scattering problems by integral equations. AMS 304 course, ENSTA Paris and AMS MSc program (2019).

[3] Bécache E., Ciarlet P., Hazard C., Lunéville E. *La méthode des éléments finis : de la théorie à la pratique. Tome II: compléments*. Presses de l'ENSTA (2010).

[4] Beck A., Teboulle M. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imag. Sci.*, **2**:183–202 (2009).

[5] Bonnet M., Frangi A., Rey C. *The finite element method in solid mechanics*. McGraw Hill Education (2014).

[6] Bonnet-Ben Dhia A.S., Joly P. Théorie spectrale des opérateurs autoadjoints et application à l'étude des ondes guidées. Lecture notes, ENSTA Paris (ANA202) (2020).

[7] Bourgeois L., Hazard C. Elementary tools of analysis for partial differential equations. MA102 Course, ENSTA Paris (2020).

[8] Bourgeois L., Moireau P. Data completion and identification in problems governed by PDEs. Lecture notes (2020).

[9] Chaillat S., Desiderio L., Ciarlet P. Theory and implementation of H-matrix based iterative and direct solvers for Helmholtz and elastodynamic oscillatory kernels. *J. Comput. Phys.*, **351**:165–186 (2017).

[10] Chatelin F. *Valeurs propres de matrices*. Masson (1988).

[11] Ciarlet P., Lunéville E. *La méthode des éléments finis : de la théorie à la pratique. Tome I: concepts généraux*. Presses de l'ENSTA (2009).

[12] Ciarlet P.G. *Introduction à l'analyse numérique matricielle et à l'optimisation*. Masson, Paris (1988).

[13] Darbas M., Darrigrand E., Lafranche Y. Combining analytic preconditioner and Fast Multipole Method for the 3D Helmholtz equation. *J. Comput. Phys.*, **236**:289–316 (2013).

[14] Darve E., Wootters M. *Numerical linear algebra with Julia*. SIAM (2021).

[15] Demmel J.W. *Applied numerical linear algebra*. SIAM (1997).

[16] Eckart C., Young G. The approximation of one matrix by one of lower rank. *Psychometrika*, **1**:211–219 (1936).

[17] Géradin M., Rixen D. *Mechanical vibrations. Theory and application to structural dynamics*. Wiley (2015).

[18] Gockenbach M.S. *Finite-dimensional linear algebra.* CRC Press (2010).

[19] Golub G.H., Van Loan C.F. *Matrix computations (third edition).* Johns Hopkins University Press, Baltimore (1996).

[20] Greenbaum A. *Iterative methods for solving linear systems.* SIAM, Philadelphia, USA (1997).

[21] Guzina B.B., Bonnet M. Effective wave motion in periodic discontinua near spectral singularities at finite frequencies and wavenumbers. *Wave Motion*, **103**:102729 (2021).

[22] Hackbusch W. *Iterative solution of large sparse systems of equations.* Springer-Verlag (2016).

[23] Hansen P.C. *Rank-deficient and discrete ill-posed problems.* SIAM, Philadelphia, USA (1998).

[24] Lunéville E. Scientific computing using C++. SIM 201 course, ENSTA Paris (2021).

[25] Mavaleix-Marchessoux D. *Modeling the fluid-structure interaction caused by a far-field underwater explosion.* Ph.D. thesis, Institut Polytechnique de Paris (2020).

[26] Mueller J.L., Siltanen S. *Linear and nonlinear inverse problems with practical applications.* SIAM (2012).

[27] Phillips D.L. A technique for the numerical solution of certain integral equations of the first kind. *JACM*, **9**:84–97.

[28] Quarteroni A., Sacco R., Saleri F. *Numerical mathematics.* Springer (2007).

[29] Saad Y. *Iterative methods for sparse linear systems.* SIAM (2003).

[30] Saad Y. *Numerical methods for large eigenvalue problems.* SIAM (2011).

[31] Saad Y., Schultz M. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, **7**:856–869 (1986).

[32] Shewchuk J.R. An introduction to the conjugate gradient method without the agonizing pain. Course notes, UC Berkeley, USA (1994).

[33] Strang G. *Linear algebra and learning from data.* Wellesley Cambridge Press (2019).

[34] Tikhonov A. Solution of incorrectly formulated problems and the regularization method. *Sov. Math. Dokl.*, **4**:1035–1038 (1963).

[35] Tikhonov A.N., Arsenin V.Y. *Solutions to ill-posed problems.* Winston-Wiley, New York (1977).

[36] Trefethen L., Bau D. *Numerical linear algebra.* SIAM (1997).

[37] Varga R.S. *Matrix iterative analysis.* Springer-Verlag (2000).

[38] Young D.M. *Iterative solution of large linear systems.* Dover (1971).

[39] Zidani H. Optimisation quadratique. AO101 course, ENSTA Paris (2020).