Chapter One

Error Analysis

1.1 Introduction and Some Mathematical Preliminaries

1.1.1 What is Numerical Analysis?

Numerical Analysis is a branch of applied mathematics that focuses on **developing**, **analyzing**, and **implementing algorithms** to obtain approximate numerical solutions to mathematical problems that cannot be solved exactly.

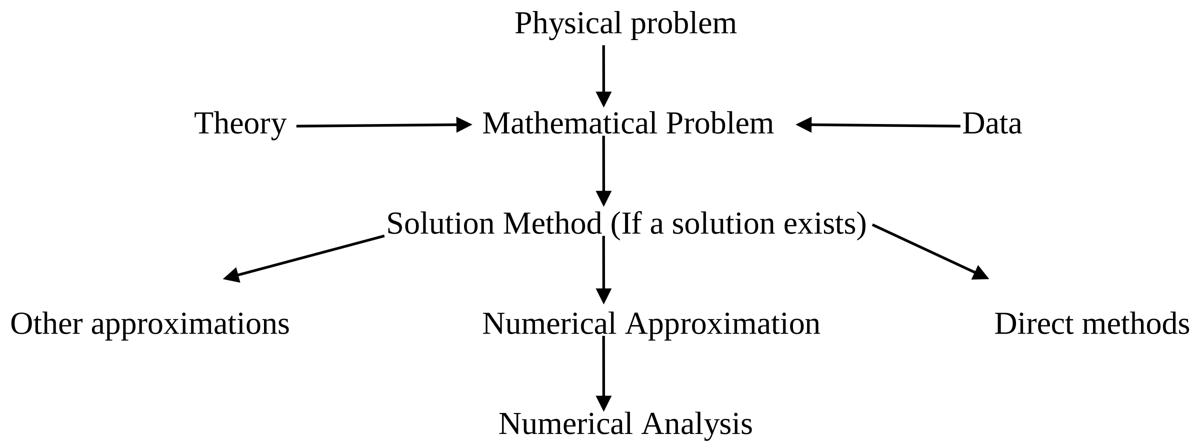
In essence, it has three fundamental components:

1. **Development of Approximation Procedures:**
   Numerical methods are designed to approximate the exact mathematical solutions using finite, discrete computations.
2. **Algorithm Construction:**
   Efficient algorithms (a defined sequence of rules or steps) are formulated to implement these numerical methods on computers.
3. **Error Analysis:**
   Every numerical result differs from the exact value to some degree. Numerical analysis provides systematic techniques for estimating, controlling, and minimizing these errors.

Remark: Numerical Analysis is incomplete without all three components - method formulation, algorithm implementation, and error investigation.

1.1.2 Importance and Applications of Numerical Analysis

Numerical analysis plays a critical role across scientific, engineering, and computational fields.



Its scope extends far beyond pure mathematics.

A. Physical and Engineering Problems

Most physical phenomena such as heat conduction, fluid motion, or electromagnetic fields are modeled by differential equations that rarely admit closed-form solutions. Numerical methods make it possible to obtain approximate but usable solutions.

B. Data Science and Machine Learning

- **Neural Networks:** Training involves minimizing large-scale nonlinear cost functions through iterative optimization algorithms (e.g., gradient descent).
- **Principal Component Analysis (PCA):** Requires solving eigenvalue problems — a core area of numerical linear algebra.
- **Recommendation Systems:** Employ numerical matrix factorization techniques for large datasets.

C. Computer Graphics and Animation

Rendering realistic 3D scenes involves solving the *rendering equation*, a numerical integration problem that models light transport. Animation also depends on numerical integration of equations of motion.

D. Computational Biology and Medicine

- **Drug Discovery:** Simulations of molecular interactions rely on numerical solutions of molecular dynamics equations.
- **Medical Imaging:** Image reconstruction from MRI or CT scan data is performed using numerical inverse problem techniques.

E. Optimization and Economics

Optimization techniques are essential in:

- Logistics and transportation planning
- Financial portfolio optimization
- Scheduling and resource allocation
- Machine learning parameter tuning

Summary: Numerical Analysis is the mathematical engine that converts abstract models into computable, practical, and predictive results.

1.1.3 When Do We Need Numerical Analysis?

Numerical methods are essential when:

1. **Exact analytical solutions are unavailable.**
   Many nonlinear, high-dimensional, or coupled systems cannot be solved symbolically.
2. **Exact solutions exist but are impractical to compute.**
   For example, integrating complex functions or solving large systems of equations.
3. **Input data are uncertain or incomplete.**
   Measurement errors or incomplete information make purely analytical solutions inadequate.
4. **Experimental approaches are costly or dangerous.**
   Numerical simulation offers a safer and cheaper "virtual laboratory."

Example: Simulating airflow over an airplane wing numerically is far cheaper than building multiple wind-tunnel models.

1.1.4 Who Uses Numerical Analysis?

Numerical analysis is a universal tool for:

- **Scientists and Researchers:** Mathematicians, physicists, chemists, and biologists.
- **Engineers:** Civil, mechanical, electrical, and aerospace engineers use numerical simulations in design and optimization.
- **Computer Scientists:** In fields like machine learning, computer vision, game development, and AI.
- **Economists and Analysts:** For forecasting, optimization, and quantitative finance.

1.1.5 Why We Study Numerical Analysis?

Numerical analysis bridges theory and computation.

- It transforms **abstract equations** into **computable algorithms**.
- It allows simulation and prediction where experimentation is expensive or impossible.

Even though modern software (like MATLAB, Python's NumPy, or Mathematica) can perform numerical tasks automatically, understanding the **mathematical foundation** is critical for:

1. **Innovation:** Developing new numerical methods for new problems.
2. **Selection:** Choosing the most accurate and efficient method for a given problem.
3. **Troubleshooting:** Diagnosing instability, divergence, or accuracy issues in computational results.

In short: Learning numerical analysis transforms you from a user of computational tools into a creator and evaluator of them.

Therefore, for a computer scientist, this course is not just a math requirement; it's a **practical toolkit for solving problems that are fundamental to modern computing.** It empowers you to move from being a programmer who *uses* libraries to an engineer who can ***design, implement, and debug*** complex computational solutions for fields like AI, graphics, and simulation. It teaches you how to make the computer solve problems that nature presents in a continuous, analog way.

QN. How to do Numerical Analysis?

This is what we are going to learn in this course.

1.1.6 Some Mathematical preliminaries

Some of mathematical results which would be useful to derive numerical methods and analyze errors associated with these numerical methods.

Definition (Sequence of Real numbers)

A sequence of real numbers is an ordered list of real numbers

$$a_1,\ \ a_2,\ \ a_3,\ \ a_4,\ \ .\ \ .\ \ \ .\,a_n,\ \ \ a_{n+1}\ .\ .\ .$$

The concept of convergence of a sequence plays an important role in Numerical Analysis. For instance, an iterative procedure is often constructed to approximate a solution x of an equation. This procedure produces a sequence of approximation which is expected to convergence to the exact solution x.

Definition (Convergence of a sequence)

Let $\{a_n\}$ be a sequence of real numbers and let a be a real number. The sequence $\{a_n\}$ is said to converge to a, written as $\lim_{n\to\infty} a_n = a$ if for every $\in > 0$, there exists a natural number N such that $|a_n - a| < \in$ whenever $n \geq N$.

Definition (Limit of a function)

Let f be a function defined on some open interval containing *a* (except possibly at *a* itself). We say that the limit of *f*(*x*) as *x* approaches a is L, and write

$$\lim_{x\to a} f(x) = L$$

if for every number $\epsilon > 0$ (no matter how small), there exists a number $\delta > 0$ such that

if $0 < | x - a | < \delta$ then $| f(x) - L | < \epsilon$.

Definition (Differentiation)

The derivative of a function f at a point a, $f'(a)$, is defined as $f'(a) = \lim_{h\to 0}\frac{f(a+h)-f(a)}{h}$ if this limit exists and we say f is differentiable at a.

Theorem 1.1 (Intermediate Value Theorem)

If f is continuous on $[a, b]$ and k is a number between $f(a)$ and $f(b)$, then there exists a number c in (a, b) such that $f(c) = k$.

Theorem 1.2 (Location Theorem)

If $f(x)$ is continuous on $[a, b]$, $f(a)$ and $f(b)$ are of opposite signs, then there exists at least one number x_0 in (a, b) such that $f(x_0) = 0$.

Theorem 1.3 (Mean Value Theorem)

If f is continuous on $[a, b]$ and differentiable on (a, b), then there exists c in (a, b) with $f'(c) = \frac{f(b)-f(a)}{b-a}$.

