

Lecture 8: Floating Point Number Systems

1.5 Floating point arithmetic.

To this point we have mainly considered pure algorithms, with only tangential concern for how these algorithms behave when executed on a computer. Yet in our comparison of two implementations of the skinny QR factorization (via classical and modified Gram–Schmidt orthogonalization), we observed that mathematically equivalent methods can yield significantly different results when implemented on a computer. Now we pause to consider the computer arithmetic that gives rise to such behavior.

First, it is important to note two important styles of computation.

- *Symbolic computing*, practiced by Mathematica and Maple, applies rules of algebra and calculus to symbolic expressions that represent variables. When numerical values are used, they are stored exactly as rational numbers, radicals, and special constants such as π . As these quantities are manipulated and combined, the size of the resulting numerators and denominators can grow rapidly: this increases memory requirements and degrades performance. Such calculations can be slow and produce results that are difficult to simplify, but they are exact.
- *Numerical computing*, practiced by MATLAB, converts all input numbers into a *floating point number system* internal to the computer. Because the floating point system contains only finitely many numbers, most input quantities must be rounded to the nearest represented number. This system is not *closed* under basic arithmetic operations, e.g., the sum of two floating point numbers need not be a floating point number. Thus, the intermediate quantities computed by algorithms are also rounded to the nearest floating point number. Floating point computations, typically performed in hardware, are typically much more efficient than symbolic ones. The key question is, how does the accumulation of rounding errors pollute the final answer an algorithm produces?

The above software descriptions are oversimplified. For example, Mathematica supports numerical computing (to a user-specified precision), while MATLAB's Symbolic Toolbox provides some symbolic operations and an interface to the Maple symbolic computing system. A notable third style of computing, *interval arithmetic*, resembles the numerical approach, but now upper and lower bounds on all rounded quantities are stored. Thus, interval algorithms can guarantee a bound on the desired solution, while still enjoying the many benefits of numerical computing. The primary challenge comes in designing algorithms that yield small intervals.

While symbolic computing and interval arithmetic have their places, the great bulk of engineering computations are performed in floating point arithmetic, and that shall be our focus.

A floating point number system is described in terms of four parameters:

- β , the base;
- t , the precision;
- e_{\min} , the minimum exponent;
- e_{\max} , the maximum exponent.

With these parameters, the floating point system is defined as follows (m and e must be integers):

$$\mathbb{F} = \{\pm m\beta^{e-t} : \text{either } \beta^{t-1} \leq m < \beta^t \text{ and } e_{\min} \leq e \leq e_{\max}, \text{ or } 0 \leq m < \beta^{t-1} \text{ and } e = e_{\min}\}.$$

The numbers $\pm m\beta^{e-t}$ for $\beta^{t-1} \leq m < \beta^t$ and $e_{\min} \leq e \leq e_{\max}$ (and zero, too) are called *normal numbers*, for a reason that shall become apparent when we see how these numbers are stored on a computer. The numbers $\pm m\beta^{e-t}$ for $0 \leq m < \beta^{t-1}$ and $e = e_{\min}$ are *subnormal numbers*.

All modern computers use the base $\beta = 2$ (i.e., binary numbers), where each binary digit (*bit*) takes a value 0 or 1. though some calculators reportedly use $\beta = 10$. The parameter t affects how densely packed the floating point numbers are, while e_{\min} and e_{\max} control the largest and smallest representable numbers.

Example. It may be difficult to visualize this definition of \mathbb{F} , but working out a small example will flesh these numbers out.[†] Take $\beta = 2$, $t = 3$, $e_{\min} = -1$, $e_{\max} = 2$. Recall the definition

$$\mathbb{F} = \{\pm m\beta^{e-t} : \text{either } \beta^{t-1} \leq m < \beta^t \text{ and } e_{\min} \leq e \leq e_{\max}, \text{ or } 0 \leq m < \beta^{t-1} \text{ and } e = e_{\min}\}.$$

For simplicity, we will only list the nonnegative floating point numbers. First, we will enumerate all the normal numbers by stepping e from e_{\min} to e_{\max} . For these fixed values of e , m can take any value between $\beta^{t-1} = 2^{3-1} = 4$ and $\beta^t - 1 = 2^3 - 1 = 7$.

