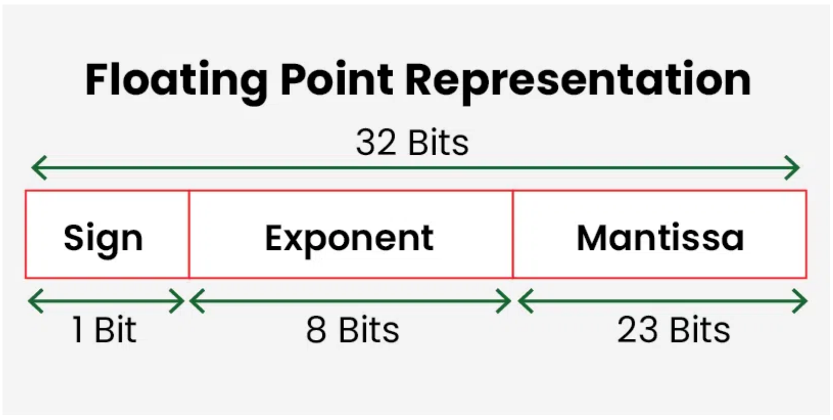
NUMERIC COMPUTATION

In the field of **numeric computation**, computers perform mathematical operations on real-world data. However, because computers represent numbers in a finite manner, special techniques are used to handle issues such as **floating-point arithmetic**, **errors**, and **iterative methods** for solving equations. This unit will cover the basic concepts of **floating-point numbers**, their operations, error analysis, and some important **iterative methods** for solving equations numerically.

1. Computer Arithmetic and Floating Point Numbers

Floating-point arithmetic is used to represent real numbers in a way that can accommodate a wide range of values by using a combination of a **mantissa** and an **exponent**. This representation allows for both very small and very large numbers, but it also introduces the possibility of errors due to the finite precision of the system.

1.1 Floating-Point Representation



A floating-point number is generally represented as:

$$\text{Floating Point Number} = \text{Mantissa} \times \text{Base}^{\text{Exponent}}$$

For example, the number 123.456 can be represented in scientific notation as:

$$1.23456 \times 10^2$$

In binary (base-2), this becomes:

$$1.111110011 \times 2^6$$

This is how **floating-point numbers** are represented in computers, using a **sign bit**, **exponent**, and **mantissa**. A floating-point number is limited by the number of bits used to store the exponent and mantissa.

1.2 Floating Point Operations

The operations on floating-point numbers, such as addition, subtraction, multiplication, and division, are carried out by performing these operations on the mantissas, adjusting the exponent accordingly.

However, because floating-point numbers have limited precision, errors may arise during these operations due to **rounding**. For instance, small discrepancies can occur when a result is truncated or rounded to fit within the available precision of the computer.

1.3 Normalization

Normalization is the process of adjusting the representation of floating-point numbers so that the mantissa falls within a certain range. In the case of binary floating-point numbers, this typically means ensuring that the first digit of the mantissa is 1 (for base-2). This process maximizes the precision of the floating-point representation and minimizes errors.

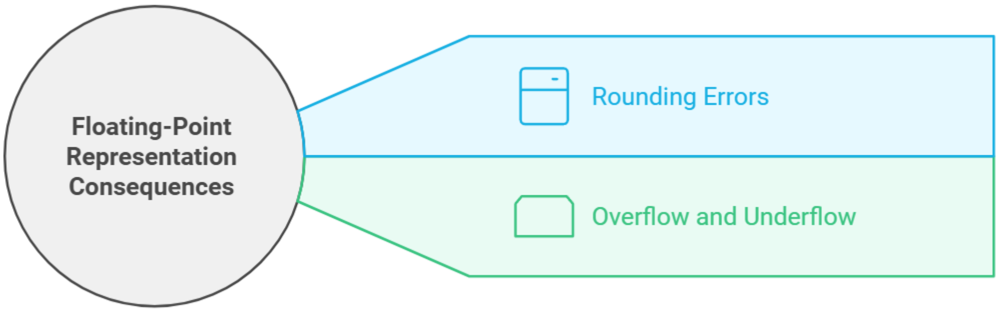
For example:

- The number 0.00123 can be normalized as 1.23×10^{-3} in scientific notation.
- Similarly, in binary, 0.0001101_2 would be normalized to 1.101×2^{-4}.

Normalization helps in ensuring that calculations are as accurate as possible given the constraints of the computer system.

1.4 Consequences of Floating-Point Representation

Floating-point representation, while allowing for a wide range of values, has its drawbacks:

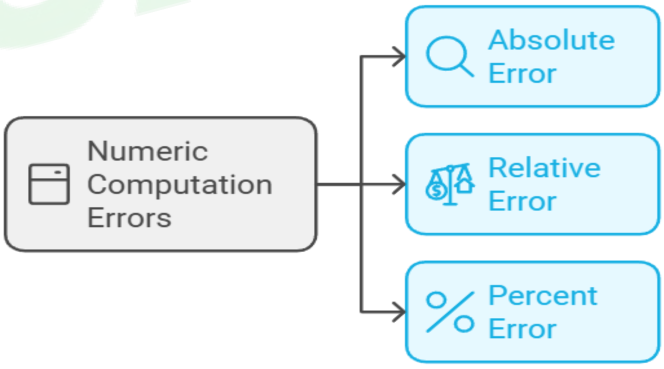


- **Rounding Errors**: Since the mantissa has finite precision, the result of an arithmetic operation may differ slightly from the exact result.
- **Overflow and Underflow**: Overflow occurs when the result of an operation exceeds the maximum representable number. Underflow occurs when the result is smaller than the smallest representable number.

