Roundoff and Truncation Errors

Berlin Chen
Department of Computer Science & Information Engineering
National Taiwan Normal University

Reference:

1. *Applied Numerical Methods with MATLAB for Engineers*, Chapter 4 & Teaching material

Chapter Objectives (1/2)

- Understanding the distinction between accuracy and precision
- Learning how to quantify error
- Learning how error estimates can be used to decide when to terminate an iterative calculation
- Understanding how roundoff errors occur because digital computers have a limited ability to represent numbers
- Understanding why floating-point numbers have limits on their range and precision

$$\frac{dv}{dt} \approx \frac{\Delta v}{\Delta t} = \frac{v(t_{i+1}) - v(t_i)}{t_{i+1} - t_i}$$

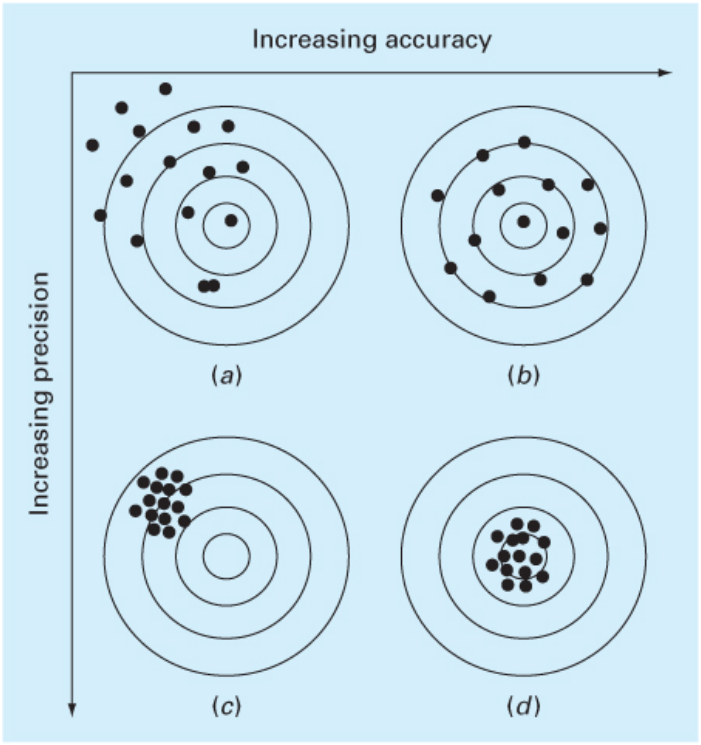
Chapter Objectives (2/2)

- Recognizing that truncation errors occur when exact mathematical formulations are represented by approximations
- Knowing how to use the Taylor series to estimate truncation errors
- Understanding how to write forward, backward, and centered finite-difference approximations of the first and second derivatives
- Recognizing that efforts to minimize truncation errors can sometimes increase roundoff errors

Accuracy and Precision

- ***Accuracy*** refers to how closely a computed or measured value agrees with the true value, while ***precision*** refers to how closely individual computed or measured values agree with each other

a) inaccurate and imprecise
b) accurate and imprecise
c) inaccurate and precise
d) accurate and precise



Error Definitions (1/2)

- **True error** (E_t): the difference between the true value and the approximation

$$E_t = \text{true value} - \text{approximation}$$

- **Absolute error** ($|E_t|$): the absolute difference between the true value and the approximation

- **True fractional relative error**: the true error divided by the true value

$$\text{True fractional relative error} = \frac{\text{true value} - \text{approximation}}{\text{true value}}$$

- **Relative error** (ε_t): the true fractional relative error expressed as a percentage

$$\varepsilon_t = \frac{\text{true value} - \text{approximation}}{\text{true value}} 100\%$$

10 → 9

$\varepsilon_t = 10\%$

10,000 → 9,999

$\varepsilon_t = 0.01\%$

Error Definitions (2/2)

- The previous definitions of error relied on knowing a true value. If that is not the case, approximations can be made to the error
- The approximate percent relative error can be given as the approximate error divided by the approximation, expressed as a percentage - though this presents the challenge of finding the approximate error!

$$\varepsilon_a = \frac{\text{approximate error}}{\text{approximation}} 100\%$$

- For iterative processes, the error can be approximated as the difference in values between successive iterations

$$\varepsilon_a = \frac{\text{present approximation - previous approximation}}{\text{present approximation}} 100\%$$

