STAT 309: MATHEMATICAL COMPUTATIONS I
FALL 2019
LECTURE 7

- we will always have errors in our computations and when we store our inputs or intermediate quantities
- to ensure accuracy of our 'final answer', we need to be able to bound errors
- one great thing about numerical analysis is the following:
  (1) it allows you to compute approximately
  (2) but it also tells you how far away your approximation is from your true solution
- e.g. we can often say something like: this computed number agrees with the true solution up to 5 decimal digits
- the ability to do this is due primarily to two things:
  (1) backward error analysis
  (2) a standard for performing floating point arithmetic that respect certain rules
- failing to understand such issues can lead to serious problems:

`http://ta.twi.tudelft.nl/users/vuik/wi211/disasters.html`

1. ERRORS

- errors are of great importance in numerical computations because they allow us to quantify how far our computed solution is from the true solution that we are seeking
- this is important because we do everything in numerical computations in the presence of rounding errors but we still want to guarantee that the solution we found is accurate to some degree
- we usually use norms to measure the size of errors in multivariate quantities like vectors or matrices
- three commonly used measures of the error in an approximation $\widehat{\mathbf{x}}$ to a vector $\mathbf{x}$ are
  - the *absolute error*
  $$\varepsilon_{\text{abs}} = \|\mathbf{x} - \widehat{\mathbf{x}}\|$$
  - the *relative error*
  $$\varepsilon_{\text{rel}} = \frac{\|\mathbf{x} - \widehat{\mathbf{x}}\|}{\|\mathbf{x}\|}$$
  - the *point-wise error*
  $$\varepsilon_{\text{elem}} = \|\mathbf{y}\|, \quad y_i = \frac{\widehat{x}_i - x_i}{x_i}$$
  where we set $y_i = 0$ if the denominator is 0
- by the equivalence of norms, errors under different norms differ at most by constant multiples
- ditto if we have matrices in place of vectors
- we will often write $\Delta\mathbf{x} \coloneqq \mathbf{x} - \widehat{\mathbf{x}}$, $\Delta A \coloneqq A - \widehat{A}$, ..., and refer to these as the error

Date: October 19, 2019, version 1.2. Comments, bug reports: `lekheng@galton.uchicago.edu`.

- if a computer word has n bits, then we can represent 2^n numbers but which 2^n numbers should we choose?
- we will describe W. Kahan's choice of these 2^n numbers, this is called the IEEE 754 floating point standard and is universally used for decades (but there's now a competing choice[1])
- $F =$ floating numbers, a finite subset of $\mathbb{Q}$, essentially numbers representable as $\pm a_1.a_2a_3\cdots a_k \times 2^{e_1e_2\cdots e_l}$ where $a_i, e_j \in \{0, 1\}$
  - $\pm$ is called the *sign*
  - $a_1.a_2a_3\cdots a_k$, called the *mantissa*, is a nonnegative number expressed in *base* 2
  - $e_1e_2\cdots e_l$ is called the *exponent*, is an integer expressed in 2*'s complement* (which allows representation of both positive and negative integers)
  - a floating number where $a_0 \neq 0$ is called *normal*
  - a floating number where $a_0 = 0$ is called *subnormal*

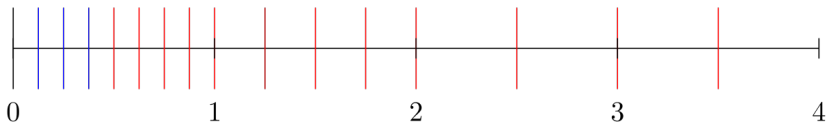

FIGURE 1. Subnormal numbers in blue, normal numbers in red. Note in particular that the set of floating point numbers is not regularly spaced.