Theorem 1.4 (Taylor's Theorem)

Suppose $f \in C^n[a,b]$ and $f^{(n+1)}$ exists on (a,b), Then, there exists a number ξ, depending on x, between x and x_0 such that

$$f(x) = P_n(x) + R_n(x) \tag{1.1}$$

Where $P_n(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \ldots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n$

and $R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)^{n+1}$ such that ξ is between x_0 and x.

Here, $P_n(x)$ is called the **nth Taylor polynomial** for f about x_0 and $R_n(x)$ is called the **remainder term, error term, truncation error** associated with $P_n(x)$.

The infinite series obtained by taking $P_n(x)$ as $n \to \infty$ is called the **Taylor series** for f about x_0.

That means the Taylor series for f about x_0 is given by:

$$P_\infty(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \ldots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n + \ . \ . \ .$$

Note:

- In the case $x_0 = 0$, the Taylor polynomial is called **Maclaurin polynomial** and the Taylor series is called **Maclaurin series**. If $\lim_{n\to\infty} R_n(x) = 0$, then the Taylor series converges to $f(x)$ and hence we have

$$f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \ldots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n + \ldots \qquad (1.2)$$

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

  With an upper bound $|R_n(x)| \le \max_{\xi\in[a,x]} \frac{|f^{(n+1)}(\xi)|}{(n+1)!}|x-x_0|^{n+1}$,
- $f(x) = P_n(x) + O(h^{n+1}),\ \ h = x - x_0$
- Error = $O(h^{n+1})$, as h gets smaller, the error gets smaller in proportion to h^{n+1}

Theorem 1.5 (Taylor's Theorem in two variables)

Suppose that $f(x, y)$ and all of its derivatives of order less than or equal to n + 1 are continuous on $D = \{(x, y) : a \le x \le b \text{ and } c \le y \le d\}$. Then, f can be expressed near $(x, y) = (x_0, y_0)$ in D as $f(x,y) = P_n(x,y) + R_n(x,y)$ (1.3)

where

$$P_n(x,y) = f(x_0, y_0) + \left[\frac{\partial f}{\partial x}(x_0,y_0)(x-x_0) + \frac{\partial f}{\partial y}(x_0,y_0)(y-y_0)\right]$$

+

$$\left[\frac{\partial^2 f}{\partial x^2}(x_0,y_0)\frac{(x-x_0)^2}{2!} + \frac{\partial^2 f}{\partial x\partial y}(x_0,y_0)(x-x_0)(y-y_0 + \frac{\partial^2 f}{\partial y^2}(x_0,y_0)\frac{(y-y_0)^2}{2!}\right] + \ldots$$

$$+ \left[\frac{1}{n!}\sum_{j=0}^{n}\binom{n}{j}\frac{\partial^n f}{\partial x^{n-j}\partial y^j}(x_0,y_0)(x-x_0)^{n-j}(y-y_0)^j\right] \text{ and}$$

$R_n(x,y) = \dfrac{1}{(n+1)!}\displaystyle\sum_{j=0}^{n+1}\binom{n+1}{j}\frac{\partial^{n+1} f}{\partial x^{n+1-j}\partial y^j}(\xi,\eta)(x-x_0)^{n+1-j}(y-y_0)^j$ such that ξ is between x_0 and x and η is between y_0 and y.

The function $P_n(x, y)$ is called the **n^th Taylor polynomial** in two variables for the function f about (x_0, y_0), $R_n(x, y)$ is the **remainder term** associated with $P_n(x, y)$

Example:

Approximate the function $\boldsymbol{f(x) = e^x}$ by a Taylor polynomial of degree $\boldsymbol{0, 1, 2, 3\ and\ 4}$ about the point $\boldsymbol{x_0 = 0}$ and determine the maximum error that comes due to approximating the given function by 4th Taylor polynomial.

Solution: For $n = 0,1,2,3,4$

$P_0(x) = f(0) = 1$

$P_1(x) = f(0) + x.f'(0) = 1 + x$

$P_2(x) = f(0) + x.f'(0) + \frac{f''(0)}{2!}x^2 = 1 + x + \frac{1}{2}x^2$

$P_3(x) = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3$

$P_4(x) = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4$

and the remainder term of the Taylor formula with $n = 4$ for $f(x) = e^x$ about $x_0 = 0$ is

$$|R_n(x)| = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1} = \frac{e^\xi}{5!}x^5$$

Where ξ lies between x and 0.

From Taylor Theorem

$$f(x) = e^x = P_4(x) + R_4(x)$$

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

Note that for $x = 1$, $e \cong 2.7183$ and $P_4(x = 1) \cong 2.7083$

Therefore, $e - P_4(x = 1) \cong 0.01$, where as $R_4(x) = \left|\frac{e^\xi}{5!}x^5\right| \le \frac{e^\xi}{5!}(1) = \frac{e}{5!} \cong 0.0227$(The max error that can come due this approximation), where $0 < \xi < x = 1$

$0.01 \cong |e - P_4(x = 1)| \le 0.0227,$

Example 2:

Consider the function $f(x) = lnx$ in the interval $[1, 2]$. Find the n^{th} Taylor polynomial for $f(x)$ at $x_0 = 1$ and determine n to estimate $ln2$ with an error less than 10^{-8} .

Solution

$f(x) = lnx,\ f'(x) = \frac{1}{x},\ f''(x) = -\frac{1}{x^2} \qquad f'''(x) = \frac{2}{x^3}$,………………

In general $f^k(x) = (-1)^{k-1}(k-1)!\,x^{-k}$, which implies that $f^k(x_0) = (-1)^{k-1}(k-1)!\,{x_0}^{-k}$

Hence, using Taylor's theorem the nth Taylor polynomial is

$f(x) = lnx = \sum_{k=1}^{n} \frac{(-1)^{k-1}}{k}(x-1)^k + \frac{(-1)^n}{n+1}\xi^{-(n+1)}(x-1)^{(n+1)}$, where $1 \leq x \leq 2$ & $1 < \xi < x$

The error term is

$|R_n(x)| = \frac{1}{(n+1)}\xi^{-(n+1)}(x-1)^{(n+1)} \leq \frac{1}{n+1}(x-1)^{n+1}$ since $\xi^{-(n+1)} < 1\ for\ \xi < 1$

To estimate $ln2$ with an error less than 10^{-8}, it would require

$$|R_n(x)| \leq \frac{1}{n+1} \leq 10^{-8}$$

Therefore $n \geq 10^8 - 1$

To estimate ln 1.5 with the same precision, it requires

$$|R_n(x)| \leq \frac{1}{n+1}\left(\frac{1}{2}\right)^{n+1} \leq 10^{-8}$$

It is sufficient to attain the accuracy if $n \geq 26$

Order of Convergence and Big O / Small o Notation

Order of Convergence

The **Order of Convergence** is a qualitative measure that describes how fast an iterative sequence converges to its limit. It tells you the relationship between the errors in successive iterations.

Definition:

Let $\{x_n\}$ be a sequence that converges to a limit L, and define the error at the $n-th$ step as

$$e_n = |\ x_n - L\ |$$

If there exists a positive constant C (with $C < 1$ for order 1) and a real number $P \geq 1$ such that

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^P} = C$$

then the sequence is said to converge to L with **order P**. The constant C is called the **asymptotic error constant**.

In numerical analysis, the order of convergence indicates how quickly a sequence of approximate values approaches the exact (true) value. If successive approximations x_k satisfy

$$|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*| \leq \boldsymbol{C}|\boldsymbol{x}_k - \boldsymbol{x}^*|^p, \text{ where } C > 0 \text{ and } p > 0,$$

then P is called the **order of convergence**.

Interpretation

• p = 1 → Linear convergence (moderate speed) -the error shrinks by a fixed percentage in each iteration (error decreases proportionally with each iteration)

• p = 2 → Quadratic convergence (fast)- error squared decreases with each iteration(the error decreases proportionally to the square of the previous error)

• p > 2 → Superlinear convergence (very fast)

• 0 < p < 1 → Sublinear convergence (slow)

The higher the order of convergence, the faster the error decreases with each iteration.

For example, the Bisection Method halves the error at each step (linear), while the Newton–Raphson Method squares the error at each step (quadratic). The Taylor polynomial approximation error is proportional to $|x - x_0|^{n+1}$, showing an order of $n + 1$.

Big O and Small o Notation

In numerical analysis, Big O and Small o notation describe how a function behaves as its argument approaches a limit (usually 0 or ∞). They help us express how fast errors or differences between functions go to zero that is, they measure **rates of convergence** and **error terms** in approximations.

