Indian Institute of Technology Roorkee

MAB-103: Numerical Methods

Unit-1 Error Analysis Session-2025-26

Real numbers, like $\pi(3.1415\ldots)$, can have endless decimal places. However, computers can only store a limited number of digits. This means that at some point, depending on the computer's memory, the remaining digits must be ignored. As a result, the real number stored on the computer is an approximation of the true value. While the discarded digits may be tiny, this approximation can sometimes lead to surprisingly large errors in calculations.

This chapter dives into the world of errors caused by using approximate numbers in calculations. Here's a breakdown of what we'll cover:

- Representing Numbers on Computers: We'll explore how computers store real numbers, which can have infinite digits, with a limited number of digits.
- Approximation Techniques for Memory: We'll discuss two methods for approximating numbers to fit within a computer's memory constraints.
- Understanding Errors and Significant Digits: We'll introduce the concept of errors and how significant digits help us understand the precision of our approximations.
- Error Propagation and Numerical Stability: This section examines how errors from approximations can grow throughout calculations. We'll also explore the concept of condition numbers and how they relate to the stability of calculating functions.

1 Floating points form:

Computers use a special format called floating-point representation to store real numbers. This section will explain how it works.

Definition 1. *Let x be a non-zero real number. An n-digit floating-point number in base β is expressed as follows:*

$$float(x) = (-1)^s \times (.d_1 d_2 \cdots d_n)_\beta \times \beta^e \tag{1}$$

where

$$(.d_1 d_2 \cdots d_n)_\beta = \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_n}{\beta^n} \tag{2}$$

is a β-fraction known as the mantissa or significand, $s = 1$ or 0 denotes the sign, and e is an integer referred to as the exponent. The number β is also termed the radix, and the point preceding d_1 in (1) is called the radix point.

- When $\beta = 2$, the floating-point representation in (1) is known as the binary floating-point representation. When $\beta = 10$, it is referred to as the decimal floating-point representation.
- Note that the floating-point representation (1) contains only a finite number of digits, whereas a real number can have an infinite sequence of digits, such as $1/3 = 0.3333\cdots$ Consequently, the representation (1) is merely an approximation of a real number.

Definition 2. *A floating-point number is considered normalized if either* $d_1 \neq 0$ *or* $d_1 = d_2 = d_3 = \cdots = d_n = 0$.

Next, we will discuss the following two examples based on decimal floating-point representation.

- The real number $x = 5.167$ can be expressed as

$$5.167 = (-1)^0 \times (.5167) \times 10^{-1},$$

  in which case, we have

$$s = 0,\ \beta = 10,\ e = 1,\ d_1 = 5,\ d_2 = 1,\ d_3 = 6 \,\text{and}\, d_4 = 7.$$

  It is important to note that this representation constitutes the normalized floating-point format.

- The real number $x = -0.00023$ can be expressed as

$$-0.00023 = (-1)^1 \times (.000023) \times 10^1,$$

  which is not in the normalized form. The normalazed representation is as follows:

$$-0.00023 = (-1)^1 \times (.23) \times 10^{-3},$$

  in which case, we have

$$s = 1,\ \beta = 10,\ e = -3,\ d_1 = 2, \,\text{and}\, d_2 = 3.$$

Definition 3. *(Overflow and underflow) The exponent* e *is constrained to a specific range*

$$m < e < M. \tag{3}$$

During the calculation, if a computed number has an exponent $e > M$*, we say there is a memory overflow. Conversely, if* $e < m$*, we say there is a memory underflow.*

- In the event of overflow, the computer typically generates meaningless results or displays the symbol NaN, indicating that the value obtained from the calculation is 'not a number'. On some systems, the symbol ∞ is also represented as NaN. Underflow, on the other hand, is less critical because the computer will usually treat the number as zero.

- The floating-point representation (1) of a number imposes two limitations: the number of digits n in the mantissa and the range of e. The number n is referred to as the precision or length of the floating-point representation.

