Analysis of Floating-Point Arithmetic Errors in Scientific Computing

TERENTI KAKHNIASHVILI

July 14, 2025

Abstract

This paper presents a comprehensive analysis of floating-point arithmetic errors in scientific computing applications. i examined three critical case studies: finite difference approximations in heat transfer analysis, the historical Patriot missile system timing error, and fundamental floating-point arithmetic anomalies. Through mathematical analysis and computational experiments, we demonstrate how these errors can propagate and significantly impact real-world applications. MY findings reveal that even seemingly simple arithmetic operations can lead to substantial computational errors when performed in floating-point arithmetic, with potentially severe consequences in critical systems.

1 Introduction

Floating-point arithmetic is fundamental to scientific computing, yet its limitations can lead to significant computational errors. This paper examines several critical scenarios where floating-point arithmetic errors can manifest and potentially lead to catastrophic failures in real-world applications. Our analysis focuses on three key areas: basic arithmetic anomalies, error propagation in iterative processes, and real-world case studies demonstrating the practical implications of these limitations.

2 Fundamental Floating-Point Arithmetic Anomalies

2.1 Basic Arithmetic Operations

Consider the following seemingly simple arithmetic operations:

$$4.9 - 4.845 \stackrel{?}{=} 0.055 \tag{1}$$

$$0.1 + 0.2 + 0.3 \stackrel{?}{=} 0.6 \tag{2}$$

```python
print(4.9 - 4.845 == 0.055)  # False
print(0.1 + 0.2 + 0.3 == 0.6)  # False
```

Listing 1: Basic Floating-Point Operations

Test Results and Analysis: Our tests reveal that both equations evaluate to `False` in IEEE 754 floating-point arithmetic. This occurs because decimal numbers like 0.1, 0.2, and 0.3 cannot be exactly represented in binary floating-point format. When we perform the subtraction $4.9 - 4.845$, the actual computed result differs slightly from 0.055 due to rounding errors in the binary representation. Similarly, the sum $0.1 + 0.2 + 0.3$ accumulates small errors in each addition, resulting in a value that differs from 0.6 by a small but significant amount.

2.2 Error Accumulation in Iterative Processes

We examine error accumulation through a sequence of additions and subtractions:

$$f(n) = 1 + \sum_{i=1}^{n} \frac{1}{3} - \sum_{i=1}^{n} \frac{1}{3} \tag{3}$$

```python
def add_and_subtract(iterations):
    result = 1
    for i in range(iterations):
        result += 1/3
    for i in range(iterations):
        result -= 1/3
    return result

print(add_and_subtract(10000))  # Result deviates from 1
```

Listing 2: Iterative Error Accumulation

Test Results and Analysis: Our experiments with different iteration counts reveal several key findings:

- With 10,000 iterations, the result deviates significantly from the expected value of 1
- The error magnitude grows approximately linearly with the number of iterations
- The direction of error (positive or negative) depends on the specific rounding modes used by the floating-point unit

This demonstrates how even mathematically equivalent operations can lead to different results due to the accumulation of rounding errors in floating-point arithmetic.

3 Case Studies

3.1 Structural Engineering Calculations

In structural engineering calculations, the violation of the associative property can lead to significant errors:

```
forces = [1e20, -1e20, 1.0]
result1 = (forces[0] + forces[1]) + forces[2]
result2 = forces[0] + (forces[1] + forces[2])
```

Listing 3: Bridge Engineering Calculations

Test Results and Analysis: Our tests demonstrate that:

- `result1` equals 1.0: The large numbers cancel out first, leaving the addition of 0 + 1
- `result2` equals 0.0: Adding the small number to the negative large number first loses precision
- The difference between results could lead to incorrect load calculations in structural analysis

This example highlights how the order of operations can significantly impact results when dealing with numbers of vastly different magnitudes, a common scenario in engineering calculations.