---

2. Errors in Numeric Computation

In numeric computation, errors can arise in various ways, especially when approximating real-world numbers. These errors are categorized into **absolute error**, **relative error**, and **percent error**.



2.1 Absolute Error

The **absolute error** is the difference between the exact value and the computed (or approximated) value. Mathematically:

$$\text{Absolute Error} = |\text{True Value} - \text{Approximate Value}|$$

For example, if the true value is 5.567 and the computed value is 5.56, the absolute error is:

$$\text{Absolute Error} = |5.567 - 5.56| = 0.007$$

2.2 Relative Error

Relative error is the absolute error divided by the true value. It provides a measure of how large the error is in relation to the size of the number:

$$\text{Relative Error} = \frac{|\text{True Value} - \text{Approximate Value}|}{|\text{True Value}|}$$

For example, if the true value is 5.567 and the approximate value is 5.56:

$$\text{Relative Error} = \frac{0.007}{5.567} \approx 0.00126$$

2.3 Percent Error

Percent error is simply the relative error multiplied by 100 to express it as a percentage:

$$\text{Percent Error} = \left(\frac{|\text{True Value} - \text{Approximate Value}|}{|\text{True Value}|}\right) \times 100$$

For the same example:

$$\text{Percent Error} = 0.00126 \times 100 \approx 0.126\%$$

These errors are important in numerical analysis because they help determine the accuracy of the results of calculations.

---

3. Iterative Methods for Solving Equations

When solving transcendental or polynomial equations numerically, iterative methods are used because analytical solutions are often not available or are difficult to compute. These methods are based on successive approximations to reach a solution.

3.1 Zeros of Transcendental Equations

A **transcendental equation** is one that cannot be solved by algebraic means and involves transcendental functions like trigonometric, exponential, and logarithmic functions. For example, the equation $f(x) = 0$ may not have a closed-form solution, but iterative methods can approximate the solution.

3.2 Bisection Method

The **bisection method** is a simple and reliable root-finding method that works on continuous functions. It is based on the **Intermediate Value Theorem**, which states that if a continuous function $f(x)$ has opposite signs at two points 'a' and 'b', then there must be 'a' root between a and 'b'.

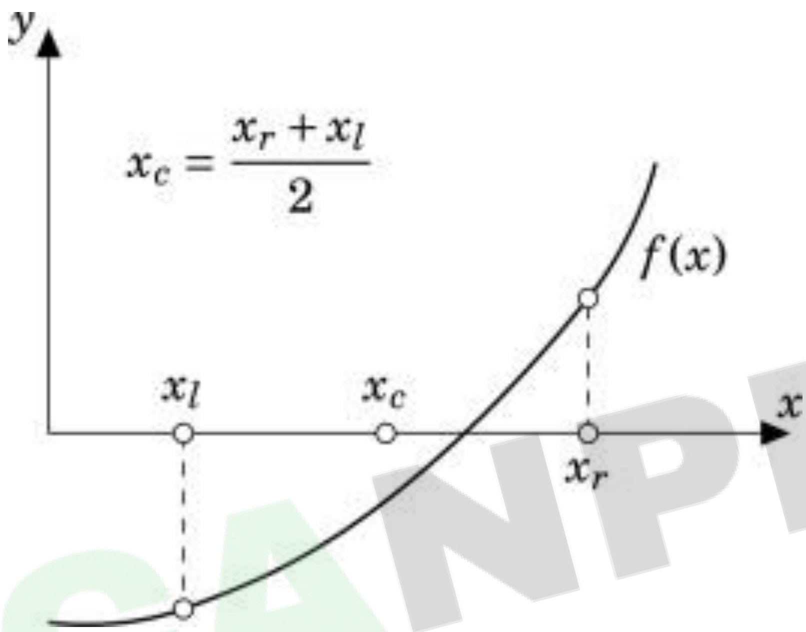
The method proceeds by:

1. Finding the midpoint $c = \frac{a+b}{2}$.

2. Evaluating $f(c)$.

3. Replacing either 'a' or 'b' with 'c', depending on the sign of $f(c)$, and repeating until the solution converges.

This method guarantees convergence, but it may take many iterations if the interval is large.