Big O Notation — Order of Magnitude

We write

$$f(x) = O(g(x)) \text{ as } x \to a \text{ if there exists a constant } C > 0 \text{ and and } \delta \text{ such that}$$

$$|f(x)| \leq C|\, g(x)\, | \text{ for all x near } a \;(\, |x - a| < \delta\,).$$

This means $f(x)$ grows no faster than $g(x)$; it gives an upper bound on growth on the size or rate of growth of $f(x)$

Example:

If $f(x) = 3x^2 + 5x + 2$, then as $x \to \infty$, $f(x) = O(x^2)$ because x^2dominates the growth rate of $f(x)$.

In numerical analysis, we may say that the **error term** in an approximation is $\boldsymbol{O}(h^2)$, meaning the error is proportional to h^2 for small step size h.

Small o Notation — Grows Slower Than

We write

$f(x) = o(\mathrm{g}(x))$ as $x \to a$ if for every $\varepsilon > 0$, there exists $\delta > 0$ such that

$$|f(x)| \leq \varepsilon\, |\mathrm{g}(x)| \quad \text{for all x near a.}$$

$f(x)$ Becomes ***negligibly small*** compared to $\mathrm{g}(x)$ as $x \to a$. That is $\lim_{x \to a} \frac{f(x)}{\mathrm{g}(x)} = 0$

Example:

$x^2 = o(x)$ as $x \to 0$ because $\frac{x^2}{x} = x \to 0$. In numerical analysis, if a method has an error term $o(h)$, it means the error decreases ***faster*** than h as $h \to 0$.

Use in Error Analysis

- **Expressing Truncation error**

  Big O notation is used to indicate the **order of truncation error** in approximation formulas.

  Example: The error in the trapezoidal rule is $\boldsymbol{O}(\boldsymbol{h}^2)$. This means the truncation error is proportional to $\boldsymbol{h}^2$. As the step size $\boldsymbol{h}$ becomes smaller, the error decreases roughly like $\boldsymbol{h}^2$.

- **Describing rates of convergence of iterative methods**.

  Big O notation also measures **how fast iterative methods** approach the true solution. **Example:** If an iterative sequence $\boldsymbol{x}^{\boldsymbol{k}+1} - \boldsymbol{x}^* = \boldsymbol{O}((\boldsymbol{x}^{\boldsymbol{k}} - \boldsymbol{x}^*)^2)$, then the method has **quadratic convergence**, meaning the number of correct digits approximately doubles with each iteration.

- **Interpreting Error Expressions**

  If $Error = O(h^p)$, it means the error decreases proportionally to h^p as step size h decreases

  If $Error = o(h^p)$, it means the error decreases faster than h^p; that is, $\lim_{x \to a} \frac{Error}{h^P} = 0$

1.2 Definition and sources of error

1.2.1 Definition of Terms

a) **Numerical Analysis**

- **Numerical Analysis** is concerned with mathematical derivation of numerical methods, designing an algorithm for the methods, implementation of these algorithms on computers and analysis of the errors associated with the methods all to solve mathematical problems.
- The study of algorithms that use numerical approximation to solve mathematical problems. It is concerned with how to solve problems (like integration, solving equations, etc.) that cannot be solved exactly with simple algebraic methods. A central focus of NA is understanding and controlling the errors that arise from these approximations.
- The study of **how** and **why** numerical algorithms work, with a focus on their stability, efficiency, and error.

b) **Numerical Methods:** The specific techniques or algorithms used in Numerical Analysis to find approximate solutions to mathematical problems (e.g., Newton's Method, Gaussian Elimination, and and Simpson's rule for numerical integration).

c) **Exact Numbers:** Numbers that are known with complete certainty, having no error or approximation.

Example: There are exactly **24 hours in a day**, **12 eggs in a dozen**.

d) **Approximate Numbers:** Numbers that are used as estimates or approximations of an exact value. Numbers that involve uncertainty due to rounding, estimation, or measurement.

Example: $\pi \approx$ **3.14**, the length of a rod measured as **2.37 m**

e) **Accuracy:** A measure of how close a computed or measured value is to the **true or exact value.**

Example: If the true value of π is 3.1416 and you compute 3.142, it is accurate.

f) **Precision:** A measure of how close repeated computations or measurements are to **each other**. It is often related to the number of significant digits or the level of detail in the measurement.

g) **Inaccuracy (also called Bias or Systematic Error)**: The systematic deviation of a computed or measured value from the true value. It is a measure of a lack of accuracy.

Example: If the true temperature is **100°C** but measured as **97°C**, the inaccuracy is 3°C.

h) **Imprecision (also called Uncertainty or Random Error)**: The lack of precision indicated by the scatter or spread in repeated measurements or computations. It is caused by unpredictable variations in the measurement process and is often statistical in nature.

Example: Measuring the same rod three times and recording **2.35 m, 2.40 m, 2.45 m**.

i) **Error:** In any scientific or computational process, the *error* is the difference between the **true value** (exact or theoretical) and the **approximate value** (measured or computed).

Let $x =$ true value, $\bar{x} =$ approximate or computed value then the **error** E is defined as:

$$\boldsymbol{Error = x - \bar{x}} = \textbf{\textit{true value – approximate value}}$$

Errors are inevitable because real-world measurements and computer arithmetic have limitations. Understanding **how errors arise** and **propagate** is essential to assess the reliability of numerical results.

1.2.2 Sources of Error

The following are the main sources of error in obtaining numerical solutions to mathematical problems.

a. Modeling Error-A mathematical modeling for physical situations is an attempt to give mathematical relationships between certain quantities or parameters of the physical situations. Because of the complexity of physical reality, a variety of simplifying assumptions are used to construct more tractable mathematical models. The resulting model has limitations on its accuracy as consequence of these assumptions; and these limitations may or may not be troublesome, depending on the uses of the model. In the case that the model is not accurate, no numerical method will improve this basic lack of accuracy.

b. Data uncertainty -Input data from a physical problem may contain error or inaccuracy within it associated with measurement or previous computations. This affects the accuracy of any calculation based on the data, limiting the accuracy of answers.

c. The numerical methods used

In most cases numerical methods are approximate by themselves, that is even if the initial data are void of errors all the arithmetic operations are ideally performed, they yield the solution of the original with some error which is called the **error of the method.**

d. Round off error arises because it is impossible to represent all real numbers exactly on a finite-state machine since a word in a computer memory stores only a finite number of digits and hence a computer numbers have limited number of digits. We are forced to cut off some of the digits of a number. For example, we are forced to approximate numbers $\frac{1}{3}$, e, and π with a fixed finite number of digits in a computer even though they have actually infinite digits representations.

e. Truncation error is caused by the fact that when an exact mathematical procedure is replaced by another to make it easier (or possible) to solve a problem. This includes truncation of an infinite series to a finite number of terms, termination of an iterative method and making a discrete approximation to something continuous and others.

Example: What problems can be created by round off errors? Twenty-eight Americans were killed on February 25, 1991. An Iraqi Scud hit the Army barracks in Dhahran, Saudi Arabia. The patriot defense system had failed to track and intercept the Scud. What was the cause for this failure?

The Patriot defense system consists of an electronic detection device called the range gate. It calculates the area in the air space where it should look for a Scud. To find out where it should aim next, it calculates the velocity of the Scud and the last time the radar detected the Scud. Time is saved in a register that has 24 bits length. Since the internal clock of the system is measured for every one-tenth of a second, 1/10 is expressed in a 24 bit-register as 0.00011001100110011001100. However, this is not an exact representation. In fact, it would need infinite numbers of bits to represent 1/10 exactly. So, the error in the representation in decimal format is

Figure 1 Patriot missile (Courtesy of the US Armed Forces, http://www.redstone.army.mil/history/archives/patriot/patriot.html)

$$\frac{1}{10}-(0\times2^{-1}+0\times2^{-2}+0\times2^{-3}+1\times2^{-4}+\ldots+1\times2^{-22}+0\times2^{-23}+0\times2^{-24})$$
$$=9.537\times10^{-8}$$

The battery was on for 100 consecutive hours, hence causing an inaccuracy of

$$= 9.537\times10^{-8}\,\frac{\mathrm{s}}{0.1\mathrm{s}}\times 100\,\mathrm{hr}\times\frac{3600\mathrm{s}}{1\mathrm{hr}}$$
$$= 0.3433\mathrm{s}$$

The shift calculated in the range gate due to $0.3433\mathrm{s}$ was calculated as $687\mathrm{m}$. For the Patriot missile defense system, the target is considered out of range if the shift was going to more than $137\mathrm{m}$. This miscalculation was large enough that the system no longer saw the Scud as a target in the area it was defending, and so it failed to launch an interceptor missile.

1.3 Computer representation of Numbers

- Since there is a fixed space of memory in a computer, a given number in a certain base must be represented in a finite space in the memory of the machine. This means that not all real numbers can be represented in the memory.

- The numbers that can be represented in the memory of the computer are called *machine numbers*. Numbers outside these cannot be represented.