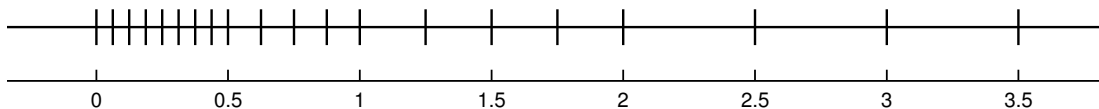
$e = -1, \beta^{e-t} = 2^{-4} = 1/16$	$e = 0, \beta^{e-t} = 2^{-3} = 1/8$																				
<table style="border-collapse: collapse; margin: auto;"> <thead> <tr> <th style="border-right: 1px solid black; padding: 5px;">m</th> <th style="padding: 5px;">$m\beta^{e-t}$</th> </tr> </thead> <tbody> <tr><td style="border-right: 1px solid black; padding: 5px;">4</td><td style="padding: 5px;">$4/16 = 1/4$</td></tr> <tr><td style="border-right: 1px solid black; padding: 5px;">5</td><td style="padding: 5px;">$5/16$</td></tr> <tr><td style="border-right: 1px solid black; padding: 5px;">6</td><td style="padding: 5px;">$6/16 = 3/8$</td></tr> <tr><td style="border-right: 1px solid black; padding: 5px;">7</td><td style="padding: 5px;">$7/16$</td></tr> </tbody> </table>	m	$m\beta^{e-t}$	4	$4/16 = 1/4$	5	$5/16$	6	$6/16 = 3/8$	7	$7/16$	<table style="border-collapse: collapse; margin: auto;"> <thead> <tr> <th style="border-right: 1px solid black; padding: 5px;">m</th> <th style="padding: 5px;">$m\beta^{e-t}$</th> </tr> </thead> <tbody> <tr><td style="border-right: 1px solid black; padding: 5px;">4</td><td style="padding: 5px;">$4/8 = 1/2$</td></tr> <tr><td style="border-right: 1px solid black; padding: 5px;">5</td><td style="padding: 5px;">$5/8$</td></tr> <tr><td style="border-right: 1px solid black; padding: 5px;">6</td><td style="padding: 5px;">$6/8 = 3/4$</td></tr> <tr><td style="border-right: 1px solid black; padding: 5px;">7</td><td style="padding: 5px;">$7/8$</td></tr> </tbody> </table>	m	$m\beta^{e-t}$	4	$4/8 = 1/2$	5	$5/8$	6	$6/8 = 3/4$	7	$7/8$
m	$m\beta^{e-t}$																				
4	$4/16 = 1/4$																				
5	$5/16$																				
6	$6/16 = 3/8$																				
7	$7/16$																				
m	$m\beta^{e-t}$																				
4	$4/8 = 1/2$																				
5	$5/8$																				
6	$6/8 = 3/4$																				
7	$7/8$																				
$e = 1, \beta^{e-t} = 2^{-2} = 1/4$	$e = 2, \beta^{e-t} = 2^{-1} = 1/2$																				
<table style="border-collapse: collapse; margin: auto;"> <thead> <tr> <th style="border-right: 1px solid black; padding: 5px;">m</th> <th style="padding: 5px;">$m\beta^{e-t}$</th> </tr> </thead> <tbody> <tr><td style="border-right: 1px solid black; padding: 5px;">4</td><td style="padding: 5px;">$4/4 = 1$</td></tr> <tr><td style="border-right: 1px solid black; padding: 5px;">5</td><td style="padding: 5px;">$5/4$</td></tr> <tr><td style="border-right: 1px solid black; padding: 5px;">6</td><td style="padding: 5px;">$6/4 = 3/2$</td></tr> <tr><td style="border-right: 1px solid black; padding: 5px;">7</td><td style="padding: 5px;">$7/4$</td></tr> </tbody> </table>	m	$m\beta^{e-t}$	4	$4/4 = 1$	5	$5/4$	6	$6/4 = 3/2$	7	$7/4$	<table style="border-collapse: collapse; margin: auto;"> <thead> <tr> <th style="border-right: 1px solid black; padding: 5px;">m</th> <th style="padding: 5px;">$m\beta^{e-t}$</th> </tr> </thead> <tbody> <tr><td style="border-right: 1px solid black; padding: 5px;">4</td><td style="padding: 5px;">$4/2 = 2$</td></tr> <tr><td style="border-right: 1px solid black; padding: 5px;">5</td><td style="padding: 5px;">$5/2$</td></tr> <tr><td style="border-right: 1px solid black; padding: 5px;">6</td><td style="padding: 5px;">$6/2 = 3$</td></tr> <tr><td style="border-right: 1px solid black; padding: 5px;">7</td><td style="padding: 5px;">$7/2$</td></tr> </tbody> </table>	m	$m\beta^{e-t}$	4	$4/2 = 2$	5	$5/2$	6	$6/2 = 3$	7	$7/2$
m	$m\beta^{e-t}$																				
4	$4/4 = 1$																				
5	$5/4$																				
6	$6/4 = 3/2$																				
7	$7/4$																				
m	$m\beta^{e-t}$																				
4	$4/2 = 2$																				
5	$5/2$																				
6	$6/2 = 3$																				
7	$7/2$																				