3.3 False Position Method

The **false position method** (also known as the **Regula Falsi method**) is similar to the bisection method but uses linear interpolation to find the root. In this method:

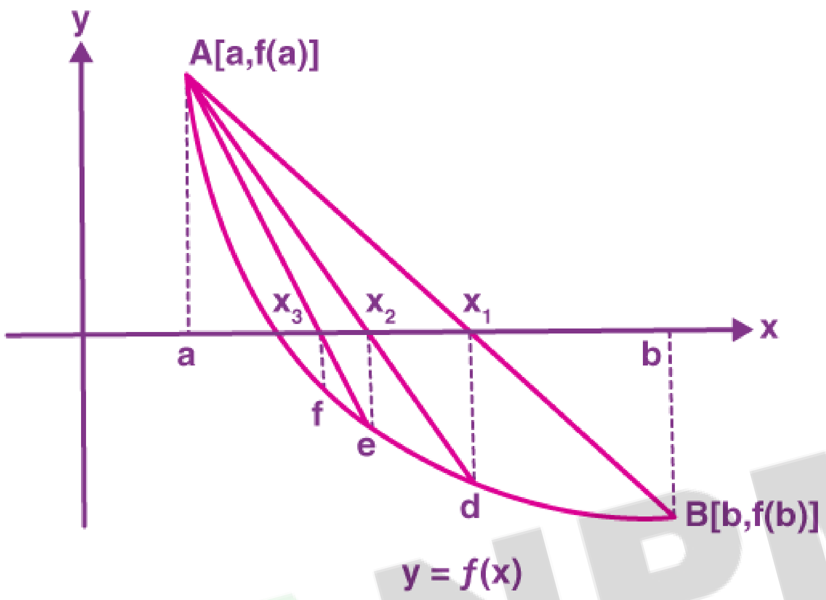
1. We choose two initial points 'a' and 'b' where $f(a)$ and $f(b)$ have opposite signs.

2. A straight line is drawn between these two points.

3. The intersection of this line with the x-axis is used as the new approximation for the root.

4. The process is repeated by replacing one of the initial points with the new approximation.

The false position method can converge faster than the bisection method but still guarantees convergence.



3.4 Newton-Raphson Method

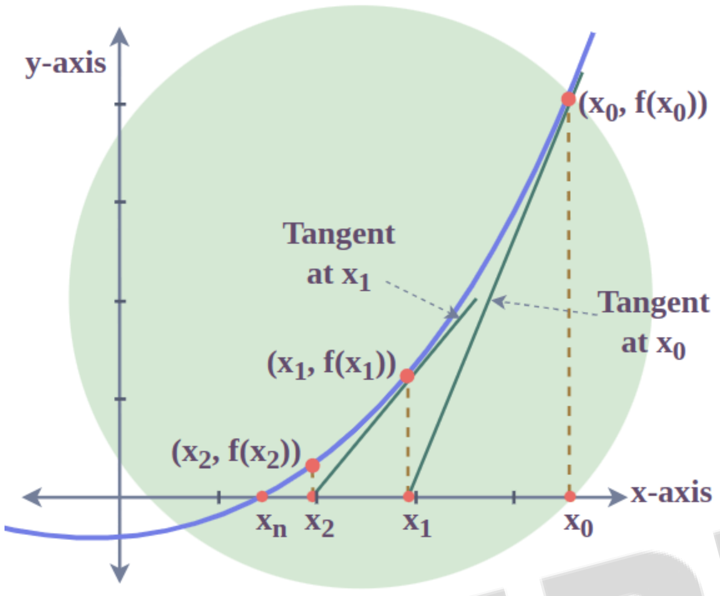
The **Newton-Raphson method** is an iterative method that uses the derivative of the function to approximate the root. The method starts with an initial guess x_0 and iteratively refines it using the formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Here:

- $f(x_n)$ is the function value at x_n,
- $f'(x_n)$ is the derivative of the function at x_n,
- x_{n+1} is the new approximation.

This method is often faster than the bisection or false position methods but requires that the derivative of the function be known and non-zero at the root.



3.5 Convergence of Solution

The convergence of these iterative methods refers to how quickly the solution approaches the true root. For example:

- The **bisection method** always converges, but it may require many iterations.
- The **false position method** typically converges faster than bisection.
- The **Newton-Raphson method** has quadratic convergence (i.e., the number of correct digits doubles with each iteration), which makes it the fastest among the three methods, provided a good initial guess is chosen.

Example

Let’s consider an example equation:

$$f(x) = x^2 - 4 = 0$$

We want to find the root using the **bisection method**.

1. Choose an initial interval, say $a = 1$ and $b = 3$.
2. Compute the midpoint $c = \frac{1+3}{2} = 2$.
3. Evaluate $f(2) = 2^2 - 4 = 0$, so $x = 2$ is the root.

This method converges quickly in this case, but if the initial guess were further from the root, more iterations would be required.

Conclusion

Numerical computation plays a critical role in the field of computer science and engineering, providing the tools and techniques to solve complex mathematical problems. **Floating-point arithmetic**, **error analysis**, and **iterative methods** are foundational for numerical methods, and understanding these concepts is essential for handling real-world computations efficiently. The methods discussed, such as **bisection**, **false position**, and **Newton-Raphson**, are indispensable for solving equations that cannot be easily tackled algebraically. By understanding how these methods work and their convergence properties, we can ensure more accurate and efficient solutions in a wide range of applications.