Using Error Estimates

- Often, when performing calculations, we may not be concerned with the sign of the error but are interested in whether the absolute value of the percent relative error is lower than a prespecified tolerance ε_s
  - For such cases, the computation is repeated until $|\varepsilon_a| < \varepsilon_s$
    - This relationship is referred to as a ***stopping criterion***
- Note that for the remainder of our discussions, we almost always employ absolute values when using relative errors
- We say that an approximation is correct to **at least *n* significant figures (significant digits)** if its $|\varepsilon_a|$ is smaller than ε_s that has a value

$$\varepsilon_a = (0.5 \times 10^{2-n})\%$$

Example: Exponential Function

- It is known that the exponential function can be computed using

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} \qquad \text{(Maclaurin series expansion)}$$

  - Try to add terms (1, 2, …, *n*) until the absolute value of the approximate error estimate $|\varepsilon_a|$ falls below a prescribed error criterion ε_s conforming to three significant figures

true value :

$e^x = 1.648721\cdots$

Terms	Result	ε_t, %	ε_a, %
1	1	39.3	
2	1.5	9.02	33.3
3	1.625	1.44	7.69
4	1.645833333	0.175	1.27
5	1.648437500	0.0172	0.158
6	1.648697917	0.00142	0.0158

Thus, after six terms are included, the approximate error falls below $\varepsilon_s = 0.05\%$, and the computation is terminated. However, notice that, rather than three significant figures, the result is accurate to five! This is because, for this case, both Eqs. (4.5) and (4.7) are conservative. That is, they ensure that the result is at least as good as they specify. Although, this is not always the case for Eq. (4.5), it is true most of the time.

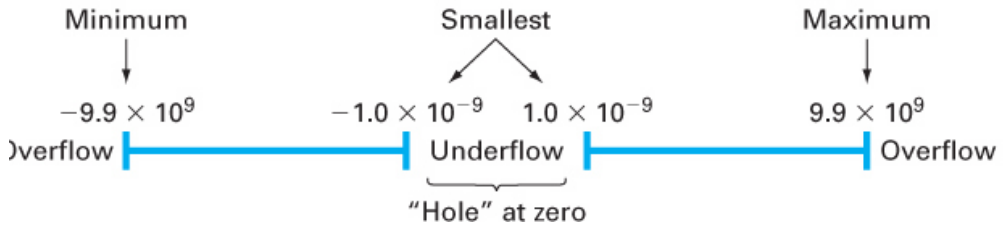
Roundoff Errors

- ***Roundoff errors*** arise because digital computers cannot represent some quantities exactly. There are two major facets of roundoff errors involved in numerical calculations:
  - Digital computers have size and precision limits on their ability to represent numbers
  - Certain numerical manipulations are highly sensitive to roundoff errors

Example: a 10-based Floating-point System

$$s_1 d_1 . d_2 \times 10^{s_0 d_0}$$

Significand (mantissa) magnitude of the exponent



– If 0.03125 is represented by the system as $3.1X10^{-2}$, a roundoff error is introduced

$$\frac{0.03125 - 0.031}{0.03125} = 0.008$$

– The roundoff error of a number will be proportional to its magnitude

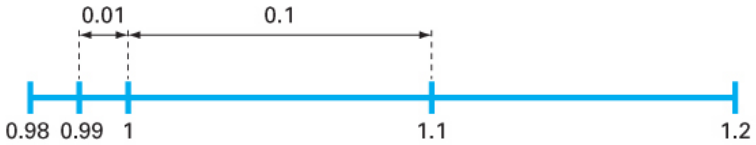


FIGURE 4.5
A small portion of the number line corresponding to the hypothetical base-10 floating-point scheme described in Example 4.2. The numbers indicate values that can be represented exactly. All other quantities falling in the "holes" between these values would exhibit some roundoff error.

Computer Number Representation

- By default, MATLAB has adopted the IEEE double-precision format in which eight bytes (64 bits) are used to represent **floating-point** numbers:

$$n = \pm(1+f) \times 2^{e}$$

Binary numbers consist exclusively of 0s and 1s. When normalized, the leading bit (always 1) does not have to be stored. Only the fractional part of the significand is stored.