- The IEEE (Institute of Electrical and Electronics Engineers) standard for floating-point arithmetic, known as IEEE 754, stands as the predominant framework for floating-point computation. It is adopted by numerous hardware components such as CPUs and FPUs, including Intel processors, and is also supported by a wide array of software implementations. Many programming languages either allow or mandate that arithmetic operations utilize IEEE 754 formats and operations. The IEEE 754 floating-point representation for a binary number x is defined as follows:

$$float(x) = (-1)^s \times (1.a_1a_2\cdots a_n)_2 \times 2^e, \tag{4}$$

  where $a_1, \cdots, a_n$ are binary digits (either 1 or 0). The IEEE 754 standard exclusively employs binary operations.

- The IEEE single precision floating-point format utilizes 4 bytes (32 bits) for storing a number. Within these 32 bits, 24 bits are allocated to store the mantissa (where each binary digit requires 1 bit of storage), 1 bit is designated for the sign (s), and the remaining 8 bits are reserved for the exponent. the standard format is as follows:

$$|(sign)\, b_1 \,|\, (exponent)\, b_2 \cdots b_9 \,|\, (mantissa)\, b_{10} b_{11} \cdots b_{32}|$$

  Note that only 23 bits are used for the mantissa. This omission occurs because the digit 1 before the binary point in (4) isn't stored in memory and is inserted during calculations instead. The exponent e is constrained within a specified range $-126 \leq e \leq 127$.

- The IEEE double precision floating-point representation of a number offers a precision equivalent to 53 binary digits, and the exponent e is constrained within a specified range $-1023 \leq e \leq 1023$.

1.1 Chopping and Rounding a Number

Any real number x can be precisely represented as follows:

$$x = (-1)^s \times (.d_1 d_2 \cdots d_n d_{n+1} \cdots)_\beta \times \beta^e.$$

where $d_1 \neq 0$ or $d_2 = d_3 = \cdots = 0$, $s = 0$ or 1, and e satisfies (3).The floating-point form (1) provides an approximate representation of x. We denote this approximation as $float(x)$. There are two methods to derive $float(x)$ from x, as defined below.

- The chopped machine approximation of x is defined as follows:

$$float(x) = (-1)^s \times (.d_1 d_2 \cdots d_n)_\beta \times \beta^e \tag{5}$$

- The rounded machine approximation of x is defined as follows:
  - If $0 \leq d_{n+1} < \beta/2$, then

$$float(x) = (-1)^s \times (.d_1 d_2 \cdots d_n)_\beta \times \beta^e$$

  - If $\beta/2 < d_{n+1} < \beta$, then

$$float(x) = (-1)^s \times (.d_1 d_2 \cdots (d_n + 1))_\beta \times \beta^e$$

  - If $d_{n+1} = \beta/2$ with one of the $d_{n+1+i} \neq 0$, where $i = 1, 2, \cdots$, then

$$float(x) = (-1)^s \times (.d_1 d_2 \cdots (d_n + 1))_\beta \times \beta^e$$

  - If $d_{n+1} = \beta/2$ with all $d_{n+1+i} = 0$, where $i = 1, 2, \cdots$, then

$$float(x) = \begin{cases} (-1)^s \times (.d_1 d_2 \cdots (d_n + 1))_\beta \times \beta^e & if\ d_n \text{ is an odd number,} \\ (-1)^s \times (.d_1 d_2 \cdots d_n)_\beta \times \beta^e & if\ d_n \text{ is an even number.} \end{cases}$$

1.2 Errors

The approximate representation of a real number inevitably differs from the actual value, and this difference is referred to as an error. The error in a computed quantity is defined as follows:

$$\text{Error} = \text{True value} - \text{Approximate value}$$

- The absolute error is the absolute value of the aforementioned error.
- The relative error quantifies the error in proportion to the magnitude of the true value, as expressed by

$$\text{Relative Error} = \frac{\text{Error}}{\text{True value}}$$

- Percentage error is the relative error multiplied by 100.
- Truncation error occurs when a Taylor series is approximated by a finite number of terms.

