1. Numerical precision and errors

- floating-point numbers
- IEEE standard and machine precision
- numerical errors
- conditioning and sensitivity
- numerical stability and efficiency

Floating-point number

a *floating-point* number is represented as

$$x = \pm(.d_1 d_2 \cdots d_n) \cdot \beta^e$$

with value

$$x = \pm\left(\frac{d_1}{\beta^1} + \cdots + \frac{d_n}{\beta^n}\right) \cdot \beta^e$$

- β is the *base* (an integer larger than 1); n is *precision* (number of digits)
- e is *exponent* ($e_{\min} \leq e \leq e_{\max}$)
- $d_1 d_2 d_3 \cdots$ is *mantissa* or *significand*
- d_i integer with $0 \leq d_i \leq \beta - 1$ and $d_1 \neq 0$ for $x \neq 0$ (normalized system)

Other convention

$$\pm(\tilde{d}_0.\tilde{d}_1\tilde{d}_2\cdots\tilde{d}_{n-1}) \cdot \beta^{\tilde{e}} = \pm\left(\tilde{d}_0 + \frac{\tilde{d}_1}{\beta^1} + \frac{\tilde{d}_2}{\beta^2} + \cdots + \frac{\tilde{d}_{n-1}}{\beta^{n-1}}\right) \cdot \beta^{\tilde{e}}$$

relation to previous representation: $\tilde{d}_i = d_{i+1}$ and $\tilde{e} = e - 1$

Floating-point numbers with base 10

$$
\begin{aligned}
x &= \pm\,(.d_1 d_2 \ldots d_n)_{10} \cdot 10^e \\
&= \pm \left(\frac{d_1}{10} + \frac{d_2}{10^2} + \cdots + \frac{d_n}{10^n} \right) \cdot 10^e
\end{aligned}
$$

- d_i integer, $0 \leq d_i \leq 9$
- $d_1 \neq 0$ if $x \neq 0$ (normalized system)
- used in pocket calculators

Example (with $n = 6$):

$$
\begin{aligned}
12.625 &= +\,(.126250)_{10} \cdot 10^2 \\
&= +\left(1 \cdot 10^{-1} + 2 \cdot 10^{-2} + 6 \cdot 10^{-3} + 2 \cdot 10^{-4} + 5 \cdot 10^{-5} + 0 \cdot 10^{-6}\right) \cdot 10^2
\end{aligned}
$$

Properties

- a finite set of numbers
- unevenly spaced: distance between floating-point numbers varies
  - smallest number greater than 1 is $(.10\cdots01)_{10} \cdot 10 = 1 + 10^{-n+1}$
  - smallest number greater than 10 is $(.10\cdots01)_{10} \cdot 10^2 = 10 + 10^{-n+2}, \ldots$
- largest positive number:

$$x_{\max} = +(.999\cdots9)_{10} \cdot 10^{e_{\max}} = (1 - 10^{-n})\, 10^{e_{\max}}$$

(here we used $\sum_{k=0}^{n} r^k = \frac{1-r^{n+1}}{1-r}$ for $r \neq 1$)

- smallest positive number:

$$x_{\min} = +(.100\cdots0)_{10} \cdot 10^{e_{\min}} = 10^{e_{\min}-1}$$

Floating-point numbers with base 2

$$
\begin{aligned}
x &= \pm\,(.d_1 d_2 \ldots d_n)_2 \cdot 2^e \\
&= \pm\left(d_1 2^{-1} + d_2 2^{-2} + \cdots + d_n 2^{-n}\right) \cdot 2^e
\end{aligned}
$$

- $d_i \in \{0, 1\}$
- $d_1 = 1$ if $x \neq 0$ (normalized system)
- used in almost all computers
- example: $x = -(.1101) \cdot 2^2$ equals $x = -(\frac{1}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16}) \cdot 2^2 = -3.25$

Properties

- a finite set of unevenly spaced numbers
- largest positive number is

$$x_{\max} = +(.111\cdots1)_2 \cdot 2^{e_{\max}} = (1 - 2^{-n})\, 2^{e_{\max}}$$

- smallest positive number is

$$x_{\min} = +(.100\cdots0)_2 \cdot 2^{e_{\min}} = 2^{e_{\min}-1}$$

Rounding

- a floating-point number system is a finite set of numbers
- all other numbers must be rounded
- $\mathrm{fl}(x)$ is the floating-point representation of x

