Linear algebra for MATH2601 Numerical methods

László Erdős

August 12, 2000

Contents

1 Introduction

In this note we discuss a couple of numerical methods of linear algebra. We rely on the theoretical material developed in the notes "Linear algebra for MATH2601: Theory".

Recall from the Introduction (Section 1.3) of "Linear algebra for MATH2601: Theory", that there are two basic problems:

(I) Give a full solution to $A\mathbf{x} = \mathbf{b}$. If there is no exact solution, then find the "best approximating solution" (Least square method)

(II) Find the eigenvalues, eigenvectors of a square matrix A. (For nonsquare matrix, find the singular value decomposition (SVD))

Based upon the theory, we know how to solve both problems. However, when implementing these methods on a computer there are always rounding errors. In addition, the eigenvalue problem requires finding the roots of a polynomial, which, in general, cannot be done exactly, only approximately (by Newton's iteration, for example). Finally, the data are measured quantities, hence they are slightly inaccurate, but we do not want huge inaccuracy in the solution. A good numerical method must deal with these issues.

There are two main concerns in every numerical scheme: How precise is it and how long does it take?

(i) The errors in long computations can accumulate. This is a serious issue even with high precision computers. Hence it is desirable to develop methods to solve problems (I) and (II) which are **stable** in a sense that computational errors remain under control: small errors in the data or small rounding errors along the computation should not influence the result too much. This requires more sophisticated methods than we have encountered so far.

(ii) The length of a computation depends on how many elementary steps (say elementary operations; additions, multiplications) one has to perform. Current high speed computers can multiply huge numbers in incredible short time. Nevertheless, speed is still an important issue. In real applications one might have to deal with matrices of dimensions several hundred thousands (for example in a discrete scheme for the solution of a PDE).

In general it is not easy to find a good tradeoff between these two issues. Of course longer calculations typically lead to more accurate results (more digits taken etc.). But does it pay off? And how much time (how elaborated method, how many number of steps, how good arithmetic precision etc.) do I need for the required accuracy in the result? If I buy twice as much time on a supercomputer, do I get a significantly better result? Or, if I buy only one hour, do I get an acceptable result at all?

Finally, let us close with saying that "Mathematics is science, computing is art". There are infinitely many methods and tricks developed by numerical experts to make things faster and more accurate. Some of them have a strong theoretical background, some of them are known to work "properly" in most cases or in certain special cases of interest, and even there some "rules of thumb". Here we consider only those aspects of numerical mathematics which are well understood from mathematical point of view.

1.1 Types of errors

There are three sources of errors in a general numerical computation:

(i) Rounding errors

(ii) Truncation errors.

(iii) Conditioning errors.

We discuss them separately.

1.1.1 Rounding errors

Rounding errors are due to explicit calculations on a computer. Every number is stored by a finite number of bits. There are various protocols, the most commonly used is the *floating point representation.* The *IEEE standard single precision* uses 32 bits to represent a number. The first bit is the *sign* of the number (s). The next eight bits is the *exponent* (e). The last 23 bits express the *fraction* ($f < 1$). The number represented is

$$N = (-1)^s \cdot 2^{e-127} \cdot (1 + f)$$

For example $N = 51.25$ is written first in binary system $N = 2^5 + 2^4 + 2^1 + 2^0 + 2^{-2}$, i.e. as $N = 110011.01$, or $N = 1.1001101 \times 2^5$. Hence $s = 0$, $e = 132 = 10000100$, $f = 0.1001101$. In the computer f is represented as 1001101, i.e. the binary point and the first zero are omitted.

The underflow threshold is 2^{-126}, the overflow threshold is 2^{128}, hence numbers between $\approx 10^{-38}$ and $\approx 10^{38}$ in absolute value can be represented.

The *double precision* representation uses 64 bits, allocated in a similar way (1 for the sign, 11 for the exponent, 52 for the fraction), and it extends the range of represented numbers from about 10^{-308} to 10^{308} in absolute value.

Let $fl(a)$ denote the floating point representation of the number a. This is of course only an approximation of the true number a. The error is the difference $\delta a = a - fl(a)$ (usually it is denoted by δa or Δa, and in this notation δ or Δ is not a number but a symbol which is meaningful only together with a). When performing computations, we usually want to compute binary operations with two numbers, a and b, for example $a + b$ or $a \cdot b$, but actually we are computing $fl(a) + fl(b)$ or $fl(a) \cdot fl(b)$, and the computer stores only the floating point representation of the result of these operations. The errors can clearly accumulate. Rounding errors in additions and substractions usually add up (in the worst case). Rounding errors for multiplications depend on the size of the numbers, in fact instead of the **absolute error**

$|\delta a| = |a - fl(a)|$, the **relative error**

$$\frac{|\delta a|}{|a|} = \frac{|a - fl(a)|}{|a|}$$

is more relevant, which typically add up (again in the worst case). Division is the least stable operation, especially if the divisor is close to zero.