- floating point representation:

$$\mathbb{R} \to F, \quad x \mapsto \mathrm{fl}(x)$$

- e.g. $\pi \mapsto \mathrm{fl}(\pi) = 3.1415926$
- special numbers like 0, $-\infty$, $+\infty$, `NaN` (not a number) requires special representation defined in the standard
- IEEE floating point standard (W. Kahan): defines a set F satisfying following properties
  - for every $x \in \mathbb{R}$, there exists $x' \in F$ such that $|x - x'| \leq \varepsilon_{\text{machine}}|x|$
  - for any $x, y \in F$,

$$\begin{aligned} \mathrm{fl}(x \pm y) &= (x \pm y)(1 + \varepsilon_1) \quad |\varepsilon_1| \leq \varepsilon_{\text{machine}} \\ \mathrm{fl}(xy) &= (xy)(1 + \varepsilon_2) \quad |\varepsilon_2| \leq \varepsilon_{\text{machine}} \\ \mathrm{fl}(x/y) &= (x/y)(1 + \varepsilon_3) \quad |\varepsilon_3| \leq \varepsilon_{\text{machine}} \end{aligned}$$

    in the last case $y \neq 0$ of course
- *machine epsilon*, a.k.a. unit roundoff, $\varepsilon_{\text{machine}} > 0$ is constant depending on computing machine used, usually defined as

$$\varepsilon_{\text{machine}} := \inf\{x \in \mathbb{R} : x > 0 \text{ and } \mathrm{fl}(1 + x) \neq 1\}$$

- caution: some people would define unit roundoff u as $\varepsilon_{\text{machine}}/2$ instead
- $\varepsilon_{\text{machine}}$ also gives an upper bound on the relative error due to rounding in floating point arithmetic
- in the IEEE floating point standard
  - floating point numbers are usually stored in the form

$\pm$	$e_1e_2\cdots e_l$	$a_1a_2\cdots a_k$

    requiring $1 + l + k$ bits
  - single precision is 32 bits

[1] `https://posithub.org`

    - 1 bit for sign, 8 bits for exponent, 23 bits for mantissa
    - allows storage of positive/negative floating numbers of 23 binary (around 7 decimal) digits with magnitude from $2^{-128} \approx 10^{-38}$ to $2^{128} \approx 10^{38}$
    - $\varepsilon_{\text{machine}} = 2^{-23} \approx 1.2 \times 10^{-7}$
  - double precision is 64 bits
    - 1 bit for sign, 11 bits for exponent, 52 bits for mantissa
    - allows for storage of positive/negative floating numbers of 52 binary (around 16 decimal) digits with magnitude from $2^{-1024} \approx 10^{-308}$ to $2^{1024} \approx 10^{308}$
    - $\varepsilon_{\text{machine}} = 2^{-52} \approx 2.2 \times 10^{-16}$
  - the latest standard (IEEE 754-2008) also defined extended precision (80 bits) and quad precision (128 bits)
- let's look at a toy example to get an idea of issues involved when dealing with floating point numbers
- for simplicity, let us assume a floating point system in base 10 (i.e., usual decimal numbers) with a 4 decimal digit mantissa and a 2 decimal digit exponent and that has no subnormal numbers, i.e., numbers of the form

$$\pm a_1.a_2a_3a_4 \times 10^e$$

  where $a_1 \in \{1, \dots, 9\}$, $a_2, a_3, a_4 \in \{0, 1, \dots, 9\}$, and $-99 \le e \le 99$
- some obvious problems

  (1) *roundoff errors*
  - storage: we can't store 1.1112×10^5 since it has a 5 digit mantissa, so

$$\text{fl}(1.1112 \times 10^5) = 1.111 \times 10^5$$

  - arithmetic: we can't store the result of the product of 1.111×10^1 and 1.111×10^2 since that requres a 7 digit mantissa

$$(1.111 \times 10^1) \times (1.111 \times 10^2) = 1.234321 \times 10^3$$

    and so

$$\text{fl}((1.111 \times 10^1) \times (1.111 \times 10^2)) = 1.234 \times 10^3$$

  (2) *overflows* and *underflows*
  - overflow: exponent too big

$$\text{fl}((1.000 \times 10^{55}) \times (1.000 \times 10^{50})) \to \texttt{overflow}$$

  - underflow: exponent too small

$$\text{fl}((1.000 \times 10^{-55}) \times (1.000 \times 10^{-50})) \to \texttt{underflow}$$

  (3) *cancellation errors*
  - if you try to compute

$$844487^5 + 1288439^5 - 1318202^5$$

    directly in MATLAB, a *numerical computing* software, you get zero as your answer — does that mean you have found a counterexample to Fermat's last theorem?
  - now try it on MATHEMATICA, a *symbolic computing* software, you get

$$844487^5 + 1288439^5 - 1318202^5 = -235305158626,$$

    which is far from zero

– the reason is cancellation error (here we need a 16-digit decimal mantissa to match what you get in MATLAB):

$$\mathrm{fl}(844487^5 + 1288439^5) = 3.980245235185639 \times 10^{30},$$
$$\mathrm{fl}(1318202^5) = 3.980245235185639 \times 10^{30},$$

i.e., the numbers have the same significant[2] digits in floating point representation