- The sign is determined by a sign bit
- The mantissa *f* is determined by a 52-bit binary number
- The exponent *e* is determined by an 11-bit binary number, from which 1023 is subtracted to get *e*

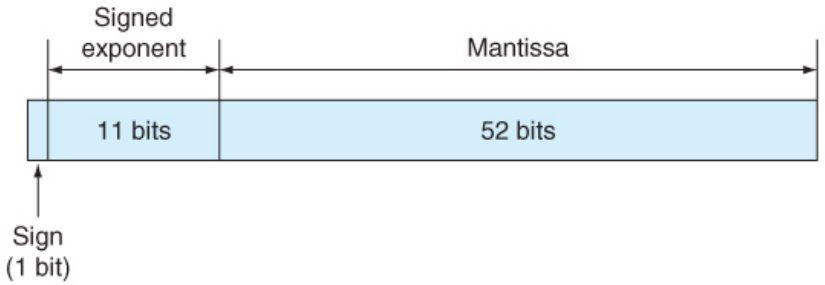


FIGURE 4.6
The manner in which a floating-point number is stored in an 8-byte word in IEEE double-precision format.

$$\text{realmax} = 1.797693134862316\text{e}+308$$
$$\text{realmin} = 2.225073858507201\text{e}-308$$

Floating Point Ranges

- Values of –1023 and +1024 for *e* are reserved for special meanings, so the exponent range is –1022 to 1023

- The largest possible number MATLAB can store has
  - *f* of all 1's, giving a significand of $2 - 2^{-52}$, or approximately 2
  - *e* of 11111111110_2, giving an exponent of 2046 – 1023 = 1023
  - This yields approximately $2^{1024} \approx 1.7997 \times 10^{308}$

- The smallest possible number MATLAB can store with full precision has
  - *f* of all 0's, giving a significand of 1
  - *e* of 00000000001_2, giving an exponent of 1–1023 = –1022
  - This yields $2^{-1022} \approx 2.2251 \times 10^{-308}$

Floating Point Precision

- The 52 bits for the mantissa *f* correspond to about 15 to 16 base-10 digits. The machine epsilon (ε) - the maximum relative error between a number and MATLAB's representation of that number, is thus $2^{-52} = 2.2204 \times 10^{-16}$

Roundoff Errors with Arithmetic Manipulations

- Roundoff error can happen in several circumstances other than just storing numbers - for example:
  - ***Large computations*** - if a process performs a large number of computations, roundoff errors may build up to become significant

```
function sout = sumdemo()
s = 0;
for i = 1:10000
   s = s + 0.0001;
% notice that 0.0001 cannot be expressed exactly in base - 2.
end
sout = s;
```

```
function sout = sumdemo()
s = 0;
for i = 1:10000
   s = s + 0.0001;
>> format long
>> sumdemo
ans =
   0.99999999999991
```

  - ***Adding a Large and a Small Number*** - Since the small number's mantissa is shifted to the right to be the same scale as the large number, digits are lost

What if $0.0010 + 4000$ is represented with 4 - digit mantissa and 1 - digit exponent?

$$\begin{array}{ll} 4.000 & \times 10^3 \\ \underline{0.000001} & \times 10^3 \\ \underline{4.000}001 & \times 10^3 \quad \text{(chopped to } 4.000 \times 10^3\text{)} \end{array}$$

  - ***Smearing*** - Smearing occurs whenever the individual terms in a summation are larger than the summation itself
    - $(x + 10^{-20}) - x = 10^{-20}$ mathematically, but $x = 1$; $(x + 10^{-20}) - x$ gives a 0 in MATLAB!

Truncation Errors

- ***Truncation errors*** are those that result from using an approximation in place of an exact mathematical procedure
- Example 1: approximation to a derivative using a finite-difference equation:

$$\frac{dv}{dt} \cong \frac{\Delta v}{\Delta t} = \frac{v(t_{i+1}) - v(t_i)}{t_{i+1} - t_i}$$

- Example 2: The Taylor Series

The Taylor Theorem and Series

- The ***Taylor theorem*** states that any smooth function can be approximated as a polynomial
- The *Taylor series* provides a means to express this idea mathematically
- A good problem context is to use Taylor series to predict a function value at one point in terms of the function value and its derivatives at another point

$$f(x_{i+1}) \approx f(x_i) \qquad \text{(constant)}$$

$$f(x_{i+1}) \approx f(x_i) + f'(x_i)h \qquad \text{(straight line)}$$

$$f(x_{i+1}) \approx f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 \qquad \text{(parabola)}$$

$$\vdots$$

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \frac{f^{(3)}(x_i)}{3!}h^3 + \cdots + \frac{f^{(n)}(x_i)}{n!}h^n + R_n$$

The Taylor Series

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \frac{f^{(3)}(x_i)}{3!}h^3 + \cdots + \frac{f^{(n)}(x_i)}{n!}h^n + R_n$$

