CMPSC/Math 451, Numerical Computation

Wen Shen

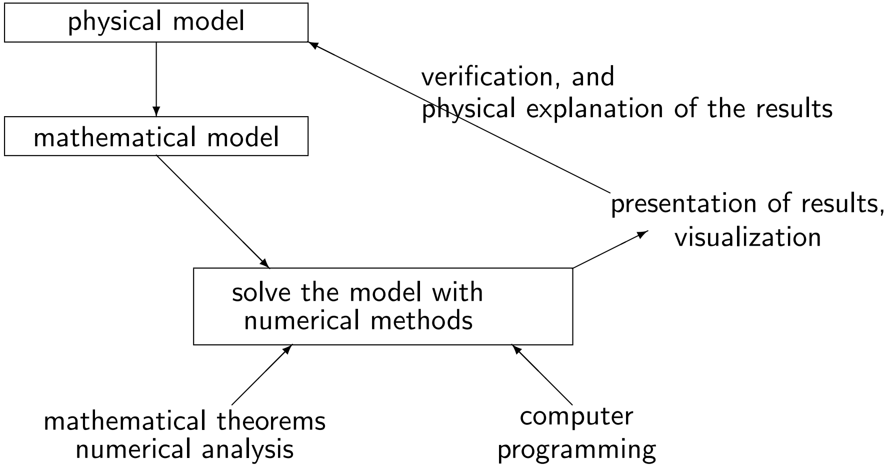
Department of Mathematics, Penn State University

What are numerical methods?

They are algorithms that compute approximations to functions, their derivatives, their integrations, and solutions to various equations etc.

Such algorithms could be implemented (programmed) on a computer.

Below is an overview on how various aspects are related.



Keep in mind that numerical methods are not about numbers. It is about mathematical ideas and insights.

We will study some basic classical types of problems:

- development of algorithms;
- implementation;
- a little bit of analysis, including error-estimates, convergence, stability etc.

We will use Matlab throughout the course for programming purpose.

Representation of numbers in different bases

Historically, there have been several bases for representing numbers:

10: decimal, daily use;

2: binary, computer use;

8: octal;

16: hexadecimal, ancient China;

20: vigesimal, used in ancient France (numbers 70-79 are counted as 60+10 to 60+19 in French, and 80 is 4×20);

60: sexagesimal, used by the Babylonians.

- etc...

In principle, one can use any number β as the base. Writing such a number with decimal point, we have

$$\begin{aligned}
&\Big(\overbrace{a_n a_{n-1}\cdots a_1 a_0}^{\text{integer part}}\,.\,\overbrace{b_1 b_2 b_3\cdots}^{\text{fractional part}}\Big)_\beta \\
= \quad & a_n\beta^n + a_{n-1}\beta^{n-1} + \cdots + a_1\beta + a_0 \qquad \text{(integer part)} \\
& \quad + b_1\beta^{-1} + b_2\beta^{-2} + b_3\beta^{-3} + \cdots \qquad \text{(fractonal part)}
\end{aligned}$$

Thinking of $\beta = 10$ above, we will understand the decimal representation.

The above formula allows us to convert a number in any base β into decimal base.

One can convert the numbers between different bases. We now go through some examples.
Example 1. octal → decimal

$$(45.12)_8 = 4 \times 8^1 + 5 \times 8^0 + 1 \times 8^{-1} + 2 \times 8^{-2} = (37.15625)_{10}$$

Example 2. octal $\rightarrow$ binary
Observe

$$\begin{aligned}
(1)_8 &= (1)_2 \\
(2)_8 &= (10)_2 \\
(3)_8 &= (11)_2 \\
(4)_8 &= (100)_2 \\
(5)_8 &= (101)_2 \\
(6)_8 &= (110)_2 \\
(7)_8 &= (111)_2 \\
(10)_8 &= (1000)_2
\end{aligned}$$

Then,

$$(5034)_8 = (\underbrace{101}_{5}\underbrace{000}_{0}\underbrace{011}_{3}\underbrace{100}_{4})_2$$

Example 3. binary $\rightarrow$ octal

$$(110\,010\,111\,001)_2 = (\underbrace{6}_{110}\ \underbrace{2}_{010}\ \underbrace{7}_{111}\ \underbrace{1}_{001})_8$$

This algorithm is possible because 8 is a power of 2, namely, $8 = 2^3$.