- issues like these require that we exercise care in designing numerical algorithms — avoid forming numbers that are too large or too small to minimize overflows; avoid subtracting two numbers that are almost equal to avoid cancellation errors
- an easy example is the evaluation of the vector 2-norm, the usual formula

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2}$$

gives a poor way of computing the value in the presence of rounding error

– assuming our toy model for F, take

$$\mathbf{x} = \begin{bmatrix} 10^{-49} \\ 10^{-50} \\ \vdots \\ 10^{-50} \end{bmatrix} \in \mathbb{R}^{101}$$

– this can be stored exactly in our toy floating point system and so there is no rounding error here

– but since $x_i^2 = 10^{-100}$ for $i = 2, \ldots, 101$,

$$\mathrm{fl}(x_2^2) = \cdots = \mathrm{fl}(x_{101}^2) = 0$$

and applying the usual formula in floating point arithmetic gives $\|\mathbf{x}\|_2 \approx 10^{-49}$ although $\|\mathbf{x}\|_2 = \sqrt{2 \times 10^{98}} \approx 1.1414 \times 10^{-49}$ — a 40% error

– a better algorithm would be a 2-step process

$$\widehat{\mathbf{x}} = \begin{cases} \mathbf{x}/\|\mathbf{x}\|_\infty & \text{if } \|\mathbf{x}\|_\infty \neq 0 \\ \mathbf{0} & \text{if } \|\mathbf{x}\|_\infty = 0 \end{cases}$$
$$\|\mathbf{x}\|_2 = \|\mathbf{x}\|_\infty \|\widehat{\mathbf{x}}\|_2$$

note that $|\widehat{x}_i| \leq 1$ for every $i = 1, \ldots, n$ and so there's no overflow; there's no underflow as long as none of the $|\widehat{x}_i|$ are much smaller than 1

- there are other less obvious examples due to all kinds of intricate errors in floating point computations, we will state two of them but won't go into the details

sample variance: if we want to compute the sample variance of n numbers $x_1, \ldots, x_n$, we could do it in either of the following ways:

(1) first compute sample mean

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

and then compute sample variance via

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})^2$$

[2] https://en.wikipedia.org/wiki/Loss_of_significance

(2) compute sample variance directly via

$$s^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right]$$

- mathematically the are equivalent and in exact arithmetic they should give identical values
- computationally both require the same number of operations but the first way requires two passes over the data while the second requires only one pass over the data, so it would appear that the second way is better
- in fact many statistics textbooks recommend the second way but it is actually a *very poor* way to compute sample variance in the presence of rounding error — the answer can even be negative
- the first way is much more accurate (and always nonengative) in floating point arithmetic

quadratic formula: the usual quadratic formula

$$x_1, x_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

is a poor way for computing the roots x_1, x_2 in floating point arithmetic

- a better way to compute them is via

$$x_1 = \frac{-(b + \operatorname{sign}(b)\sqrt{b^2 - 4ac})}{2a}, \qquad x_2 = \frac{c}{ax_1}$$

- reason again is cancellation error:

`https://en.wikipedia.org/wiki/Loss_of_significance`

3. BACKWARD ERROR ANALYSIS

- of course we can never compute any of these errors in reality since we do not know the exact solution $\mathbf{x}$ but the point is that:

we can bound these errors

- say we want to determine bounds on the error in a computed solution to $A\mathbf{x} = \mathbf{b}$ where $A \in \mathbb{R}^{n\times n}$ is nonsingular
- let $\mathbf{x}$ be exact solution, i.e., $\mathbf{x} = A^{-1}\mathbf{b}$ analytically,[3] and $\widehat{\mathbf{x}}$ be solution computed via floating-point arithmetic — therefore there will be rounding error in $\widehat{\mathbf{x}}$
- *backward error analysis* means we view $\widehat{\mathbf{x}}$ as the exact solution of the "nearby" system

$$(A + \Delta A)\widehat{\mathbf{x}} = \mathbf{b} + \Delta\mathbf{b}$$

  - if

$$\frac{\|\Delta A\|}{\|A\|} \le \varepsilon, \quad \frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|} \le \varepsilon$$

