Preliminaries to Numerical Analysis

num. linear algebra

- Square linear systems: $A\vec{x} = \vec{b}$ (polynomial interpolation)
- Overdetermined linear systems: $A\vec{x}$ "=" $\vec{b}$ (polynomial approximation)
- spectral theory (eigenvalues & singular values)

nonlinear

- nonlinear rootfinding
- piecewise polynom. interp.
- num. calculus (diff., integ., ODE theory)

Contents

Main sources of errors

1. Floating-Point Numbers (how #'s are represented or stored on computer w/ finite memories)

2. Conditioning (inherent nature of a problem)

3. Stability (inherent prop. of computer algorithm.)

Absolute and Relative Errors

In numerical analysis, we use an **algorithm** to *approximate* some quantity of interest.

- We estimate of the accuracy of the computed value via an **absolute error** or a **relative error**:

$$e_{\text{abs}} = A_{\text{approx}} - A_{\text{exact}} \qquad \text{(absolute error)}$$

$$e_{\text{rel}} = \frac{A_{\text{approx}} - A_{\text{exact}}}{A_{\text{exact}}} = \frac{A_{\text{approx}}}{A_{\text{exact}}} - 1\,, \qquad \text{(relative error)}$$

"percentage" error

• dimensionless

  where A_{exact} is the exact, analytical answer and A_{approx} is the approximate, numerical answer.

- If e_{abs} or e_{rel} is small, we say that the approximate answer is **accurate**.

Example: Stirling's Formula

Stirling's formula provides a "good" approximation to $n!$ for large n:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n . \qquad (\star)$$

Try in MATLAB:

```
n = ...;
err_abs = sqrt(2*pi*n)*(n/exp(1))^n - factorial(n);
err_rel = err_abs/factorial(n);
disp(err_abs)
disp(err_rel)
```

When $n = 10$:

Abs. err. $= -30104.\cdots$

Rel. err. $= -0.008296$ (0.8%)

When $n = 100$:

Abs. err. $= -7.77\ldots \times 10^{154}$

Rel. err. $= -0.00083298$ (0.08%)

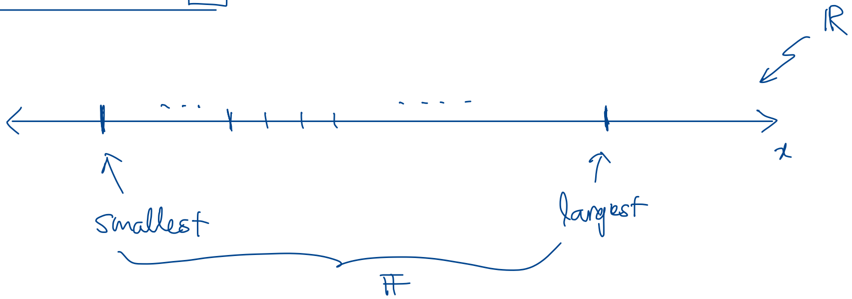
Floating-Point Numbers

Limitations of Digital Representations

A digital computer uses a finite number of bits to represent a real number and so it cannot represent all real numbers.

- The represented numbers cannot be arbitrarily large or small;
- There must be gaps between them.

So for all operations involving real numbers, it uses a subset of $\mathbb{R}$ called the **floating-point numbers**, $\boxed{\mathbb{F}.}$



Scientific notation (base 10)

Ignore signs; consider positive numbers for now.

cannot be 0

Ex

$$\boxed{3}.14 \times 10^{5} = 314000$$

$$3.14 \times 10^{-2} = 0.0314$$

$$\left(3 + 1\cdot 10^{-1} + 4\cdot 10^{-2}\right) \times 10^{5}$$

$$= 3\cdot 10^{5} + 1\cdot 10^{4} + 4\cdot 10^{3}$$

$$\begin{aligned} = \ & 300000 \\ + \ & \ \ 10000 \\ + \ & \ \ \ \ 4000 \\ = \ & \underline{314000} \end{aligned}$$

Scientific notation (base 2)

- Only binary bits (0 or 1)
- expansion in powers of 2.

cannot be 0; always 1.