Example 4. decimal $\to$ binary. Write $(12.45)_{10}$ in binary base.
Answer. This example is of particular interests. Since the computer uses binary base, how would the number $(12.45)_{10}$ look like in binary base?
The conversion takes two steps.
First, we treat the integer part and convert $(12)_{10}$ into binary.
Procedure: keep divided by 2, and store the remainders of each step, until one can not divide anymore.

$$\begin{array}{r|lc} & & (\textit{remainder}) \\ 2 & \underline{12} & 0 \\ 2 & \underline{6} & 0 \\ 2 & \underline{3} & 1 \\ 2 & \underline{1} & 1 \\ & 0 & \end{array} \qquad \Rightarrow (12)_{10} = (1100)_2$$

We now convert $(0.45)_{10}$ into binary.
Procedure: multiply the fractional part by 2, and store the integer part for the result.

$$\begin{array}{l|ll} 0.45 & \times & 2 \\ \underline{0}.9 & \times & 2 \\ \underline{1}.8 & \times & 2 \\ \underline{1}.6 & \times & 2 \\ \underline{1}.2 & \times & 2 \\ \underline{0}.4 & \times & 2 \\ \underline{0}.8 & \times & 2 \\ \underline{1}.6 & \times & 2 \\ \ldots & & \end{array} \qquad \Rightarrow (0.45)_{10} = (0.01110011001100\cdots)_2.$$

Note that the fractional part does not end!

Putting them together, we get

$$(12.45)_{10} = (1100.01110011001100\cdots)_2$$

It is surprising to observe that, a simple finite length decimal number such as 12.45 could be have infinite length of fractional numbers in binary form!

How does a computer store such a number? – next video!

Floating point representation

Recall normalized scientific notation for a real number x:

Decimal: $x = \pm r \times 10^n, \quad 1 \leq r < 10$. (Example: $2345 = 2.345 \times 10^3$)

Binary: $x = \pm r \times 2^n, \quad 1 \leq r < 2$

Octal: $x = \pm r \times 8^n, \quad 1 \leq r < 8$ (Just for an example.)

For any base β: $x = \pm r \times \beta^n, \quad 1 \leq r < \beta$

Information to be stored:
(1) the sign,
(2) the exponent n,
(3) the value of r.

The computer uses the binary version of the number system. They represent numbers with finite length. These are called *machine numbers*.

r: normalized mantissa. For binary numbers, we have

$$r = 1.(\text{fractional part})$$

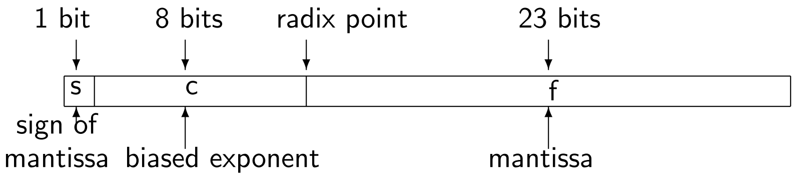
Therefore, in the computer we will only store the fractional part of the number.

n: exponent. If $n > 0$, then $x > 1$. If $n < 0$, then $x < 1$.

s: the sign of the number. If $s = 0$, it is positive; if $s = 1$, it is negative.

Each bit can store the value of either 0 or 1.

Single-precision IEEE standard floating-point, in a 32-bit computer



The exponent: $2^8 = 256$. It can represent numbers from -127 to 128.

The value of the number:

$$(-1)^s \times 2^{c-127} \times (1.f)_2$$

The smallest representable number in absolute value is

$$x_{\min} = 2^{-127} \approx 5.9 \times 10^{-39}.$$

The largest representable number in absolute value is

$$x_{\max} = 2^{128} \approx 2.4 \times 10^{38}.$$

Computers can only handle numbers with absolute values between $x_{\min}$ and $x_{\max}$.