  - then

$$\frac{\|\mathbf{x} - \widehat{\mathbf{x}}\|}{\|\mathbf{x}\|} \le \frac{2\varepsilon}{1-\rho}\kappa(A) \tag{3.1}$$

    where

$$\rho = \|\Delta A\|\|A^{-1}\| = \|\Delta A\|\kappa(A)/\|A\|$$

  - precise statement is in (4.5)
- it is really relative error that we bound

[3] you should never ever compute inverse explicitly but using it in mathematical expressions is OK

  - absolute error $\|\mathbf{x} - \widehat{\mathbf{x}}\|$ is difficult to bound and is dependent on the choice of units of measurement
  - relative error $\|\mathbf{x} - \widehat{\mathbf{x}}\|/\|\mathbf{x}\|$ can be more readily bounded and is independent of units
- as was pointed out earlier, in either case we can't compute the error (absolute or relative) exactly since we don't know $\mathbf{x}$
- but it's enough to be able to *bound* errors: e.g. if we know that the error is less than 10^{-6}, we know our answer has at least 5 digits of accuracy
- the number
$$\kappa(A) = \|A\|\|A^{-1}\|$$
is the *condition number* of A — a singularly important notion
- why important? $\kappa(A)$ measures how an error in the system $A\mathbf{x} = \mathbf{b}$ is amplified in the solution
- even if ε is small, the computed solution can be useless if $\kappa(A)$ is large
- a system $A\mathbf{x} = \mathbf{b}$ where $\kappa(A)$ is large is an example of an *ill-conditioned* problem
- no algorithm, no matter how accurate, will be an effective tool for solving such an ill-conditioned problem
- it is important to distinguish between ill-conditioned problems from unstable algorithms
- informally, a problem or an algorithm is *stable* if a small change in its input yields a small change in its output
- ensuring that a problem is well-conditioned is the responsibility of the modeller, who formulates the mathematical problem from the original application
- ensuring the stability of an algorithm is the responsibility of the numerical analyst
- for a problem, the output is the exact solution, whereas for an algorithm, the output is the computed solution

4. Simple perturbation theory

- in homework 2, you will be asked to do a more accurate version of this analysis
- as an illustration, we will do a simplified version where we assume that the error occurs only in $\mathbf{b} \in \mathbb{R}^n$ but $A \in \mathbb{R}^{n\times n}$ is known exactly and is nonsingular
- let $\mathbf{x} \in \mathbb{R}^n$ be the unique exact solution to
$$A\mathbf{x} = \mathbf{b} \tag{4.1}$$
- $\mathbf{x}$ is the 'true solution' we seek and it's unique because A is nonsingular
- taking norms, we get
$$\|\mathbf{b}\| \leq \|A\|\|\mathbf{x}\|$$
where the norm on A is the operator norm
- hence
$$\frac{1}{\|\mathbf{x}\|} \leq \|A\|\frac{1}{\|\mathbf{b}\|} \tag{4.2}$$
- suppose the solution to (4.1) with the right-hand side perturbed to $\mathbf{b} + \Delta\mathbf{b}$ is given by[4] $\mathbf{x} + \Delta\mathbf{x}$
$$A(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{b} + \Delta\mathbf{b}$$
- then $A\Delta\mathbf{x} = \Delta\mathbf{b}$ and so $\Delta\mathbf{x} = A^{-1}\Delta\mathbf{b}$
- taking norms, we get
$$\|\Delta\mathbf{x}\| \leq \|A^{-1}\|\|\Delta\mathbf{b}\| \tag{4.3}$$

[4] note that this always works: if the solution is $\widehat{\mathbf{x}}$, then we just set $\Delta\mathbf{x} := \widehat{\mathbf{x}} - \mathbf{x}$