Ex

$$\boxed{1}.01_{(2)} \times 2^{1_{(2)}} = 2.5$$

$$1.11_{(2)} \times 2^{-11_{(2)}} = \boxed{}$$

$$1.01_{(2)} \times 2^{1_{(2)}} = 10.1_{(2)}$$

$$= \left(1\cdot 2^{0} + 0\cdot 2^{-1} + 1\cdot 2^{-2}\right) \times 2^{1}$$

$$= 1\cdot 2^{1} + 0\cdot 2^{0} + 1\cdot 2^{-1}$$

$$= 2 + 0 + \frac{1}{2} = 2.5$$

Floating-Point Numbers

base-2 scientific notation w/ given # of ~~digits~~ bits.

A *floating-point number* is written in the form $\pm(1+F)2^E$ where

- E, the *exponent*, is an integer;
- F, the *mantissa*, is a number $F = \sum_{i=1}^{d} b_i 2^{-i}$, with $b_i = 0$ or $b_i = 1$.

$\to$ # of digits after .

expansion in neg. powers of 2.

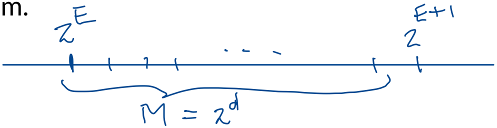
$$1.\underbrace{\boxed{11_{(2)}}}_{=F} \times 2^{\overbrace{\boxed{10_{(2)}}}^{E=}} \quad (d=2)$$

Note that F can be rewritten as

$$F = 2^{-d} \underbrace{\sum_{k=0}^{d-1} b_{d-k} 2^k}_{=:M},$$

where M is an integer in $\mathbb{N}[0, 2^d - 1]$. $= \{0, 1, 2, \cdots, 2^d - 1\}$

Consequently, there are 2^d evenly-spaced numbers between 2^E and 2^{E+1} in the floating-point number system.



Example $d = 2$ consider $(+)$ #'s only.

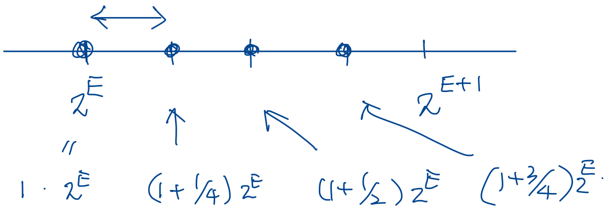
$$x = (1 + F)\, 2^E$$

$$= 1.\square\square_{(2)} \times 2^E$$

$\frac{1}{2}$	$\frac{1}{4}$	F
0	0	0
0	1	$\frac{1}{4}$
1	0	$\frac{1}{2}$
1	1	$\frac{3}{4}$

"resolution" "location"

$$\text{Gap} = \frac{1}{4} 2^E = 2^{-2}\, 2^E = 2^{-d}\, \boxed{2^E}$$



$$2^d = 2^2 = 4 \text{ FP \#'s on } [2^E, 2^{E+1})$$

Floating-Point Numbers – IEEE 754 Standard

$-1022 \le E \le 1023$

$E = -1023$ for 0 ; $E = 1024$ for "Inf"

- MATLAB, by default, uses *double precision* floating-point numbers, stored in memory in 64 bits (or 8 bytes):

S	$(-1)^S$
0	+
1	−

$$\pm\, \underbrace{1.\text{xxxxxxxx}\cdots\text{xxxxxxxx}_{(2)}}_{\text{mantissa (base 2): 52+1 bits}} \times 2^{\overbrace{\text{xxxx}\cdots\text{xxxx}_{(2)}}^{\text{exponent: 11 bits}} - 1023}.$$

offset

"d" "sign bit"

$$\underbrace{11\cdots1_{(2)}}_{11} = 1 + 2^1 + 2^2 + \cdots + 2^{10} = \frac{2^{11}-1}{2-1} = 2048 - 1 = 2047$$