Note that for each fixed value of e , the gap between successive numbers is always the same: β^{e-t} . Finally, we enumerate zero and the subnormal numbers, for which the gap between successive numbers is the same as for the smallest normal numbers, $\beta^{e_{\min}-t}$.

$e = e_{\min} = -1, \beta^{e-t} = 2^{-4} = 1/16$	
m	$m\beta^{e-t}$
0	$0/16 = 0$
1	$1/16$
2	$2/16 = 1/8$
3	$3/16$

[†]We draw this example from Nicholas J. Higham's outstanding treatise, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002.

It helps to see where these quantities fall on the number line.



As the numbers get larger, the spacing between them increases. (This is a hallmark of floating point numbers, as opposed to the *fixed point* number systems that were seen as viable alternatives in the 1950s.) There is a convenient way to describe the precision of a floating point number system.

Definition. The *machine epsilon*, written $\varepsilon_{\text{mach}}$ or $\varepsilon_{\text{machine}}$, is the gap between 1 and the next largest floating point number, i.e., $\varepsilon_{\text{mach}} = \beta^{1-t}$.

Suppose that $x \in \mathbb{R}$ is a real number within the limits of the normalized floating point numbers, and let $\text{fl}(x) \in \mathbb{F}$ denote the floating point approximation of x . Then we can be sure that

$$\text{fl}(x) = x(1 + \delta), \quad |\delta| \leq \varepsilon_{\text{mach}}, \quad (8.1)$$

i.e., the *relative error* is controlled by $\varepsilon_{\text{mach}}$:

$$\frac{|\text{fl}(x) - x|}{|x|} \leq \varepsilon_{\text{mach}}.$$

In the floating point system with $t = 3$ that we worked out above, $\varepsilon_{\text{mach}} = \beta^{1-t} = 1/4$. Returning to our earlier example, suppose $x = 2.25$. This number is not in the floating point system, so it is rounded to the nearest number in \mathbb{F} , e.g., $\text{fl}(2.25) = 2.5$, giving

$$\frac{|\text{fl}(2.25) - 2.25|}{2.25} = \frac{.25}{2.25} = 1/9,$$

which indeed is less than $\varepsilon_{\text{mach}} = 1/4$. (Actually, the key property (8.1) would also hold if we define $\varepsilon_{\text{mach}}$ to instead be $\frac{1}{2}\beta^{1-t}$, which is preferred by some authors. In this case, $1 + \varepsilon_{\text{mach}}$ would be the first real number that could be rounded to a floating point number larger than 1.)