Example 1. *Find the truncation error of a second degree polynomial approximation to*

$$f(x) = 1/\sqrt{1+x}, \quad x \in [0,1]$$

using the Taylor series expansion about $x = 0$.

1.3 Significant Digits

Significant digits are often used instead of relative error.

Definition 4. *We say* x_A *approximates* x *to* r *significant* β*-digits if*

$$|x - x_A| \leq \frac{1}{2}\beta^{s-r+1}$$

with s *the largest integer such that* $\beta^s \leq |x|$.

- For $x = 1/3$, the approximate number $x_A = 0.333$ has three significant digits, given that

$$|x - x_A| \approx 0.00033 < 0.0005 = 0.5 \times 10^{-3}$$

  with $10^{-1} < 0.333\cdots = x$. Thus, we have $s = -1$ and $r = 3$.

- For $x = 0.02138$, the approximate number $x_A = 0.02144$ has two significant digits, since

$$|x - x_A| \approx 0.0006 < 0.0005 = 0.5 \times 10^{-3}$$

  with $10^{-2} < 0.02138\cdots = x$. Thus, we have $s = -2$ and $r = 2$.

- Simply put, the number of leading non-zero digits in x_A that match the corresponding digits in the true value x is referred to as the number of significant digits in x_A.

- Consider two real numbers

$$x = 7.6545428, \quad y = 7.6544201.$$

The numbers

$$x_A = 7.6545421, \quad y_A = 7.6544200$$

are approximation to x and y, accurate to six and seven significant digits, respectively. In eight-digit floating-point arithmetic,

$$z_A = x_A - y_A = 0.12210000 \times 10^{-3}$$

is the exact difference between x_A and y_A and

$$z = x - y = 0.12270000 \times 10^{-3}$$

is the exact difference between x and y. Therefore,

$$z - z_A = 0.6 \times 10^{-6} < 0.5 \times 10^{-5}$$

and hence z_A has only two significant digits with respect to z as $10^{-4} < z = 0.0001227$. Thus, we began with two approximate numbers x_A and y_A which are accurate to six and seven significant digits with respect to x and y respectively. However, their difference, z_A, has only two significant digits relative to z. This indicates a loss of significant digits during the subtraction process.

1.4 Propagation of Error

Once an error is introduced, it influences subsequent results as it propagates through subsequent calculations. Initially, we will examine how results are affected when using approximate numbers instead of exact ones, and then we will address function evaluation.

Let x_A and y_A represent the approximate numbers used in the calculation, while x and y denote their corresponding true values. Next, we will explore how errors propagate through the four fundamental arithmetic operations.

1.4.1 Propagated error in addition and subtraction

Let $x = x_A + \epsilon$ and $y = y_A + \eta$ be positive numbers. The relative error $E_r(x_A \pm y_A)$ is given by

$$E_r(x_A \pm y_A) = \frac{(x \pm y) - (x_A \pm y_A)}{x \pm y} = \frac{\epsilon \pm \eta}{x \pm y}$$

This demonstrates that relative errors propagate gradually with addition, but amplify significantly with subtraction, especially when $x_T \approx y_T$, as illustrated in the previous examples.

1.4.2 Propagated error in multiplication

The relative error $E_r(x_A \times y_A)$ is given by

$$E_r(x_A \times y_A) = \frac{(x \times y) - (x_A \times y_A)}{x \times y} = E_r(x_A) + E_r(y_A) - E_r(x_A)E_r(y_A).$$

This indicates that relative errors propagate gradually with multiplication.