- Positive integers including zero, can be represented as **unsigned numbers**. However, to represent negative integers, we need a notation for negative values.

- In computers the convention is to make the sign bit equal to 0 for positive and to 1 for negative and represent the sign with a bit placed in the leftmost position of the number.

- In addition to the sign, a number may have a binary (or decimal) point.

- The position of the binary point is needed to represent fractions, integers, or mixed integer-fraction numbers.

- The presentation of the binary point in a register is complicated by the fact that it is characterized by a position in the computer memory (register).

- There are two ways of specifying the position of the binary point in a register: by giving it fixed position(**Fixed point representation**) or employing a **floating-point representation** as is going to be discussed in the following subsections.

1.3.1 Fixed point representation

The fixed point method of representing numbers assumes that the binary point is always fixed in one position.

Example:

Suppose we have an 8-bit register:

- 1 bit for the sign
- 4 bits for the integer part
- 3 bits for the fractional part

Then

- The largest number that can be represented is **$01111.111_2 = +15.875_{10}$**
- The smallest number is **$11111.111_2 = -15.875_{10}$**

In fixed-point systems:

- If the binary point is on the **left**, the number is a *fraction*.
- If the binary point is on the **right**, the number is an *integer*.

In all cases the binary point is not actually present, but its presence is assumed from the fact that the number stored in the register is treated as **mixed, fraction or as an integer**.

Of course the second alternative is becoming the standard fixed point representation for integers and thus in a 32 bit register 31 bits are available for the integer, because only 1 bit is needed for the sign. Consequently, the range for integers is from $-(2^{31}-1)\ \textit{to}\ (2^{31}-1) = 2147483647.$

Why Fixed-Point Isn't Always Enough

While fixed-point representation works for integers, it struggles with very large or very small numbers.

1. Small numbers lose accuracy.

   Example: with only two decimal places, both 0.014 and 0.006 would round to 0.01, even though one is more than double the other!

2. Large numbers can't be stored precisely.

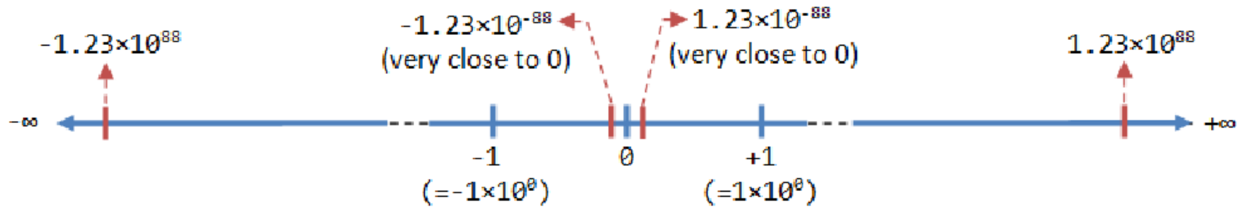
   For instance, the speed of light ≈ 299,792,458 m/s.

   But in fixed-point form, small variations like 299,792,458.63 vs 299,792,457.88 can't be meaningfully distinguished.

We need a more flexible system - and that's where **floating-point representation** comes in.

1.3.2 Floating Point Number Representation

-A floating-point number (or real number) can represent a very large value (e.g., $1.23 \times 10\text{^}88$) or a very small value (e.g., $1.23 \times 10\text{^} - 88$). It could also represent very large negative number (e.g., $-1.23 \times 10\text{^}88$) and very small negative number (e.g., $-1.23 \times 10\text{^} - 88$), as well as zero, as illustrated:



-A floating-point number is typically expressed in the scientific notation, with a *fraction* (***m***), and an *exponent* (ϵ) of a certain *radix* ($r = 2\ for\ binary\ r = 10\ for\ decimal\ numbers$), in the form of

$$y = \sigma \times m \times r^p .$$

Where σ = sign of number (negative or positive – use 0 for positive and 1 for negative),

m = mantissa, $(1)_2 \leq m < (10)_2$, that is, $(1)_{10} \leq m < (2)_{10}$, and

p = integer exponent

-Representation of floating point number is **not unique**.

Example:

The number 55.66 can be represented as $5.566 \times 10^1, 0.5566 \times 10^2, 0.05566 \times 10^3$, and so on.

-The fractional part can be **normalized** (using explicit normalization or implicit normalization) in order to remove non unique representation of numbers.

-In the normalized form (implicit normalization), there is only a single non-zero digit before the radix point.

Example:

Decimal number 123.4567 can be normalized as 1.234567×10^2

Binary number 1010.1011B can be normalized as $1.0101011B \times 2^3$.

Example 1:

Represent $(54.75)_{10}$ in floating point binary format. Assuming that the number is written to a hypothetical word that is 9 bits long where the first bit is used for the sign of the number, the second bit for the sign of the exponent, the next four bits for the mantissa, and the next three bits for the exponent,

Solution: $(54.75)_{10} = (110110.11)_2 = (1.1011011)_2 \times 2^{(5)_{10}}$

The exponent 5 is equivalent in binary format as $(5)_{10} = (101)_2$

Hence $(54.75)_{10} = (1.1011011)_2 \times 2^{(101)_2}$

The sign of the number is positive, so the bit for the sign of the number will have zero in it.

$$\sigma = 0$$

The mantissa $m = 1011$

(There are only 4 places for the mantissa, and the leading 1 is not stored as it is always expected to be there), and the exponent $e = 101$.

we have the representation as

0	0	1	0	1	1	1	0	1

Example 2:

What number does the below given floating point format

0	1	1	0	1	1	1	1	0

represent in base-10 format. Assume a hypothetical 9-bit word, where the first bit is used for the sign of the number, second bit for the sign of the exponent, next four bits for the mantissa and next three for the exponent.

Bit Representation	Part of Floating point number
0	Sign of number
1	Sign of exponent
1011	Magnitude of mantissa
110	Magnitude of exponent

Solution: Given the first bit is 0, so the number is positive.

The second bit is 1, so the exponent is negative.

The next four bits, 1011, are the magnitude of the mantissa, so

$$m = (1.1011)_2 = (1\times 2^0 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3} + 1\times 2^{-4})_{10} = (1.6875)_{10}$$

The last three bits, 110, are the magnitude of the exponent, so

$$e = (110)_2 = (1\times 2^2 + 1\times 2^1 + 0\times 2^0)_{10} = (6)_{10}$$

The number in binary format then is $(1.1011)_2 \times 2^{-(110)_2}$.

The number in base-10 format is $= 1.6875\times 2^{-6} = 0.026367$

Example 3:

A machine stores floating-point numbers in a hypothetical 10-bit binary word. It employs the first bit for the sign of the number, the second one for the sign of the exponent, the next four for the exponent, and the last four for the magnitude of the mantissa.

a) Find how 0.02832 will be represented in the floating-point 10-bit word.

b) What is the decimal equivalent of the 10-bit word representation of part (a)?

Solution

a) For the number, we have the integer part as 0 and the fractional part as 0.02832

Let us first find the binary equivalent of the integer part

Integer part $(0)_{10} = (0)_2$. Now we find the binary equivalent of the fractional part

Fractional part: $0\underline{.02832\times 2} = 0.05664$

$\underline{0.05664\times 2} = 0.11328$

$\underline{0.11328\times 2} = 0.22656$

$\underline{0.22656\times 2} = 0.45312$

$\underline{0.45312\times 2} = 0.90624$

$\underline{0.90624\times 2} = 1.81248$

$\underline{0.81248\times 2} = 1.62496$

$\underline{0.62496\times 2} = 1.24992$

$\underline{0.24992\times 2} = 0.49984$

$\underline{0.49984\times 2} = 0.99968$

$\underline{0.99968\times 2 = 1.99936}$

0.99936

Hence $(0.02832)_{10} \cong (0.0000011100\ 1)_2$

$$= (1.11001)_2 \times 2^{-6}$$

$$\cong (1.1100)_2 \times 2^{-6}$$

The binary equivalent of exponent is found as follows

So $(6)_{10} = (110)_2$

So $(0.02832)_{10} = (1.1100)_2 \times 2^{-(110)_2}$

$$= (1.1100)_2 \times 2^{-(0110)_2}$$

Part of Floating point number	Bit Representation
Sign of number is positive	0
Sign of exponent is negative	1
Magnitude of the exponent	0110
Magnitude of mantissa	1100

The ten-bit representation bit by bit is

0	1	0	1	1	0	1	1	0	0

b) Converting the above floating point representation from part (a) to base 10 by following Example 2 gives $(1.1100)_2 \times 2^{-(0110)_2}$

$$= \left(1\times 2^0 + 1\times 2^{-1} + 1\times 2^{-2} + 0\times 2^{-3} + 0\times 2^{-4}\right) \times 2^{-\left(0\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 2^0\right)}$$

$$= (1.75)_{10} \times 2^{-(6)_{10}} = 0.02734375$$

Q: **How do you determine the accuracy of a floating-point representation of a number?**

A: The machine epsilon, $\in_{mach}$ is a measure of the accuracy of a floating point representation and is found by calculating the difference between 1 and the next number that can be represented.