$2^0 = 1$

- Predefined variables:
  - `eps` = the distance from 1.0 to the next ~~largest~~ double-precision number:

Take E=0

$$\text{eps} = 2^{-52} \approx 2.2204 \times 10^{-16}.$$

  - `realmin` = the smallest positive floating-point number that is stroed to full accuracy; the actual smallest is `realmin/2^52`.
  - `realmax` = the largest positive floating-point number

→ anything smaller ↦ 0 (underflow)

→ anything larger ↦ Inf (overflow)

realmax

$$1.\underbrace{11\cdots1}_{52}{}_{(2)} \times 2^{1023}$$

realmin

$$1.00\cdots0_{(2)} \times 2^{-1022}$$

$$\hat{x} = x(1+\epsilon_1)$$

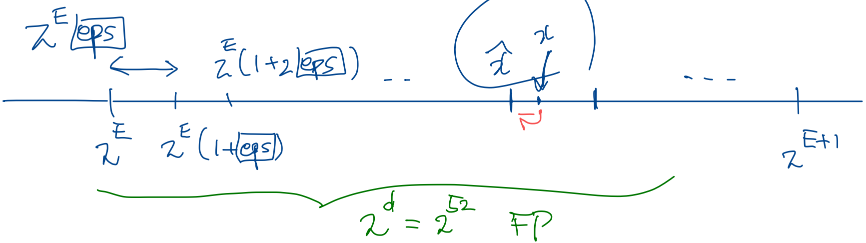
$$\hat{x} = x + x\epsilon_1$$

$$\frac{\hat{x}-x}{x} = \epsilon_1 \quad \text{(relative error)}$$

Suppose $\underline{2^E \le x} < 2^{E+1}$

$$\begin{cases} |x - \hat{x}| \le \frac{1}{2} 2^E \boxed{eps} \\ x \ge 2^E \end{cases}$$

$$\left|\frac{x-\hat{x}}{x}\right| \le \frac{1}{2}\,\frac{\cancel{2^E}\boxed{eps}}{\cancel{2^E}}$$



Machine Epsilon and Relative Errors

eps $= 2^{-52}$ ← # of bits of mantissa

$\approx 2.2 \times 10^{-16}$

The IEEE standard guarantees that the *relative representation error* and the *relative computational error* have sizes smaller than $\boxed{\text{eps}}$, the *machine epsilon*:

- **Representation**: The floating-point representation, $\hat{x} \in \mathbb{F}$, of $x \in \mathbb{R}$ satisfies

percentage error, i.e., relative error

$$\hat{x} = x(1 + \epsilon_1), \qquad \text{for some } |\epsilon_1| \leqslant \frac{1}{2}\boxed{\text{eps}}.$$

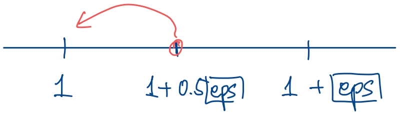
- **Arithmetic**: The floating-point representation, $\hat{x} \oplus \hat{y}$, of the result of $\hat{x} + \hat{y}$ with $\hat{x}, \hat{y} \in \mathbb{F}$ satisfies

$$\hat{x} \oplus \hat{y} = (\hat{x} + \hat{y})(1 + \epsilon_2), \qquad \text{for some } |\epsilon_2| \leqslant \frac{1}{2}\boxed{\text{eps}}.$$

Similarly with $\ominus, \otimes, \oslash$ corresponding to $-, \times, \div$, respectively.

Round-Off Errors

" # in the middle gets rounded down "



Computers CANNOT usually

- represent a number correctly;
- add, subtract, multiply, or divide correctly!!

