Numerical Analysis

- Introduction -

Hanyang University

Jong-Il Park

Numerical Analysis ?

- Solving problems that cannot be solved analytically in a closed form

  eg. nonlinear equations…

- Calculating special functions such as sin(), cos(), exp(),…

  eg. Calculation of sin function using Taylor series

- Solving scientific and engineering problems using computers

  eg. Analysis of electro-magnetic field,

  optimization of engineering problem, etc.

Objective and Scope

- Emphasis is placed on **the essential strategies for solving the given problem**, rather than on the computational algorithm itself.

- Objective

  - design of numerical algorithms ➔ computational theory

  - using numerical packages ➔ ordinary users, scientists/engineers

  ☞ This course aims at somewhere in-between.

- Students will be able to solve scientific or engineering problems using their own code. When necessary, they may make use of external libraries.

- Scope: solving nonlinear eq., solving a large set of linear eqs., maximization/minimization of function, data modeling, solving differential eq., calculation for DSP etc.

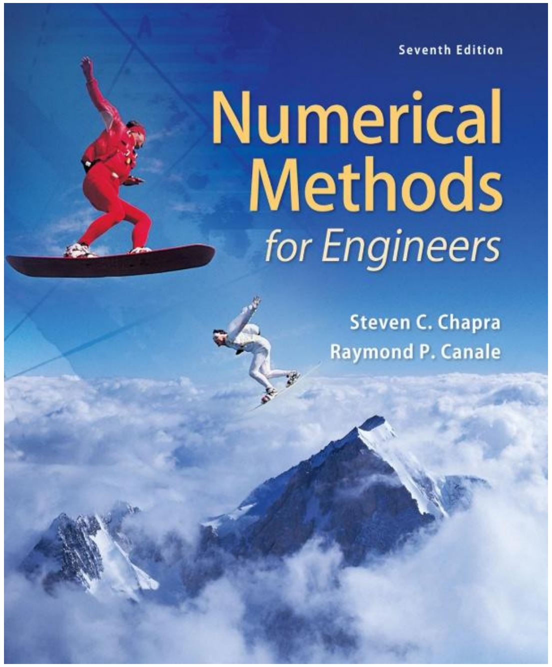
In this course

- Repeated practices of solving practical problems using learnt algorithms.
- Several homeworks on solving exercises by computers
- Learning and practicing how to exploit many useful algorithms in *Numerical Recipes in C* (All of the necessary routines will be provided through our course website)
- MATLAB, Mathematica, Maple ? → Self-study

Textbooks and References

[1] S.C. Chapra and R.P. Canale, *Numerical Methods for Engineers*, 7th ed., McGraw-Hill, 2015. (or 8th ed. 2020)

[2] W.Press, S.Teukolski, W.Vetterling, and B.Flannery, *Numerical Recipes in C, 2nd edition*, Cambridge University Press, 1992.



References

<1> 이관수, 공학도를 위한 수치해석, 2판, 세화, 2014.
<2> S.C. Chapra, *Applied Numerical Methods with MATLAB*, 5th ed., McGraw-Hill, 2022.
<3> J.D.Faires and R.Burden, Numerical Methods, 4th ed., Thomson, 2012.

Teaching Fellow

- 윤준영 박사과정
  - 02-2220-4368
  - 010-3604-8736
  - jyyun@hanyang.ac.kr

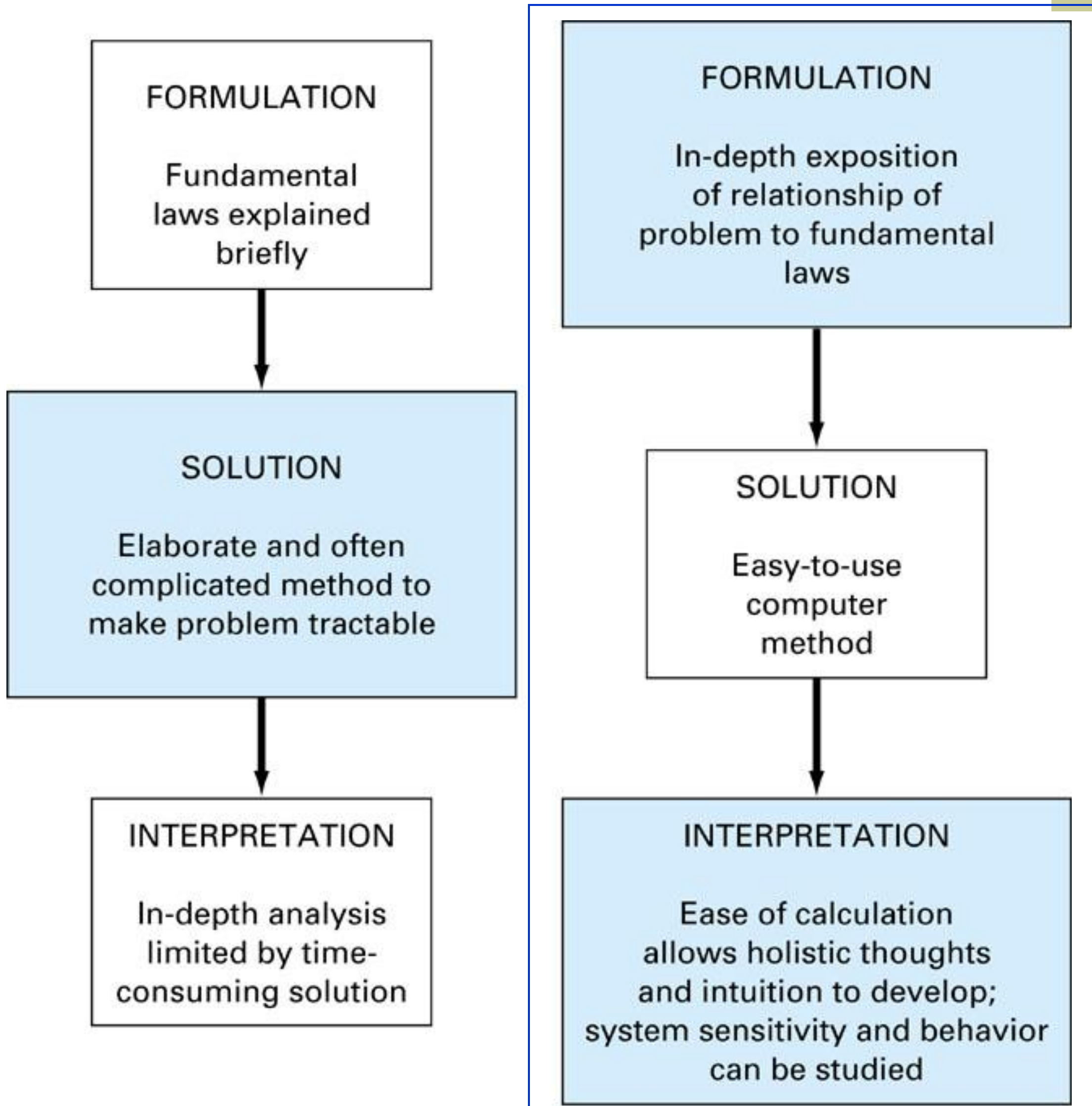
Why numerical methods?