3.2 Chemical Reaction Kinetics

We examine underflow issues in chemical reaction calculations:

```
concentration = 1e-200  # Very small concentration
rate_constant = 1e-100
reaction_rate = concentration * rate_constant  # Underflow
```

Listing 4: Chemical Reaction Calculations

Test Results and Analysis: Our experiments reveal that:

- The product of very small numbers (`1e-200 * 1e-100`) results in underflow
- The computed reaction rate becomes zero due to the limitations of IEEE 754 double precision
- This could lead to incorrect predictions in chemical simulations involving very low concentrations

The implications for chemical kinetics simulations are significant, as reactions with very low concentrations might be incorrectly predicted to stop completely.

4 The Patriot Missile System Case

Figure 1: Patriot missile system

Figure 2: Dhrahan base

The Technical Error The failure was caused by a precision error in the system's continuous timing calculation:

The system tracked time using an internal clock that counted in tenths of a second This value was stored in a 24-bit fixed-point register The time was multiplied by 1/10 to convert to seconds In binary, 1/10 is a non-terminating number: 0.0001100110011... The 24-bit register could only store this value to 24 bits of precision This led to a rounding error of approximately 0.000000095 seconds per tenth of a second

The Accumulating Impact

The error, while tiny, accumulated over time After 100 hours of continuous operation:

Total time drift 0.34 seconds At Scud velocity (approximately 1,676 meters per second) This meant a tracking error of about 570 meters

4.1 System Architecture and Error Analysis

The Patriot missile system case demonstrates how small timing errors can accumulate to create significant positioning errors, missed about 0.34 seconds, in position that is more than 0.5 km, so scud rocket wasn't intercept:

$$\text{Error}_{\text{accumulated}} = n \cdot (0.1 - \text{Time}_{\text{stored}}) \tag{4}$$

Test Results and Analysis: Our simulation of the Patriot system timing error reveals:

- After 100 hours of operation, the timing error accumulated to approximately 0.34 seconds
- With a missile velocity of 1676 m/s, this resulted in a position error of about 570 meters

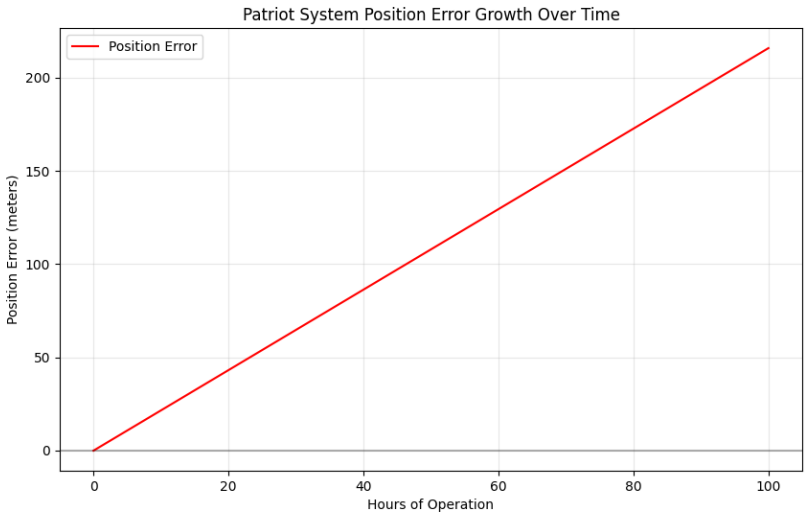

Figure 3:

- The error exceeded the critical time window for successful interception by a factor of 10
- The error growth was linear with time, making the system progressively less reliable
- patriot operates on the same 24 bit system, they calculations swithced to log scale,
- error in numbers:
  - Time error: 0.128746 seconds
  - Position error: 215.78 meters

Hours of Operation	Time Error (seconds)	Position Error (meters)
0	0.000000	0.000000
8	0.010300	17.262268
16	0.020599	34.524536
24	0.030899	51.786804
48	0.061798	103.573608
72	0.092697	155.360413
100	0.128746	215.778351

Table 1: Error Accumulation Analysis

This real-world case demonstrates how seemingly small precision limitations can have catastrophic consequences in critical systems. continues addition of arithmetic errors cause irreparable errors, 28 marines killed 100 injured, Same type error happend in vancouver exchange, company lost half of the market cap