Rounding

- x_- is the nearest floating point number to x that is $\leq x$
- x_+ is the nearest floating point number to x that is $\geq x$
- numbers are rounded to the nearest floating-point number

$$\mathrm{fl}(x) = \begin{cases} x_- & \text{if } x - x_- < x_+ - x \\ x_+ & \text{if } x_+ - x < x - x_- \end{cases}$$

for ties we round to nearest even

for binary case we round to number with least significant bit 0

Example: 3-digit calculator

$$x = \pm\,(.d_1 d_2 d_3)_{10} \cdot 10^e, \quad -9 \le e \le 9$$

- largest/smallest positive numbers: $x_{\max} = 0.99 \cdot 10^9$ and $x_{\min} = 0.100 \cdot 10^{-9}$
- not enough “room” to store exactly the results from most arithmetic operations

$$(1.23 \times 10^1) \times (4.56 \times 10^2) = 5608.8$$
$$(1.23 \times 10^6) + (4.56 \times 10^4) = 1275600$$

  involve more than three significant digits

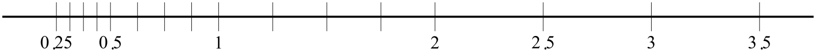
- results must be rounded in order to “fit” the 3-digit format,

$$\mathrm{fl}(5608.8) = .561 \times 10^4, \quad \mathrm{fl}(1275600) = .128 \times 10^7$$

Example: small binary system

we enumerate all positive floating-point numbers for

$$n = 3, \quad e_{\min} = -1, \quad e_{\max} = 2$$



$$\begin{aligned}
&+(.100)_2 \cdot 2^{-1} = 0.2500, \qquad &&+(.100)_2 \cdot 2^{0} = 0.500 \\
&+(.101)_2 \cdot 2^{-1} = 0.3125, \qquad &&+(.101)_2 \cdot 2^{0} = 0.625 \\
&+(.110)_2 \cdot 2^{-1} = 0.3750, \qquad &&+(.110)_2 \cdot 2^{0} = 0.750 \\
&+(.111)_2 \cdot 2^{-1} = 0.4375, \qquad &&+(.111)_2 \cdot 2^{0} = 0.875
\end{aligned}$$

$$\begin{aligned}
&+(.100)_2 \cdot 2^{1} = 1.00, \qquad &&+(.100)_2 \cdot 2^{2} = 2.0 \\
&+(.101)_2 \cdot 2^{1} = 1.25, \qquad &&+(.101)_2 \cdot 2^{2} = 2.5 \\
&+(.110)_2 \cdot 2^{1} = 1.50, \qquad &&+(.110)_2 \cdot 2^{2} = 3.0 \\
&+(.111)_2 \cdot 2^{1} = 1.75, \qquad &&+(.111)_2 \cdot 2^{2} = 3.5
\end{aligned}$$

numbers not represented are rounded (*e.g.*, $x = 0.26$ is rounded to $\mathrm{fl}(x) = 0.25$)

Overflow and underflow

- overflow means number is too large to fit into floating-point system ($e > e_{\max}$)
- underflow is obtained when $e < e_{\min}$
- underflow is nonfatal: system sets number to 0 (MATLAB does this)

Example: consider computing $c = \sqrt{a^2 + b^2}$ in a floating-point system with four decimal digits and two exponent digits

- for $a = 10^{60}$ and $b = 1$, correct result is $c = 10^{60}$
- squaring a gives 10^{120}, which cannot be represented in this system (overflow)
- can be avoided if we rescale $c = s\sqrt{(a/s)^2 + (b/s)^2}$ for any $s \neq 0$
- using $s = a = 10^{60}$ gives an underflow when b/s is squared, which is set to zero
- this yields the most accurate answer given this particular floating-point system

Outline

- floating-point numbers
- **IEEE standard and machine precision**
- numerical errors
- conditioning and sensitivity
- numerical stability and efficiency