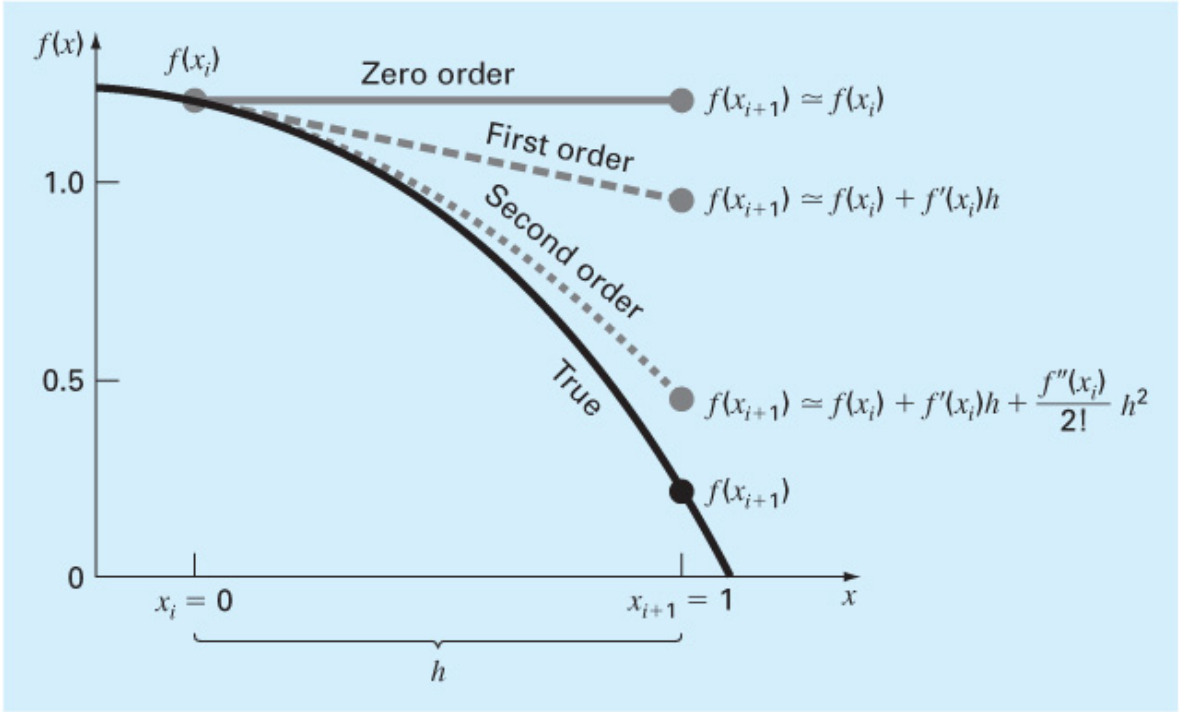


FIGURE 4.7
The approximation of $f(x) = -0.1x^4 - 0.15x^3 - 0.5x^2 - 0.25x + 1.2$ at $x = 1$ by zero-order, first-order, and second-order Taylor series expansions.

The Taylor Series: Remainder Term

- In the complete Taylor series expansion, a remainder term R_n is included to account for all terms from n+1 to infinite

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!} h^{n+1}$$

  - Where the subscript n connote that this is the remainder for the nth-order approximation and ξ is a value of x that lies somewhere between x_i and x_{i+1}

More on Truncation Errors

- In general, the *n*th order Taylor series expansion will be exact for an *n*th order **polynomial**
  - Any smooth function can be approximated as a polynomial

- In other cases, the remainder term R_n is of the order of h^{n+1}, meaning:
  - The more terms are used, the smaller the error, and
  - The smaller the spacing, the smaller the error for a given number of terms

Numerical Differentiation (1/2)

- The first order Taylor series can be used to calculate approximations to derivatives:
  - Given:

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + O(h^2)$$

  - Then:

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + O(h)$$

- This is termed a "**forward**" difference because it utilizes data at *i* and *i*+1 to estimate the derivative

Numerical Differentiation (2/2)

- There are also **backward** and **centered** difference approximations, depending on the points used:
  - Forward:

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + O(h)$$

  - Backward:

$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h} + O(h)$$

  - Centered

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + O(h^2)$$

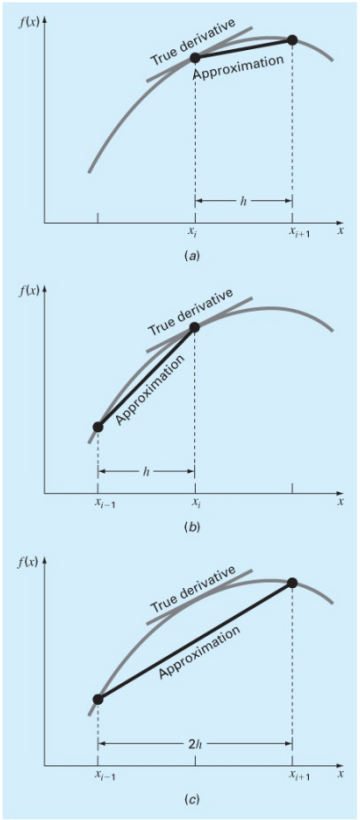


FIGURE 4.10
Graphical depiction of (a) forward, (b) backward, and (c) centered finite-difference approximations of the first derivative.