Run the following and examine the answers:

```
format long
1.23456789012345678 90
12345678901234567890
(1 + eps) - 1          :  eps
(1 + .5*eps) - 1       :  0
(1 + .51*eps) - 1      :  eps
n = input(' n = '); ( n^(1/3) )^3 - n
```

$1 + 0.5\,\boxed{eps} \longrightarrow 1$

$- 1 \longrightarrow 0$

$1 + 0.51\,\boxed{eps} \longrightarrow 1 + \boxed{eps}$

$- 1 \longrightarrow \boxed{eps}$

Catastrophic Cancellation

In finite precision storage, two numbers that are close to each other are indistinguishable. So subtraction of two nearly equal numbers on a computer can result in loss of many significant digits.

Catastrophic Cancellation

Consider two real numbers stored with 10 digits of precision:

$$\begin{aligned} e &= 2.7182818284, \\ b &= 2.7182818272. \end{aligned}$$

- Suppose the actual numbers e and b have additional digits that are not stored.
- The stored numbers are good approximations of the true values.
- However, if we compute $e - b$ based on the stored numbers, we obtain $0.0000000012 = 1.2 \times 10^{-9}$, a number with only two significant digits.

Example 1: Cancellation for Large Values of x

Question

Compute $f(x) = e^x \underbrace{(\cosh x - \sinh x)}_{= e^{-x}}$ at $x = 1, 10, 100$, and 1000.

NaN (not a number)

Numerically:

$f(x) = e^x \cdot e^{-x} = 1$.

```
format long
x = input(' x = ');
y = exp(x) * ( cosh(x) - sinh(x) );
disp([x, y])
```

$$\begin{cases} \cosh x = \dfrac{e^x + e^{-x}}{2} \\ \sinh x = \dfrac{e^x - e^{-x}}{2} \end{cases}$$

for large x, e^{-x} is negligible.

So $\cosh x \approx \sinh x$

Example 2: Cancellation for Small Values of x

Question

Compute $f(x) = \dfrac{\sqrt{1+x}-1}{x}$ at $x = 10^{-12}$.

Numerically:

- $\sqrt{1+x} \approx 1 + \frac{1}{2}x$ for small x. (binomial series; Taylor series)

```
x = 1e-12;
fx = (sqrt(1+x) - 1)/x;
disp( fx )
```

- For small x, $\sqrt{1+x} \approx 1$

$$f(x) = \frac{\sqrt{1+x}-1}{x} \cdot \frac{\sqrt{1+x}+1}{\sqrt{1+x}+1} = \frac{(1+x)-1}{x(\sqrt{1+x}+1)} = \frac{x}{x(\sqrt{1+x}+1)} = \frac{1}{\sqrt{1+x}+1}$$

To Avoid Such Cancellations …

- Unfortunately, there is no universal way to avoid loss of precision.
- One way to avoid catastrophic cancellation is to remove the source of cancellation by simplifying the given expression before computing numerically.
- For Example 1, rewrite the given expression recalling that

$$\cosh x = (e^x + e^{-x})/2 \qquad \sinh x = (e^x - e^{-x})/2.$$

- For Example 2, try again after rewriting $f(x)$ as

$$f(x) = \frac{\sqrt{1+x} - 1}{x} \cdot \frac{\sqrt{1+x}+1}{\sqrt{1+x}+1} = \frac{1}{\sqrt{1+x}+1}.$$

- Do you now have an improved accuracy?

nature of problem

Conditioning

Errors A_{exact} , A_{approx}.

$$\begin{cases} \text{Abs. err.} = A_{\text{exact}} - A_{\text{approx}} \\ \text{Rel. err.} = \dfrac{A_{\text{exact}} - A_{\text{approx}}}{A_{\text{exact}}} \end{cases}$$

(percentage error ; dimensionless)

Floating-point numbers ($\mathbb{F}$)

- base-2 scientific notation

$$\left.\underbrace{\pm}_{1} 1 . \underbrace{xx\cdots x}_{52}{}_{(2)} \times 2^{\overbrace{x\cdots\cdots x}^{11}{}_{(2)} - 1023}\right\} \text{64 bits}$$

$-1022 \leq E \leq 1023$

abstractly written as $\pm(1+F)\, 2^{E}$

$0 \leq F < 1$

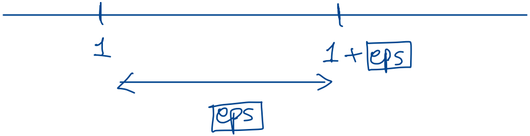
Code / writing

- machine epsilon (eps / $\boxed{eps}$)

  - $\boxed{eps} = 2^{-52} \approx 2.22 \times 10^{-16}$

  - Significance?