1. When there is no analytic solution
2. Economic reason: Simulation costs less than experiments in terms of time, cost, and effort.
3. Learning basic theory for smart use of commercial software packages
4. No available numerical software package for a given problem
5. Overpriced software
6. As a practical tool for learning computer language
7. Help understanding complicated/ambiguous problems

※ Downside

- Coding and running first, thinking later ➔ Stupid!

Engineering Problem Solving



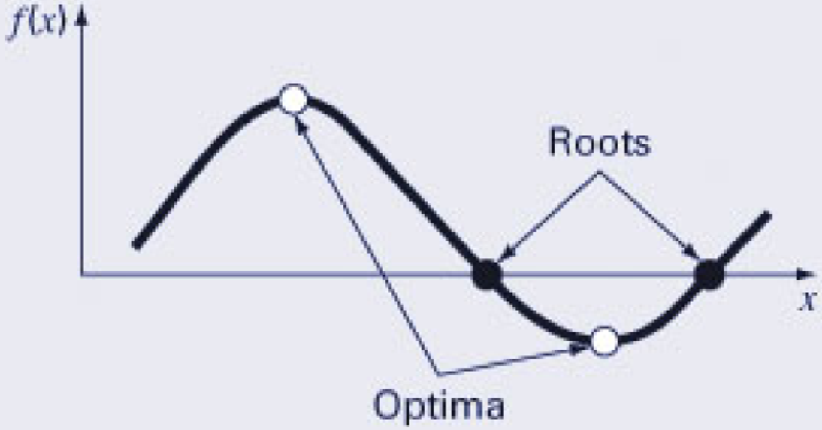
(a) (b)

Overview of the topics(I)

(a) *Part 2*: Roots and optimization

Roots: Solve for x so that $f(x) = 0$

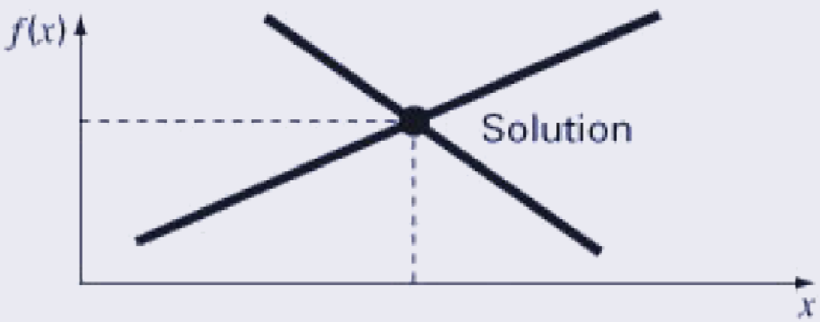
Optimization: Solve for x so that $f'(x) = 0$