IEEE standard for binary arithmetic

- two binary ($\beta = 2$) floating-point number formats
- used in almost all modern computers

IEEE standard single precision

$$n = 24, \quad e_{\min} = -125, \quad e_{\max} = 128$$

requires 32 bits:

- 23 bits for mantissa ($d_1 = 1$ not stored)
- 1 sign bit and 8 bits for exponent

IEEE standard double precision

$$n = 53, \quad e_{\min} = -1021, \quad e_{\max} = 1024$$

requires 64 bits:

- 52 bits for mantissa ($d_1 = 1$ not stored)
- 1 sign bit and 11 bits for exponent

Machine precision

for binary number system the value

$$\epsilon_{\mathrm{M}} = 2^{-n}$$

is called *machine precision* or *machine epsilon*

Rounding error: ϵ_{M} gives rounding error bound

$$\frac{|x - \mathrm{fl}(x)|}{|x|} \le \epsilon_{\mathrm{M}}$$

fundamental limitations of numerical computations

Example: IEEE standard double precision (used by MATLAB)

$$n = 53, \quad \epsilon_{\mathrm{M}} = 2^{-53} \simeq 1.1102 \cdot 10^{-16}$$

number of correct digits is roughly $-\log_{10} \epsilon_{\mathrm{M}} \approx 16$

Example

- the smallest floating-point number greater than 1 is

$$(.10\cdots01)_2 \cdot 2^1 = 1 + 2^{1-n} = 1 + 2\epsilon_{\mathrm{M}}$$

- numbers $x \in (1, 1 + 2\epsilon_{\mathrm{M}})$ are rounded to 1 or $1 + 2\epsilon_{\mathrm{M}}$

$$\begin{array}{ll} \mathrm{fl}(x) = 1 & \text{for } 1 \leq x \leq 1 + \epsilon_{\mathrm{M}} \\ \mathrm{fl}(x) = 1 + 2\epsilon_{\mathrm{M}} & \text{for } 1 + \epsilon_{\mathrm{M}} < x \leq 1 + 2\epsilon_{\mathrm{M}} \end{array}$$

- therefore numbers between 1 and $1 + \epsilon_{\mathrm{M}}$ are indistinguishable from 1

Outline

- floating-point numbers
- IEEE standard and machine precision
- **numerical errors**
- conditioning and sensitivity
- numerical stability and efficiency

Error sources

Errors in the problem to be solved

- mathematical model errors (model approximation)
- error in the input data (arising from physical measurements)
- input data may have been produced by a previous approximate computational step

Truncation or discretization errors

- due to using approximate formula
  - replacing derivatives by finite differences
  - evaluating function by truncating a Taylor series
- *convergence errors* in iterative methods, which converge to the exact solution in infinitely many iterations, but are cut off after a finite number of iterations

Roundoff errors

- arise from finite precision representation of real numbers on computers
- truncation or discretization errors usually dominate roundoff errors in magnitude

Example

surface area of the Earth might be computed using the formula

$$A = 4\pi r^2$$

for the surface area of a sphere of radius r

- earth is modeled as a sphere, which is an approximation of its true shape
- $r \approx 6370\ \mathrm{km}$, is based on empirical measurements and previous computations
- π is given by an infinite limiting process, which must be truncated at some point
- numerical values for the input data, as well as the results of the arithmetic operations performed on them, are rounded in a computer or calculator

Absolute and relative errors

given actual value x and its approximation $\hat{x}$

- *absolute error:* $|x - \hat{x}|$
- *relative error:* $\dfrac{|x - \hat{x}|}{|x|}$ (assuming $x \neq 0$)

  gives percentage of error compared to the actual value

Example

x	$\hat{x}$	absolute error	relative error
1	0.99	0.01	0.01
1	1.01	0.01	0.01
100	99.99	0.01	0.0001
100	99	1	0.01

- when $|x| \approx 1$, little difference between absolute and relative error
- when $|x| >> 1$, relative error more meaningful