- combining (4.2) and (4.3), we get
$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \|A^{-1}\|\|A\|\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}$$
or
$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \kappa(A)\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}$$
- in this simple case, the relative error in $\mathbf{x}$ is bounded by the relative error in $\mathbf{b}$ scaled by the condition number of A
- suppose the error is only in A and $\mathbf{b}$ is known perfectly, i.e., the case
$$(A + \Delta A)(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{b}$$
- we can show that
$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\kappa(A)\dfrac{\|\Delta A\|}{\|A\|}}{1 - \kappa(A)\dfrac{\|\Delta A\|}{\|A\|}} \tag{4.4}$$
under some mild assumptions
- if the error is in both A and $\mathbf{b}$, i.e.,
$$(A + \Delta A)(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{b} + \Delta\mathbf{b}$$
- we can show that
$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\kappa(A)\left(\dfrac{\|\Delta A\|}{\|A\|} + \dfrac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}\right)}{1 - \kappa(A)\dfrac{\|\Delta A\|}{\|A\|}} \tag{4.5}$$
under some mild assumptions
- (4.4) and (4.5) will be in homework 2
- we will discuss such *backward error analysis* for other problems and in greater detail in the next lecture

5. Condition number of a matrix

- we defined the 2-norm condition number for a nonsingular square matrix $A \in \mathbb{C}^{n\times n}$ as
$$\kappa_2(A) = \|A\|_2\|A^{-1}\|_2 \tag{5.1}$$
- what if A is singular? one way is to set $\kappa_2(A) = \infty$
- this is natural though not very useful — the only information it conveys is what you already know, namely, A is singular
- if we apply SVD of A and the unitary invariance of the 2-norm, then an alternative expression for (5.1) is
$$\kappa_2(A) = \frac{\sigma_1(A)}{\sigma_n(A)} = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}$$
- note that $\operatorname{rank}(A) = n$ and we could have written
$$\kappa_2(A) = \frac{\sigma_1(A)}{\sigma_{\operatorname{rank}(A)}(A)} = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)} \tag{5.2}$$
where $\sigma_{\min}(A)$ denotes the smallest non-zero singular value of A
- this last expression extends to any singular and even rectangular $A \in \mathbb{C}^{m\times n}$ as long as $A \neq O$
- note that $\sigma_{\operatorname{rank}(A)}(A)$ is the smallest non-zero singular value of A
- we call (5.2) the *generalized condition number* to distinguish it from (5.1)

- another expression for (5.2) is

$$\kappa_2(A) = \|A\|_2\|A^\dagger\|_2 \tag{5.3}$$

- proof: use SVD to see that $\|A\|_2 = \sigma_1(A)$ and $\|A^\dagger\|_2 = \sigma_{\operatorname{rank}(A)}(A)$
- (5.3) can be used to extend generalized condition number to any matrix norm, for example

$$\kappa_p(A) = \|A\|_p\|A^\dagger\|_p, \quad \kappa_F(A) = \|A\|_F\|A^\dagger\|_F$$

6. Condition number of a problem

- a *well-posed* instance of a problem is one whose existence and uniqueness of solutions hold, otherwise the instance is *ill-posed*
- for example for the problem $A\mathbf{x} = \mathbf{b}$,

$$A = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

  is an ill-posed instance because a solution does not exist; and

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{6.1}$$

  is an ill-posed instance because solutions are not unique; but

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{6.2}$$

  is a well-posed instance
- you will often hear of people speaking of "ill-posed problem" and "well-posed problem" — what they really mean are "ill-posed instance of a problem" and "well-posed instance of a problem"
- the notion of conditioning is a further refinement of the notion of well-posedness, for example, the instance in (6.2) and the instance

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 10^{-1000} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{6.3}$$

  are both well-posed but intuitively we know that (6.3) is worse because the coefficient matrix

$$\begin{bmatrix} 1 & 0 \\ 0 & 10^{-1000} \end{bmatrix} \approx \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

  is nearly singular and so this instance is very close to the ill-posed instance (6.1)
- in fact, in floating point arithmetic of any realistic precision,

$$\operatorname{fl}\left(\begin{bmatrix} 1 & 0 \\ 0 & 10^{-1000} \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

- the respective condition numbers of the coefficient matrices are

$$\kappa\left(\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}\right) = 2, \qquad \kappa\left(\begin{bmatrix} 1 & 0 \\ 0 & 10^{-1000} \end{bmatrix}\right) = 10^{1000}, \qquad \kappa\left(\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\right) = \infty$$

- the first is *well-conditioned*, the second is *ill-conditioned*, both are *well-posed*; but the third is *ill-posed*
- more generally, the condition number of an instance of a problem is the reciprocal of the normalized distance to the nearest ill-posed instance

