STAT 309: MATHEMATICAL COMPUTATIONS I
FALL 2015
LECTURE 3

1. errors

- three commonly used measures of the error in an approximation $\hat{\mathbf{x}}$ to a vector $\mathbf{x}$ are
  - the *absolute error*
$$\varepsilon_{\text{abs}} = \|\mathbf{x} - \hat{\mathbf{x}}\|$$
  - the *relative error*
$$\varepsilon_{\text{rel}} = \frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|}$$
  - the *point-wise error*
$$\varepsilon_{\text{elem}} = \|\mathbf{y}\|, \quad y_i = \frac{\hat{x}_i - x_i}{x_i}$$
    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

2. perturbation theory and backward error analysis

- 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[1], and $\mathbf{y}$ be solution computed via floating-point arithmetic — therefore there will be rounding error in $\mathbf{y}$
- backward error analysis: view $\mathbf{y}$ as the exact solution of the "nearby" system
$$(A + \delta A)\mathbf{y} = \mathbf{b} + \delta\mathbf{b}$$
  - if
$$\frac{\|\delta A\|}{\|A\|} \leq \epsilon, \quad \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} \leq \epsilon$$
  - then
$$\frac{\|\mathbf{x} - \mathbf{y}\|}{\|\mathbf{x}\|} \leq \frac{2\epsilon}{1-\rho}\kappa(A) \tag{2.1}$$
    where
$$\rho = \|\delta A\|\|A^{-1}\| = \|\delta A\|\kappa(A)/\|A\|$$
- it is really relative error that we bound
  - absolute error $\|\mathbf{x} - \mathbf{y}\|$ is difficult to bound and is dependent on the choice of units of measurement
  - relative error $\|\mathbf{x} - \mathbf{y}\|/\|\mathbf{x}\|$ can be more readily bounded and is independent of units
- note that in either case we can't compute the error (absolute or relative) exactly since we don't know $\mathbf{x}$

Date: October 10, 2015, version 1.0. Comments, bug reports: `lekheng@galton.uchicago.edu`.

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

- 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
- in homework 1, 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
$$A\mathbf{x} = \mathbf{b} \tag{2.2}$$
- $\mathbf{x}$ 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
$$\frac{1}{\|\mathbf{x}\|} \leq \|A\|\frac{1}{\|\mathbf{b}\|} \tag{2.3}$$
- suppose the solution to (2.2) with the right-hand side perturbed to $\mathbf{b} + \delta\mathbf{b}$ is given by[2] $\mathbf{x} + \delta\mathbf{x}$
$$A(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}$$
- then $A\delta\mathbf{x} = \delta\mathbf{b}$ and so $\delta\mathbf{x} = A^{-1}\delta\mathbf{b}$
- taking norms, we get
$$\|\delta\mathbf{x}\| \leq \|A^{-1}\|\|\delta\mathbf{b}\| \tag{2.4}$$
- combining (2.3) and (2.4), 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

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

- 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
$$\frac{\|\delta \mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\kappa(A) \dfrac{\|\delta A\|}{\|A\|}}{1 - \kappa(A) \dfrac{\|\delta A\|}{\|A\|}} \tag{2.5}$$
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
$$\frac{\|\delta \mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\kappa(A) \left(\dfrac{\|\delta A\|}{\|A\|} + \dfrac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|} \right)}{1 - \kappa(A) \dfrac{\|\delta A\|}{\|A\|}} \tag{2.6}$$
under some mild assumptions
- (2.5) and (2.6) will be in homework 1

3. Floating point numbers

- we will always have errors in our computations and representations
- 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) the backward error analysis in the previous lecture
  (2) a standard for performing floating point arithmetic that respect certain rules
- $F =$ floating numbers, a finite subset of $\mathbb{Q}$, essentially numbers representable as $\pm a_1.a_2a_3 \cdots a_k \times 2^{e_1 e_2 \cdots e_l}$ 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*

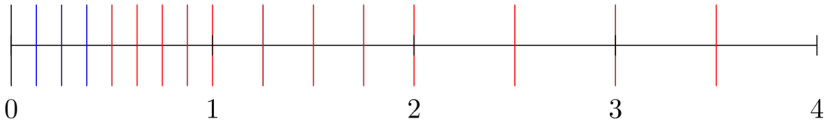

Figure 1. Subnormal numbers in blue, normal numbers in red. Note in particular that the set of floating point numbers is not regularly spaced.

- 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

$\pm$	$e_1 e_2 \cdots e_l$	$a_1 a_2 \cdots a_k$

    requiring $1 + l + k$ bits
  - single precision is 32 bits
    - 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 $a_1 \in \{1, \ldots, 9\}$, $a_2, a_3, a_4 \in \{0, 1, \ldots, 9\}$, and $-99 \leq e \leq 99$
- two obvious problems
  (1) *roundoff errors*
    - storage: we can't store 1.1112×10^5 since it has a 5 digit mantissa, so

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

– 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

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

(2) *overflows* and *underflows*

– overflow: exponent too big

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

– underflow: exponent too small

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

- issues like these require that we exercise care in designing numerical algorithms
- 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, 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) 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

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