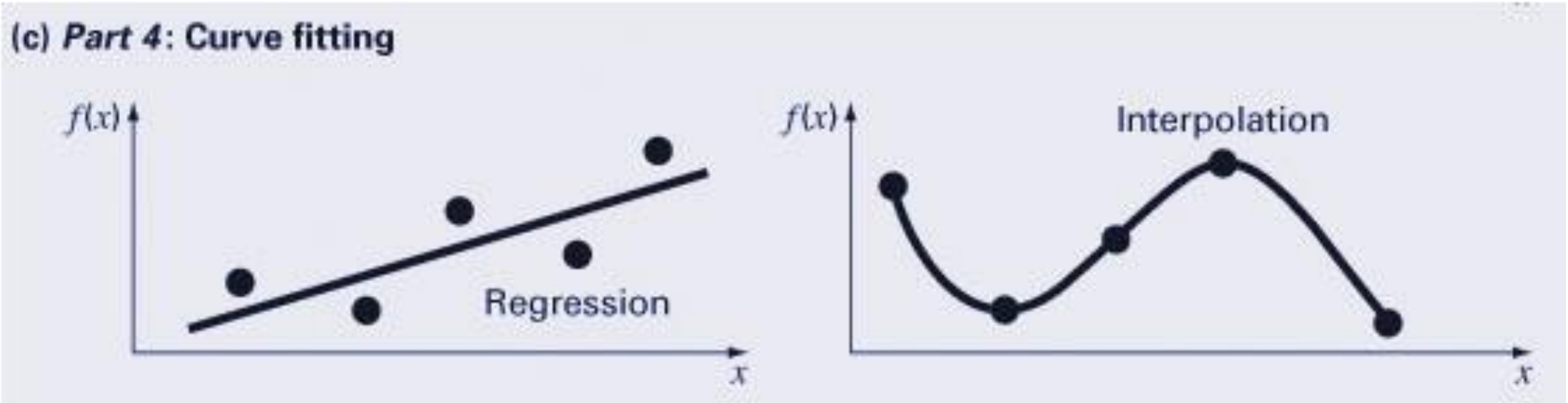
(b) *Part 3*: Linear algebraic equations

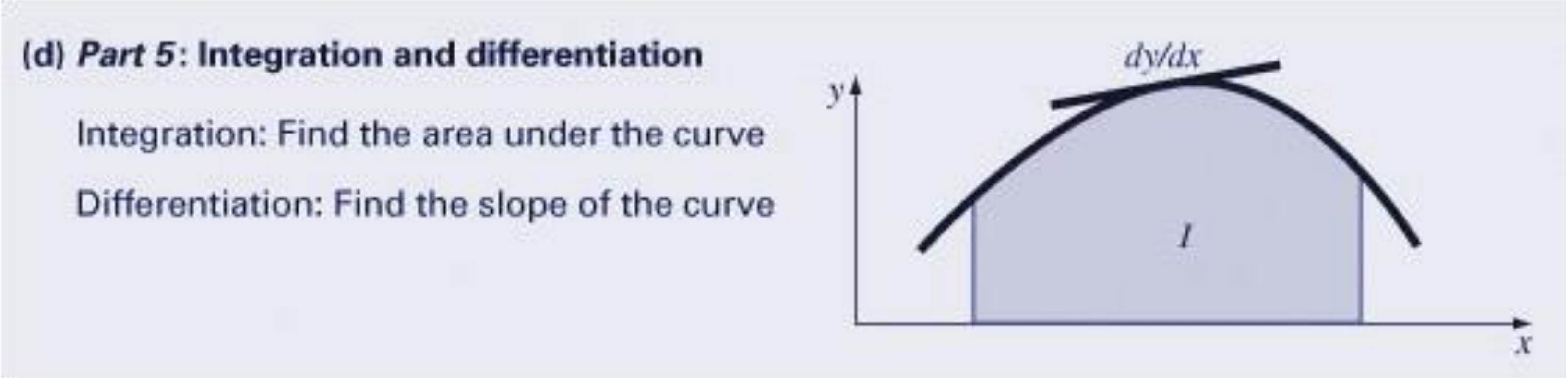
Given the a's and the b's, solve for the x's

$$a_{11}x_1 + a_{12}x_2 = b_1$$
$$a_{21}x_1 + a_{22}x_2 = b_2$$



Overview of the topics(II)





Overview of the topics(III)

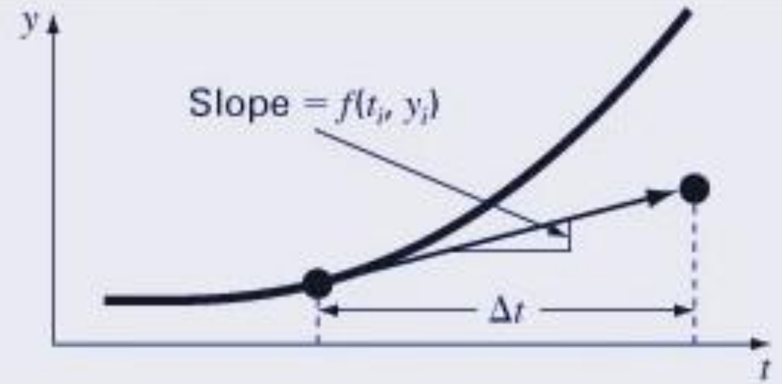
(e) *Part 6*: Differential equations

Given

$$\frac{dy}{dt} \approx \frac{\Delta y}{\Delta t} = f(t, y)$$

solve for y as a function of t

$$y_{i+1} = y_i + f(t_i, y_i)\Delta t$$



+Sorting

+DSP (FFT, convolution, etc.)