One might think that the range of numbers stored even with the single precision arithmetic should be enough for any real life applications without further care. However, there are two phenomena which require special attention:

(i) Many algorithms proceed differently if a number is zero or not ("IF... THEN... ELSE"). Rounding errors could completely mislead the algorithm. (E.g. the choice of pivot in the Gauss elimination depends on whether a number is zero or not. There could be an entry that is zero in reality, if precise computation were done, but happens to be 10^{-30} on the computer. A carelessly programmed Gauss algorithm could choose this number as pivot.) The most expensive error known to have been caused by an improperly interpreted "almost zero" is the crash of the Ariane 5 rocket of the European Space Agency on June 4, 1996. (This example was pointed out in "Applied numerical linear algebra" by J. W. Demmel (SIAM, 1997), see `http://www.cs.berkeley.edu/~demmel/ma221/ariane5rep.html`).

(ii) Many algorithms have a systematical instability. This means that that the rounding errors do not "cancel", rather they magnify each other. I.e. "bad becomes worse", and this is not due to a bad luck, but is inherent in the system. Once a small error is made, it will be magnified.

1.1.2 Truncation errors

Gauss elimination and Cramer's rule are examples of *direct* methods. This means that they would produce an *exact* solution to a problem in finite number of elementary steps (addition, subtraction, multiplication, division). In contrast to these methods, for example, the Newton's

method is *iterative*. It produces a sequence of approximate solutions x_k instead of the true solution x, hopefully with $\|x_k - x\| \to 0$. This means that the approximate solutions converge to the true solution in some sense (measured in some distance or norm). This is called the *consistency* of the method. But you must end the sequence after some k_0 steps, since eventually you have to present the result to your boss within a finite time. Even if the method were consistent and no rounding errors were made, what you present as a solution is x_{k_0} and not the true x. The error $\|x - x_{k_0}\|$ is called *truncation error*.

Estimating the truncation error is especially important in order to decide when to stop the iteration. Usually you know how accurate result you want, e.g. you know that you stop as soon as $\|x - x_{k_0}\|$ falls below 10^{-4}. But you do not know x, only the sequence $x_1, x_2, \ldots x_{k_0}$. You need an estimate on the distance of x_{k_0} from the true but unknown solution x based upon the sequence you already computed.

You have seen similar error estimates at numerical integration which were needed to set the stepsize (number of points in the subdivision). For example when you computed a definite integral by the rectangular or trapezoid or Simpson rule, the accuracy depended on the number of intervals in the subdivision (See Salas-Hille, Section 8.7). Finer subdivision meant more precise result and the error estimates gave you a hint on the necessary number of intervals to reach a given accuracy.

Similar estimates are well known for most iterative methods and they are related to the *speed of convergence* of the method.

1.1.3 Conditioning errors

The data used in any applications are not exact. They usually come from measurements with a certain accuracy. In fact, instead of saying that the length of a rod is 3.2 meters, it would be more honest to say that it is between two numbers, depending on how accurately you measured it. E.g. it is probably between 2.7 and 3.7 meters if you just estimated it by

"looking at it" and it is between 2.19 and 3.21 meters if you measured it with a ruler, while it can be between 2.1999 and 3.2001 meters if you measured it with more sophisticated tools. In engineering literature this is usually expressed as 3.2 ± 10^{-2} or 3.2 ± 10^{-4} meters. Sometimes the number of shown digits indicate the precision: 3.20 refers to two accurate digits, while 3.2000 refers to four.

This is called *interval arithmetic*, since each number is replaced by an interval of a certain length around its "most likely" value.