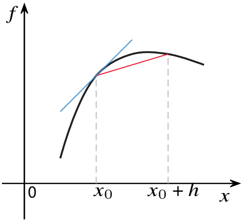
Example: derivative approximation

Taylor theorem: for differentiable f, we have

$$f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{h^2}{2} f''(\theta) \quad \text{for some } x \le \theta \le x_0 + h$$

we can approximate $f'(x_0)$ by

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h}$$



with the truncation (discretization) error being

$$\left| f'(x_0) - \frac{f(x_0 + h) - f(x_0)}{h} \right| = \left| \frac{h}{2} f''(\theta) \right| \le Mh/2$$

where $|f''(\theta)| \le M$

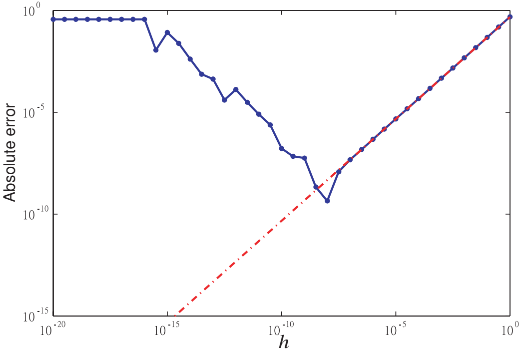
- assume error in evaluating $f(x)$ is bounded by ϵ
- rounding error in evaluating $\frac{f(x_0+h)-f(x_0)}{h}$ is bounded by $2\epsilon/h$
- total error is

$$\frac{Mh}{2} + \frac{2\epsilon}{h}$$

- first term decreases as h decreases
- second term increases as h decreases

Example

- $f(x) = \sin(x)$ and $x_0 = 1.2$
- exact value of derivative is $f'(x_0) = \cos(1.2)$
- a log-log plot of the error versus h is provided below



- solid curve shows $\left|f'(x_0) - \frac{f(x_0+h)-f(x_0)}{h}\right|$ for $f(x) = \sin(x)$, $x_0 = 1.2$
- dash-dot style line depicts the truncation error without roundoff error
- when $h < 10^{-8}$, discretization error becomes small, and roundoff error dominate

Cancellation

$$\hat{a} = a(1 + \Delta a), \quad \hat{b} = b(1 + \Delta b)$$

- a, b: exact values
- $\hat{a}, \hat{b}$: approximations with unknown relative errors $\Delta a, \Delta b$
- relative error in $\hat{x} = \hat{a} - \hat{b} = (a - b) + (a\Delta a - b\Delta b)$ is

$$\frac{|\hat{x} - x|}{|x|} = \frac{|a\Delta a - b\Delta b|}{|a - b|}$$

  if $a \simeq b$, small Δa and Δb can lead to very large relative errors in x

this is called **cancellation**; cancellation occurs when:

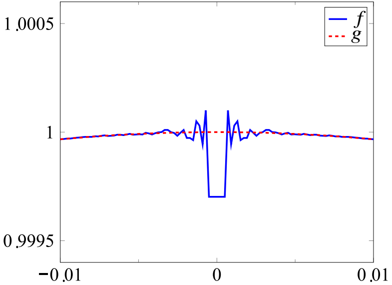
- we subtract two numbers that are almost equal
- one or both numbers are subject to error

Example

two expressions for the same function

$$f(x) = \frac{1 - (\cos x)^2}{x^2}$$

$$g(x) = \frac{(\sin x)^2}{x^2}$$



- results of $\cos x$ and $\sin x$ were rounded to 10 significant digits
- other calculations are exact
- cancellation occurs when we evaluate the numerator of $f(x) = \frac{1-(\cos x)^2}{x^2}$
  - $1 \simeq (\cos x)^2$ when x is small
  - there is a rounding error in $\cos x$

Evaluation of f: evaluate $f(x)$ at $x = 5 \cdot 10^{-5}$

- calculate $\cos x$ and round result to 10 digits

$$\cos x = 0.99999999875000\ldots$$
$$\rightsquigarrow 0.9999999988$$

- evaluate $f(x) = \left(1 - \cos(x)^2\right)/x^2$ using rounded value of $\cos x$

$$\frac{1 - (0.9999999988)^2}{(5 \cdot 10^{-5})^2} = 0.9599\ldots$$

has only one correct significant digit (correct value is $0.9999\ldots$)