    1. Gap btw 1 and its successor.



    2. $\left| \dfrac{x - \hat{x}}{x} \right| \le \dfrac{1}{2} \boxed{eps}$

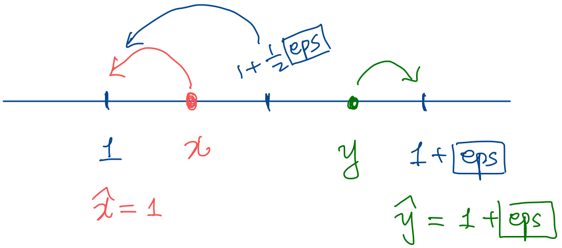
- Roundoff error.

>> (1 + 0.5 * eps) - 1

ans = 0

>> (1 + 0.51 * eps) - 1

ans = 2.2 ... e-16 (eps)



<u>Convention</u> : Midpoint → round down

– Distribution of F-P numbers

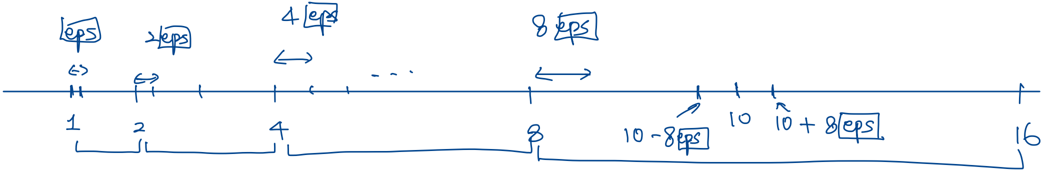
On $[2^E, 2^{E+1})$

- there are $2^d = 2^{52}$ f-p #'s
- they are uniformly spaced
- the gap b/w two adjacent f-p #'s <u>here</u>

  is $\underline{2^E \boxed{eps}} = 2^E 2^{-d} = 2^{E-52}$

<u>Ex</u> $E=0$, $[1,2)$: Gap $= 2^0 \boxed{eps} = \boxed{eps}$

$E=1$, $[2,4)$: Gap $= 2^1 \boxed{eps} = 2 \boxed{eps}$ . . .



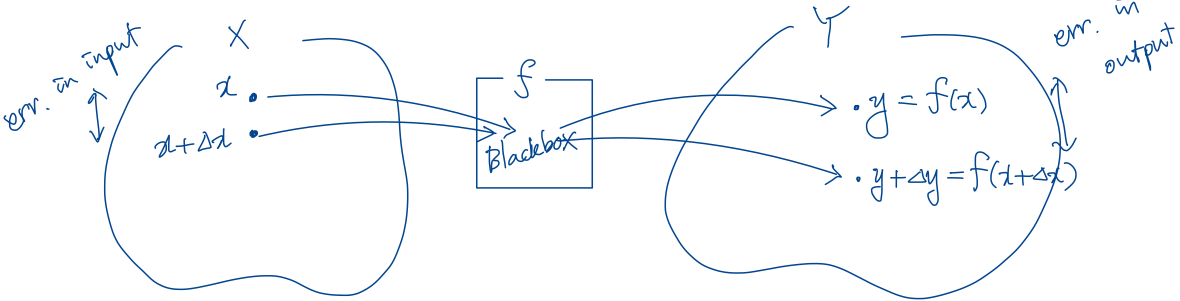
$$\ldots,\ 8 - 4\,\boxed{eps},\ \overset{2^3}{\overset{\|}{8}},\ 8 + 8\,\boxed{eps},\ \ldots$$

$$\ldots,\ 10 - 8\,\boxed{eps},\ 10,\ 10 + 8\,\boxed{eps},\ \ldots$$

Problems and Conditioning

inputs / data outputs / soln.