Representation of numbers

- Finite number of bits for representing a number
- floating point representation

$$\boxed{s \times M \times B^{e-E}}$$

s: sign bit

M: exact positive integer mantissa

B: base(2 or 16)

e: exact integer exponent

E: bias of the exponent (machine dependent)

IEEE Binary Floating Point Arithmetic Standard 754 - 1985

- Eg. Double precision real numbers (64 bits)

$$(-1)^s * 2^{c-1023} * (1+f)$$

c: 11 bit exponent(0 ~ 2^{11}-1)

f: 52 bit binary fraction

$$0\ \underbrace{10000000011}_{c}\ \underbrace{1011100100010000000000000000000000000000000000000000}_{f}$$

$$c = 1 \cdot 2^{10} + 0 \cdot 2^{9} + \cdots + 0 \cdot 2^{2} + 1 \cdot 2^{1} + 1 \cdot 2^{0} = 1024 + 2 + 1 = 1027$$

$$f = 1 \cdot \left(\frac{1}{2}\right)^{1} + 1 \cdot \left(\frac{1}{2}\right)^{3} + 1 \cdot \left(\frac{1}{2}\right)^{4} + 1 \cdot \left(\frac{1}{2}\right)^{5} + 1 \cdot \left(\frac{1}{2}\right)^{8} + 1 \cdot \left(\frac{1}{2}\right)^{12}$$

Eg. 64 bit floating point number

$$0\ \underbrace{10000000011}_{c}\ \underbrace{1011100100010000000000000000000000000000000000000000}_{f}$$

$$c = 1 \cdot 2^{10} + 0 \cdot 2^{9} + \cdots + 0 \cdot 2^{2} + 1 \cdot 2^{1} + 1 \cdot 2^{0} = 1024 + 2 + 1 = 1027$$

$$f = 1 \cdot \left(\frac{1}{2}\right)^{1} + 1 \cdot \left(\frac{1}{2}\right)^{3} + 1 \cdot \left(\frac{1}{2}\right)^{4} + 1 \cdot \left(\frac{1}{2}\right)^{5} + 1 \cdot \left(\frac{1}{2}\right)^{8} + 1 \cdot \left(\frac{1}{2}\right)^{12}$$

$$\boxed{\begin{aligned}
&(-1)^{s} * 2^{c-1023} * (1+f) \\
&\qquad = (-1)^{0} \cdot 2^{1027-1023}\left(1 + \left(\frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{4096}\right)\right) \\
&\qquad = 27.56640625.
\end{aligned}}$$

Accuracy, range…

- Accuracy
  - 27.56640625 represents the interval

$$[27.5664062499999998889776975374843459576368331909179687 5,$$

$$27.5664062500000001110223024625156540423631668090820312 5).$$

- Smallest number

$$2^{-1022} \cdot (1+0) \approx 0.2225 \times 10^{-307}$$

- Largest number

$$2^{1023} \cdot \left(1 + \left(1 - 2^{-52}\right)\right) \approx 0.17977 \times 10^{309}$$

- Underflow: when smaller than the smallest number
  - Generally set to 0
- Overflow: when larger than the largest number
  - Generally cause the computation to stop

Machine Accuracy

Def. The smallest floating-point number which, when added to the floating-point number 1.0, produces a floating-point result different from 1.0

$$\varepsilon = b^{1-m} \quad \text{(m=\# of bit for mantissa)}$$

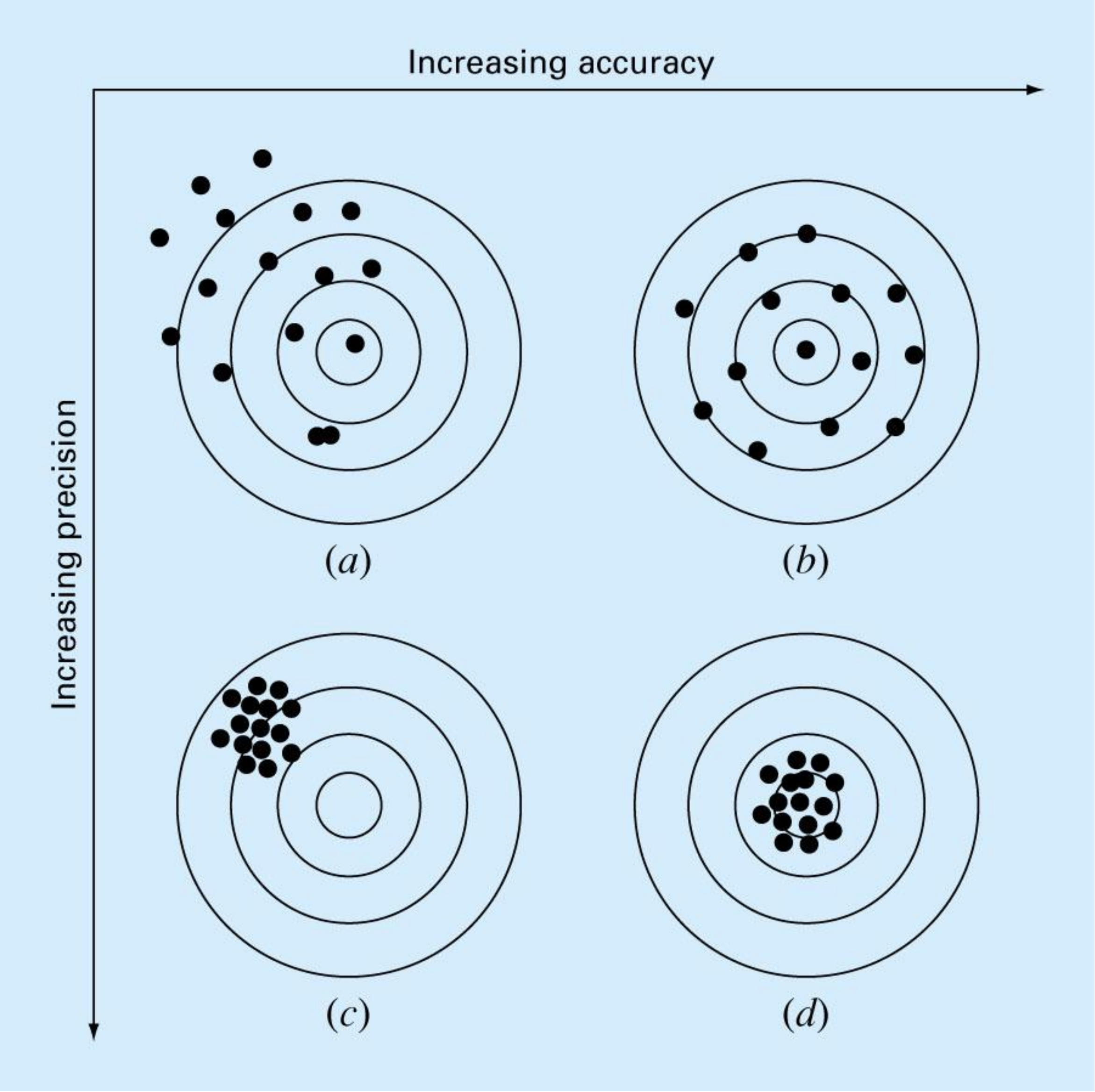
eg. typical machines with b=2 and a 32-bit word length ~ 10^{-8} .