- for example, if the problem is solving a linear system with a *fixed* right-hand side $\mathbf{b}$, or, equivalently, matrix inversion , i.e.,
$$f : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}, \qquad f(X) = X^{-1},$$
then the condition number of a problem instance $A \in \mathbb{R}^{n\times n}$ is
$$\kappa(A) = \begin{cases} \|A\|\|A^{-1}\| & A \text{ invertible;} \\ \infty & \text{otherwise} \end{cases}$$
- why? because the set of ill-posed problem is the set of singular matrices $\mathcal{M} = \{X \in \mathbb{R}^{n\times n} : \det(X) = 0\}$ and the distance of any nonsingular A to this set is (exercise: verify this for the matrix 2-norm)
$$\operatorname{dist}(A, \mathcal{M}) := \min_{X\in\mathcal{M}} \|A - X\| = \frac{1}{\|A^{-1}\|}$$
and so the normalized distance is
$$\frac{\operatorname{dist}(A, \mathcal{M})}{\|A\|} = \frac{1}{\|A\|\|A^{-1}\|} = \frac{1}{\kappa(A)},$$
the reciprocal of the usual condition number
- in other words, the "condition number of a matrix" that we defined earlier really comes from the condition number of the problem of matrix inversion or the problem of solving linear systems with fixed right-hand side
- we can also do this for the problem of solving for a minimum norm least squares problem with a fixed $\mathbf{b}$, which is equivalent to finding pseudoinverse
$$f : \mathbb{R}^{m\times n} \to \mathbb{R}^{n\times m}, \qquad f(X) = X^{\dagger}$$
- in this case, the set of ill-posed problem is the set of rank-deficiency matrices $\mathcal{M} = \{A \in \mathbb{R}^{m\times n} : \operatorname{rank}(A) < \min(m, n)\}$
- the condition number of a problem instance $A \in \mathbb{R}^{m\times n}$ is
$$\frac{\operatorname{dist}_F(A, \mathcal{M})}{\|A\|_F} = \min_{X\in\mathcal{M}} \|A - X\|_F = \frac{\sigma_{\min}(A)}{\sigma_{\max}(A)} = \frac{1}{\kappa_F(A)},$$
- in other words, the "generalized condition number of a matrix" that we defined earlier really comes from the condition number of the problem of computing pseudoinverse or the problem of solving minimum norm least squares problem with a fixed $\mathbf{b}$
- there are many others: linear system, least squares, linear programming, eigenvalue problems, polynomial eigenvalue problems
- for example, for linear programming, the condition number is given by
$$\frac{1}{\kappa_2(A, \mathbf{b})} = \frac{\operatorname{dist}_2([A, \mathbf{b}], \mathcal{M})}{\|[A, \mathbf{b}]\|_2}$$
where $\mathcal{M}$ = boundary of feasible pairs $(A, \mathbf{b}) \in \mathbb{R}^{m\times(n+1)}$
- in the next lecture, we will discuss the condition number for eigenvalue problems

7. Why orthogonal/unitary

- unitary and orthogonal matrices are awesome because they preserve length
- it also preserves the length of your errors and so your errors don't get magnified during your computations
- more precisely, if we multiply a vector $\mathbf{a} \in \mathbb{C}^n$ or a matrix $A \in \mathbb{C}^{n\times k}$ by another matrix $X \in \mathrm{GL}(n)$ we usually magnify whatever error there is in $\mathbf{a}$ or A by $\kappa_2(X)$, the condition number of X

- more precisely, unitary and orthogonal matrices are awesome because they are perfectly conditioned, i.e., $\kappa_2(U) = 1$ for all $U \in U(n)$ (but converse is not true)
- the vector case $\mathbf{a} \in \mathbb{C}^n$ is the same as the matrix case $A \in \mathbb{C}^{n\times k}$ with $k = 1$ so we will do the more general one
- for simplicity, let us assume that we know X precisely but
  - we don't have XA, only $\mathrm{fl}(XA)$, which differs from XA by an error term E

$$\mathrm{fl}(XA) = XA + E$$

  - we have also assumed that all errors arise from rounding in floating point arithmetic and storage
- we will do a backward error analysis again, i.e., we want to find the smallest perturbation ΔA in A so that $XA + E$ is the *exact* answer had $A + \Delta A$ been the input
- we will measure how good our method is by asking what is the relative error in the input