We say that x **underflows** if $|x| < x_{\min}$. In this case, we consider $x = 0$.

We say that x **overflows** if $|x| > x_{\max}$. In this case, we consider $x = \infty$.

Error in the floating point representation.

Let $\mathrm{fl}(x)$ denote the floating point representation of the number x. In general it contains error (roundoff or chopping).

$$\mathrm{fl}(x) = x \cdot (1 + \delta)$$

$$\text{relative error: } \delta = \frac{\mathrm{fl}(x) - x}{x}$$

$$\text{absolute error: } = \mathrm{fl}(x) - x = \delta \cdot x$$

Computer errors in representing numbers:

- relative error in rounding off: $\delta \leq 0.5 \times 2^{-23} \approx 0.6 \times 10^{-7}$
- relative error in chopping: $\delta \leq 1 \times 2^{-23} \approx 1.2 \times 10^{-7}$

Error propagation (through arithmetic operation)

Example 1. Consider an addition, say $z = x + y$, done in a computer. How would the errors be propagated?

Let $x > 0, y > 0$, and let $\mathrm{fl}(x), \mathrm{fl}(y)$ be their floating point representation.

$$\mathrm{fl}(x) = x(1+\delta_x), \qquad \mathrm{fl}(y) = y(1+\delta_y)$$

where δ_x, δ_y are the relative errors in x, y.
Then

$$\begin{aligned}
\mathrm{fl}(z) &= \mathrm{fl}\left(\mathrm{fl}(x) + \mathrm{fl}(y)\right) \\
&= \left(x(1+\delta_x) + y(1+\delta_y)\right)(1+\delta_z) \\
&= (x+y) + x\cdot(\delta_x+\delta_z) + y\cdot(\delta_y+\delta_z) + (x\delta_x\delta_z + y\delta_y\delta_z) \\
&\approx (x+y) + x\cdot(\delta_x+\delta_z) + y\cdot(\delta_y+\delta_z)
\end{aligned}$$

Here, δ_z is the round-off error for z.

Then, we have

$$\begin{aligned}\text{absolute error} &= \text{fl}(z) - (x+y) = x\cdot(\delta_x+\delta_z) + y\cdot(\delta_y+\delta_z) \\ &= \underbrace{\underbrace{x\cdot\delta_x}_{\text{abs. err. for } x} + \underbrace{y\cdot\delta_y}_{\text{abs. err. for } y}}_{\text{propagated error}} + \underbrace{(x+y)\cdot\delta_z}_{\text{round off err}}\end{aligned}$$

$$\text{relative error} = \frac{\text{fl}(z)-(x+y)}{x+y} = \underbrace{\frac{x\delta_x + y\delta_y}{x+y}}_{\text{propagated err}} + \underbrace{\delta_z}_{\text{round off err}}$$

Loss of significance

Loss of significance typically happens when one gets too few significant digits in subtraction of two numbers very close to each other.

Example 1. Find the roots of $x^2 - 40x + 2 = 0$. Use 4 significant digits in the computation.
Answer. The roots for the equation $ax^2 + bx + c = 0$ are

$$r_{1,2} = \frac{1}{2a}\left(-b \pm \sqrt{b^2 - 4ac}\right)$$

In our case, we have

$$x_{1,2} = 20 \pm \sqrt{398} \approx 20.00 \pm 19.95$$

so

$$x_1 \approx 20 + 19.95 = 39.95, \quad \text{(OK)}$$

$$x_2 \approx 20 - 19.95 = 0.05, \quad \text{not OK, lost 3 sig. digits}$$

To avoid this: change the algorithm. Observe that $x_1 x_2 = c/a$. Then

$$x_2 = \frac{c}{ax_1} = \frac{2}{1 \cdot 39.95} \approx 0.05006$$

We get back 4 significant digits in the result.

Example 2. Compute the function

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

in a computer. Explain what problem you might run into in certain cases. Find a way to fix the difficulty.

Answer. We see that, for large values of x with $x > 0$, the values $\sqrt{x^2+2x}$ and $x+1$ are very close to each. Therefore, in the subtraction we will lose many significant digits.