Evaluation of g: evaluate $g(x)$ at $x = 5 \cdot 10^{-5}$

- calculate $\sin x$ and round result to 10 digits

$$\sin x = 0.499999999791667\ldots \cdot 10^{-5}$$
$$\rightsquigarrow 0.4999999998 \cdot 10^{-5}$$

- evaluate $f(x) = \sin(x)^2/x^2$ using rounded value of $\cos x$

$$\frac{(\sin x)^2}{x^2} \approx \frac{\left(0.4999999998 \cdot 10^{-5}\right)^2}{(5 \cdot 10^{-5})^2} = 0.9999\ldots$$

  has about ten correct significant digits

Conclusion: f and g are equivalent mathematically, but not numerically

Outline

- floating-point numbers
- IEEE standard and machine precision
- numerical errors
- **conditioning and sensitivity**
- numerical stability and efficiency

Condition (conditioning) of a problem

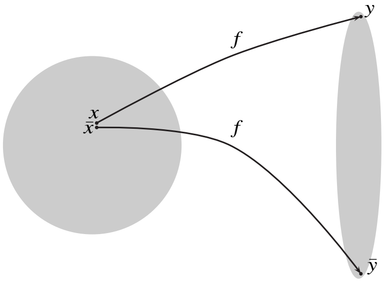
describes sensitivity of the solution to changes in the problem data

well-conditioned problem

- small changes in the data produce small changes in the solution

ill-conditioned (badly conditioned) problem

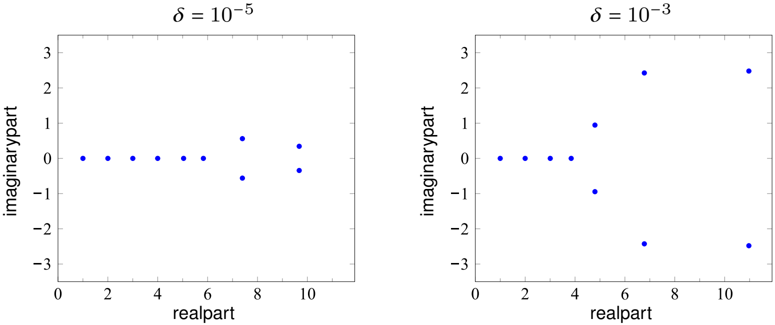
- small changes in the data can produce large changes in the solution



Roots of a polynomial

$$p(x) = (x-1)(x-2)\cdots(x-10) + \delta \cdot x^{10}$$

roots of p computed by MATLAB for two values of δ



roots can be very sensitive to errors in the coefficients

Condition number of differentiable functions

given x, evaluate $y = f(x)$

- if x is changed to $x + \Delta x$, solution changes to $y + \Delta y = f(x + \Delta x)$
- condition with respect to absolute error in x and y

$$|\Delta y| \approx |f'(x)|\,|\Delta x|$$

problem is ill-conditioned with respect to absolute error if $|f'(x)|$ is very large

Condition number: condition with respect to relative errors in x and y

$$\frac{|\Delta y|}{|y|} \approx \frac{|f'(x)||x|}{|f(x)|} \frac{|\Delta x|}{|x|}$$

- $|f'(x)|\,|x|/|f(x)|$ is the *condition number*
- ill-conditioned with respect to relative error if condition number is very large

Examples

consider $f(x) = \sqrt{x}$; since $f'(x) = 1/(2\sqrt{x})$, the condition number is

$$\left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{x/(2\sqrt{x})}{\sqrt{x}}\right| = \frac{1}{2}$$

- any relative change in input causes relative change in output of about half that size
- the square root problem is well-conditioned

consider $f(x) = \tan(x)$; since $f'(x) = 1 + \tan^2(x)$, the condition number is

$$\left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{x\left(1+\tan^2(x)\right)}{\tan(x)}\right| = \left|x\left(\frac{1}{\tan(x)} + \tan(x)\right)\right|$$

