AMATH141: Numerical Methods for Applied Sciences

1. Floating-Point Arithmetic and Error Analysis

Ahmed Ratnani

January 25, 2026

University Mohammed VI Polytechnic

THE UM6P VANGUARD CENTER

#Empowering Minds.

Plan

Introduction & Motivation

From mathematical truth to computational reality

A Simple Question

"Do computers compute real numbers?"

- We write formulas using $\mathbb{R}$
- We trust numerical outputs
- We rarely question the arithmetic itself

This lecture challenges that assumption.

The Hidden Assumption

- Mathematics assumes infinite precision
- Computers use finite memory
- Real numbers must be approximated

Computers do not compute $\mathbb{R}$ – they compute *approximations* of $\mathbb{R}$.

Why This Matters

- Space missions have failed
- Defense systems have malfunctioned
- Financial systems have drifted
- Microprocessors have been recalled

> *"These failures were not caused by bad mathematics. They were caused by misunderstanding how numbers behave on computers."*

What You Will Learn

By the end of this lecture, you will be able to:

- Explain why floating-point errors are unavoidable
- Interpret absolute and relative error correctly
- Recognize unstable formulas
- Understand the meaning of numerical stability
- Know the limits of achievable accuracy

This is about numerical responsibility.

Reference Material

- **N. J. Higham**, `Accuracy and Stability of Numerical Algorithms`
  SIAM – the reference on floating-point error analysis and stability
- **D. Goldberg**, `What Every Computer Scientist Should Know About Floating-Point Arithmetic`
  ACM Computing Surveys – a classic, accessible introduction
- **L. N. Trefethen & D. Bau**, `Numerical Linear Algebra`
  SIAM – intuition-driven approach to conditioning and stability

Why Numerical Errors Matter

From rockets to microprocessors

Vancouver Stock Exchange Index Drift (1982–1983)

Context

- Index recomputed thousands of times per day
- Market stable, index slowly drifted downward

Numerical cause

- Truncation applied after each update
- Systematic rounding bias
- No error cancellation

Lesson

- Rounding errors are not random
- Bias accumulates deterministically

> Rounding is a modeling choice.


INTERNATIONAL REPORT

Mexico Threatened By Oil Price Cuts

International Market Cited

Scottish Financiers Stay Wary Of Fads

Vancouver Exchange Struggles to Repair Its Tattered Image

Profit at British Aerospace Up

Venezuela Debt Pact

Pakistani Bonds Sell Heavily

Patriot Missile Failure (1991)

Context

- Dhahran, Gulf War
- Patriot missile system fails to intercept a Scud
- 28 soldiers killed

Numerical cause

- Time stored in tenths of seconds
- Binary approximation of 0.1 truncated
- Error $\approx 10^{-7}$ seconds per second
- System ran continuously for ~ 100 hours

Result

- Accumulated timing error ≈ 0.34 seconds
- Target position miscomputed by hundreds of meters

Tiny errors accumulate. Physics does not forgive.

Intel Pentium FDIV Bug (1994)

Context

- Certain divisions produced wrong results
- Rare but systematic
- Public discovery led to massive backlash
- $4195835 - 4195835/3145727 * 3145727$ gives 256 instead of 0

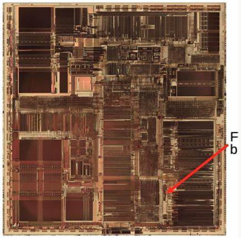
Numerical cause

- Division implemented via lookup tables
- 5 missing entries out of ~ 1000
- Errors in the 4th–5th decimal digit

Impact

- Intel initially: "users will not notice"
- Mathematicians noticed immediately
- $475 million recall

Rare errors become inevitable at scale.



Ariane 5 Flight 501 (1996)

Context

- Maiden flight of Ariane 5
- Rocket self-destructs 40 seconds after launch
- Loss: $370 million

Numerical cause

- Horizontal velocity stored as 64-bit floating-point
- Converted to 16-bit signed integer
- Value exceeded representable range
- Overflow triggered an exception

Deeper issue

- Software reused from Ariane 4
- Assumptions on variable ranges silently violated
- Overflow checks disabled for performance

Numerical assumptions are part of the specification.

