

# Numerical Matrix Analysis

## Notes #10 — Conditioning and Stability

### Floating Point Arithmetic / Stability

Peter Blomgren  
(blomgren@sdsu.edu)

Department of Mathematics and Statistics  
Dynamical Systems Group  
Computational Sciences Research Center  
San Diego State University  
San Diego, CA 92182-7720  
<http://terminus.sdsu.edu/>

Spring 2024  
(Revised: January 18, 2024)

## Outline

- 1 Student Learning Targets, and Objectives
  - SLOs: Floating Point Arithmetic & Stability
- 2 Finite Precision
  - IEEE Binary Floating Point (from Math 541<sup>R.I.P.</sup>)
  - Non-representable Values — a Source of Errors
- 3 Floating Point Arithmetic
  - “Theorem” and Notation
  - Fundamental Axiom of Floating Point Arithmetic
  - Example
- 4 Stability
  - Introduction: What is the “correct” answer?
  - Accuracy — Absolute and Relative Error
  - Stability, and Backward Stability

## Student Learning Targets, and Objectives

### Target Floating Point Arithmetic

Objective Know how to express a floating point number using the IEEE-754-1985 (and successor) standard

Objective Know how to express the limits of the floating point environment using  $\epsilon_{\text{mach}}$ .

### Target Stability

Objective Know the definitions of absolute and relative error.

Objective Know the formal and informal definitions of stable and backward stable algorithms.

## Finite Precision

A 64-bit real number, double

The **Binary Floating Point Arithmetic Standard 754-1985** (IEEE — The Institute for Electrical and Electronics Engineers) standard specified the following layout for a 64-bit real number:

$$s \ c_{10} \ c_9 \ \dots \ c_1 \ c_0 \ m_{51} \ m_{50} \ \dots \ m_1 \ m_0$$

Where

Symbol	Bits	Description
$s$	1	The sign bit — 0=positive, 1=negative
$c$	11	The characteristic (exponent)
$m$	52	The mantissa

$$r = (-1)^s 2^{c-1023} (1 + f), \quad c = \sum_{n=0}^{10} c_n 2^n, \quad f = \sum_{k=0}^{51} \frac{m_k}{2^{52-k}}$$

## IEEE-754-1985 Special Signals

In order to be able to represent **zero**,  $\pm\infty$ , and **NaN** (not-a-number), the following special signals are defined in the IEEE-754-1985 standard:

Type	S (1 bit)	C (11 bits)	M (52 bits)
signaling NaN	u	2047 (max)	.0uuuuu—u (*)
quiet NaN	u	2047 (max)	.1uuuuu—u
negative infinity	1	2047 (max)	.000000—0
positive infinity	0	2047 (max)	.000000—0
negative zero	1	0	.000000—0
positive zero	0	0	.000000—0

(\*) with at least one 1 bit.

From <http://www.freesoft.org/CIE/RFC/1832/32.htm>

If you think IEEE-754-1985 is too “simple.” There are some interesting additions in the IEEE 754-2008 revision; e.g. fused-multiply-add (*fma*) operations.

Some environments (e.g. AVX/AVX2/AVX-512 extensions) combine multiple *fma* operations into a single step, e.g. performing a four-element dot-product on two 128-bit SIMD registers  $a_0 \times b_0 + a_1 \times b_1 + a_2 \times b_2 + a_3 \times b_3$  with single cycle throughput.





## That's Quite a Range!

In summary, we can represent

$$\{ \pm 0, \pm 1.113 \times 10^{-308}, \pm 1.798 \times 10^{308}, \pm \infty, \text{NaN} \}$$

and a whole bunch of numbers in

$$(-1.798 \times 10^{308}, -1.113 \times 10^{-308}) \cup (1.113 \times 10^{-308}, 1.798 \times 10^{308})$$

**Bottom line:** Over- or under-flowing is usually not a problem in IEEE floating point arithmetic.

The problem in **scientific computing** is what we **cannot** represent.



Fun with Matlab...

...Integers

---


$$\begin{aligned} (2^{53} + 2) - 2^{53} &= 2 \\ (2^{53} + 2) - (2^{53} + 1) &= 1 \\ (2^{53} + 1) - 2^{53} &= 0 \\ 2^{53} - (2^{53} - 1) &= 1 \end{aligned}$$


---

---


$$\begin{aligned} \text{realmax} &= 1.7977 \cdot 10^{308} & \text{realmin} &= 2.2251 \cdot 10^{-308} \\ \text{eps} &= 2.2204 \cdot 10^{-16} \end{aligned}$$


---

The smallest not-exactly-representable integer is  
 $(2^{53} + 1) = 9,007,199,254,740,993.$







## The Relative Gap

It makes more sense to factor the exponent out of the discussion and talk about the relative gap:

Exponent	Gap	Relative Gap (Gap/Exponent)
$2^{-1023}$	$2^{-1075}$	$2^{-52} \approx 2.22 \times 10^{-16}$
$2^1$	$2^{-51}$	$2^{-52}$
$2^{1023}$	$2^{971}$	$2^{-52}$

Any difference between numbers smaller than the local gap is not representable, e.g. any number in the interval

$$\left[ 3.0, 3.0 + \frac{1}{2^{51}} \right)$$

is represented by the value 3.0.

