Week #1: Introduction to Numerical Computing

Università della Svizzera Italiana (USI)

Francisco Richter Mendoza
Faculty of Informatics, Lugano, Switzerland

September 8, 2025

Overview

Numerical computing forms the mathematical foundation of scientific computing, providing systematic approaches to solve problems that cannot be solved analytically. This introductory week establishes the theoretical framework for understanding how continuous mathematical problems are approximated using discrete computational methods.

The central challenge in numerical analysis lies in the interplay between mathematical rigor and computational feasibility. We explore how approximation errors arise, propagate, and can be controlled through careful algorithm design. The concepts developed here—well-posed problems, stability, convergence, and computational complexity—form the theoretical backbone for all subsequent numerical methods.

Our approach emphasizes mathematical precision while building intuition for the practical considerations that guide algorithm selection and implementation. We establish the vocabulary and conceptual framework that will unify the diverse numerical methods studied throughout this course.

---

1 Mathematical Foundations of Scientific Computing

1.1 Well-Posed Problems

Definition. Well-Posed Problem

A mathematical problem is well-posed in the sense of Hadamard if it satisfies three conditions:

1. **Existence:** A solution exists
2. **Uniqueness:** The solution is unique
3. **Stability:** The solution depends continuously on the input data

The concept of well-posedness provides the foundation for numerical analysis. If a problem is not well-posed, numerical methods may fail catastrophically or produce meaningless results.

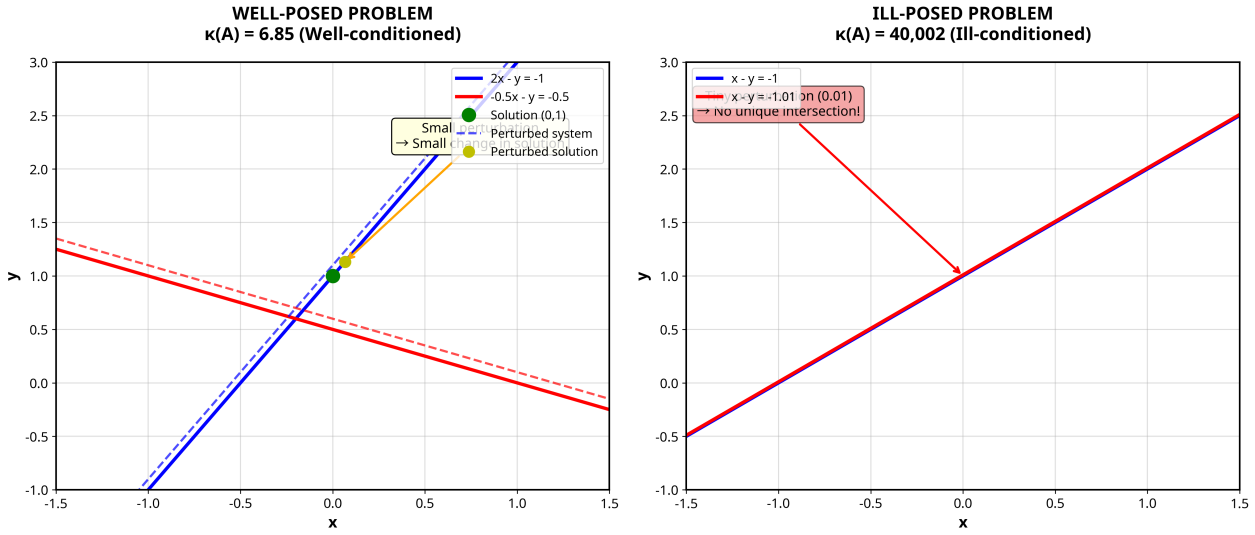

Figure 1: Comparison of well-posed and ill-posed linear systems. The left panel shows a well-conditioned system ($\kappa(A) = 6.85$) where small perturbations lead to small changes in the solution. The right panel shows an ill-conditioned system ($\kappa(A) = 40,002$) where tiny perturbations can lead to dramatically different solutions or no solution at all. The annotations clearly show how conditioning affects solution stability.