Mars Climate Orbiter (1999)

Context

- NASA spacecraft lost on arrival to Mars
- Entered atmosphere too low and disintegrated
- Loss: $125 million

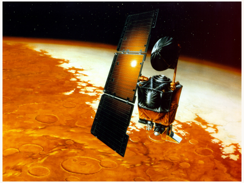
Numerical cause

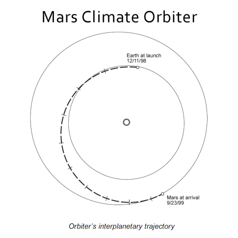
- One subsystem used imperial units
- Another used metric units
- No unit conversion performed

Lesson

- Numbers were computationally correct
- Physical meaning was wrong

Numbers without units are meaningless.





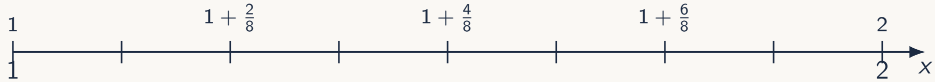
What Do These Failures Have in Common?

- Finite representation of real numbers
- Implicit numerical assumptions
- Error accumulation and amplification
- Confusion between mathematics and computation

> *"If two formulas are mathematically equivalent, that tells you nothing about their numerical behavior."*

Now: how floating-point arithmetic really works.

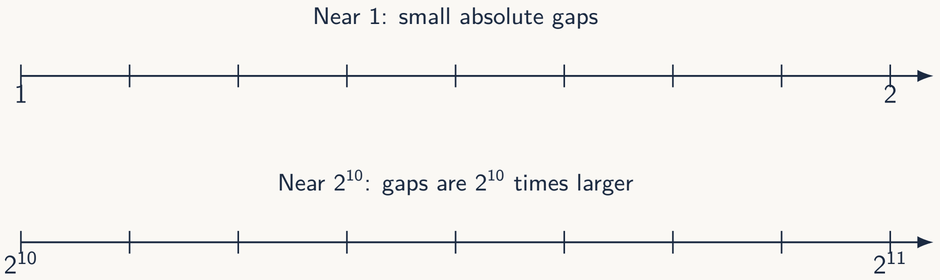
Floating-Point Numbers Form a Non-Uniform Grid



Toy model: numbers in $[1, 2)$ are equally spaced (here step $1/8$).
But scaling by powers of 2 changes the spacing globally.

- Around 1, spacing is about ε.
- Around 2^k, spacing is about $2^k\varepsilon$ (resolution worsens with magnitude).

Same Relative Precision, Larger Absolute Gaps



*Floating-point has roughly constant **relative** precision, not constant **absolute** precision.*

```python
# Accumulation of a tiny rounding error
dt = 0.1          # supposed to be exact
t = 0.0

for _ in range(1_000_000):
    t += dt

print(t)
print("Expected:", 100_000.0)
print("Error:", t - 100_000.0)
```

```python
import math

def f_bad(x):
    return math.sqrt(x + 1) - math.sqrt(x)

def f_good(x):
    return 1.0 / (math.sqrt(x + 1) + math.sqrt(x))

for x in [1e2, 1e6, 1e10, 1e16]:
    print(f"x = {x:.0e}")
    print(" bad :", f_bad(x))
    print(" good:", f_good(x))
    print()
```

```python
a = 1e16
b = -1e16
c = 1.0

print((a + b) + c)
print(a + (b + c))
```

```python
x = 1e-12
x_tilde = 0.0

abs_error = abs(x - x_tilde)
rel_error = abs_error / abs(x)

print("Absolute error:", abs_error)
print("Relative error:", rel_error)
```

Exercises

1. **Non-representability (warm-up)**
   In Python, compute `0.1 + 0.2 - 0.3`.
   Question: Is the result a bug? What does it reveal about base-2 representation?
2. **Accumulation (Patriot analogy)**
   Compute $t = \sum_{i=1}^{10^6} 0.1$ in Python. Compare to 10^5.
   Question: Why does a tiny error per step accumulate? Is the error random?
3. **Catastrophic cancellation (the key skill)**
   Evaluate for $x = 10^k$ with $k \in \{2, 6, 10, 16\}$:

   $$f(x) = \sqrt{x+1} - \sqrt{x}, \qquad g(x) = \frac{1}{\sqrt{x+1} + \sqrt{x}}.$$

   Question: Which is numerically stable and why?
4. **Associativity failure**
   Let $a = 10^{16}$, $b = -10^{16}$, $c = 1$. Compare $(a+b)+c$ and $a+(b+c)$.
   Question: Which step loses information? What does this say about rearranging computations?
5. **Absolute vs relative error near zero**
   Take $x = 10^{-12}$ and $\tilde{x} = 0$. Compute absolute and relative error.
   Question: Which metric is meaningful here, and why?
6. **Mini-design question (Ariane mindset)**
   You store a velocity in float and later cast it to a 16-bit signed integer.
   Question: What must be specified/checked before that cast, and what should happen on overflow?