Example:

Assume a 10-bit hypothetical computer where the first bit is used for the sign of the number, the second bit for the sign of the exponent, the next four bits for the exponent and the next four for the mantissa.

We represent 1 as

0	0	0	0	0	0	0	0	0	0

and the next higher number that can be represented is

0	0	0	0	0	0	0	0	0	1

The difference between the two numbers is

$$(1.0001)_2 \times 2^{(0000)_2} - (1.0000)_2 \times 2^{(0000)_2}$$

$$= (0.0001)_2$$

$$= (1 \times 2^{-4})_{10}$$

$$= (0.0625)_{10}\,.$$

The machine epsilon is $\in_{mach} = 0.0625$.

-The machine epsilon, $\in_{mach}$ is also simply calculated as two to the negative power of the number of bits used for mantissa. As far as determining accuracy, machine epsilon, $\in_{mach}$ is an upper bound of the magnitude of relative error that is created by the approximate representation of a number (See Example 4).

Example 4:

A machine stores floating-point numbers in a hypothetical 10-bit binary word. It employs the first bit for the sign of the number, the second one for the sign of the exponent, the next four for the exponent, and the last four for the magnitude of the mantissa. Confirm that the magnitude of the relative true error that results from approximate representation of 0.02832 in the 10-bit format (as found in previous example) is less than the machine epsilon.

Solution:

From Example 2, the ten-bit representation of 0.02832 bit-by-bit is

0	1	0	1	1	0	1	1	0	0

Again from Example 2, converting the above floating point representation to base-10 gives

$$(1.1100)_2 \times 2^{-(0110)_2}$$

$$= (1.75)_{10} \times 2^{-(6)_{10}}$$

$$= (0.02734375)_{10}$$

The absolute relative true error between the number 0.02832 and its approximate representation 0.02734375 is

$$|\varepsilon_t| = \left| \frac{0.02832 - 0.02734375}{0.02832} \right| = 0.034472 \le \varepsilon_{mach} = 2^{-4} = 0.0625$$

which is less than the machine epsilon for a computer that uses 4 bits for mantissa

IEEE-754 Single and double precision Representations of Real Numbers

-Modern computers adopt IEEE 754(Institute of Electrical and Electronics Engineers) standard for representing floating-point numbers.

-There are two representation schemes: 32-bit single-precision and 64-bit double precision.

-Let us point out the salient features of the **single precision** format.

- A single precision number uses 32 bits.
- A number y is represented as

  $N = (-1)^s \times (1.a_1 a_2 \cdots a_{23}) \cdot 2^p$. Where

  s = sign of the number (positive or negative)

  a_i = entries of the mantissa, can be only 0 or 1, $i = 1,..,23$ (**note:** 1 before the radix point.)

  $P = e - 127$ is the actual exponent e = biased exponent
- The first bit represents the sign of the number (0 for positive number and 1 for a negative number).

- The next eight bits represent the exponent. Note that there is no separate bit for the sign of the exponent. The sign of the exponent is taken care of by normalizing by adding 127 to the actual exponent.

Example:

In the previous example, the exponent was 6. It would be stored as the binary equivalent of $127+6=133$. Why is 127 and not some other number added to the actual exponent? Because in eight bits the largest integer that can be represented is $(11111111)_2=255$, and halfway of 255 is 127. This allows negative and positive exponents to be represented equally. The normalized (also called biased) exponent has the range from 0 to 255, and hence the exponent p has the range of $-127\le p\le 128$.

Bias= $\mathbf{2^{r-1}-1}$**,** where **r** is the number bits assigned for the exponent

- If instead of using the biased exponent, let us suppose we still used eight bits for the exponent but used one bit for the sign of the exponent and seven bits for the exponent magnitude. In seven bits, the largest integer that can be represented is $(1111111)_2=127$ in which case the exponent p range would have been smaller, that is, $-127\le p\le 127$. By biasing the exponent, the unnecessary representation of a negative zero and positive zero exponent (which are the same) is also avoided.
- Actually, the biased exponent range used in the **IEEE-754 format** is not 0 to 255, but 1 to 254. Hence, exponent p has the range of $-126\le p\le 127$. So what are $p=-127$ and $p=128$ used for?
  - ✓ If $p=128$ $(e=255)$ and all the **mantissa entries are zeros**, the number is $\pm\infty$ (the sign of infinity is governed by the sign bit),
  - ✓ if $p=128$ $(e=255)$and the mantissa entries are **not zero**, the number being represented is Not a Number (NaN).
  - ✓ Because of the leading 1 in the floating point representation, the number zero cannot be represented exactly. That is why the number zero (0) is represented $p=-127$ $(e=0)$ and all the mantissa entries are being zero.
- The next twenty-three bits are used for the mantissa.
- The largest number by magnitude that is represented by this format is

$$\left(1\times 2^{0}+1\times 2^{-1}+1\times 2^{-2}+\cdots+1\times 2^{-22}+1\times 2^{-23}\right)\times 2^{127}=3.40\times 10^{38}$$

The smallest number by magnitude that is represented, other than zero, is

$$\left(1\times 2^{0}+0\times 2^{-1}+0\times 2^{-2}+\cdots+0\times 2^{-22}+0\times 2^{-23}\right)\times 2^{-126}=1.18\times 10^{-38}$$

- Since 23 bits are used for the mantissa, the machine epsilon, $\in_{mach}=2^{-23}=1.19\times 10^{-7}$

Summary

In summary, the value (N) is calculated as follows:

- For $1 \le e \le 254, N = (-1)^{S} \times 1.f \times 2^{(e-127)}$. These numbers are in the so called ***normalized* form**. The sign bit represents the sign of the number.
- Fractional part $(1.f)$ are normalized with an implicit leading 1. The exponent is bias (or in excess) of 127, so as to represent both positive and negative exponent. The range of actual exponent is -126 to +127.
- For e = 0, $N == (-1)^{S} \times 0.f \times 2^{-126}$ These numbers are in the so-called ***denormalized* form**. The exponent of 2^{-126} evaluates to a very small number.
- Denormalized form is needed to represent zero (with $f = 0$ and $e = 0$). It can also represent very small positive and negative number close to zero.
- For e = 255, it represents special values, such as ±INF (positive and negative infinity) and NaN (not a number). This is beyond the scope of this article.

Note: Denormalized (subnormal) numbers are numbers for which the exponent field contains a bit string of all zeros and the fraction field contains a nonzero bit string. Subnormal numbers cannot be normalized because this would result in an exponent that does not fit into the exponent field. These subnormal numbers are less accurate than normal numbers because there are fewer significant digits in the mantissa. But without this special convention for the all-zero exponent it would not be possible to represent them at all, and the designers of the IEEE standard felt that a degraded approximation is better than none.

For instance, the mantissa used for 3 x 2^{-129} is 0.011 (3 x 2^{-129} = 11B x 2^{-129} = 0.011B x 2^{3} x 2^{-129} = 0.011B x 2^{-126} = ¼+1/8 x 2^{-126} =0.375D x 2^{-126}) . Once again, only the part of the mantissa that follows the binary point is stored explicitly,

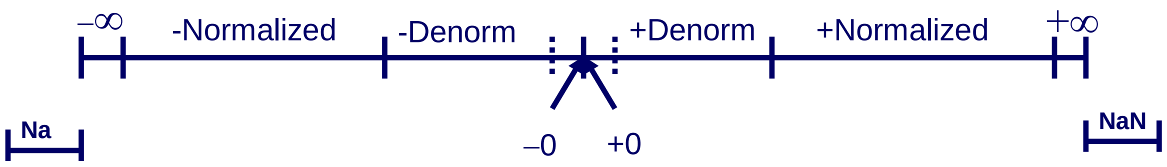
so the representation of 3 x 2^-129=0.011B x 2^{-126} is

0 00000000 01100000000000000000000

(sign positive, exponent -126, mantissa 0.01100000000000000000000)

Note: A number cannot be represented exactly if it contains more than t bits in mantissa. In that case it has to be rounded off. Moreover if, in the course of computation, a number x is produced of the form $\pm q\times N^r$ with r outside the computer's permissible range, then we say that an **overflow** or an **underflow** has occurred or that x is outside the range of the computer. Generally, an overflow results in a fatal error(or exception), and the normal execution of the program stops. An underflow, however, is usually treated automatically by setting x to zero without any interruption of the program but with a warning message in most computers.