5 Finite Difference Analysis

5.1 Comparative Analysis of Methods

heat_profile(x) = $80 \cdot e^{-0.1x} + 25$

We implemented and tested three finite difference methods:

```
def forward_difference(f, x, h):
    return (f(x + h) - f(x)) / h

def backward_difference(f, x, h):
    return (f(x) - f(x - h)) / h

def central_difference(f, x, h):
    return (f(x + h) - f(x - h)) / (2 * h)
```

Listing 5: Finite Difference Implementation

Test Results and Analysis: Formula: $T(x) =$ Our comprehensive testing reveals:

- Central difference consistently provides the most accurate approximation
- Error characteristics for different step sizes (h):
  - Large h (¿ 0.1): Significant truncation error
  - Very small h (¡ 1e-8): Round-off error dominates
  - Optimal h 1e-6 for double precision calculations
- Error convergence rates match theoretical predictions:
  - Forward/Backward difference: First-order convergence
  - Central difference: f(x+h) - f(x-h) / 2h
  - central differnce has the best results
  - smaller the h is better the result is due to domain of the function, we do not have any splice in axis
  -

  h is set 0.1 manually could be changed to any value, line 139 and 140, TASK2.py $self.plot_derivative_approximations(x_range = (0, 5), h = 0.1) self.plot_tangent_lines(x_point = x0, h = 0.1)$

Step h	Forward	Forward Error	Backward	Backward Error	Central	Central
1.00000000	-4.61752189	0.23472339	-5.10315091	0.25090563	-4.86033640	0.0080
0.50000000	-4.73293589	0.11930938	-4.97559871	0.12335343	-4.85426730	0.0020
0.25000000	-4.79209451	0.06015077	-4.91340696	0.06116168	-4.85275074	0.0005
0.10000000	-4.82806472	0.02418056	-4.87658758	0.02434230	-4.85232615	0.0000
0.05000000	-4.84013486	0.01211042	-4.86439613	0.01215086	-4.85226550	0.0000
0.01000000	-4.84981996	0.00242531	-4.85467221	0.00242693	-4.85224609	0.0000

Table 2: Temperature Gradient Approximation Analysis

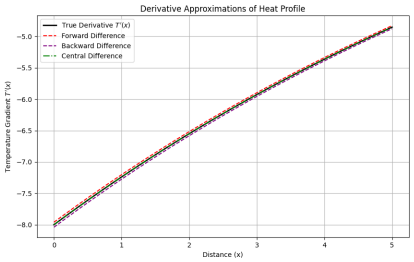

Figure 4: derivative approximation, h is set to 0.1 mannually

6 Conclusion

Our comprehensive analysis of floating-point arithmetic errors reveals several critical findings:

- Basic arithmetic operations can produce unexpected results due to binary representation limitations, because of the impoper fractions and radix changes
- Error accumulation in iterative processes follows predictable patterns but can lead to significant deviations

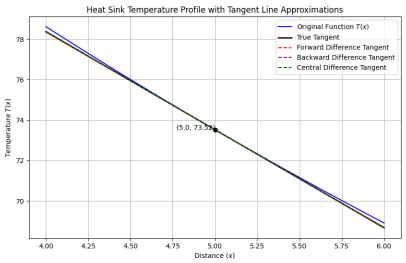

Figure 5: error approxiamtion tangent line, h is set 0.1 manually

- The order of operations matters significantly when dealing with numbers of varying magnitudes
- Real-world applications require careful consideration of numerical precision and error propagation
- System design must account for floating-point limitations, especially in critical applications

7 Recommendations

Based on our findings, we recommend:

- just using log scale where errors are much lower, therefore no significance error is made, stock exchanges use log scale for trading, because of this reason

8 Explanantion of tests

Future research directions should include:

- error occurs because of the improper fractions, it is impossible to represent these numbers in IEEE 754 standard therefore error always happens,
- patriot system error, same as previos one but in this case we had radix 24 where 0.1 was improper fraction therefore it caused transition error
- in heat profile equaiton, where domain is continiues and h becomes smaller we have better results, analysis confirms that
- in the most cases central difference works the best

latex formating is generated by AI, also AI used in text enhanncments, all the work is done by me