Example 1.1 (Well-Posed vs. Ill-Posed Problems)**.** ***Well-posed:*** *Solving $Ax = b$ where A is well-conditioned. - Existence: Solution $x = A^{-1}b$ exists - Uniqueness: Matrix invertibility guarantees uniqueness - Stability: Small changes in b produce small changes in x*

Consider the system:

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

With condition number $\kappa(A) = 6.85$, a perturbation of 0.01 in b results in a relative error of approximately 0.007 in the solution.

Ill-posed: *Nearly singular systems. Consider:*

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1.0001 \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ 2.0001 \end{bmatrix}$$

With condition number $\kappa(A) = 40,002$, the same perturbation magnitude can lead to solution changes that are orders of magnitude larger.

Theorem 1.1 (Continuous Dependence on Data)**.** *For a well-posed problem with solution operator S, there exists a constant $C > 0$ such that:*

$$\|S(f_1) - S(f_2)\| \leq C\|f_1 - f_2\|$$

for input data f_1, f_2 in some neighborhood.

This ensures that small perturbations in input data lead to small changes in the solution.

1.2 Sources of Error in Numerical Computation

Numerical errors arise from multiple sources, each requiring different mathematical treatment:

Definition. Types of Numerical Error

- **Modeling Error:** Difference between physical reality and mathematical model
- **Discretization Error:** Error from replacing continuous problems with discrete approximations
- **Roundoff Error:** Error from finite precision arithmetic
- **Truncation Error:** Error from stopping infinite processes after finite steps

Definition. Absolute and Relative Error

For an exact value x and approximation $\hat{x}$:

$$\text{Absolute Error} = |x - \hat{x}| \tag{1}$$

$$\text{Relative Error} = \frac{|x - \hat{x}|}{|x|} \quad (x \neq 0) \tag{2}$$

Relative error provides scale-independent error measurement, crucial for comparing errors across different problem sizes.