Summary of floating point representation



Example 1:

Suppose that IEEE-754 32-bit floating-point representation pattern is

0 10000000 110 0000 0000 0000 0000 0000.

Solution:

Sign bit S = 0 ⇒ positive number

e = 1000 0000B = 128D (in normalized form)

Fraction is 1.11B (with an implicit leading 1) = 1 + 1×2^-1 + 1×2^-2 = 1.75D

The number is +1.75 × 2^(128-127) = +3.5D

Example 2:

Suppose that IEEE-754 32-bit floating-point representation pattern is

1 01111110 100 0000 0000 0000 0000 0000.

Solution:

Sign bit S = 1 ⇒ negative number

e = 0111 1110B = 126D (in normalized form)

Fraction is 1.1B (with an implicit leading 1) = 1 + 2^-1 = 1.5D

The number is -1.5 × 2^(126-127) = -0.75D

Example 3:

Suppose that IEEE-754 32-bit floating-point representation pattern is

1 01111110 000 0000 0000 0000 0000 0001.

Solution:

Sign bit S = 1 ⇒ negative number

e = 0111 1110B = 126D (in normalized form)

Fraction is 1.000 0000 0000 0000 0000 0001B (with an implicit leading 1) = 1 + 2^-23

The number is -(1 + 2^-23) × 2^(126-127) = -0.500000059604644775390625 (may not be exact in decimal!)

Example 4 (De-Normalized Form):

Suppose that IEEE-754 32-bit floating-point representation pattern is

1 00000000 000 0000 0000 0000 0000 0001.

Solution:

Sign bit S = 1 ⇒ negative number

e = 0 (in de-normalized form)

Fraction is 0.000 0000 0000 0000 0000 0001B (with an implicit leading 0) = 1×2^-23

The number is -2^-23 × 2^(-126) = -2×(-149) ≈ -1.4×10^-45

Exercises (Floating-point Numbers)

Q: **How are numbers represented in floating point in double precision in a computer?**

A: In double precision IEEE-754 format, a real number is stored in 64 bits.

- The first bit is used for the sign,
- the next 11 bits are used for the exponent, and
- the rest of the bits, that is 52, are used for mantissa.

Q : Find in double precision the

- range of the biased exponent,
- smallest number that can be represented,
- largest number that can be represented, and
- Machine epsilon?

1.4 Absolute, Percentage and Relative Errors

There are three ways in which we measure the magnitude or quantity of an error. If x^* is an approximation to the exact value x, then

- the **absolute error** in x^* $\boxed{e_{abs}(x^*) = |x - x^*|}$
- the **relative error** in x^* $\boxed{e_{rel}(x^*) = \frac{|x - x^*|}{|x|} = \frac{e_{abs}(x^*)}{|x|}}$, provided that $x \neq 0$
- the **percentage error** in x^* $\boxed{e_{per}(x^*) = \frac{|x - x^*|}{|x|} \times 100\% = e_{rel}(x^*) \times 100\%}$

Example: The three approximation of $\frac{1}{3}$ are given as 0.30, 0.33, and 0.34.

a. Find the absolute, relative and percentage errors for each approximation.
b. Which of these is the best approximation?

Solution: Here $x = \frac{1}{3}$.

a. (i) $x^* = 0.30$

$$e_{abs}(x^*) = |x - x^*| = \left|\frac{1}{3} - 0.30\right| = \frac{0.1}{3} = \frac{1}{30}$$

$$e_{rel}(x^*) = \frac{|x - x^*|}{|x|} = \frac{\frac{1}{3}}{\frac{1}{30}} = \frac{1}{10}$$

$$e_{per}(x^*) = e_{rel}(x^*) \times 100\% = \frac{1}{10} \times 100\% = 10\%$$

(ii) $x^* = 0.33$

$$e_{ab}(x^*) = |x - x^*| = \left|\frac{1}{3} - 0.33\right| = \frac{0.01}{3} = \frac{1}{300}$$

$$e_{rel}(x^*) = \frac{|x - x^*|}{|x|} = \frac{\frac{1}{300}}{\frac{1}{3}} = \frac{1}{100}$$

$$e_{per}(x^*) = e_{rel}(x^*) \times 100\% = \frac{1}{100} \times 100\% = 1\%$$

(iii) $x^* = 0.34$

$$e_{abs}(x^*) = |x - x^*| = \left|\frac{1}{3} - 0.34\right| = \frac{0.02}{3} = \frac{2}{300}$$

$$e_{rel}(x^*) = \frac{|x - x^*|}{|x|} = \frac{\frac{2}{300}}{\frac{1}{3}} = \frac{2}{100}$$

$$e_{per}(x^*) = e_{rel}(x^*) \times 100\% = \frac{2}{100} \times 100\% = 2\%$$

b. The smallest the error gives the best approximation. Hence 0.33 is the best approximation.

Example: Compute the absolute and relative errors of the approximation $x = \pi = 3.1415926536$ by $x^* = \frac{22}{7} = 3.142857$.

Solution: Here $x = 3.1415926536$ and $x^* = 3.142857$. Hence,

$$e_{abs}(x^*) = |x - x^*| = |3.1415926536 - 3.142857| = 0.0012643464$$

$$e_{rel}(x^*) = \frac{|x - x^*|}{|x|} = \frac{0.0012643464}{3.1415926536} = 0.0004024540$$

Example: Consider the absolute and relative errors in the approximation of x by x^*.

a. If $x = 4$ and $x^* = 4.2$, $e_{abs}(x^*) = 0.2$ and $e_{rel}(x^*) = 0.05$.

b. If $x = 0.0004$ and $x^* = 0.00042$, $e_{abs}(x^*) = 0.00002$ and $e_{rel}(x^*) = 0.05$.

c. If $x = 4000$ and $x^* = 4200$, $e_{abs}(x^*) = 200$ and $e_{rel}(x^*) = 0.05$.

What you conclude from these?

1.5 Decimal Places and Significant Digits

1. Let x be a real number having decimal representation

$x = a_n a_{n-1} a_{n-2} \ldots a_2 a_1 a_0 . d_1 d_2 \ldots d_k d_{k+1} d_{k+2} \ldots$ We say that x has been correctly rounded to a **k decimal places number**, which we denote it by $d_k(x)$,

$$d_{k(x)} = \begin{cases} a_n a_{n-1} a_{n-2} \ldots a_2 a_1 a_0 . d_1 d_2 \ldots d_k, & if\ d_{k+1} < 5 \\ a_n a_{n-1} a_{n-2} \ldots a_2 a_1 a_0 . d_1 d_2 .. d_k + 1 \times 10^{-k}, & if\ d_{k+1} > 5 \\ a_n a_{n-1} a_{n-2} \ldots a_2 a_1 a_0 . d_1 d_2 \ldots d_k & if\ d_k = 5\, and\, d_{k+1}\, even \\ a_n a_{n-1} a_{n-2} \ldots a_2 a_1 a_0 . d_1 d_2 \ldots d_k + 1x10^{-k} & if\ d_k = 5\, and\, d_{k+1}\, odd \end{cases} \quad (1.7)$$

Example: If $x = 92.507623,$ then $d_3(x) = 92.508$

$d_4(x) = 92.5076$

$d_8(x) = 92.50762300$

2. We say that x^* approximates x correct to k decimal places, if k is the largest positive integer such that the absolute error

$$e_{abs}(x^*) = \left|x - x^*\right| \le 0.5\times 10^{-k}.$$

Example: If $x = 0.0031758,$ then $x^* = 0.0032$ approximates x correct to four decimal places since:

$$\left|x - x^*\right| = \left|0.0031758 - 0.0032\right| = 0.0000242 = 0.242\times 10^{-4} \le 0.5\times 10^{-4}.$$

4. The following rules apply to find the **number of significant digits** in a number.

i. All nonzero digits in the number are significant.

ii. Zeros between nonzero digits in the number are significant.

iii. Zeros to the left of the first nonzero digit in the number are not significant.

iv. When the number ends in zeros that are to the right of the decimal point, these zeros are significant.

v. When the number ends in zeros that are not to the right of the decimal point, these zeros are not necessary significant.

Example: a. 0.31416 has five significant digits.

b. 102.3 have four significant digits.

c. 0.000045 has two significant digits.

d. 127 has three significant digits.

e. 0.0494200000 has nine significant.

f. 29000 may have two, three, four, or five significant digits.

5. If a number x is correctly rounded to n significant digit number (using 1 and 4), we denote the resulting approximation of x by $s_n(x)$.

Example: If $x = 53.261894,$ then $s_4(x) = 53.26$, $s_5(x) = 53.262$ and $s_7(x) = 53.26189$.

If $x = 0.009148570,$ then $s_3(x) = 0.00915$, and $s_5(x) = 0.0091486$.

