ME 261: Numerical Analysis

- 3.00 credit hours Prereq.: ME 163/ME 171

- **Course content**
  - Approximations and error types
  - Roots of polynomials and transcendental equations
  - Determinants and matrices
  - Solution of linear and non-linear algebraic equations
  - Eigenvalues and eigenvectors
  - Interpolation methods
  - Curve fitting
  - Numerical differentiation and integration
  - Solution of first-order differential equations
  - Solving equations by finite differences

1

ME 261: Numerical Analysis

- **Course teachers**

1. **Dr. S. Reaz Ahmed** (Sunday)
   Professor
2. **Dr. M Zakir Hossain** (Saturday, Tuesday)
   Assistant Professor
   Dept. of Mechanical Engineering
   BUET, Dhaka-1000, Bangladesh.

   Room: M602 (EME Building)
   email: zakir(at)me.buet.ac.bd, zakir92(at)yahoo.com
   web: teacher.buet.ac.bd/zakir

2

ME 261: Numerical Analysis

- **Text book**
  -Numerical Methods for Engineers
  by Steven C. Chapra and Raymond P. Canale
  Publishing company: Tata McGraw-Hill

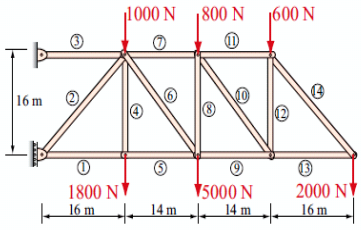
- **Reference book**
  1. Applied Numerical Analysis
  by Curtis F. Gerald and Patrik O. Wheatley
  Publishing company: Pearson
  2. Numerical Analysis
  by Timothy Sauer
  Publishing company: Pearson

- **ACADEMIC OFFENCES**
  Students must write their assignments in their own words.
  If students take an idea, or a passage of text from book, journal, web etc, they must acknowledge this by proper referencing such as footnotes or citations.
  Plagiarism is a major academic offence.

3

Why Numerical Analysis ?

- Example: computing the forces in a TRUSS



$$
\begin{aligned}
-F_1 + F_5 &= 0\\
-0.71071F_2 - F_3 + 0.6585F_6 + F_7 &= 0\\
-0.7071F_2 - F_4 - 0.7525F_6 &= 1000\\
F_4 &= 1800\\
-F_7 + 0.6585F_{10} + F_{11} &= 0\\
-F_8 - 0.7525F_{10} &= 800\\
-F_5 - 0.6585F_6 + F_9 &= 0\\
0.7525F_6 + F_8 &= 5000\\
-F_{11} + 0.7071F_{14} &= 0\\
-F_{12} - 0.7071F_{14} &= 600\\
-F_9 - 0.6585F_{10} + F_{13} &= 0\\
0.7525F_{10} + F_{12} &= 0\\
-F_{13} + 0.7071F_{14} &= 0\\
0.7071F_{14} &= 2000
\end{aligned}
$$

- 14 equations with 14 unknowns

4

Floating Point Arithmetic

- fractional quantities are typically represented in computers using floating point format
- this approach is very much similar to scientific notation
- for example, *fixed point number* 17.542 is the same as the floating point number $.17542*10^2$ which is often displayed as .17542e2
- another example, -.004428 is same as $-.4428*10^{-2}$

General form of floating-point number in Computers:

$$\pm .d_1 d_2 d_3 \ldots . d_p * B^{e.}$$

where, d_j's are digits or bits with values from 0 to B-1
B=the number base that is used, e.g. 2, 10,16,8
P=number of significand bits (digits), that is, the precision.
e=an integer exponent, ranging from E_{min} to E_{max}

- $d_1 d_2 d_3 \ldots d_p$ constitute the fractional part of the number

5

Floating Point Arithmetic

Normalization:

- the fractional digits are shifted and the exponents are adjusted so that d_1 is NONZERO

e.g.
$-.004428 \cong -0.4428*10^{-2}$ is the normalized form

- In base-2(binary) system normalization means *the first bit is always 1*

IEEE Standard for the floating-point numbers:

- consists of a set of binary representations of real numbers
- normalization gives the advantage that the first bit does not need to be stored
- Hence, nonzero binary floating point numbers can be expressed as,

$$\pm (1+ f)*2^e$$

where f =the mantissa (or the fraction)
e= exponent

6

Floating Point Arithmetic

- e.g. the decimal number 9, which is 1001 in binary, would be stored as,

$$+1.001 * 2^3$$

although the original number has 4 significant digits, we only have to store the 3 fractional bits: .001

- There are three commonly used levels of precision for floating point numbers:
  - single precision (32 bits)
  - double precision (64 bits)
  - long double (80 bits)

The bits are divided as follows:

precision	sign	exponent	mantissa
single	1	8	23
double	1	11	52
long double	1	15	64

7

Floating Point Arithmetic

- By default, MATLAB has adopted IEEE double precision format

Figure: The manner in which a floating point number is stored in IEEE double precision format

Note: because of normalization, 53 bits can be stored in mantissa.

