Numerical Matrix Analysis
Lecture Notes #10
— Conditioning and Stability —
Floating Point Arithmetic / Stability

Peter Blomgren,
⟨blomgren.peter@gmail.com⟩

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 2016
Outline

1. Finite Precision
   - IEEE Binary Floating Point (from Math 541)
   - Non-representable Values — a Source of Errors

2. Floating Point Arithmetic
   - “Theorem” and Notation
   - Fundamental Axiom of Floating Point Arithmetic
   - Example

3. Stability
   - Introduction: What is the “correct” answer?
   - Accuracy — Absolute and Relative Error
   - Stability, and Backward Stability
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:

\[ s \ c_{10} \ c_9 \ldots \ c_1 \ c_0 \ m_{51} \ m_{50} \ldots \ m_1 \ m_0 \]

Where

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s)</td>
<td>1</td>
<td>The sign bit — 0=positive, 1=negative</td>
</tr>
<tr>
<td>(c)</td>
<td>11</td>
<td>The characteristic (exponent)</td>
</tr>
<tr>
<td>(m)</td>
<td>52</td>
<td>The mantissa</td>
</tr>
</tbody>
</table>

\[
r = (-1)^s 2^{c - 1023} (1 + f), \quad c = \sum_{k=0}^{10} c_n 2^n, \quad f = \sum_{k=0}^{51} \frac{m_k}{2^{52-k}}
\]
IEEE-754-1985 Special Signals

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

<table>
<thead>
<tr>
<th>Type</th>
<th>S (1 bit)</th>
<th>C (11 bits)</th>
<th>M (52 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>signaling NaN</td>
<td>u</td>
<td>2047 (max)</td>
<td>.0uuuuu—u (*)</td>
</tr>
<tr>
<td>quiet NaN</td>
<td>u</td>
<td>2047 (max)</td>
<td>.1uuuuu—u</td>
</tr>
<tr>
<td>negative infinity</td>
<td>1</td>
<td>2047 (max)</td>
<td>.000000—0</td>
</tr>
<tr>
<td>positive infinity</td>
<td>0</td>
<td>2047 (max)</td>
<td>.000000—0</td>
</tr>
<tr>
<td>negative zero</td>
<td>1</td>
<td>0</td>
<td>.000000—0</td>
</tr>
<tr>
<td>positive zero</td>
<td>0</td>
<td>0</td>
<td>.000000—0</td>
</tr>
</tbody>
</table>

(*) with at least one 1 bit.

From [http://www.freesoft.org/CIE/RFC/1832/32.htm](http://www.freesoft.org/CIE/RFC/1832/32.htm)
Examples: Finite Precision

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

**Example #1**

\[
0, 10000000000, 000000000000000000000000000000000000000000000000000
\]

\[
r_1 = (-1)^0 \cdot 2^{2^{10} - 1023} \cdot \left(1 + \frac{1}{2}\right) = 1 \cdot 2^1 \cdot \frac{3}{2} = 3.0
\]

**Example #2 (The Smallest Positive Real Number)**

\[
0, 00000000000, 0000000000000000000000000000000000000000000000000001
\]

\[
r_2 = (-1)^0 \cdot 2^{0 - 1023} \cdot (1 + 2^{-52}) \approx 1.113 \times 10^{-308}
\]
Examples: Finite Precision

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

Example #3 (The Largest Positive Real Number)

0, 11111111110, 111111111111111111111111111111111111111111111111111

\[
\begin{align*}
r_3 &= (-1)^0 \cdot 2^{1023} \cdot \left(1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^{51}} + \frac{1}{2^{52}}\right) \\
&= 2^{1023} \cdot \left(2 - \frac{1}{2^{52}}\right) \approx 1.798 \times 10^{308}
\end{align*}
\]
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...

\[
\begin{align*}
(2^{53} + 2) - 2^{53} &= 2 \\
(2^{53} + 2) - (2^{53} + 1) &= 2 \\
(2^{53} + 1) - 2^{53} &= 0 \\
2^{53} - (2^{53} - 1) &= 1
\end{align*}
\]

realmax = $1.7977 \cdot 10^{308}$  
realmin = $2.2251 \cdot 10^{-308}$  
eps = $2.2204 \cdot 10^{-16}$

The smallest not-exactly-representable integer is 
\[(2^{53} + 1) = 9,007,199,254,740,993.\]
There are gaps in the floating-point representation!

Given the representation

\[ 0,0000000000000000000000000000000001 \]

for the value \( v_1 = 2^{-1023}(1 + 2^{-52}) \),

the next larger floating-point value is

\[ 0,0000000000000000000000000000000010 \]

\( i.e. \) the value \( v_2 = 2^{-1023}(1 + 2^{-51}) \)

The difference between these two values is \( 2^{-1023} \cdot 2^{-52} = 2^{-1075} \) (\( \sim 10^{-324} \)).

Any number in the interval \((v_1, v_2)\) is not representable!
A gap of $2^{-1075}$ doesn’t seem too bad...

However, the size of the gap depend on the value itself...

Consider $r = 3.0$

- $0, 100000000000000000000000000000000000000000000000000$
- $0, 1000000000100000000000000000000000000000000000000000000001$

Here, the difference is $2 \cdot 2^{-52} = 2^{-51}$ ($\sim 10^{-16}$).