How Numbers Behave on a Computer

Finite representation, rounding, and machine precision

From $\mathbb{R}$ to Floating-Point Numbers

- Computers do *not* store real numbers
- They store **finite approximations**
- Most real numbers are not representable exactly

Floating-point model

$$x = \pm m \times 2^{e}$$

- m: mantissa (finite precision)
- e: exponent (finite range)

Floating-Point Numbers Are Not Uniformly Spaced

- Floating-point numbers form a **non-uniform grid**
- Dense near zero
- Sparse for large magnitudes

> *"Floating-point arithmetic has roughly constant **relative** precision, not constant **absolute** precision."*

- Explains why $10^{16} + 1 = 10^{16}$
- Explains loss of small increments at large scales

Machine Precision

Definition (conceptual)

Machine epsilon ε is the smallest number such that

$$1 + \varepsilon \neq 1$$

- Measures resolution of the machine near 1
- Around x, smallest distinguishable increment $\approx x\varepsilon$
- Precision depends on scale

"Floating-point precision is relative."

Why Rounding Is Inevitable

- Most decimal numbers have infinite binary expansions
- Example: 0.1_{10} cannot be represented exactly in base 2

- Numbers are rounded when stored
- Results are rounded after every arithmetic operation

“Floating-point arithmetic is exact arithmetic on approximations.”

The Floating-Point Error Model

Standard model

$$\mathrm{fl}(x \circ y) = (x \circ y)(1 + \delta), \qquad |\delta| \leq \varepsilon$$

- $\mathrm{fl}(\cdot)$: result computed by the machine
- δ: small relative error
- Every operation introduces a small error

This is a model, not a bug.

Floating-Point Arithmetic Is Not Algebra

- Associativity fails:
$$(a+b)+c \neq a+(b+c)$$
- Distributivity may fail:
$$a(b+c) \neq ab+ac$$

"Algebraic equivalence does not imply numerical equivalence."

Absolute Error vs Relative Error

Definitions

$$\text{Absolute error: } |x - \tilde{x}|$$

$$\text{Relative error: } \frac{|x - \tilde{x}|}{|x|}$$

- Absolute error depends on scale
- Relative error meaningless near zero

"Errors must always be interpreted in context."

Sources of Numerical Error

1. **Modeling error** – wrong equations
2. **Discretization error** – continuous $\to$ discrete
3. **Round-off error** – floating-point arithmetic

> *"This lecture focuses on round-off error."*

Catastrophic Cancellation

$$\sqrt{x+1} - \sqrt{x}$$

- Subtraction of nearly equal numbers
- Significant digits are lost
- Relative error explodes

Key idea

Cancellation is not a mistake – it is an error amplifier.

Conditioning vs Stability

Conditioning (problem)

How sensitive is the *problem* to small input errors?

Stability (algorithm)

How much additional error does the *algorithm* introduce?

> *"An ill-conditioned problem can be solved stably.*
> *A well-conditioned problem can be solved unstably."*

Numerical Responsibility

- Floating-point errors are unavoidable
- Ignoring them is a design choice
- Numerical thinking is part of mathematical rigor

"A numerical result without error analysis is not a result."

The Rules of Floating-Point Arithmetic

What the machine can and cannot do

Floating-Point Arithmetic Has Limits

- Finite precision
- Finite range
- Not all real numbers are representable

Key idea

Floating-point arithmetic is a *finite model* of $\mathbb{R}$.

Machine Precision (Machine Epsilon)

Definition

The machine epsilon ε is the smallest number such that

$$1 + \varepsilon \neq 1$$

in floating-point arithmetic.

- Measures relative precision near 1
- Smallest resolvable relative change
- Around x, resolution $\approx x\varepsilon$

Rounding

- Most real numbers are rounded when stored
- Every arithmetic operation is rounded

Standard model

$$\mathrm{fl}(z) = z(1+\delta), \qquad |\delta| \leq \varepsilon$$

"Floating-point arithmetic is exact arithmetic on rounded numbers."

Overflow

- Occurs when a result exceeds the largest representable number
- Example: multiplying very large numbers

Consequence

- Result becomes $\pm\infty$
- Computation may silently continue

Overflow is a failure of range, not precision.

Underflow

- Occurs when numbers are too small in magnitude
- They may be flushed to zero

Consequence

- Sudden loss of relative accuracy
- Gradual behavior may become discontinuous

Underflow breaks relative error guarantees.