Total Numerical Error

- The ***total numerical error*** is the summation of the truncation and roundoff errors
- The truncation error generally ***increases*** as the step size increases, while the roundoff error ***decreases*** as the step size increases - this leads to a point of diminishing returns for step size

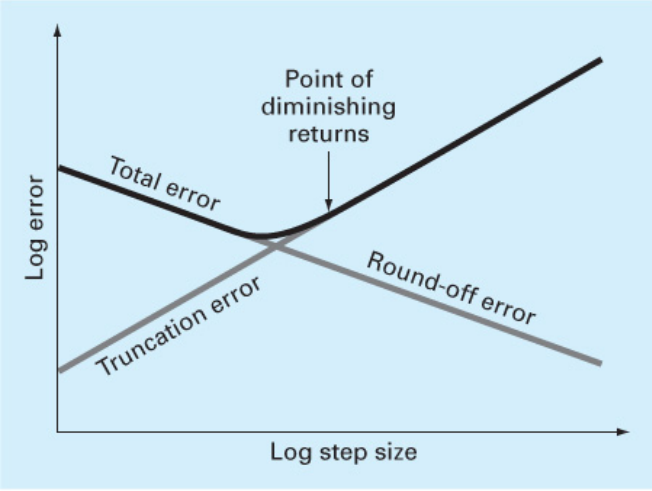


FIGURE 4.11
A graphical depiction of the trade-off between roundoff and truncation error that sometimes comes into play in the course of a numerical method. The point of diminishing returns is shown, where roundoff error begins to negate the benefits of step-size reduction.

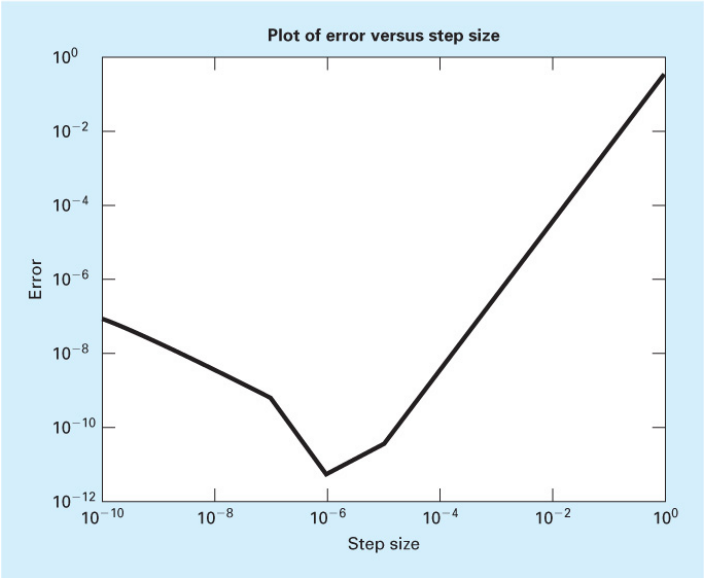
Finite-Difference Approximation of Derivatives

- Given a function:

$$f(x) = -0.1x^4 - 0.15x^3 - 0.5x^2 - 0.25x + 1.2$$

- we can use a **centered difference approximation** to estimate the first derivative of the above function at x=0.5
  - However, if we progressively divide the step size by a factor of 10, roundoff errors become dominant as the step size is reduced

FIGURE 4.12



Other Errors (1/2)

- **Blunders** - errors caused by malfunctions of the computer or human imperfection
  - In early years of computers, erroneous numerical results could sometimes be attributed to malfunctions of the computer itself
  - Today, most blunders must be attributed to human imperfection
    - Can be avoided only by sound knowledge of fundamental principles and by the care when approaching and designing our solutions to the problem
- **Model errors** - errors resulting from incomplete mathematical models
  - When some latent effects are not taken into account or ignored

Other Errors (2/2)

- **Data uncertainty** - errors resulting from the accuracy and/or precision of the data
  - When with biased (underestimation/overestimation) or imprecise instruments
  - We can use descriptive statistics (viz. mean and variance) to provide a measure of the bias and imprecision

- For most of this course, we will assume that we have not made gross errors (blunders), we have a sound model, and we are dealing with error-free measurements