6. An approximation x^* approximates x to correct to n significant digits if n is the largest positive integer such that the absolute error

$e_{abs}(x^*) = |x - x^*| \le 0.5\times 10^{s-n+1}$ where s is the largest integer such that $10^s \le |x|$

Example: If $x = 135.2469208,$ then $x^* = 135.247$ approximates x correct to six significant digits since 2 is the largest integer such that $10^2 \le |x|$ and 6 is the largest integer with

$$|x - x^*| \le 0.0000792 \le 0.0005 = 0.5\times 10^{-3} = 0.5\times 10^{2-6+1}$$

7. Writing a number x in **scientific notation(standard notation)** is to put x in the form $\pm a \times 10^n$ where $1 \le a < 10$ and n is an integer. Here a is the mantissa and n is the exponent.

Example: The scientific notation of 0.00025678 is 2.5678×10^{-4} and the scientific notation 4560 is 4.56×10^3. We can fix the mantissa to a specified number of significant digits to round off x.

Example: The scientific notation of 63459286 correct to three significant digits is 6.35×10^4.

1.6 Computer Arithmetic

Let x and y be decimal numerals and assume a computer uses rounding to n significant digits for its arithmetic calculations. Let $*$ denote any of the basic operations $+,\ -,\ \times,$ and / and $*'$ denote the computer version of the same operation. Then, we have

$$x *' y = s_n(s_n(x) * s_n(y)) \qquad (1.8)$$

Hence, the rule for arithmetic calculation by computers is

"Round input, perform exact arithmetic, and round its result."

Example: Let $x = \frac{2}{3} = 0.666...,$ and $y = \frac{5}{7} = 0.714285714285....$ Find the sum of x and y using five significant digits rounding computer arithmetic and find its absolute and relative errors.

<u>Solution</u>: Let the exact sum be S and the required approximate sum be S^*. Here, we have $s_5(x) = 0.66667$ and $s_5(y) = 0.71429$.

Thus, $s_5(x) + s_5(y) = 1.38096$ and hence $S^* = s_5(s_5(x) + s_5(y)) = 1.3810$.

The exact sum $S = \frac{2}{3} + \frac{5}{7} = \frac{29}{21}$.

The absolute error $e_a(S^*) = |S - S^*| = \left|\frac{29}{21} - 1.3810\right| = \frac{0.001}{21} = 0.000047619$.

The relative error $e_{rel}(S^*) = \dfrac{|S - S^*|}{|S|} = \dfrac{0.000047619}{\frac{29}{21}} = 0.000034483.$

1.7 Propagation of Errors

Errors of input numbers of an arithmetic procedure cause errors in output numbers. The transmission of this initial error through the procedure is called **propagation of errors**. In addition, errors may be **generated** at each step in the procedure, and we speak of the total cumulative error of any step as **accumulated error**.

Since we wish to produce results with some chosen limit of error, it is useful to consider the propagation of error. Roughly speaking, from experience, the propagated error depends on the mathematical algorithm chosen, whereas the generated error is more sensitive to the actual ordering of the computational steps.

Let us consider the propagation of errors of two numbers in the basic arithmetic operations. Let x^* and y^* approximate the true values x and y respectively. Let ε and η be the absolute errors such that $x = x^* + \varepsilon$ and $y = y^* + \eta$. Let ω denote one of the operations $+,\ -,\ \times,$ and / and ω' be the corresponding computer version of the same operation including rounding. Then,

$$x\omega y - x^*\omega' y^* = (x\omega y - x^*\omega y^*) + (x^*\omega y^* - x^*\omega' y^*)$$

Thus, we have

$$\underbrace{\left|x\omega y - x^*\omega' y^*\right|}_{\textit{accumulated error}} \le \underbrace{\left|x\omega y - x^*\omega y^*\right|}_{\textit{propagated error}} + \underbrace{\left|x^*\omega y^* - x^*\omega' y^*\right|}_{\textit{generated error}}$$

So, we observe that the accumulated error does not exceed the sum of the propagated and generated errors.

Let us estimate bounds of the absolute and relative propagated errors associated with each operations.

(i) Addition and subtraction

$$e_{abs}(x^* \pm y^*) = \left|\,(x \pm y) - (x^* \pm y^*)\right| = \left|(x - x^*) \pm (y - y^*)\right| = \left|(x - (x - \varepsilon)) \pm ((y - (y - \eta))\right|$$

$$= |\varepsilon \pm \eta| \ \le |\varepsilon| + |\eta| = e_{abs}(x^*) + e_{abs}(y^*)\ .$$

Hence: $e_{abs}(x^* \pm y^*) \le e_{abs}(x^*) + e_{abs}(y^*)$.

$$e_{rel}(x^* \pm y^*) = \frac{e_{abs}(x^* \pm y^*)}{|x \pm y|} \le \frac{e_{abs}(x^*) + e_{abs}(y*)}{|x \pm y|} = \frac{\varepsilon}{|x \pm y|} + \frac{\eta}{|x \pm y|}.$$

(ii) Multiplication

$$e_{abs}(x^* y^*) = |xy - x^* y^*| = |xy - (x-\varepsilon)(y-\eta| = |x\eta + y\varepsilon - \varepsilon\eta| \approx |x\eta + y\varepsilon|.$$

Hence, $e_{abs}(x^* y^*) \approx |x\eta + y\varepsilon| \le |x|\eta + |y|\varepsilon$

This means, $e_{abs}(x^* y^*) \le |x|\eta + |y|\varepsilon$

$$e_{rel}(x^* y^*) = \frac{e_{abs}(x^* y^*)}{|xy|} \le \frac{|x|\eta + |y|\varepsilon}{|xy|} = \frac{\varepsilon}{|x|} + \frac{\eta}{|y|} = e_{rel}(x^*) + e_{rel}(y^*)$$

Hence, $e_{rel}(x^* y^*) \le e_{rel}(x^*) + e_{rel}(y^*)$

(iii) Division

$$e_{abs}(x^*/y^*) = \left|\frac{x}{y} - \frac{x^*}{y^*}\right| = \left|\frac{x}{y} - \frac{x-\varepsilon}{y-\eta}\right| = \left|\frac{-x\eta + y\varepsilon}{y^2 - y\eta}\right| \le \frac{|x|\eta + |y|\varepsilon}{y^2}$$

Therefore, $e_{abs}(x^*/y^*) \le \dfrac{|x|\eta + |y|\varepsilon}{y^2}$

$$e_{rel}(x^*/y^*) = \frac{e_{abs}(x^*/y^*)}{|x/y|} \le \frac{\dfrac{|x|\eta + |y|\varepsilon}{y^2}}{|x/y|} = \frac{\varepsilon}{|x|} + \frac{\eta}{|y|} = e_{rel}(x^*) + e_{rel}(y^*)$$

Thus, $e_{rel}(x^*/y^*) \le e_{rel}(x*) + e_{rel}(y*)$

Example: Determine the sum of the approximate numbers $x^* = 5.3$ and $y^* = 48.97$ where these numbers are correct to all the written digits.

<u>**Solution**</u>**:** Let *x* and *y* be the true numbers approximated by the numbers x^* and y^* respectively.

x^* Approximates *x* correct to one decimal places implies that $5.25 \le x < 5.35$.

y^* Approximates *y* correct to one decimal places implies that $48.965 \le y < 48.975$.

Hence, $|x - x^*| \le 0.05$, $|y - y^*| \le 0.005$

Hence the maximum absolute error of x^* is 0.05 and that of y^* is 0.005

$e_{abs}(x^* + y^*) \le e_{abs}(x^*) + e_{abs}(y^*) \le 0.055$. Here we observe that the maximum absolute error of $x^* + y^*$ is 0.055. The required sum is therefore 54.27 ± 0.055.

Example : Find the bounds for the propagation error in adding two numbers. For example if one is calculating $X+Y$ where $X=1.5\pm0.05$, $Y=3.4\pm0.04$.

Solution : By looking at the numbers, the maximum possible value of X and Y are

$X=1.55$ and $Y=3.44$

Hence $X+Y=1.55+3.44=4.99$

is the maximum value of $X+Y$.

The minimum possible value of X and Y are $X=1.45$ and $Y=3.36$.

Hence $X+Y=1.45+3.36=4.81$ is the minimum value of $X+Y$.

Hence $4.81\le X+Y\le 4.99$.

One can find similar intervals of the bound for the other arithmetic operations of $X-Y, X*Y,\text{and}\, X/Y$. What if the evaluations we are making are function evaluations instead?

Propagation of Errors in Function Evaluations

Let $y=f(x)$ be a function that depends on x which is in error. Let the absolute error in x be Δx.