Important Floating-Point Concepts

- Finite precision $\longrightarrow$ rounding
- Finite range $\longrightarrow$ overflow / underflow
- Machine epsilon $\longrightarrow$ resolution
- Arithmetic is deterministic, not exact

"Understanding the machine is a prerequisite for trusting results."

How Errors Appear and Grow

Finite representation, rounding, and machine precision

Exercise 1 – Floating-Point Surprise

Compute in Python:

- `0.1 + 0.2 - 0.3`
- `(1e16 + 1) - 1e16`

Questions

- Are these results bugs?
- Which property of floating-point arithmetic explains this?

Goal: realize that representation matters.

Exercise 2 – Error Accumulation

Compute:

$$t = \sum_{k=1}^{10^6} 0.1$$

Compare with the exact value 10^5.

Questions

- Is the error random or systematic?
- Why does repeating a tiny error matter?

Goal: understand accumulation over time.

Exercise 3 – Catastrophic Cancellation

Evaluate for $x = 10^k$, $k = 2, 6, 10, 16$:

$$f(x) = \sqrt{x+1} - \sqrt{x}$$

$$g(x) = \frac{1}{\sqrt{x+1} + \sqrt{x}}$$

Questions

- Which formula is more accurate?
- Why do two equivalent formulas behave differently?

Goal: learn to distrust naive algebra.

Exercise 4 – Non-Associativity

Let:

$$a = 10^{16}, \quad b = -10^{16}, \quad c = 1$$

Compute:

$$(a + b) + c \quad \text{and} \quad a + (b + c)$$

Questions

- Which operation loses information?
- Why does the order matter?

Goal: see algebra break at the hardware level.

Exercise 5 – Interpreting Error

Let:

$$x = 10^{-12}, \quad \tilde{x} = 0$$

Compute:

- Absolute error
- Relative error

Questions

- Is this approximation good or bad?
- Which error metric is meaningful here?

Goal: avoid blind use of relative error.

Fixing a Bad Formula

Reformulation as a numerical skill

Mini–Case Study – A Dangerous Formula

We want to compute:

$$h(x) = 1 - \cos(x) \quad \text{for small } x$$

Question

- What happens numerically when $x \to 0$?

Hint: $\cos(x) \approx 1$ for small x.

Mini–Case – What Goes Wrong?

- $\cos(x)$ is close to 1
- Subtraction cancels significant digits
- Result dominated by rounding error

Diagnosis

This is **catastrophic cancellation**.

The problem is well-conditioned, the formula is not.

Mini–Case – A Stable Reformulation

Use the identity:

$$1 - \cos(x) = 2\sin^2\left(\frac{x}{2}\right)$$

- No subtraction of close numbers
- Numerically stable for small x

Key lesson

Mathematically equivalent does not mean numerically equivalent.

Mini–Case – What You Should Learn

- Floating-point errors are predictable
- Bad formulas amplify errors
- Reformulation is often the solution

"Good numerical algorithms respect the arithmetic they run on."

Final Message

- Computers do not compute real numbers
- Floating-point arithmetic has rules
- Ignoring them leads to disasters

> *"Numerical thinking is part of mathematical rigor."*

Next: numerical methods that work *because* of this understanding.

Conditioning, stability, and what accuracy really means

Problems, Algorithms, and Trust

Stability: Formal View (Advanced)

Key question

Does the algorithm solve a *nearby problem* exactly?

- Forward error: how far is the output from the true answer?
- Backward error: what input perturbation makes the output exact?

"Backward error analysis is the gold standard of numerical stability."

Forward and Backward Error

Let $y = f(x)$ and $\tilde{y}$ be the computed result.

Forward error

$$\|\tilde{y} - y\|$$

Backward error

Find Δx such that:

$$\tilde{y} = f(x + \Delta x)$$

- Forward error measures *accuracy*
- Backward error measures *stability*

Conditioning as the Bridge

Key principle

$$\text{Forward error} \;\approx\; (\text{condition number}) \times (\text{backward error})$$

- Conditioning belongs to the *problem*
- Stability belongs to the *algorithm*

"Even a perfectly stable algorithm cannot fix an ill-conditioned problem."