It is certainly important to predict the sensitivity of the output on the data in any mathematical model and method. We know for example that dividing by small numbers is a sensitive operation, if we want to solve the equation

$$0.001x = 34 \tag{1.1}$$

then we'd better make sure that 0.001 and 34 are precise, otherwise the solution is very inaccurate. If the true coefficient is 0.0011 instead of the measured 0.001, then the true result $x = \frac{34}{0.0011} \approx 30909$ instead of the computed $x = \frac{34}{0.001} = 34000$. A tiny inaccuracy in the input gave a huge inaccuracy in the result. You might say that the *relative* inaccuracy of the data was not too small, eventually 0.001 differs from 0.0011 by 10%. This is true, but 0.0011 could have been obtained as a result of some earlier calculation, for example as a difference of two "big" numbers: $0.0011 = 1.234 - 1.2329$. Here a tiny inaccuracy in 1.2329 (e.g. using 1.233 instead) changes the solution of (1.1) drastically.

It is important to emphasize that this problem has nothing to do with computers, running times or rounding errors. It *inherently* comes from the type of question we asked: the original *mathematical* question to solve $ax = b$ is **ill-conditioned** for small a.

You might think that it is easy to recognize when and why things go wrong. Here is a

warning example (from "Introduction to numerical linear algebra and optimisation" by P. Ciarlet, Cambridge Univ. Press, 1989)

The problem is to solve the following system of four equations for four unknowns of the form $A\mathbf{x} = \mathbf{b}$ (I already wrote it in the matrix notation)

$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$$

The solution is

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

as you can easily convince yourself.

Now suppose that the numbers in the $\mathbf{b}$ vector are slightly modified (due to measurement error) and we consider the system

$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 32.1 \\ 22.9 \\ 33.1 \\ 30.9 \end{pmatrix}$$

Certainly you would not expect too much deviation in the solution just by changing the entries of the vector $\mathbf{b}$ by 10^{-1}, which correspond to a relative error of order $\approx 10^{-1}/23 \approx 10^{-2}$. But the solution of the new system is

$$\mathbf{x} = \begin{pmatrix} 9.2 \\ -12.6 \\ 4.5 \\ -1.1 \end{pmatrix}$$

which even does not resemble the original solution! The (biggest) relative error is around $\frac{13.6}{1} \approx 10$. The amplification of the relative errors is of order 1000!!

As another example, let us suppose now that the entries of the matrix are measured incorrectly and the modified problem is

$$\begin{pmatrix} 10 & 7 & 8.1 & 7.2 \\ 7.08 & 5.04 & 6 & 5 \\ 8 & 5.98 & 9.89 & 9 \\ 6.99 & 4.99 & 9 & 9.98 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$$

The solution is

$$\mathbf{x} = \begin{pmatrix} -81 \\ 137 \\ -34 \\ 22 \end{pmatrix}$$

and it is again very far from the original solution despite a tiny change in A.

Since the solution is $\mathbf{x} = A^{-1}\mathbf{b}$, one might think that the problem is that the original matrix A has a "bad" inverse, analogously to (1.1). But its inverse is "nice"

$$A^{-1} = \begin{pmatrix} 25 & -41 & 10 & -6 \\ -41 & 68 & -17 & 10 \\ 10 & -17 & 5 & -3 \\ -6 & 10 & -3 & 2 \end{pmatrix}$$

and even $det(A) = 1$, so Cramer's formula should be OK, too. What goes wrong?

The real reason why this matrix behaves so badly is that it has a very small eigenvalue. The eigenvalues of A are

$$\lambda_1 \approx .01015 < \lambda_2 \approx .8431 < \lambda_3 \approx 3.858 < \lambda_4 \approx 30.2887$$

None of them are big, but the really important quantity is the *ratio of the biggest and smallest eigenvalue*, which is

$$\frac{\lambda_4}{\lambda_1} \approx 2984 \tag{1.2}$$

The fact is that the relative error in the data can be amplified by this amount in the worst case.

To understand this phenomenon, we first have to introduce some tools.

1.2 Matrix norms

Recall that a vector $\mathbf{v} \in \mathbf{R}^n$ has a norm or length, defined as

$$\|\mathbf{v}\| := \sqrt{\mathbf{v}^t \cdot \mathbf{v}} = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2} \tag{1.3}$$

This *number* gives an information about "how large" the vector is. There are other ways to measure the size of a vector, for example you could have taken the maximum of $|v_i|$'s or their sum $|v_1| + \ldots + |v_n|$. Which one to use, depends on the applications. Here we will always use the norm (1.3) as this is the most common vector norm.

The basic properties of the norm are the following

$$\|\mathbf{v}\| = 0 \iff \mathbf{v} = \mathbf{0} \tag{1.4}$$

$$\|\alpha\mathbf{v}\| = |\alpha| \, \|\mathbf{v}\| \qquad \text{for every } \alpha \in \mathbf{R} \tag{1.5}$$

$$\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\| \tag{1.6}$$

(triangle inequality).