- A mathematical *problem* can be viewed as a function $f : X \to Y$ from a data/input space X to a solution/output space Y.
- We are interested in changes in $f(x)$ caused by small perturbations of x.
- A *well-conditioned* problem is one with the property that all small perturbations of x lead to only small changes in $f(x)$



Condition Number

perturbed input (e.g. f-p rep'n) true input

Let $f : \mathbb{R} \to \mathbb{R}$ and $\hat{x} = x(1 + \epsilon)$ be the representation of $x \in \mathbb{R}$.

relative error

- The ratio of the relative error in f due to the change in x to the relative error in x simplifies to

$$\frac{\text{rel. err. output}}{\text{rel. err. input}} = \frac{\left|\frac{f(\hat{x}) - f(x)}{f(x)}\right|}{\left|\frac{\hat{x} - x}{x}\right|} = \frac{\left|f(x) - f(x(1+\epsilon))\right|}{\left|\epsilon f(x)\right|}.$$

- In the limit of small error (ideal computer), we obtain

$$\kappa_f(x) := \lim_{\epsilon \to 0} \frac{\left|f(x) - f(\underbrace{x(1+\epsilon)}_{x + \epsilon x})\right|}{\left|\epsilon f(x)\right|}$$

$$= \left|\lim_{\epsilon \to 0} \frac{f(x + \epsilon x) - f(x)}{\epsilon x} \cdot \frac{x}{f(x)}\right| = \left|\frac{x f'(x)}{f(x)}\right|, \quad (\star)$$

which is called the **(relative) condition number**.

"magnification ratio of error"

Example: Conditioning of Subtraction

Consider $f(x) = x - c$ where c is some constant. Using the formula ($\star$), we find that the associated condition number is

$$\kappa(x) = \left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{x}{x-c}\right|.$$

- It is large when $x \approx c$. (Catastrophic cancellation)

Example: Conditioning of Multiplication

The condition number of $f(x) = cx$ is

$$\kappa(x) = \left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{x \cdot c}{cx}\right| = 1.$$

- No magnification of error. (well- conditioned)

Example: Conditioning of Function Evaluation

The condition number of $f(x) = \cos(x)$ is

$$\kappa(x) = \left|\frac{x f'(x)}{f(x)}\right| = \left|\frac{-x \sin x}{\cos x}\right| = |x \tan x| \,.$$

- The condition number is large when $x = (n + 1/2)\pi$, where $n \in \mathbb{Z}$.

Vert. asympt. of tan(x).

Example: Conditioning of Root-Finding

parameter

Let $r = f(a; b, c)$ be a root of $ax^2 + bx + c = 0$. Instead of direct differentiation, use implicit differentiation

$ar^2 + br + c = 0.$

Full problem

$\vec{f} : \mathbb{R}^3 \to \mathbb{C}^2$

$\vec{f}\left(\begin{bmatrix} a \\ b \\ c \end{bmatrix}\right) = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}$

$f'(a) = \frac{dr}{da}$

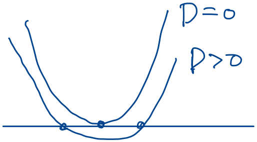
$$r^2 + 2ar\frac{dr}{da} + b\frac{dr}{da} = 0.$$

Solve for the derivative,

$$f'(a) = \frac{dr}{da} = -\frac{r^2}{2ar + b} = -\frac{r^2}{\pm\sqrt{b^2 - 4ac}},$$

then compute the condition number using the formula (⋆) to get

$$\kappa(a) = \left|\frac{af'(a)}{f(a)}\right| = \left|\frac{ar^2}{\pm r\sqrt{b^2 - 4ac}}\right| = \left|\frac{ar}{\sqrt{b^2 - 4ac}}\right|.$$



- Conditioning is poor for small discriminant, *i.e.*, near repeated roots.