Theorem – Stability of Floating-Point Addition

Theorem

Let $x, y \in \mathbb{R}$ and assume no overflow/underflow. Then the floating-point sum satisfies:

$$\mathrm{fl}(x + y) = (x + y)(1 + \delta), \qquad |\delta| \leq \varepsilon$$

- The computed result is the exact sum of slightly perturbed inputs
- The operation is **backward stable**

Theorem – Stability of Summation (Advanced)

Theorem

Let $S = \sum_{i=1}^{n} x_i$ be computed by sequential floating-point summation. Then:

$$\mathrm{fl}\left(\sum_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} x_i(1+\delta_i), \qquad |\delta_i| \leq \gamma_n$$

where $\gamma_n = \frac{n\varepsilon}{1-n\varepsilon}$.

- Error grows linearly with n
- Order of summation matters

Theorem – Cancellation and Conditioning

Statement

The subtraction $f(x, y) = x - y$ is ill-conditioned when $x \approx y$.

Reason

Relative condition number:

$$\kappa_f \approx \frac{|x| + |y|}{|x - y|}$$

which blows up as $x \to y$.

- Loss of significance is unavoidable
- No algorithm can be stable here

Theorem – Stability of Horner's Method (Preview)

Theorem

Evaluating a polynomial using Horner's method is backward stable.

- Computed value equals exact evaluation of a nearby polynomial
- Coefficients are perturbed by $\mathcal{O}(\varepsilon)$

"This is why Horner's method is universally used."

Advanced Takeaways

- Stability is about **nearby problems**
- Backward error analysis is the right lens
- Conditioning limits achievable accuracy
- Good algorithms respect floating-point arithmetic

"Numerical analysis is applied analysis with responsibility."

Stability: Theorems and Guarantees

Backward error analysis and limits of computation

Theorem – Backward Stability of Basic Arithmetic

Theorem

For $\circ \in \{+, -, \times, /\}$ (assuming no overflow/underflow),

$$\mathrm{fl}(x \circ y) = (x \circ y)(1 + \delta), \qquad |\delta| \leq \varepsilon$$

- Each elementary operation is backward stable
- Computed result equals the exact result of slightly perturbed inputs

"Instability comes from algorithms, not from single operations."

Theorem – Error Growth in Sequential Summation

Theorem

Let $S = \sum_{i=1}^{n} x_i$ be computed sequentially in floating-point arithmetic. Then:

$$\mathrm{fl}(S) = \sum_{i=1}^{n} x_i(1+\delta_i), \qquad |\delta_i| \le \gamma_n, \quad \gamma_n = \frac{n\varepsilon}{1-n\varepsilon}$$

- Error grows linearly with the number of terms
- Order of summation matters

Theorem – Stability Improvement via Kahan Summation

Theorem (informal)

Kahan compensated summation reduces the forward error of summation to $\mathcal{O}(\varepsilon)$ independent of n (under mild assumptions).

- Tracks lost low-order bits
- Dramatically reduces cancellation error

"Stability can often be fixed without changing the problem."

Theorem – Conditioning of Elementary Functions

Statement

Let $f : \mathbb{R} \to \mathbb{R}$ be differentiable. The relative condition number at x is:

$$\kappa_f(x) = \left| \frac{xf'(x)}{f(x)} \right|$$

- Measures intrinsic sensitivity of the problem
- Large κ_f means unavoidable loss of accuracy

Theorem – Limits of Stability

Theorem (principle)

If a problem is ill-conditioned at x, no algorithm can compute $f(x)$ with small relative forward error for all nearby inputs.

- Stability cannot beat conditioning
- Accuracy is fundamentally limited

"Numerical analysis cannot fix bad problems – only bad algorithms."

Theorem – Backward Stability of Gaussian Elimination (Preview)

Theorem (informal)

Gaussian elimination with partial pivoting is backward stable for most matrices:

$$(A + \Delta A)\tilde{x} = b, \qquad \|\Delta A\| \leq \mathcal{O}(\varepsilon)\|A\|$$

- Solution is exact for a nearby system
- Conditioning of A determines final accuracy

Theorem – Polynomial Evaluation and Stability

Theorem

Naive polynomial evaluation is generally unstable. Horner's method is backward stable.

- Same polynomial
- Same arithmetic
- Different stability behavior

"Algorithmic structure matters more than formulas."

Meta-Theorem – What Stability Really Means

Unifying principle

A numerically stable algorithm computes the exact solution of a slightly perturbed problem.

- Backward error small
- Conditioning determines forward error

"Stability is about trust, not perfection."

Advanced Summary

- Floating-point operations are backward stable
- Algorithms may or may not be
- Conditioning limits achievable accuracy
- Reformulation is a mathematical act

> *"Numerical analysis is the mathematics of approximation with guarantees."*