It is important to note that the norm is insensitive to the direction of the vector. This means that if you rotate or reflect the vector, its norm (length) does not change. However, rotation and reflection makes sense only in $n = 2, 3$ dimensions. The right *generalization* of rotations and reflections is precisely defined via the property that the norm is preserved. Recall these are exactly the orthogonal transformations.

How to measure the size of a matrix $A = (a_{ij})$? Again, there are various ways to do this. It seems that the analogue of the vector norm

$$\sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij}^2} \tag{1.7}$$

is a convenient quantity, in particular it satisfies all the three properties above (in fact this is exactly the vector norm above if you forget about the rectangular shape of the matrix, just write their entries into a long mn-vector). This norm is actually often used, and is called the *Hilbert-Schmidt norm* of the matrix. We already used this norm in the singular value decomposition.

However, this is not the most common way to measure the size of a matrix. The following norm is more complicated than (1.7), but it will be a very convenient tool.

Definition 1.1 *The norm of an $n \times m$ matrix A is*

$$\|A\| := \max_{\substack{\mathbf{v}\in\mathbf{R}^n \\ \mathbf{v}\neq\mathbf{0}}} \frac{\|A\mathbf{v}\|}{\|\mathbf{v}\|} \tag{1.8}$$

REMARK 1: Notice that this matrix norm is defined via the vector norms in the spaces $\mathbf{R}^n$ and $\mathbf{R}^m$. The ratio $\frac{\|A\mathbf{v}\|}{\|\mathbf{v}\|}$ measures the amount of magnification of the vector $\mathbf{v}$ under the action of A (recall that a matrix can be viewed as a linear transformation). Hence the norm $\|A\|$ measures the biggest possible magnification of A.

REMARK 2: The norm of any orthogonal matrix Q is $\|Q\| = 1$. Recall that orthogonal transformations preserve the norm of any vector, i.e $\|Q\mathbf{v}\| = \|\mathbf{v}\|$.

REMARK 3: The Maple command to compute the norm of a matrix A is `norm(A,2)`. Here 2 refers exactly to the norm we defined in (1.8). There are several other norms, and the default of Maple (command `norm(A)`) is *not* the norm we defined.

The following crucial property follows immediately from the definition:

$$\|A\mathbf{v}\| \leq \|A\| \, \|\mathbf{v}\| \qquad \text{for any } \mathbf{v} \in \mathbf{R}^n. \tag{1.9}$$

Although we use the same notation $\|\ \ \|$ for vector and matrix norms, these are separately defined by (1.3) and (1.8). This should not cause confusion, since what you have "inside the

norm" should tell you which norm you use. For example on the left hand side of (1.9) $\|\ \ \|$ means the vector norm in $\mathbf{R}^m$, the first norm on the right hand side is the matrix norm defined in (1.8) and finally the last norm is the vector norm in $\mathbf{R}^n$.

Lemma 1.2 *The matrix norm $\|A\|$ satisfies the analogues of the properties (1.4)-(1.6), i.e.*

$$\|A\| = 0 \iff A = 0 \tag{1.10}$$

$$\|\alpha A\| = |\alpha|\ \|A\| \qquad \textit{for every} \quad \alpha \in \mathbf{R} \tag{1.11}$$

$$\|A + B\| \leq \|A\| + \|B\| \tag{1.12}$$

In addition, if $m = n$, then

$$\|AB\| \leq \|A\|\ \|B\| \tag{1.13}$$

The proof is left as an exercise (DO IT (*))

Since the matrix norm expresses the amount of magnification, it could be related to eigenvalues, at least for square matrices. Plugging an eigenvector into (1.9) we easily see that $\|A\| \geq |\lambda|$ for all eigenvalue λ. It would be nice if $\|A\|$ were actually the largest eigenvalue. This is indeed true for symmetric matrices, but not for general square matrices (GIVE AN EXAMPLE (*) Hint: $A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$). However, we have the following theorem:

Theorem 1.3 *(i) Let A be an $n \times n$ symmetric matrix. Then*

$$\|A\| = \max_i |\lambda_i(A)|$$

where $\lambda_i(A)$ for $i = 1, 2, \ldots n$ are the eigenvalues of A.