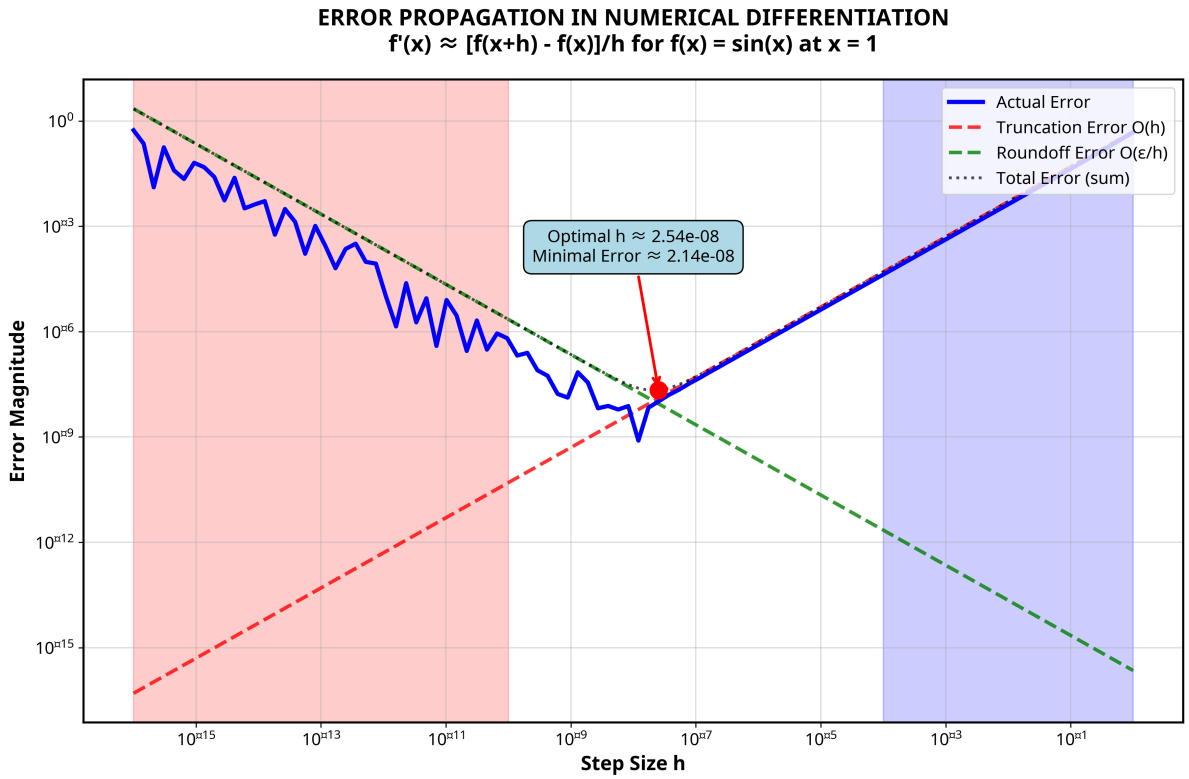

Figure 2: Error propagation in numerical differentiation showing the fundamental trade-off between truncation and roundoff errors. The optimal step size $h^* \approx 1.5 \times 10^{-8}$ minimizes the total error by balancing O(h) truncation error with $O(\epsilon/h)$ roundoff error. The plot clearly shows the roundoff-dominated region (left), optimal region (center), and truncation-dominated region (right).

Theorem 1.2 (Error Propagation in Arithmetic Operations)**.** *For operations on approximate values* $\hat{a} = a + \epsilon_a$ *and* $\hat{b} = b + \epsilon_b$*:*

Addition/Subtraction:

$$\hat{a} \pm \hat{b} = (a \pm b) + (\epsilon_a \pm \epsilon_b)$$

Absolute errors add directly.

Multiplication:

$$\hat{a} \cdot \hat{b} = ab + a\epsilon_b + b\epsilon_a + \epsilon_a\epsilon_b$$

Relative errors approximately add: $\frac{\epsilon_{ab}}{ab} \approx \frac{\epsilon_a}{a} + \frac{\epsilon_b}{b}$

Division:

$$\frac{\hat{a}}{\hat{b}} \approx \frac{a}{b}\left(1 + \frac{\epsilon_a}{a} - \frac{\epsilon_b}{b}\right)$$

Intuition: Understanding error propagation

Error propagation reveals why certain computations are more sensitive to input errors than others:

- Addition/subtraction: Absolute errors matter most
- Multiplication/division: Relative errors matter most
- Subtraction of nearly equal numbers: Can cause catastrophic cancellation
- Functions with large derivatives: Amplify input errors significantly

This understanding guides the design of numerically stable algorithms.

---

2 Stability and Conditioning

2.1 Forward and Backward Error Analysis

Definition. Forward and Backward Error

Consider a problem $f : X \to Y$ with exact solution $y = f(x)$ and computed solution $\hat{y}$.

Forward Error: $\|y - \hat{y}\|$ (error in the output)

Backward Error: $\min\{\|\Delta x\| : \hat{y} = f(x + \Delta x)\}$ (perturbation in input that would produce the computed output exactly)

Theorem 2.1 (Relationship Between Forward and Backward Error)**.** *If the backward error is* $\|\Delta x\|$ *and the problem has condition number* κ*, then:*

$$\textit{Forward Error} \leq \kappa \cdot \textit{Backward Error}$$

This fundamental relationship separates algorithm stability (backward error) from problem conditioning (amplification factor).