Homework #1

[Due: 9/10]

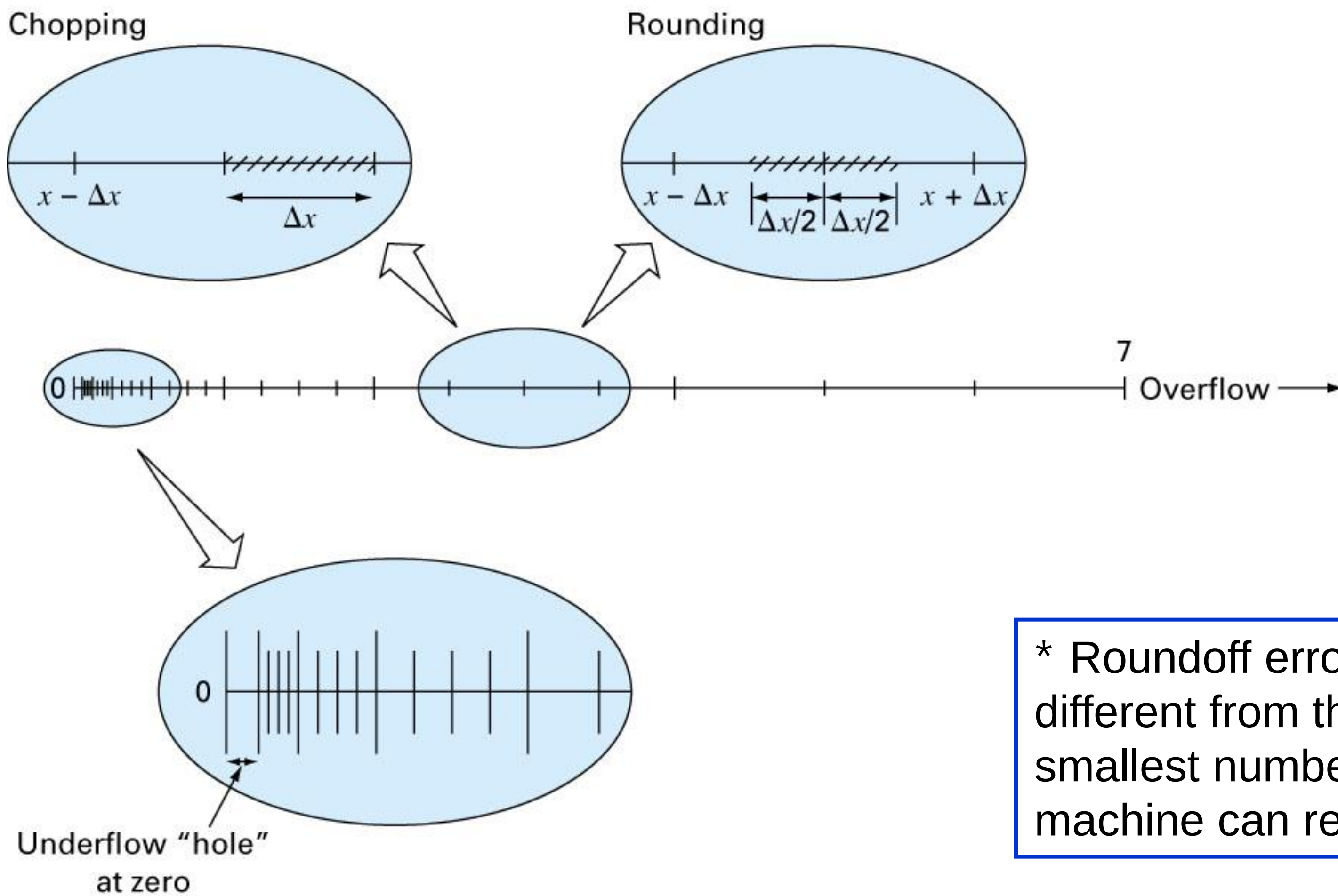
- Programming: Obtain the machine accuracy of “float” and “double” of your computer in two ways.
  - Method 1: Use the routine machar() in NR in C (Modification is required for “double”)
  - Method 2: Use your own code, get_eps(), which is based on finding minimum n that satisfies $1+2^{-n}=1$