(ii) Let A be a general $n \times n$ matrix. Then

$$\|A\| = \max_i \sqrt{\lambda_i(AA^t)}$$

where $\lambda_i(AA^t)$ for $i = 1, 2, \ldots n$ are the eigenvalues of AA^t.

REMARK 1.: Notice that we omitted the absolute value under the square root. This is because the eigenvalues of AA^t are always nonnegative. (WHY(*)? Hint: let $AA^t\mathbf{v} = \lambda\mathbf{v}$, and take the scalar product of this equation with $\mathbf{v}^t$)

REMARK 2.: In fact the nonzero eigenvalues of AA^t and A^tA coincide, hence $\max_i \lambda_i(AA^t) = \max_i \lambda_i(A^tA)$. The proof of this fact is easy for invertible matrices (PROVE IT (*)) and slightly harder for singular matrices.

Proof of (i) of Theorem 1.3. Recall the Spectral Theorem

$$A = \sum_{i=1}^{n} \lambda_i \mathbf{v}_i \mathbf{v}_i^t$$

where $\lambda_i, \mathbf{v}_i$ are the eigenvalues-eigenvectors of A. Recall that the eigenvectors can be assumed orthonormal. Then for any $\mathbf{u} \in \mathbf{R}^n$

$$A\mathbf{u} = \sum_{i=1}^{n} \lambda_i (\mathbf{v}_i^t \cdot \mathbf{u}) \mathbf{v}_i$$

and by the Parseval identity

$$\|A\mathbf{u}\|^2 = \sum_{i=1}^{n} \lambda_i^2 (\mathbf{v}_i^t \cdot \mathbf{u})^2.$$

We can estimate all eigenvalues by the largest one in absolute value, i.e.

$$\|A\mathbf{u}\|^2 \leq \Big(\max_{i=1,2,\ldots} \lambda_i^2 \Big) \sum_{i=1}^{n} (\mathbf{v}_i^t \cdot \mathbf{u})^2$$

Again by the Parseval identity

$$\sum_{i=1}^{n} (\mathbf{v}_i^t \cdot \mathbf{u})^2 = \|\mathbf{u}\|^2$$

hence

$$\|A\mathbf{u}\|^2 \leq \Big(\max_{i=1,2,\ldots} \lambda_i^2 \Big) \|\mathbf{u}\|^2$$

from which $\|A\| \leq \max_{i=1,2,\ldots} |\lambda_i|$ follows.

The opposite inequality $\|A\| \geq \max_{i=1,2,\ldots} |\lambda_i|$ is easy to see just by plugging into (1.9) the eigenvector that belongs to the biggest (in absolute value) eigenvalue. (THINK IT OVER WHY(*))

The proof of part (ii) of Theorem 1.3 is not much harder, but we omit it here. □

1.3 Condition number

Let A be an invertible matrix. Assume that we solve the equation $A\mathbf{x} = \mathbf{b}$ and we also solve the equation $A\mathbf{x}' = \mathbf{b} + \delta\mathbf{b}$ for some small perturbation of $\mathbf{b}$ by $\delta\mathbf{b}$ (this is the situation of the example in Section 1.1.3). Write $\mathbf{x}' = \mathbf{x} + \delta\mathbf{x}$, where $\delta\mathbf{x}$ is the deviation of the solution of the perturbed system from the original solution. We emphasize that both systems are solved *exactly*, there are no rounding errors. Comparing

$$A\mathbf{x} = \mathbf{b} \qquad \text{and} \qquad A(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}$$

we see that $\delta\mathbf{x} = A^{-1}\delta\mathbf{b}$, and we also have $\mathbf{b} = A\mathbf{x}$. Hence (see (1.9))

$$\|\delta\mathbf{x}\| \leq \|A^{-1}\| \, \|\delta\mathbf{b}\| \qquad \text{and} \qquad \|\mathbf{b}\| \leq \|A\| \, \|\mathbf{x}\|$$

Hence the relative error of the solution, measured by $\|\delta\mathbf{x}\|/\|\mathbf{x}\|$, is bounded in terms of the relative error $\|\delta\mathbf{b}\|/\|\mathbf{b}\|$ in the datum $\mathbf{b}$ as follows

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \|A\| \, \|A^{-1}\| \, \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} \tag{1.14}$$

The number

$$\mathrm{cond}(A) := \|A\| \, \|A^{-1}\|$$

i.e., product of the norms of A and A^{-1} is called the **condition number** of the matrix A. This number shows how much a small error in the input data can be magnified in the output. Is is clear that

$$\mathrm{cond}(A) \geq 1$$

for any matrix (WHY(*)?), i.e. errors can only get amplified.