To avoid this problem, we manipulation the function $f(x)$ into an equivalent one that does not perform the subtraction. This can be achieved by multiplying both numerator and denominator by the conjugate of the denominator.

$$\begin{aligned} f(x) &= \frac{\sqrt{x^2+2x}+x+1}{\left(\sqrt{x^2+2x}-x-1\right)\left(\sqrt{x^2+2x}+x+1\right)} \\ &= \frac{\sqrt{x^2+2x}+x+1}{x^2+2x-(x+1)^2} = -\left(\sqrt{x^2+2x}+x+1\right). \end{aligned}$$

Review of Taylor Series

Given that $f(x) \in C^\infty$ is a smooth function. Its Taylor expansion about the point $x = c$ is:

$$\begin{aligned} f(x) &= f(c) + f'(c)(x-c) + \frac{1}{2!}f''(c)(x-c)^2 + \frac{1}{3!}f'''(c)(x-c)^3 + \cdots \\ &= \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(c)(x-c)^k. \end{aligned}$$

This is called *Taylor series of f at the point c*.

If $c = 0$, this is the *MacLaurin series*:

$$f(x) = f(0) + f'(0)x + \frac{1}{2!}f''(0)x^2 + \frac{1}{3!}f'''(0)x^3 + \cdots = \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(0)x^k.$$

Familiar examples of MacLaurin Series:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots, \qquad |x| < \infty$$

$$\sin x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots, \qquad |x| < \infty$$

$$\cos x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots, \qquad |x| < \infty$$

$$\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k = 1 + x + x^2 + x^3 + x^4 + \cdots, \qquad |x| < 1$$

Since the computer performs only algebraic operation, these series are actually actually how a computer calculates many “fancier” functions!

For example, the exponential function is calculated as

$$e^x \approx \sum_{k=0}^{N} \frac{x^k}{k!}$$

for some large integer N such that the error is sufficiently small.

Note that this is rather “expensive” in computing time! This is why in many algorithm we take care in doing fewer function evaluations!

Example 1. Compute e to 6 digit accuracy.
Answer. We have

$$e = e^1 = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \cdots$$

And

$$\begin{aligned} \frac{1}{2!} &= 0.5 \\ \frac{1}{3!} &= 0.166667 \\ \frac{1}{4!} &= 0.041667 \\ \cdots & \\ \frac{1}{9!} &= 0.0000027 \qquad \text{(can stop here)} \end{aligned}$$

so

$$e \approx 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \cdots \frac{1}{9!} = 2.71828$$

Error and convergence:

Assume $f^{(k)}(x)$ $(0 \le k \le n)$ are continuous functions.

$$\text{partial sum:} \quad f_n(x) = \sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(c)(x-c)^k$$

Taylor Theorem:

$$E_{n+1} = f(x) - f_n(x) = \sum_{k=n+1}^{\infty} \frac{1}{k!} f^{(k)}(c)(x-c)^k = \frac{1}{(n+1)!} f^{(n+1)}(\xi)(x-c)^{n+1}$$

where ξ is some value between x and c.

This says, for the infinite sum for the error, if it converges, then the sum is "dominated" by the first term in the series.

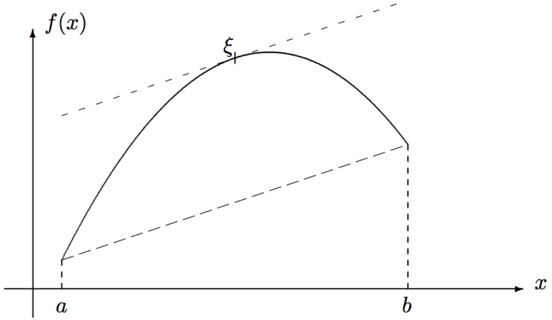
Observation: A Taylor series convergence rapidly if x is near c, and slowly (or not at all) if x is far away from c.