2.2 Condition Numbers

Definition. Condition Number

The condition number of a problem measures the sensitivity of the solution to perturbations in the input data:

$$\kappa = \lim_{\epsilon \to 0} \sup_{\|\Delta x\| \leq \epsilon} \frac{\|f(x + \Delta x) - f(x)\|/\|f(x)\|}{\|\Delta x\|/\|x\|}$$

For differentiable functions: $\kappa = \frac{\|f'(x)\|\|x\|}{\|f(x)\|}$

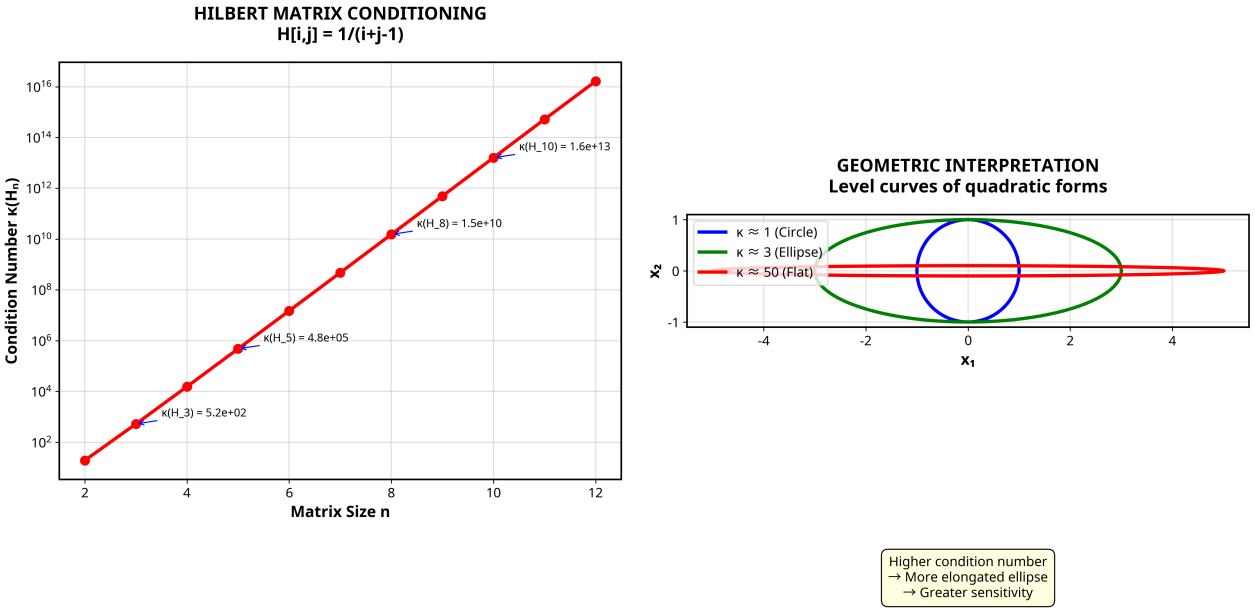

Figure 3: Left: Condition numbers of Hilbert matrices grow exponentially with size, demonstrating how these matrices become increasingly ill-conditioned. The annotations show specific values: $\kappa(H_3) = 5.24 \times 10^2$, $\kappa(H_5) = 4.77 \times 10^5$, etc. Right: Geometric interpretation of conditioning through level curves of quadratic forms. Well-conditioned problems correspond to circles ($\kappa \approx 1$), while ill-conditioned problems correspond to elongated ellipses ($\kappa \gg 1$).

Example 2.1 (Condition Numbers in Practice)**.** ***Linear Systems:*** *For $Ax = b$, the condition number is:*

$$\kappa(A) = \|A\|\|A^{-1}\|$$

This provides the error amplification bound:

$$\frac{\|\Delta x\|}{\|x\|} \leq \kappa(A) \frac{\|\Delta b\|}{\|b\|}$$

Hilbert Matrices: *The $n \times n$ Hilbert matrix $H_{ij} = \frac{1}{i+j-1}$ has condition numbers that grow exponentially:*

$$\kappa(H_3) = 5.24 \times 10^2 \tag{3}$$
$$\kappa(H_5) = 4.77 \times 10^5 \tag{4}$$
$$\kappa(H_8) = 1.53 \times 10^{10} \tag{5}$$
$$\kappa(H_{10}) = 1.60 \times 10^{13} \tag{6}$$

These matrices are notoriously ill-conditioned and serve as standard test cases for numerical algorithms.