## The Floating Point “Theorem”

$\epsilon_{\text{mach}}$

“Theorem”

Floating point “numbers” represent intervals!

Notation

We let  $\text{fl}(x)$  denote the floating point representation of  $x \in \mathbb{R}$ .

Let the symbols  $\oplus$ ,  $\ominus$ ,  $\otimes$ , and  $\oslash$  denote the floating-point operations: addition, subtraction, multiplication, and division.

The Floating Point  $\varepsilon_{\text{mach}}$ 

The relative gap defines  $\varepsilon_{\text{mach}}$ ; and

$\forall x \in \mathbb{R}$ , there exists  $\varepsilon$  with  $|\varepsilon| \leq \varepsilon_{\text{mach}}$ , such that  $\text{fl}(x) = x(1 + \varepsilon)$ .

In 64-bit floating point arithmetic  $\varepsilon_{\text{mach}} \approx 2.22 \times 10^{-16}$ .

In matlab, `eps` returns this value.

In Python, `print(np.finfo(float).eps)`

In C, `#include <float.h>` to define the value of `__DBL_EPSILON__`

## Floating Point Arithmetic

 $\epsilon_{\text{mach}}$ 

All floating-point operations are performed up to some precision, *i.e.*

$$\begin{aligned}x \oplus y &= \text{fl}(x + y), & x \ominus y &= \text{fl}(x - y), \\x \otimes y &= \text{fl}(x * y), & x \oslash y &= \text{fl}(x/y)\end{aligned}$$

This paired with our definition of  $\epsilon_{\text{mach}}$  gives us

Axiom (The Fundamental Axiom of Floating Point Arithmetic)

For an  $n$ -bit floating point environment —

For all  $x, y \in \mathbb{F}_{64}$  (where  $\mathbb{F}_{64}$  is the set of 64-bit floating point numbers), there exists  $\epsilon$  with  $|\epsilon| \leq \epsilon_{\text{mach}}(\mathbb{F}_{64})$ , such that

$$\begin{aligned}x \oplus y &= (x + y)(1 + \epsilon), & x \ominus y &= (x - y)(1 + \epsilon), \\x \otimes y &= (x * y)(1 + \epsilon), & x \oslash y &= (x/y)(1 + \epsilon)\end{aligned}$$

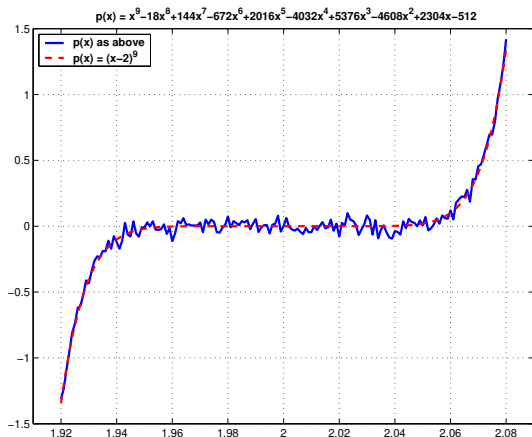
That is **every operation of floating point arithmetic is exact up to a relative error of size at most  $\epsilon_{\text{mach}}$** .



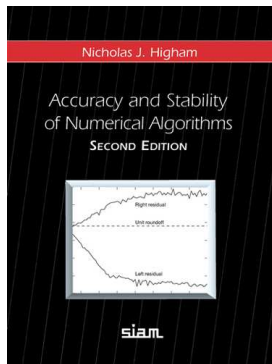
## Example: Floating Point Error

Scaled by  $10^{10}$ Consider the following polynomial on the interval  $[1.92, 2.08]$ :

$$\begin{aligned} p(x) &= (x - 2)^9 \\ &= x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512 \end{aligned}$$



# Stability



680 pages of details...

## Stability: Introduction

1 of 3

With the knowledge that “**(floating point) errors happen,**” we have to re-define the concept of the “**right answer.**”

Previously, in the context of **conditioning** we defined a mathematical problem as a map

$$f : X \mapsto Y$$

where  $X \subseteq \mathbb{C}^n$  is the set of data (input), and  $Y \subseteq \mathbb{C}^m$  is the set of solutions.

## Stability: Introduction

2 of 3

We now define an implementation of an **algorithm** — on a floating-point device, where  $\mathbb{F}$  satisfies the fundamental axiom of floating point arithmetic — as another map

$$\tilde{f} : X \mapsto Y$$

i.e.  $\tilde{f}(\vec{x}) \in Y$  is a numerical solution of the problem.