Geometric interpretation for the error estimate with $n = 0$.

$$f(b) - f(a) = (b-a)f'(\xi), \qquad \text{for some } \xi \text{ in } (a,b)$$

This implies

$$f'(\xi) = \frac{f(b)-f(a)}{b-a}, \qquad \text{The Mean-Value Theorem}$$



Finite Difference Approximation to derivatives

Given $h > 0$ sufficiently small, we have 3 ways of approximating $f'(x)$:

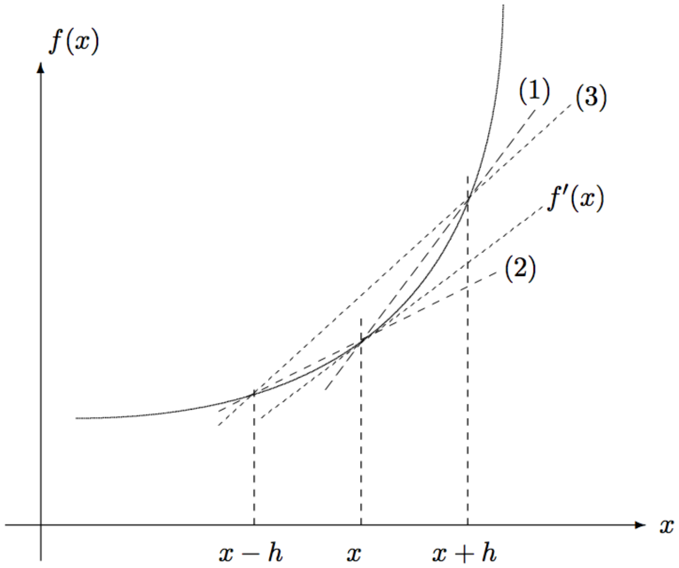
$$(1) \qquad f'(x) \approx \frac{f(x+h) - f(x)}{h} \qquad \text{Forward Euler}$$

$$(2) \qquad f'(x) \approx \frac{f(x) - f(x-h)}{h} \qquad \text{Backward Euler}$$

$$(3) \qquad f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \qquad \text{Central Finite Difference}$$

For the second derivative f'', we also have a central finite difference approximation:

$$f''(x) \approx \frac{1}{h^2} \left(f(x+h) - 2f(x) + f(x-h) \right)$$


$f(x)$
(1)
(3)
$f'(x)$
(2)
x
$x-h$
x
$x+h$

Local Truncation Error

The Taylor expansion for $f(x+h)$ about x:

$$f(x+h) = \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(x) h^k = \sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(x) h^k + E_{n+1}$$

where

$$E_{n+1} = \sum_{k=n+1}^{\infty} \frac{1}{k!} f^{(k)}(x) h^k = \frac{1}{(n+1)!} f^{(n+1)}(\xi) h^{n+1} = \mathcal{O}(h^{n+1})$$

Here the notation $\mathcal{O}(h^4)$ indicate a quantity bounded by Ch^4 where C is a bounded constant.

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{6}h^3 f'''(x) + \mathcal{O}(h^4), \\ f(x-h) &= f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) - \frac{1}{6}h^3 f'''(x) + \mathcal{O}(h^4). \end{aligned}$$

Forward Euler:

$$\frac{f(x+h)-f(x)}{h} = f'(x) + \frac{1}{2}hf''(x) + \mathcal{O}(h^2) = f'(x) + \mathcal{O}(h^1), \quad (1^{st}\text{order}),$$

Backward Euler:

$$\frac{f(x)-f(x-h)}{h} = f'(x) - \frac{1}{2}hf''(x) + \mathcal{O}(h^2) = f'(x) + \mathcal{O}(h^1), \quad (1^{st}\text{order}),$$

Central finite difference:

$$\frac{f(x+h)-f(x-h)}{2h} = f'(x) - \frac{1}{6}h^2 f'''(x) + \mathcal{O}(h^2) = f'(x) + \mathcal{O}(h^2), \quad (2^{nd}\text{order}),$$

Central finite difference for the second derivative

$$\frac{f(x+h)-2f(x)+f(x-h)}{h^2} = f''(x) + \frac{1}{12}h^2 f^{(4)}(x) + \mathcal{O}(h^4) = f''(x) + \mathcal{O}(h^2),$$

second order method.