P. 6

CHAPTER 2

Complexity and numerical stability

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

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

2.1. Floating-point arithmetic

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

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

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

whose elements are 0 and the rational numbers of the form

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

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

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

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

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

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

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

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

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

FIGURE 2.1.1. A simple floating-point number system

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

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

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

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

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

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

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

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

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

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

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

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

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

$$0.100\times 2^1 \oplus 0.110\times 2^{-1} = \mathrm{fl}(0.100\times 2^1 + 0.101\times 2^{-1}) = \mathrm{fl}(0.10101\times 2^1) = 0.101\times 2^1.$$

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

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

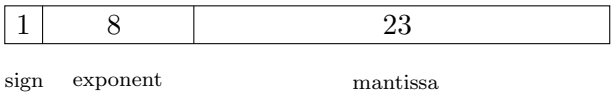

FIGURE 2.1.2. IEEE single precision encoding

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

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

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

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

The underflow and overflow thresholds of this system are

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

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

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

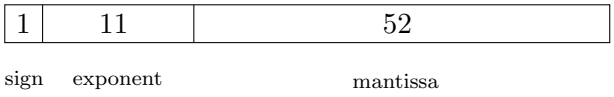

FIGURE 2.1.3. IEEE double precision encoding

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

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

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

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

The underflow and overflow thresholds of this system are

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

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

2.2. The complexity of GEPP

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

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

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

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

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

which is also the size of the input matrix A.

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

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

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

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

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

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

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

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

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

2.3. Vector and matrix norms

Norms are used to measure errors in matrix computations. Errors appears event when the input data is exact, because the operation are performed in floating-point arithmetic, as explained in §2.1. A *vector norm* is a function

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

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

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

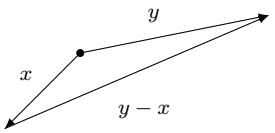

FIGURE 2.3.1. The triangle inequality

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.4. Perturbation theory in linear equation solving

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

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

Some matrices behave really bad with respect to perturbations.

EXAMPLE 2.4.1. Set

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

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

$$\delta b = 0.00017,$$

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

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

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

To compare the *errors*

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

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

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

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

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

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

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

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

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

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

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

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

by the submultiplicativity of the operator norm.

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

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

which readily implies that

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

reflecting the behavior in (2.11). Moreover,

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

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

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

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

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

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

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

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

Remark 2.4.3. Even though the condition number allows to estimate the error in linear equation solving, in practice it may be too expensive to compute it. However, there are algorithm to estimate it, see Exercise 2.4.