Wiki-History: Pentium FDIV bug ( $\approx$  1994)

The Pentium FDIV bug was a bug in Intel's original Pentium FPU. Certain FP division operations performed with these processors would produce incorrect results. According to Intel, there were a few missing entries in the lookup table used by the divide operation algorithm.

Although encountering the flaw was extremely rare in practice (*Byte Magazine* estimated that 1 in 9 billion FP divides with random parameters would produce inaccurate results), both the flaw and Intel's initial handling of the matter were heavily criticized. Intel ultimately recalled the defective processors.

## Stability: Introduction

3 of 3

The task at hand is to make **useful** statements about  $\tilde{f}(\vec{x})$ .

Even though  $\tilde{f}(\vec{x})$  is affected by many factors — roundoff errors, convergence tolerances, competing processes on the computer\*, etc; we will be able to make (maybe surprisingly) clear statements about  $\tilde{f}(\vec{x})$ .

- \* Note that depending on the memory model, the previous state of a memory location *may* affect the result in e.g. the case of cancellation errors: If we subtract two 16-digit numbers with 13 common leading digits, we are left with 3 digits of valid information. We tend to view the remaining 13 digits as “random.” But really, there is nothing random about what happens inside the computer (we hope!) — the “randomness” will depend on what happened previously...

## Accuracy

The **absolute error** of a computation is

$$\|\tilde{f}(\vec{x}) - f(\vec{x})\|$$

and the **relative error** is

$$\frac{\|\tilde{f}(\vec{x}) - f(\vec{x})\|}{\|f(\vec{x})\|}$$

this latter quantity will be our standard measure of error.

If  $\tilde{f}$  is a good algorithm, we expect the relative error to be small, of the order  $\varepsilon_{\text{mach}}$ . We say that  $\tilde{f}$  is **accurate** if  $\forall \vec{x} \in X$

$$\frac{\|\tilde{f}(\vec{x}) - f(\vec{x})\|}{\|f(\vec{x})\|} = \mathcal{O}(\varepsilon_{\text{mach}})$$

Interpretation:  $\mathcal{O}(\varepsilon_{\text{mach}})$

Since all floating point errors are functions of  $\varepsilon_{\text{mach}}$  (the relative error in each operation is bounded by  $\varepsilon_{\text{mach}}$ ), the relative error of the algorithm must be a function of  $\varepsilon_{\text{mach}}$ :

$$\frac{\|\tilde{f}(\vec{x}) - f(\vec{x})\|}{\|f(\vec{x})\|} = e(\varepsilon_{\text{mach}})$$

The statement

$$e(\varepsilon_{\text{mach}}) = \mathcal{O}(\varepsilon_{\text{mach}})$$

means that  $\exists C \in \mathbb{R}^+$  such that

$$e(\varepsilon_{\text{mach}}) \leq C\varepsilon_{\text{mach}}, \quad \text{as } \varepsilon_{\text{mach}} \searrow 0$$

In practice  $\varepsilon_{\text{mach}}$  is fixed; the notation means that **if** we were to decrease  $\varepsilon_{\text{mach}}$ , **then** our error would decrease at least proportionally to  $\varepsilon_{\text{mach}}$ .

## Stability

If the **problem**  $f : X \mapsto Y$  is ill-conditioned, then the accuracy goal

$$\frac{\|\tilde{f}(\vec{x}) - f(\vec{x})\|}{\|f(\vec{x})\|} = \mathcal{O}(\varepsilon_{\text{mach}})$$

may be unreasonably ambitious. Instead we aim for **stability**.

We say that  $\tilde{f}$  is a **stable algorithm** if  $\forall \vec{x} \in X$

$$\frac{\|\tilde{f}(\vec{x}) - f(\tilde{\vec{x}})\|}{\|f(\tilde{\vec{x}})\|} = \mathcal{O}(\varepsilon_{\text{mach}})$$

for some  $\tilde{\vec{x}}$  with

$$\frac{\|\tilde{\vec{x}} - \vec{x}\|}{\|\vec{x}\|} = \mathcal{O}(\varepsilon_{\text{mach}})$$

**“A stable algorithm gives approximately the right answer, to approximately the right question.”**



## Backward Stability

For many algorithms we can tighten this somewhat vague concept of stability.

An algorithm  $\tilde{f}$  is **backward stable** if  $\forall \vec{x} \in X$

$$\tilde{f}(\vec{x}) = f(\tilde{\vec{x}})$$

for some  $\tilde{\vec{x}}$  with

$$\frac{\|\tilde{\vec{x}} - \vec{x}\|}{\|\vec{x}\|} = \mathcal{O}(\varepsilon_{\text{mach}})$$

**“A backward stable algorithm gives exactly the right answer, to approximately the right question.”**

**Next:** Examples of stable and unstable algorithms;  
Stability of Householder triangularization.