Then , we have $y+\Delta y=f(x+\Delta x)$

The error in y is given by $\Delta y=f(x+\Delta x)-f(x)$ (1.9)

From Taylor series, we $f(x+\Delta x)=f(x)+f'(x)\Delta x+\dfrac{f''(x)}{2!}\Delta x^2+\dfrac{f'''(x)}{3!}\Delta x^3+\dots$

Dropping the second and higher order derivatives and rearranging, we get

$$f(x+\Delta x)-f(x)=f'(x)\Delta x$$

Hence the absolute error in y can be estimated by $\Delta y=\left|f'(x)\right|\Delta x$

(1.10)

Similarly, if $y=f(x_1, x_2, x_3, \dots, x_n)$ is a function of several variables with an error in each x_i be Δx_i, then

$$\Delta y=\left|\frac{\partial y}{\partial x_1}\right|\Delta x_1+\left|\frac{\partial y}{\partial x_2}\right|\Delta x_2+\left|\frac{\partial y}{\partial x_3}\right|\Delta x_3+\dots+\left|\frac{\partial y}{\partial x_n}\right|\Delta x_n.$$

Example: Let $f(x)=2x^3-5x^2+2$ .Let x be approximated by 1 with an error $\Delta x=0.001$.

Now, $f'(x)=6x^2-10x=2x(3x-5)$

We have $\left|f'(1)\right|=4.$ Hence, the error in Δy , $\Delta y=\left|f'(1)\right|\Delta x=4\times0.001=0.004$

Example: Let $w = 5x^3y^2z$ and x= 3, y =1, and z = 2 with errors 0.003, 0.001, and 0.002 respectively. Determine the error in w, Δw.

Solution: Here, $\frac{\partial w}{\partial x} = 15x^2y^2z$, $\frac{\partial w}{\partial y} = 10x^3yz$, $\frac{\partial w}{\partial x} = 5x^3y^2$

Hence, $\Delta w = \left|\frac{\partial w}{\partial x}(x,\ y,\ z)\right|\Delta x + \left|\frac{\partial w}{\partial y}(x,\ y,\ z)\right|\Delta y + \left|\frac{\partial w}{\partial z}(x,\ y,\ z)\right|\Delta z$

$= 15\times 9\times 1\times 2\times 0.03 + 10\times 27\times 1\times 2\times 0.01 + 5\times 27\times 1\times 0.02$

= 1. 62

Example: The strain in an axial member of a square cross-section is given by $\in = \frac{F}{h^2E}$

where F =axial force in the member, N

h = length or width of the cross-section, m

E =Young's modulus, Pa

Given $F = 72 \pm 0.9$ N

$h = 4 \pm 0.1$ mm

$E = 70 \pm 1.5$ GPa

Find the maximum possible error in the measured strain.

Solution: $\in = \frac{72}{(4\times 10^{-3})^2(70\times 10^9)}$

$= 64.286\times 10^{-6}$

$= 64.286\mu$

$$\Delta \in = \left|\frac{\partial \in}{\partial F}\Delta F\right| + \left|\frac{\partial \in}{\partial h}\Delta h\right| + \left|\frac{\partial \in}{\partial E}\Delta E\right|$$

$$\frac{\partial \in}{\partial F} = \frac{1}{h^2E}\ ,\ \frac{\partial \in}{\partial h} = -\frac{2F}{h^3E}\ ,\ \frac{\partial \in}{\partial E} = -\frac{F}{h^2E^2}$$

$$\Delta E = \left|\frac{1}{h^2E}\Delta F\right| + \left|\frac{2F}{h^3E}\Delta h\right| + \left|\frac{F}{h^2E^2}\Delta E\right|$$

$$= \left|\frac{1}{(4\times 10^{-3})^2(70\times 10^9)}\times 0.9\right| + \left|\frac{2\times 72}{(4\times 10^{-3})^3(70\times 10^9)}\times 0.0001\right|$$

$$+ \left|\frac{72}{(4\times 10^{-3})^2(70\times 10^9)^2}\times 1.5\times 10^9\right|$$

$$= 8.0357\times10^{-7} + 3.2143\times10^{-6} + 1.3776\times10^{-6}$$

$$= 5.3955\times10^{-6}$$

$$= 5.3955\mu$$

Hence $\in = (64.286\mu \pm 5.3955\mu)$ implying that the axial strain, $\in$ is between 58.8905μ and 69.6815μ

Example : Subtraction of numbers that are nearly equal can create unwanted inaccuracies. Using the formula for error propagation, show that this is true.

Solution: Let $z = x - y$

Then $|\Delta z| = \left|\frac{\partial z}{\partial x}\Delta x\right| + \left|\frac{\partial z}{\partial y}\Delta y\right|$

$$= |(1)\Delta x| + |(-1)\Delta y|$$

$$= |\Delta x| + |\Delta y|$$

So the absolute relative change is $\left|\frac{\Delta z}{z}\right| = \frac{|\Delta x| + |\Delta y|}{|x - y|}$

As x and y become close to each other, the denominator becomes small and hence create large relative errors.

For example if $x = 2 \pm 0.001$, $y = 2.003 \pm 0.001$

$$\left|\frac{\Delta z}{z}\right| = \frac{|0.001| + |0.001|}{|2 - 2.003|} \quad = 0.6667 \quad = 66.67\%$$

Adama Science and Technology University
Department of Applied Mathematics
Numerical Analysis(math3201) Worksheet 1

1. Determine the sources of error that arise in computing the surface area of the earth using the formula S=$4\pi r^2$, where r is the radius of the earth.

2. Compute the absolute and relative error in approximating a number x by x^*

a) $x = \pi$ and $x^* = \frac{22}{7}$ b) $x = \sqrt{2}$ and $x^* = 1.414$ c) $x = e$ and $x^* = 2.718$

3. The derivative of $f(x)$ at a particular value of x can be approximately be calculated by

$$f'(x) = \frac{f(x+h)-f(x)}{h} \text{ for } f(x) = 7e^{0.5x} \text{ and } h = 0.3, x = 2. \text{ Find}$$

a) The exact value of $f'(x)$ b) The approximate value of $f'(x)$ c) The absolute and relative errors of this approximation

4. If a number x is approximated by x^*=123.456 and the relative error is 0.1, then what is the possible range of the values x?

5. If $u = \frac{x^2y}{z}$ and the error in x, y and z are 0.001,0.002 and 0.003 respectively, compute the maximum relative error in u when $x = y = z = 3$.

6. Identify the IEEE standard floating-point numbers corresponding to the following bit strings.

	0000000	0000000000000000000000

	0000000	0000000000000000000000
	1111111	0000000000000000000000

	1111111	0000000000000000000000

	0000001	0000000000000000000000

7. Compute the largest and smallest positive numbers that can be represented in the 32-bit normalized form.

8. Compute the largest and smallest negative numbers can be represented in the 32-bit normalized form.

9. Repeat (1) for the 32-bit denormalized form.

10. Repeat (2) for the 32-bit denormalized form.

11. Find the minimum number of iteration required for the Bisection Method to obtain the root of an equation $f(x) = 0$ on [4,5] with a tolerance of $\varepsilon = 10^{-3}$.

12. Use the Bisection method to find solutions accurate to within 10^{-2} for the following problems:

(a) $x - 2^{-x} = 0$ for $0 \le x \le 1$

(b) $x = \tan x$ for $4 \le x \le 4.5$ (c) $2 + \cos(e^x - 2) - e^x = 0$ for $0.5 \le x \le 1.5$

13. Using Bisection Method solve the following equations:

a) $f(x) = \sqrt{x} - cosx$ on [0,1]. Perform only the first three iterations.

b) $x = \sqrt{5}$ correct to two decimal places.

14. Let $f(x) = -x^3 - cosx$. Taking $x_0 = -1$ and $x_1 = 0$. find x_3 using each of the following methods.

a) False position method b) Secant method

15. Obtain the root of the equation $x^x = 100$ correct to three significant digits, using the Secant method .Given that the root lies between 3 and 4.

16. Let $x^3 + 2x^2 - 2x - 4 = 0$ be a given equation. Then find

a) An iteration function $\emptyset$ so that the iteration $x_{n+1} = \emptyset(x_n)$ converges to the desired root.

b) The root of the given equation correct to three decimal places(use $x_0 = 1$)

17. Using Newton-Raphson method solve the following equations

a) $3xe^x = 0$ with absolute error tolerance $\varepsilon = 10^{-5}$. Take $x_0 = 1.5$

b) $x^4 - x - 10 = 0$ correct to three decimal places.

18. Find a bound for the number of iterations needed to achieve an approximation with an absolute accuracy 10^{-4} to the solution of $x^3 - x - 1 = 0$ lying in the interval [1,2].Find an approximation to the root with this degree of accuracy.