1.5.1 Accuracy of simple floating point computations.

The property (8.1) describes the error made when a real number is stored in a floating point system, but in order to analyze algorithms, we require more information. As mentioned above, the floating point system is not *closed* under basic arithmetic operations. For example, the numbers $7/8$ and 1 are both in our toy system detailed above, but their sum, $7/8 + 1 = 15/8$ is not. How should the computer handle the addition of such numbers? Computer engineers design floating point arithmetic units to ensure that several basic axioms analogous to the property (8.1) hold. Suppose that x, y are both normalized floating point numbers. Then provided the result of the exact arithmetic operation is also within the range of the normalized floating point numbers, we have

$$\begin{aligned} \text{fl}(x + y) &= (x + y)(1 + \delta), & |\delta| &\leq \varepsilon_{\text{mach}}, \\ \text{fl}(x - y) &= (x - y)(1 + \delta), & |\delta| &\leq \varepsilon_{\text{mach}}, \\ \text{fl}(x \times y) &= (x \times y)(1 + \delta), & |\delta| &\leq \varepsilon_{\text{mach}}, \\ \text{fl}(x \div y) &= (x \div y)(1 + \delta), & |\delta| &\leq \varepsilon_{\text{mach}}. \end{aligned}$$

In other words, basic arithmetic operations are implemented so that there is a small *relative* error. This may seem to be just about perfect, but perplexing results can still occur. Note that the above axioms required that $x, y \in \mathbb{F}$. If we wish, for example, to subtract two *real* numbers, we must first convert each number to something in \mathbb{F} , then add those two floating point numbers, resulting in a total of three rounding errors. More precisely, suppose $x, y \in \mathbb{R}$, and define

$$\begin{aligned}\hat{x} &= \text{fl}(x) = x(1 + \delta_1) \\ \hat{y} &= \text{fl}(y) = y(1 + \delta_2),\end{aligned}$$

with $|\delta_1|, |\delta_2| \leq \varepsilon$. Now subtract these numbers:

$$\begin{aligned}\text{fl}(\hat{x} - \hat{y}) &= (\hat{x} - \hat{y})(1 + \delta_3) \\ &= (x - y + \delta_1x - \delta_2y)(1 + \delta_3).\end{aligned}$$

Suppose that we have the good fortune that $\delta_3 = 0$, i.e., the subtraction is performed exactly. Then

$$\frac{|\text{fl}(\hat{x} - \hat{y}) - (x - y)|}{|x - y|} \leq \frac{|\delta_1x - \delta_2y|}{|x - y|} \leq \max\{|\delta_1|, |\delta_2|\} \frac{|x| + |y|}{|x - y|}.$$

This upper bound suggests that when $x \approx y$, there can be a very large relative error. Suppose we wish to subtract $y = 2$ from $x = 2.25$ in the demonstration system above. Then $\hat{x} = \text{fl}(x) = 2.5$, rounding up, and $\hat{y} = \text{fl}(y) = 2$ (no rounding). We can then compute the relative error in subtraction (note that $\hat{x} - \hat{y}$ is exactly represented in this floating point system):

$$\frac{|\text{fl}(\hat{x} - \hat{y}) - (x - y)|}{|x - y|} = \frac{|.5 - .25|}{|.25|} = 1.$$

We were hoping for a relative error like $\varepsilon_{\text{mach}}$, but instead we have a much larger error; no digits in the computed solution are correct. This phenomenon is known as *catastrophic cancellation*: the difference of two nearby floating point numbers may have very few accurate digits, and this error can go on to spoil subsequent computations.

1.4.3 Binary representation of floating point numbers.

In the past, different computers used different floating point systems, meaning that a program need not produce the same answer when executed on various computers. In the 1980s there was a significant move toward standardization, and now almost all modern computers use the IEEE floating point standard, which specifies formats for ‘single’ and ‘double’ precision systems, along with several special numbers (Not a Number, Infinity, etc.).

	β	t	e_{\min}	e_{\max}	$\varepsilon_{\text{mach}}$	range	storage
IEEE single precision	2	24	-125	128	6×10^{-8}	$10^{\pm 38}$	32 bits
IEEE double precision	2	53	-1021	1024	1×10^{-16}	$10^{\pm 308}$	64 bits

By default, MATLAB uses IEEE double precision arithmetic. You can view $\varepsilon_{\text{mach}}$ by typing the command `eps`. Type `help eps` to find other interesting properties of this command; see also `realmin`, and `realmax`. Recent versions of MATLAB also allow you to experiment with single precision arithmetic; type `help single` for details.

An excellent resource to learn about the IEEE floating point system is: Michael L. Overton, *Numerical Computing with IEEE Floating Point Arithmetic*, SIAM, Philadelphia, 2001.