The Maple command to compute the condition number of a matrix A is `cond(A,2)`.

In case of symmetric matrices $\|A\| = |\lambda_{max}|$, where λ_{max} is the eigenvalue biggest in absolute value (i.e. $|\lambda_{max}| \geq |\lambda|$ for all other eigenvalues). The eigenvalues of A^{-1} are the inverses of the eigenvalues of A (WHY(*)?), hence the eigenvalue of A^{-1} biggest in absolute value is actually the inverse of the eigenvalue of A that is smallest in absolute value. (WARNING: The norm of the inverse matrix A^{-1} is NOT the inverse of the norm of A, i.e. $\|A^{-1}\| \neq \frac{1}{\|A\|}$).

Hence for symmetric matrices

$$\text{cond}(A) = \frac{|\lambda_{max}(A)|}{|\lambda_{min}(A)|}$$

where $|\lambda_{min}| \leq |\lambda| \leq |\lambda_{min}|$ for all other eigenvalues λ.

For nonsymmetric matrices $\|A\|$ is the square root of the biggest eigenvalue of the symmetric matrix AA^t by Theorem 1.3. Similarly, $\|A^{-1}\|$ is obtained from the biggest eigenvalue of $A^{-1}(A^{-1})^t = (A^tA)^{-1}$, which is just the inverse of the smallest eigenvalue of A^tA. Recall (Remark 2. after Theorem 1.3) that the nonzero eigenvalues of AA^t and A^tA coincide. Hence we have

$$\text{cond}(A) = \sqrt{\frac{\lambda_{max}(AA^t)}{\lambda_{min}(AA^t)}} \tag{1.15}$$

In the example of Section 1.1.3 we have $\text{cond}(A) \approx 2984$ (see (1.2)). Despite the fact that both A and A^{-1} look "reasonable" matrices, the condition number of A is quite big, mainly because the smallest eigenvalue is near zero. This is why a small error in the right hand side could become a huge error in the solution.

Similar estimate is valid for the case when the matrix A is perturbed instead of $\mathbf{b}$. As before, let $A\mathbf{x} = \mathbf{b}$ be the original equation and let $(A + \Delta A)(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b}$ be the perturbed

equation with exact solutions. Then (PROVE IT (*))

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}+\delta\mathbf{x}\|} \leq \mathrm{cond}(A)\frac{\|\Delta A\|}{\|A\|}$$

For small enough ΔA it is expected that the ratio $\|\delta\mathbf{x}\|/\|\mathbf{x}+\delta\mathbf{x}\|$ is close to the relative error $\|\delta\mathbf{x}\|/\|\mathbf{x}\|$.

To summarize, we have:

CONCLUSION: When solving an equation $A\mathbf{x} = \mathbf{b}$, a small error either in $\mathbf{b}$ or in A can get multiplied by a factor of $\mathrm{cond}(A) = \|A\|\ \|A^{-1}\| \geq 1$ in the solution. If this number is big, then we say that the problem is *ill-conditioned*. In general a mathematical problem is ill-conditioned, if a small change in the input data yields a big change in the output.

REMARK: These arguments are just estimates on the error. In fact they are optimal in a sense that there is always an error which gets magnified by the amount $\mathrm{cond}(A)$. However, this is only the worst case scenario. Most errors do not get magnified so much. Play with the example in Section 1.1.3 by perturbing the data differently and find the exact solution. You will see that most perturbations actually do not cause a big deviation in the solution. This example was actually a carefully designed "worst-case" scanario. However, it can be shown that the condition number correctly estimates the error-amplification in the worst case scenario. In other words for any matrix A there is a perturbation which gets magnified by $\mathrm{cond}(A)$.

You might say that if "most" perturbations do not lead to disaster, then "most likely" you will never encounter with it and you do not have to worry about it. This is certainly an irresponsible attitude. Rare disasters are still disasters; if one plane crashes, it is not a satisfactory excuse to say that the other five hundred planes did not...

Problem 1.4 *Find the condition number of* $A = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}$

SOLUTION: Using formula (1.15) we need to compute the eigenvalues of

$$AA^t = \begin{pmatrix} 5 & 3 \\ 3 & 1 \end{pmatrix}$$

which are $\lambda_1 = 0.1459$ and $\lambda_2 = 6.854$, hence

$$\text{cond}(A) = \sqrt{\frac{6.854}{0.1459}} = 6.782$$