1.4.3 Propagated error in division

The relative error $E_r(x_A/y_A)$ is given by

$$E_r(x_A/y_A) = \frac{(x/y) - (x_A/y_A)}{x/y} = \frac{1}{1 - E_r(y_A)}(E_r(x_A) - E_r(y_A))$$

This indicates that relative errors propagate gradually with division, unless $E_r(y_A) \approx 1$. However, this scenario is highly improbable as we typically expect the error to be very small, close to zero, in which case the right-hand side is approximately equal to $E_r(x_A) - E_r(y_A)$.

1.4.4 Total calculation error

When performing floating-point arithmetic on a computer, computing $x_A \omega y_A$ (where ω represents one of the basic arithmetic operations '$+$', '$-$', '$\times$' and '$/$') introduces an additional rounding or truncation error. The computed value of $x_A \omega y_A$ will encompass the propagated error along with a rounding or truncation error. To be precise, let $\hat{\omega}$ denote the complete operation as executed on the computer, encompassing any rounding or truncation. Therefore, the total error is expressed as:

$$x\omega y - x_A \hat{\omega} y_A = (x\omega y - x_A \omega y_A) + (x_A \omega y_A - x_A \hat{\omega} y_A)$$

The first term on the right represents the propagated error, while the second term denotes the error arising from rounding or truncating the computed result $x_A \omega y_A$.

1.5 Propagated error in function evaluation

Consider evaluating $f(x)$ at the approximate value instead of at x. Next, assess how accurately $f(x_A)$ approximates $f(x)$? By applying the mean value theorem, we find that

$$f(x) - f(x_A) = f'(\xi)(x - x_A),$$

where ξ is an unknown point between x and x_A. The relative error of $f(x)$ with respect to $f(x_A)$ is expressed as follows:

$$E_r(f(x_A)) = \frac{f'(\xi)}{f(x)} x E_r(x_A).$$

Given that x_A and x are assumed to be very close to each other and ξ lies between x and x_A, we can approximate as follows:

$$f(x) - f(x_A) \approx f'(x_A)(x - x_A) \approx f'(x)(x - x_A).$$

Definition 5. *The condition number of a function f at the point $x = c$ is defined as*

$$\left| \frac{f'(c)}{f(c)} c \right|$$

Definition 6. *Consider a function $f(x)$ that requires n steps for evaluation. The overall evaluation process is deemed unstable if at least one step is ill-conditioned. Conversely, if all steps are well-conditioned, the process is considered stable.*

Error Propagation in Multivariable Functions

Problem Statement

Let $f = f(x_1, x_2, \ldots, x_n)$ be a function of several variables. Each variable x_i has an associated small uncertainty or error Δx_i. We aim to estimate the resulting uncertainty Δf in the function f.

First-Order (Linear) Approximation

Using the first-order Taylor expansion, the propagated error in f is approximated by:

$$\Delta f \approx \left|\frac{\partial f}{\partial x_1}\right| \Delta x_1 + \left|\frac{\partial f}{\partial x_2}\right| \Delta x_2 + \cdots + \left|\frac{\partial f}{\partial x_n}\right| \Delta x_n$$

This gives an upper bound on the total error assuming worst-case additive behavior.

Example: First-Order Error Propagation

Let:

$$f(x, y) = x \cdot y$$

Given:

$$x = 5.0 \pm 0.1, \quad y = 3.0 \pm 0.2$$

Step 1: Compute partial derivatives

$$\frac{\partial f}{\partial x} = y, \quad \frac{\partial f}{\partial y} = x$$

Step 2: Evaluate at the given values

$$\frac{\partial f}{\partial x} = 3.0, \quad \frac{\partial f}{\partial y} = 5.0$$

Step 3: Apply the first-order error propagation formula

$$\Delta f \approx \left|\frac{\partial f}{\partial x}\right| \Delta x + \left|\frac{\partial f}{\partial y}\right| \Delta y$$

$$\Delta f \approx (3.0)(0.1) + (5.0)(0.2) = 0.3 + 1.0 = 1.3$$

Final Answer

$$f = xy = (5.0)(3.0) = 15.0$$

$$\boxed{f = 15.0 \pm 1.3}$$