- Sign bit is 0 for positive numbers and 1 for negative numbers

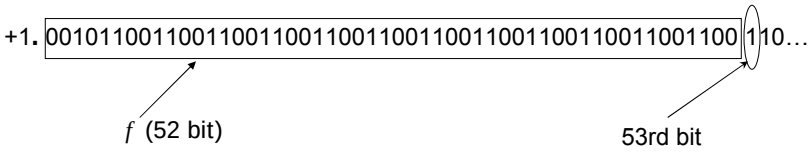
8

Floating Point Arithmetic

The decimal number $9.4 = (1001.\overline{0110})_2 = (1.00\overline{10110})_2 * 2^3$

Normalized

It will be stored in exponent



f (52 bit)

53rd bit

How do we fit the infinite binary number representing 9.4 in a finite number of bits?

9

We must truncate the number in some way, and in doing so we necessarily make a small error

We can use two methods to truncate

Chopping:

- Simply through away the bits that fall off the end

  -that is, those beyond the 52nd bit

- This method is simple, but it is biased in that it always moves the result toward zero

10

Rounding:

- In base 10, numbers are customarily rounded up if the next digit is 5 or higher, and rounded down otherwise
- In binary, this corresponds to rounding up if the bit is 1
- Specifically, the important bit in the double precision format is the 53rd bit

IEEE Rounding to Nearest Rule:

if bit 53 is 0,then truncate after the 52nd bit (round down)
if bit 53 is 1, add 1 to 52nd bit (round up)
exception: if the bits 53rd and beyond are like 10000000...,
then add 1 to 52nd bit if and only if 52nd bit is 1.

11

For the number (9.4) discussed previously, the 53rd bit in mantissa is 1

+1. 0010110011001100110011001100110011001100110011001100 110…

53rd bit

So, the Rounding to the Nearest Rule says to round up, i.e. to add 1 to bit 52

Therefore the mantissa of floating point number that represents 9.4 is

+1. 0010110011001100110011001100110011001100110011001101

12

Now let's see how much error occurs due to this rounding of the mantissa

We have discarded the bits after 53rd, so error associated with discarding (Ed) is

$$= .\overline{110} * 2^{-52} * 2^{3} \quad = .\overline{0110} * 2^{-51} * 2^{3} \quad = .4 * 2^{-48}$$

We have added 1 in the 52nd bit, so error associated with roundup (Er) is

$$2^{-52} * 2^{3} = 2^{-49}$$

Therefore, fl(9.4)=9.4+Er-Ed

$$= 9.4 + 2^{-49} - 0.4 * 2^{-48} = 9.4 + (1 - 0.8) * 2^{-49}$$

$$= 9.4 + 0.2 * 2^{-49}$$

Roundoff error

In other words, a computer using double precision representation and the Rounding to the nearest Rule makes as error of $0.2*2^{-49}$

13

Machine Epsilon ε_{mach}

52 bits of mantissa

The double precision number 1 is

+1. 0000000000000000000000000000000000000000000000000000

The next floating point number greater than 1 is

+1. 0000000000000000000000000000000000000000000000000001

$\Rightarrow 1+2^{-52}$

So, the number machine Epsilon, denoted by ε_{mach}, is the distance between 1 and the smallest floating point number greater than 1.

For the IEEE double precision floating point standard,

$$\varepsilon_{mach} = 2^{-52} = 2.2204 * 10^{-16}$$

It is a quantitative measure of the machine's ability to perform computations accurately, depending upon the precision in use.

14

Machine Epsilon ε_{mach}

- Double precision numbers can be computed reliably

up to machine epsilon.

- MATLAB has a library function **eps** to display ε_{mach}

15

Absolute and relative error

Let X_{comp} be a computed version of the exact quantity X_{exact}

Absolute error $= |X_{comp} - X_{exact}|$ Relative error $= \dfrac{|X_{comp} - X_{exact}|}{|X_{exact}|}$

- absolute error does not account the order of magnitude of the value under examination.
- Relative error can be expressed in %
- For example, we want to measure the length of bridge and a rivet.
- We measured the length of bridge is 9999 cm and the length of the rivet is 9 cm.
- The exact values are 10000 and 10 cm, respectively.
- The absolute error for both cases is 1 cm.
- The percentage relative errors are 0.01% and 10%.
- That means, measurement of the length of the bridge is quite accurate, but the measurement of the rivet has significant error.