Accuracy and Precision



Roundoff error

- Due to machine accuracy
  - Chopping
  - Symmetric rounding



* Roundoff error is different from the smallest number that a machine can represent

Minimizing roundoff errors

- Keep intermediate values around ± 1 (close to middle of all floating point numbers) to avoid overflow or underflow
- Minimize the number of arithmetic manipulations in order to suppress the accumulation of errors.
  - Eg.) nested multiplication
- Avoid subtractive cancellation
  - Avoid subtraction between similar numbers
  - Start from small numbers
- Use double precision

Loss of significant digit

- Eg. Evaluate F(x)=x(sqrt(x+1)-sqrt(x)) at x=100 with 6s (significant digit = 6). True value: F(100)=4.98756

  - Method 1: Direct calculation

    sqrt(x+1)=10.0498, sqrt(x)=10.0000

    x(sqrt(x+1)-sqrt(x))=100(10.0498-10.0000)=4.98000

    Error=0.00756

  - Method 2: Modification of formula

    F(x)= x/(sqrt(x+1)+sqrt(x)) =4.98758

    Error=0.00002

Losing significant digit

0.0498 ➔ 3s

Truncation error

- The error between exact solution and computed solution using practical machines
- Due to approximation of formula
- The goal of numerical analysis: Minimizing truncation error!
- Round-off error is out of our control (depends on machine)
- Truncation error depends on algorithm (how we calculate)

Taylor series

Suppose $f \in C^n[a, b]$ and $f^{(n+1)}$ exists on $[a, b]$. Let x_0 be a number in $[a, b]$. For every x in $[a, b]$, there exists a number $\xi(x)$ between x_0 and x with