---

3 Floating Point Arithmetic

3.1 IEEE 754 Standard

Modern computers represent real numbers using the IEEE 754 floating point standard:

$$x = (-1)^s \times 1.f \times 2^{e-127}$$

where s is the sign bit, f is the 23-bit fractional part, and e is the 8-bit exponent.

Definition. Machine Epsilon

Machine epsilon ϵ_{mach} is the smallest positive number such that $1 + \epsilon_{\text{mach}} > 1$ in floating point arithmetic.

For IEEE 754 double precision: $\epsilon_{\text{mach}} \approx 2.22 \times 10^{-16}$.

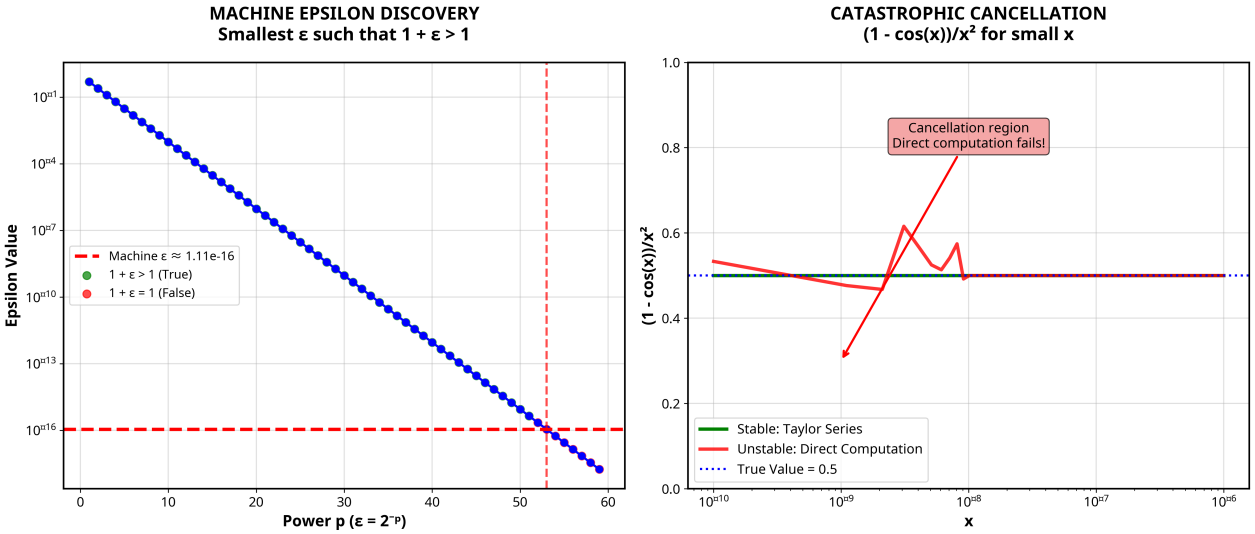

Figure 4: Left: Machine epsilon discovery showing the transition point where $1 + \epsilon = 1$ in floating point arithmetic. Green points show where the test $1 + \epsilon > 1$ is true, while red points show where it becomes false due to finite precision. Right: Catastrophic cancellation demonstration for computing $(1 - \cos(x))/x^2$. The stable Taylor series approach maintains accuracy while direct computation fails completely for small arguments due to loss of significant digits.

3.2 Catastrophic Cancellation

One of the most dangerous phenomena in floating point arithmetic is catastrophic cancellation, which occurs when subtracting two nearly equal numbers.