- ill-conditioned around an integer multiple of $\pi/2$, where its value becomes infinite
- for $x = 1.57079$, the condition number is approximately 2.48275×10^5
- to see the effect of this, we evaluate the function at two nearby points,

$$\tan(1.57079) \approx 1.58058 \times 10^5, \quad \tan(1.57078) \approx 6.12490 \times 10^4$$

difference is on order of approximately 10

Outline

- floating-point numbers
- IEEE standard and machine precision
- numerical errors
- conditioning and sensitivity
- **numerical stability and efficiency**

Stability, efficiency, and robustness

Stability: refers to the accuracy of an algorithm in the presence of rounding errors

- an algorithm is unstable if rounding errors cause large errors in the result
- instability is often, but not always, caused by cancellation

Efficiency

- a numerical algorithm is inefficient if it takes an unreasonable amount of run-time
- efficiency depends on both cpu time and storage space requirements
- theoretical properties, like the rate of convergence, can indicate efficiency

Robustness

- major effort in writing numerical software is ensuring it works under all conditions
- a robust routine should yield correct results within an acceptable error tolerance

Example: roots of a quadratic equation

$$ax^2 + bx + c = 0 \quad (a \neq 0)$$

Algorithm 1: use the formulas

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

unstable if $b^2 \gg |4ac|$

- if $b^2 \gg |4ac|$ and $b \geq 0$, cancellation occurs in x_1 ($b \simeq \sqrt{b^2 - 4ac}$)
- if $b^2 \gg |4ac|$ and $b \leq 0$, cancellation occurs in x_2 ($-b \simeq \sqrt{b^2 - 4ac}$)
- in both cases b may be exact, but the square root introduces small errors

Example: roots of a quadratic equation

$$ax^2 + bx + c = 0 \quad (a \neq 0)$$

Algorithm 2: use fact that roots x_1, x_2 satisfy $x_1 x_2 = c/a$

- if $b \leq 0$, calculate

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{c}{ax_1}$$

- if $b > 0$, calculate

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}, \quad x_1 = \frac{c}{ax_2}$$

no cancellation when $b^2 \gg |4ac|$

Example: polynomial evaluation

$$p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$$

Naive method

- compute $c_n x^n$ using n multiplications, $c_{n-1} x^{n-1}$ using $n-1$ multiplications, ...
- total is $n(n+1)/2$ multiplications and n additions

Horner's rule: write in nested form:

$$p_n(x) = c_0 + x\Big(c_1 + x\Big(c_2 + x\big(c_3 + \cdots + x(c_{n-1} + c_n x)\cdots\big)\Big)\Big)$$

$p = c_n$
for $j = n-1, \ldots, 1, 0$
$p = px + c_j$

reduces the operation to n multiplications and n additions

Error accumulation

if E_k measures the relative error at the kth operation of an algorithm, then

- $E_k \simeq c_0 k E_0$ represents linear error growth, for some constant c_0
- $E_k \simeq c_1^k E_0$, for some constant $c_1 > 1$, represents exponential error growth

an algorithm with exponential error growth is unstable and should be avoided

Example

consider evaluating integrals $y_k = \int_0^1 \frac{x^k}{x+10}\, dx$ for $k = 1, 2, \dots, 30$

observe at first that analytically

$$y_k + 10y_{k-1} = \int_0^1 \frac{x^k + 10x^{k-1}}{x+10} dx = \int_0^1 x^{k-1} dx = \frac{1}{k}$$

and

$$y_0 = \int_0^1 \frac{1}{x+10} dx = \ln(11) - \ln(10)$$

- a simple algorithm is constructed as follows:
  1. evaluate $y_0 = \ln(11) - \ln(10)$
  2. for $k = 1, \dots, 30$, evaluate
  $$y_k = \frac{1}{k} - 10y_{k-1}$$
- this algorithm is in fact unstable
- magnitude of roundoff errors gets multiplied by 10 at each iteration; there is exponential error growth with $c_1 = 10$

References and further readings

- Uri M. Ascher. *A First Course on Numerical Methods*. Society for Industrial and Applied Mathematics, 2011.

- L. Vandenberghe. *EE133A lecture notes,* University of California, Los Angeles. (`http://www.seas.ucla.edu/~vandenbe/ee133a.html`)