$$f(x) = P_n(x) + R_n(x),$$

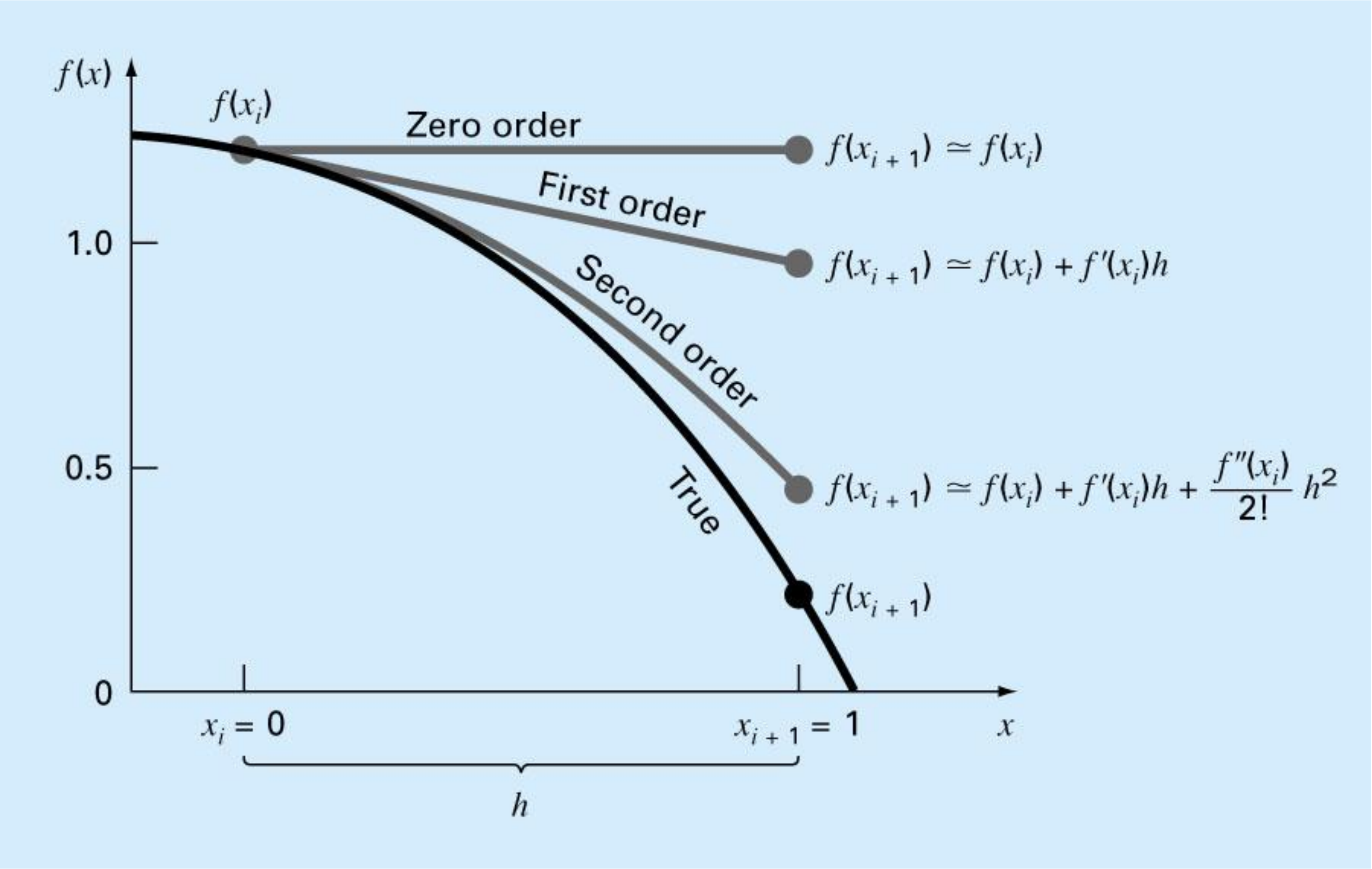
where

$$\begin{aligned} P_n(x) &= f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \\ &= \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k \end{aligned}$$

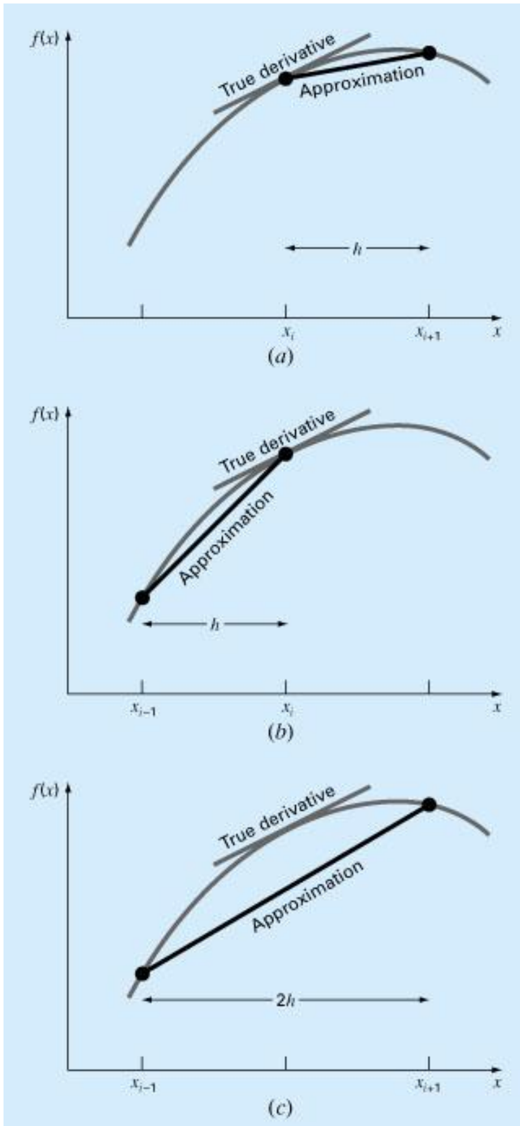
and

$$R_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n + 1)!}(x - x_0)^{n+1}.$$

Approximation



Approximation of the 1st derivative



Forward difference

Backward difference

Centered difference

Error

- Absolute error: E_t = (true value-approximation)
- Relative error: e_t =(true value - approximation) / true value
- Approximate relative error :

  e_a =(Approx.TrueValue-approxi)/ Approx.TrueValue

Stopping condition in iterative algorithms:

Terminate computation when

$$e_a < e_s$$

where e_s is the desired relative error

Data errors

- data error

$$\Delta f(\tilde{x}) = | f(x) - f(\tilde{x}) | \cong | f'(\tilde{x}) | (x - \tilde{x})$$

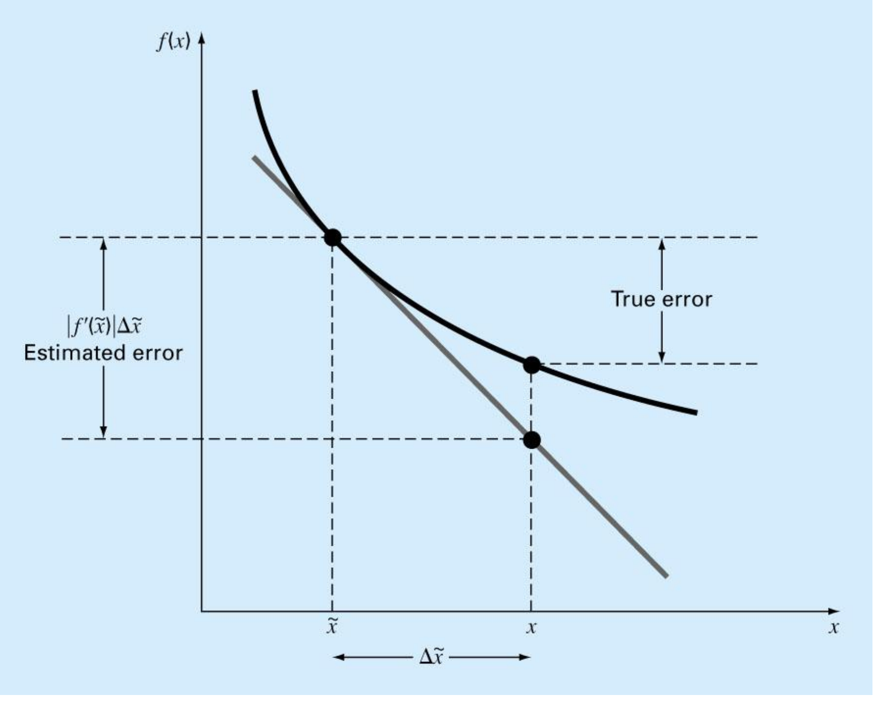
Ignoring higher order terms of Taylor series