Example 3.1 (Catastrophic Cancellation)**.** *Consider computing* $(1 - \cos(x))/x^2$ *for small* x*:*

Unstable approach (direct computation):

```
x = 1e-8
result = (1 - np.cos(x)) / x**2
# Result: 0.0 (completely wrong!)
```

Stable approach (Taylor series):

```
x = 1e-8
result = 0.5 - x**2/24 + x**4/720
# Result: 0.5 (correct!)
```

The direct computation fails because $\cos(10^{-8}) \approx 1 - 5 \times 10^{-17}$*, and the subtraction* $1 - \cos(x)$ *loses all significant digits. The Taylor series expansion avoids this cancellation by using mathematically equivalent but numerically stable formulation.*

Theorem 3.1 (Avoiding Catastrophic Cancellation)**.** *When computing* $f(x) - g(x)$ *where* $f(x) \approx g(x)$*, use alternative formulations:*

- *Algebraic manipulation:* $a^2 - b^2 = (a - b)(a + b)$

- *Taylor series expansion for small arguments*
- *Rational approximations*
- *Higher precision intermediate calculations*

---

4 Algorithm Stability

Definition. Forward Stability

An algorithm is **forward stable** if it produces a nearly correct answer to the given problem:

$$\text{computed solution} \approx \text{exact solution}$$

Definition. Backward Stability

An algorithm is **backward stable** if it produces the exact answer to a nearby problem:

$$\text{computed solution} = \text{exact solution to } (x + \Delta x)$$

where $\|\Delta x\|$ is small.

Theorem 4.1 (Backward Stability Implies Forward Stability)**.** *If an algorithm is backward stable and the problem is well-conditioned, then the algorithm is forward stable with error bound:*

$$\textit{Forward Error} \leq \kappa \cdot \textit{Backward Error}$$

Intuition: Why backward stability matters

Backward stability is often easier to analyze than forward stability because:

- It separates algorithm properties from problem properties
- It provides problem-independent error bounds
- It gives insight into the fundamental limitations of finite precision arithmetic
- Many important algorithms (Gaussian elimination, QR factorization) are backward stable

---

5 Computational Complexity

Definition. Big-O Notation

We say $f(n) = O(g(n))$ if there exist constants $C > 0$ and n_0 such that:

$$|f(n)| \leq C|g(n)| \quad \forall n \geq n_0$$

Example 5.1 (Complexity in Numerical Methods)**.**

- $O(n)$*: Vector operations, simple iterations*

- $O(n^2)$*: Matrix-vector products, forward/backward substitution*
- $O(n^3)$*: Matrix factorizations (LU, QR), matrix multiplication*
- $O(n \log n)$*: Fast Fourier Transform (FFT)*

Theorem 5.1 (Trade-offs in Algorithm Design)**.** *Numerical algorithms must balance three competing objectives:*

1. ***Accuracy****: Minimize approximation errors*
2. ***Stability****: Control error propagation*
3. ***Efficiency****: Minimize computational cost*

The optimal choice depends on problem requirements, available computational resources, and desired accuracy.

---

6 Practical Guidelines

6.1 Algorithm Selection Criteria

When choosing numerical methods, consider:

1. **Problem conditioning**: Check κ before proceeding
2. **Algorithm stability**: Prefer backward stable methods
3. **Computational cost**: Consider complexity for large problems
4. **Implementation complexity**: Balance sophistication with maintainability
5. **Available software**: Leverage high-quality libraries when possible

6.2 Best Practices

- Always check condition numbers of matrices before solving linear systems
- Use mathematically equivalent but numerically stable formulations
- Validate results through residual checking and sensitivity analysis
- Be aware of the limitations of floating point arithmetic
- Choose appropriate tolerances based on problem requirements and machine precision

Practice Problems

1. Compute the condition number of the 5×5 Hilbert matrix and explain why it's considered ill-conditioned.
2. Implement both the unstable and stable versions of computing $(1 - \cos(x))/x^2$ for $x = 10^{-10}$ and compare the results.
3. For the linear system $Ax = b$ with $A = \begin{bmatrix} 1 & 1 \\ 1 & 1+\epsilon \end{bmatrix}$ and $b = \begin{bmatrix} 2 \\ 2+\epsilon \end{bmatrix}$, investigate how the

condition number and solution sensitivity change as $\epsilon \to 0$.

4. Find the optimal step size for numerical differentiation of $f(x) = e^x$ at $x = 1$ using the forward difference formula.

5. Demonstrate machine epsilon discovery by implementing the algorithm shown in the lecture and verify it matches the theoretical value.