In general, in the interval $[2^n, 2^{n+1}]$ the gap is $2^{n-52}$.
At the other extreme, the difference between

\[
0, 11111111110, 111111111111111111111111111111111111111111111111110
\]

and the next value

\[
0, 11111111110, 111111111111111111111111111111111111111111111111111
\]

is \(2^{1023} \cdot 2^{-52} = 2^{971} \approx 1.996 \cdot 10^{292}\).

That’s a fairly significant gap!!! (A number large enough to comfortably count all the particles in the universe...)

See, e.g. http://home.earthlink.net/~mrob/pub/math/numbers-10.html
The Relative Gap

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

<table>
<thead>
<tr>
<th>Exponent</th>
<th>Gap</th>
<th>Relative Gap (Gap/Exponent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2^{-1023}</td>
<td>2^{-1075}</td>
<td>2^{-52} = 2.22 \times 10^{-16}</td>
</tr>
<tr>
<td>2^{1}</td>
<td>2^{-51}</td>
<td>2^{-52}</td>
</tr>
<tr>
<td>2^{1023}</td>
<td>2^{971}</td>
<td>2^{-52}</td>
</tr>
</tbody>
</table>

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”

“Theorem”

Floating point “numbers” represent intervals!

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

The relative gap defines \( \epsilon_{\text{mach}} \) —

\[
\forall x \in \mathbb{R}, \text{ there exists } \epsilon \text{ with } |\epsilon| \leq \epsilon_{\text{mach}}, \\
\text{ such that } \text{fl}(x) = x(1 + \epsilon).
\]

In 64-bit floating point arithmetic \( \epsilon_{\text{mach}} \approx 2.22 \times 10^{-16} \). In matlab, the command `eps` returns this value.

Let the symbols \( \oplus, \ominus, \otimes, \oslash \) denote the floating-point operations: addition, subtraction, multiplication, and division.
All floating-point operations are performed up to some precision, i.e.

\[ x \oplus y = \text{fl}(x + y), \quad x \ominus y = \text{fl}(x - y), \]
\[ x \otimes y = \text{fl}(x \ast y), \quad x \oslash y = \text{fl}(x/y) \]

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

**Axiom (The Fundamental Axiom of Floating Point Arithmetic)**

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

\[ x \oplus y = (x + y)(1 + \epsilon), \quad x \ominus y = (x - y)(1 + \epsilon), \]
\[ x \otimes y = (x \ast y)(1 + \epsilon), \quad x \oslash y = (x/y)(1 + \epsilon) \]

That is every operation of floating point arithmetic is exact up to a relative error of size at most \( \epsilon_{\text{mach}} \).
Consider the following polynomial on the interval [1.92, 2.08]:

\[ p(x) = (x - 2)^9 = x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512 \]
Stability
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

\[ f : X \rightarrow Y \]

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

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

i.e. $\tilde{f}(\bar{x}) \in Y$ 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.
The task at hand is to make useful statements about \( \tilde{f}(\bar{x}) \).

Even though \( \tilde{f}(\bar{x}) \) 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 \( \tilde{f}(\bar{x}) \).

\(^*\) 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

\[ \| \tilde{f}(\bar{x}) - f(\bar{x}) \| \]

and the **relative error** is

\[ \frac{\| \tilde{f}(\bar{x}) - f(\bar{x}) \|}{\| f(\bar{x}) \|} \]

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 \( \epsilon_{\text{mach}} \). We say that \( \tilde{f} \) is **accurate** if \( \forall \bar{x} \in X \)

\[ \frac{\| \tilde{f}(\bar{x}) - f(\bar{x}) \|}{\| f(\bar{x}) \|} = O(\epsilon_{\text{mach}}) \]
Since all floating point errors are functions of $\epsilon_{\text{mach}}$ (the relative error in each operation is bounded by $\epsilon_{\text{mach}}$), the relative error of the algorithm must be a function of $\epsilon_{\text{mach}}$:

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

The statement

$$e(\epsilon_{\text{mach}}) = O(\epsilon_{\text{mach}})$$

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

$$e(\epsilon_{\text{mach}}) \leq C\epsilon_{\text{mach}}, \quad \text{as} \quad \epsilon_{\text{mach}} \downarrow 0$$

In practice $\epsilon_{\text{mach}}$ is fixed, and the notation means that if we were to decrease $\epsilon_{\text{mach}}$, then our error would decrease at least proportionally to $\epsilon_{\text{mach}}$. 
If the problem \( f : X \rightarrow Y \) is ill-conditioned, then the accuracy goal

\[
\frac{\| \tilde{f}(\tilde{x}) - f(\tilde{x}) \|}{\| f(\tilde{x}) \|} = O(\epsilon_{\text{mach}})
\]

may be unreasonably ambitious.

Instead we aim for stability. We say that \( \tilde{f} \) is a stable algorithm if \( \forall \tilde{x} \in X \)

\[
\frac{\| \tilde{f}(\tilde{x}) - f(\tilde{x}) \|}{\| f(\tilde{x}) \|} = O(\epsilon_{\text{mach}})
\]

for some \( \tilde{x} \) with

\[
\frac{\| \tilde{x} - x \|}{\| x \|} = O(\epsilon_{\text{mach}})
\]

“A stable algorithm gives approximately the right answer, to approximately the right question.”
For many algorithms we can tighten this somewhat vague concept of stability.

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

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

for some $\tilde{x}$ with

$$\frac{\|\tilde{x} - \bar{x}\|}{\|\bar{x}\|} = O(\epsilon_{\text{mach}})$$

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

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