16

Arithmetic accuracy in Computers

- *Roundoff errors* arise because digital computers cannot represent some quantities exactly.
- Aside from the limitations of a computer's number system, the actual arithmetic manipulation involving this numbers can also result in roundoff error.
- To understand how this occurs, let's look at how the computer performs simple addition and subtraction.
- Because of their familiarity, normalized base-10 numbers will be employed to illustrate the effect of roundoff errors on simple arithmetic manipulations.
- Other number bases would behave in a similar fashion.
- To simplify the discussion, we assume a hypothetical computer with a 3-digit mantissa, and a 1-digit exponent, i.e. B=10, p=3, $-9 \le e \le 9$.

17

Arithmetic accuracy in Computers

- When two floating-point numbers are added /or subtracted,
  - the numbers are first expressed so that they have the same exponents.

examples,

Addition:

- if we want to add 1.371 + 0.02693
- the computer would express the numbers as

$$\begin{array}{rl} & 0.1371 \times 10^{1} \\ + & 0.002693 \times 10^{1} \\ \hline & 0.139793 \times 10^{1} \end{array}$$

Align the decimal points

- Because this hypothetical computer only carries 3-digit mantissa
  if rounded, → 0.140×10^{1}
  if chopped, → 0.139×10^{1}

18

Arithmetic accuracy in Computers

Subtraction:

- compute 76.4 – 76.3 on a 3-decimal-digit computer.
- The computer would express the numbers as

$$\begin{array}{rll} & 0.764 & * 10^2 \\ - & 0.763 & * 10^2 \\ \hline & 0.001 & * 10^2 \end{array}$$

3 significant digits

- After normalization, $0.100 * 10^0$

Two non-significant zeros are appended

Subtractive cancelation/loss of significant:

- subtraction of two nearly equal numbers.
- a major source of error in floating-point operations.

19

Subtractive cancellation/loss of significant

- subtraction of two nearly equal numbers
- a major source of error in floating-point operations

Example-1

Example-2

Lesson

- It is important to find ways to avoid subtracting nearly equal numbers in calculations

20

Arithmetic accuracy in Computers

- 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
    - Example
  - *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
  - *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; $(x+1e^{-20})-x$ gives a 0 in MATLAB!

21

Adding a Large and a Small Number

- Suppose we want to add a small number 0.001 to a large number 4000 using a hypothetical computer with the 4-digit mantissa and the 1-digit exponent

4000	0.400	$* 10^4$
0.001	0.000 000 1	$* 10^4$
	0.400 000 1	$*10^4$

- The final result will be $0.4000 *10^4$ (since we have 4-digit mantissa)
- It seems that we have not performed any addition !!!
- This type of error may occur in the computation of an infinite series

Example

- One way to mitigate this type of error is to sum the series in reverse order
- In this way each new term will be of comparable magnitude to the accumulated sum

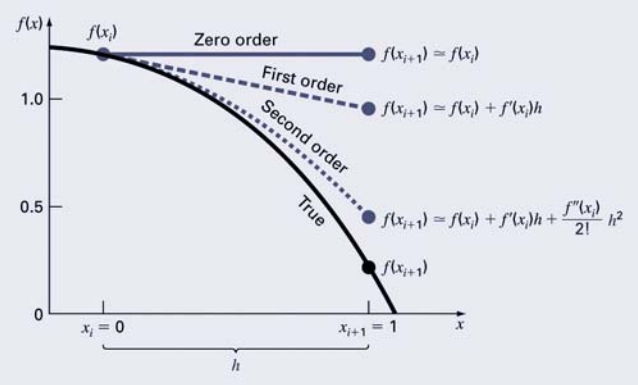
22

Truncation Error

Truncation errors are those that result from using an approximation in place of an exact mathematical procedure.

Example: The Taylor Series

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



23

Truncation Error

R_n =O(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.

Example 2: Numerical differentiation

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

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

If h is decreased (↓) truncation error decreases(↓)

24

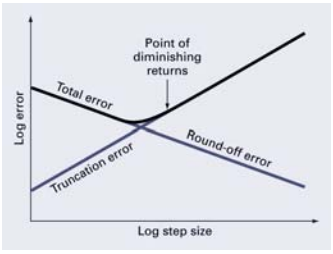
Error Propagation

- Functions of a single variable
  - Derivation using Taylor series for a single variable
  - Example
- Functions of more than one variable
  - Derivation using multivariable version of Taylor series
  - Example

25

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,
- To minimize the roundoff error, increase the number of precision (bits in mantissa) in computers
- Round off error increases due to subtractive cancellation
  - Or due to an increase in the number of computations in an analysis.
- A decrease in stepsize can lead to subtractive cancellation, or to an increase in computations, $\Rightarrow$ the round off errors are increased.



26

Other Errors

- Blunders - errors caused by malfunctions of the computer or human imperfection.
- Model errors - errors resulting from incomplete mathematical models.
- Data uncertainty - errors resulting from the inaccuracy and/or imprecision of the measurement of the data.

27