$$\frac{\|\Delta A\|_2}{\|A\|_2} \tag{7.1}$$

  required so that the relative error of the output is

$$\frac{\|E\|_2}{\|XA\|_2} \leq \varepsilon \tag{7.2}$$

  for some $\varepsilon > 0$
- terminologies: the ratio in (7.1) is called the *relative backward error*, the ratio in (7.2) is called the *relative forward error*
- in this case, it is trivial to derive the relative backward error: by assumption $XA + E$ is the *exact* answer of multiplying X to $A + \Delta A$, so

$$XA + E = X(A + \Delta A)$$

  and so

$$\Delta A = X^{-1}E$$

- from (7.2), we get $\|E\|_2 \leq \varepsilon\|XA\|_2 \leq \varepsilon\|X\|_2\|A\|_2$ and so

$$\|\Delta A\|_2 \leq \|X^{-1}\|_2\|E\|_2 \leq \varepsilon\kappa_2(X)\|A\|_2$$

  and so the relative backward error is

$$\frac{\|\Delta A\|_2}{\|A\|_2} \leq \varepsilon\kappa_2(X) \tag{7.3}$$

- this may seem a little wierd the first time you see it: why don't we assume that the error is in the input and then see how big it becomes in the output — this is called forward error analysis
- forward error analysis is in general much hard than backward error analysis
- recap of backward error analysis
  - we assume that the error E in the final computed output comes from the *exact* solution of a perturbed problem $A + \Delta A$
  - we start by assuming that the relative error in the output is ε, i.e., (7.2)
  - then we try to find how far away (i.e., ΔA) the input must be from the given one (i.e., A) in order to produce such an error ε in the output, i.e., (7.3), when everything is done without error
- we will cover backward error analysis and condition number in greater details later

- rounding errors also makes the exact rank of a matrix difficult to determine and far less useful than in pure math
- note that matrix rank is a discrete notion that is sometimes too imprecise, for example both

$$\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 0 \\ 0 & 10^{-1000} \end{bmatrix}$$

  have rank 2
- another example: take a randomly generated vector $\mathbf{x} \sim N(\mathbf{0}, I_n)$ and consider the $n \times n$ matrices

$$X = [\mathbf{x}, 2\mathbf{x}, \dots, n\mathbf{x}] \quad \text{and} \quad \text{fl}(X) = [\text{fl}(\mathbf{x}), \text{fl}(2\mathbf{x}), \dots, \text{fl}(n\mathbf{x})]$$

- in the presence of rounding error, we will get

$$\text{rank}(X) = 1 \quad \text{and} \quad \text{rank}(\text{fl}(X)) = n$$

- the singular values are much more informative

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$$

- the profile or decay rate of these can often tell us the 'true rank' of a matrix
- exercise: plot the singular value profile of $\text{fl}(X)$ in MATLAB
- this is the notion of what is often called *numerical rank*
- SVD tells us about numerical rank
- but some folks insist that numerical rank of a matrix must be a number like rank, not a decomposition
- there are several proposals on how it could be defined, three of the most common ones are defined as follows
- let $\tau > 0$ be some predetermined tolerance level (in practice a small number ≈ 0.1 or $\tau = \max(m, n) \times \varepsilon_{\text{machine}}$) and $A \in \mathbb{C}^{m \times n}$ be a non-zero matrix
- the term *numerical rank* of A have variously been given to
  - the positive integer

$$\sigma\,\text{rank}(A) := \min\left\{ r \in \mathbb{N} : \frac{\sigma_r(A)}{\sigma_1(A)} \geq \tau \right\}$$

  - the positive integer

$$\rho\,\text{rank}(A) := \min\left\{ r \in \mathbb{N} : \frac{\sigma_{r+1}(A)}{\sigma_r(A)} \leq \tau \right\}$$

  - or the positive integer

$$\mu\,\text{rank}(A) := \min\left\{ r \in \mathbb{N} : \frac{\sum_{i \geq r+1} \sigma_i(A)^2}{\sum_{i \geq 1} \sigma_i(A)^2} \leq \tau \right\}$$

  - or the positive real number

$$\nu\,\text{rank}(A) = \frac{\|A\|_F^2}{\|A\|_2^2} = \frac{\sum_{i=1}^{\min(m,n)} \sigma_i(A)^2}{\sigma_1(A)^2}$$

  depending on the application