Implicit diff. Hit with $\frac{d}{da}$, keeping in mind that r is a func. of a.

$$\frac{d}{da}\left(a r^2\right) \overset{CR}{=} 1 \cdot r^2 + a \cdot 2r \cdot \frac{dr}{da}$$

$$= r^2 + 2ar\frac{dr}{da}$$

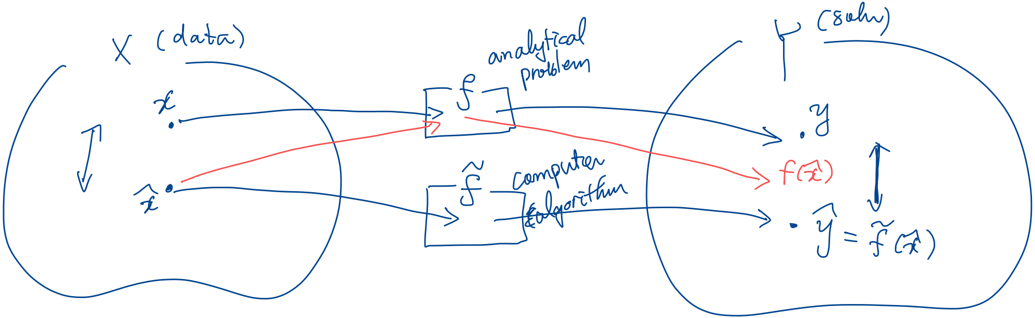
Main sources of errors

- numbers (f-p)
- problems (cond.)
- algorithm (stab.)

Stability

Algorithms

- Recall that we defined a *problem* as a function $f : X \to Y$.
- An *algorithm* can be viewed as another map $\tilde{f} : X \to Y$ between the same two spaces, which involves errors arising in
  - representing the actual input x as $\hat{x}$;
  - implementing the function f numerically on a computer.



Example: Horner's Method

Consider evaluating a polynomial

$$p(x) = c_n x^{n-1} + c_{n-1} x^{n-2} + \cdots + c_2 x + c_1.$$

Can rewrite it as

$$p(x) = (\cdots((c_n x + c_{n-1})x + c_{n-2})x + \cdots + c_2)x + c_1,$$

```
function p = horner(c, x)
% HORNER evaluates polynomial using Horner's method.
    n = length(c);
    p = c(n);
    for k = n-1:-1:1
        p = p*x + c(k);
    end
end
```

The relative error of our interest is

$$\left|\frac{\tilde f(\hat x) - f(x)}{f(x)}\right| \leqslant \left|\frac{\tilde f(\hat x) - f(\hat x)}{f(x)}\right| + \left|\frac{f(\hat x) - f(x)}{f(x)}\right|$$

(triangle inequality)

$$\lessapprox \underbrace{\left|\frac{\tilde f(\hat x) - f(\hat x)}{f(\hat x)}\right|}_{\text{numerical error}} + \underbrace{\left|\frac{f(\hat x) - f(x)}{f(x)}\right|}_{\text{perturbation error}} \leqslant (\hat\kappa_{\text{num}} + \kappa_f)\,\boxed{\text{eps}}.$$

where $\kappa = \kappa_f$ be the (relative) condition number of the exact problem f and

$$\hat\kappa_{\text{num}} = \max \left|\frac{\tilde f(\hat x) - f(\hat x)}{f(\hat x)}\right| \Bigg/ \left|\frac{\hat x - x}{x}\right| .$$

Example: Root-Finding Revisited

Consider again solving the quadratic problem $ar^2 + br + c = 0$.

- Taking $a = c = 1$ and $b = -(10^6 + 10^{-6})$, the roots can be computed exactly by hand: $r_1 = 10^6$ and $r_2 = 10^{-6}$.
- If numerically computed in MATLAB using the quadratic equation formula, r_1 is correct but r_2 has only 5 correct digits.
- Fix it using $r_2 = (c/a)/r_1$.