- data relative error

$$e_f = \frac{| f'(\tilde{x}) | (x - \tilde{x})}{f(\tilde{x})}$$

- relative error of x

$$e_x = \frac{x - \tilde{x}}{\tilde{x}}$$



Condition number

$$\frac{e_f}{e_x} = \frac{\tilde{x} f'(\tilde{x})}{f(\tilde{x})}$$

if condition number < 1 ➔ error reduction

if condition number >> 1 ➔ ill-conditioned

Error propagation

- Single variable function

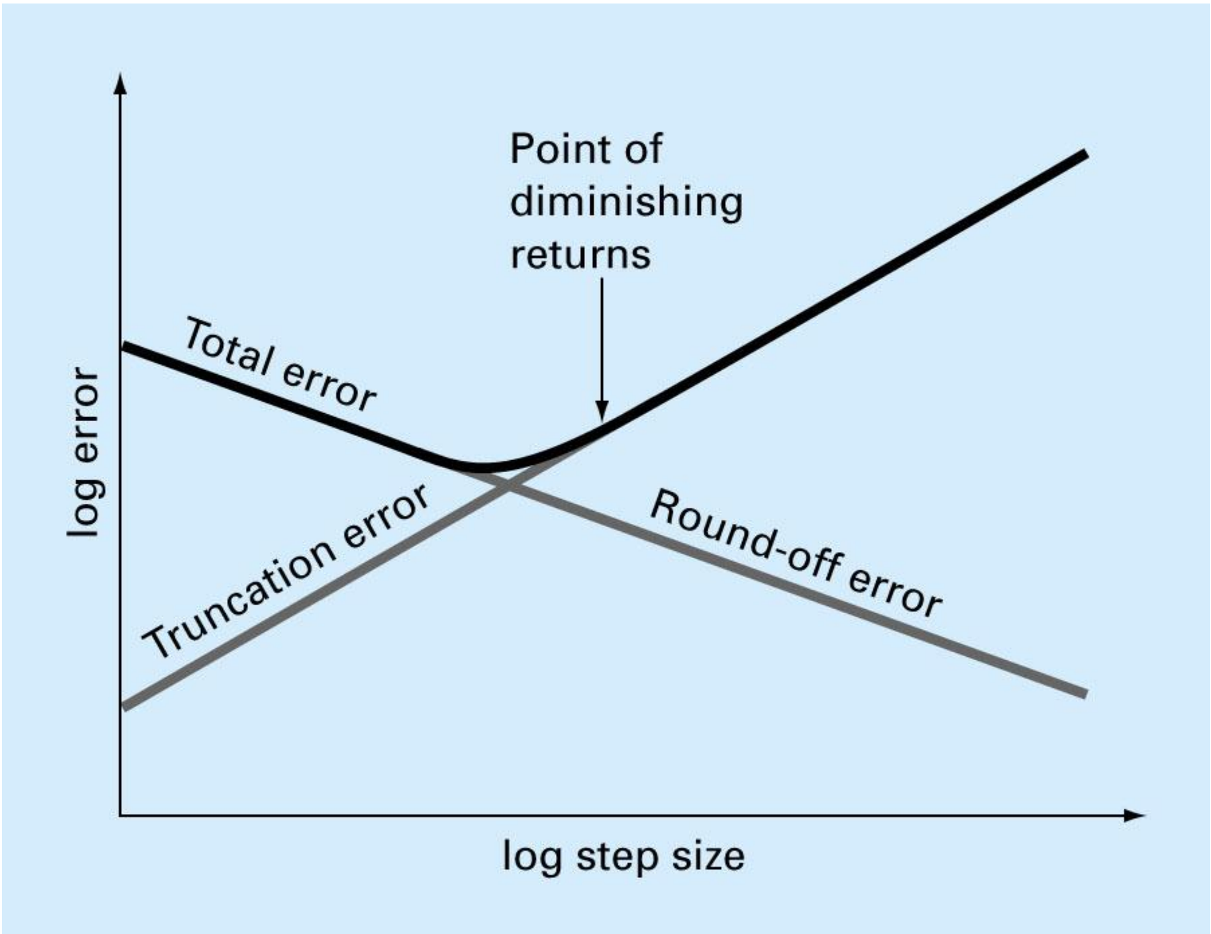
$$\Delta f(\tilde{x}) = | f(x) - f(\tilde{x}) | = | f'(\tilde{x}) | (x - \tilde{x})$$

- Multivariable function

$$\Delta f(\tilde{x}_1, \tilde{x}_2, \cdots, \tilde{x}_n) \cong \left|\frac{\partial f}{\partial x_1}\right| \Delta \tilde{x}_1 + \left|\frac{\partial f}{\partial x_2}\right| \Delta \tilde{x}_2 + \cdots + \left|\frac{\partial f}{\partial x_n}\right| \Delta \tilde{x}_n$$

Total error

- Total error = roundoff error + truncation error



Trade-off

Interval h

Homework

- Read Chapter 1, Numerical Recipes in C
  - how to use pointers for memory allocation
  - how to use pointer to function
- Solve the problems: 3.6, 3.7, 4.2, 4.5, 4.12