CPSC 303: NORMAL AND SUBNORMAL NUMBERS IN DOUBLE PRECISION

JOEL FRIEDMAN

Contents

Disclaimer: The material may sketchy and/or contain errors, which I will elaborate upon and/or correct in class. For those not in CPSC 303: use this material at your own risk...

The main goal of this note is to explain why the largest positive number MATLAB reports is 1.7977e+308 (roughly 2^{1024}), while the smallest positive number is 4.9407e-324 (or 2^{-1074}). The key is to understand *normal* and *subnormal* numbers in double precision.

The content of this note all appears in the solution to Homework 2 in CPSC 303 in Spring 2020; this note is slightly edited and explains some three-term recursions studied in Homework 1 and 2 in Spring 2020.

1. Normal Numbers, Subnormal Numbers, and Special Values

Double precision mostly works with numbers written in base 2 scientific notation as

$$s \ \times \ 1.b_1 b_2 \dots b_{52} \times 2^m$$

where $s = \pm$, $b_1, \dots, b_{52} \in \{0, 1\}$ are bits (binary digits), and $m = -1022, \dots, 1023$; such a number is called a *normal number*. The 52 bits $b_1, \dots, b_{52}$ gives you "53 bits of precision" (or 52 bits of precision, depending on if you view 3.45 having 2 or 3 digits of precision). There are 2046 possible values of m, from -1022 to 1023, which requires 11 bits (binary digits) to describe, and leaves two (since $2^{11} - 2046 = 2$) special values:

(1) one of these two special values (of the 2048 possible values) is for values like Inf, -Inf, and Nan (this special value has a sign, s, plus 52 bits $b_1, \dots, b_{52}$ to describe the particular special value you mean);

Research supported in part by an NSERC grant.

(2) the other special value is for *subnormal numbers*, where when double precision understands that you mean the number

$$\pm\, 0.b_1 b_2 \dots b_{52} \times 2^{-1022}.$$

In this way you can express numbers as small as

$$0.\underbrace{000\dots000}_{51\ 0\text{'s}}1 \ \times\ 2^{-1022},$$

as a *subnormal numbers* in double precision, which is $2^{-52}\times 2^{-1022} = 2^{-1074}$; of course, for this number you only have one bit (binary digit) of precision (or zero bits of precision, depending on how you count); you can only count on a full 53-bits of precision if your number is 2^{-1022} or larger, and the smaller a *subnormal* number is, the more precision you will lose.

The (current) Wikipedia article on Double-precision floating-point format has a good explanation of this with examples; for example, the largest number in double precision is

$$1.\underbrace{1111\dots1111}_{52\ 1\text{'s}} \times 2^{1023} = \big(2-(1/2)^{52}\big)2^{1023} \approx 2^{1024}, \tag{1}$$

and this is a *normal* number—the kind you *should* be working with—since you get a full 53 bits of precision, because *normal* numbers are written as 1. followed by 52 more bits (binary digits). The smallest positive normal number is 2^{-1022}.

In CPSC 303 we consider the following three-term recurrences: we fix an $r \in \mathbb{R}$, typically with $0 < r < 1$, and consider the recurrence

$$x_{n+2} = (1+r)x_{n+1} - rx_n, \tag{2}$$

whose general solution is

$$C_1 + C_2 r^n. \tag{3}$$

The recurrence subject to the initial condition $x_0 = 1$, $x_1 = r$ has the solution $x_n = r^n$ (i.e., $C_1 = 0$ and $C_2 = 1$) in exact arithmetic. However, for a typical value of $0 < r < 1$ the numerical solution ressembles a sequence that looks like $C_2 = 1$ and C_1 is very small (typically on the order of magnitude $\pm 10^{-18}$) but not exactly 0. The exception to this are values like $r = 1/2, 1/4, 3/4$, where r is a rational number whose denominator is a power of 2.

On Homework 2 in Spring 202, we worked with $r = 1/8$; in this case the recurrence looks like

$$x_{n+2} = (9/8)x_{n+1} - (1/8)x_n,$$

and the exact solution for $x_0 = 1$ and $x_0 = 1/8$ is $x_n = (1/8)^n$. Since we are working with powers of 1/8, each power of 8 looks like

$$1.\underbrace{000\dots000}_{52\ 0\text{'s}} \times 2^m$$

for $-1022 \le m \le 1023$, which are normal numbers, but for $m \le -1023$ these powers of 2 (or 8) are the special *subnormal numbers* that look like

$$0.\underbrace{000\dots000}_{\text{some 0's}}1\underbrace{000\dots000}_{\text{more 0's}} \times 2^{-1022}$$

So in binary arithmetic, the smallest positive number is $2^{-1074} = 8^{-358}$, so I'll (probably) trust the numerical computation as exact until the division by 8 in the recurrence dips below 8^{-358}.

Note that textbook [A&G] **does not mention subnormal numbers**. Similarly, if you type `realmin` into MATLAB, it will return `2.2251e-308`, since you can't count on 53 bits of precision for smaller positive numbers, i.e., subnormal numbers (and you should realize this caveat in working with smaller positive numbers). However, MATLAB will report subnormal numbers without telling you this, which explains why you can see positive numbers as small as $2^{-1074} \approx 4.94 \times 10^{-324}$, while you only see positive numbers as large $2^{1024} \approx 1.80 \times 10^{308}$ (where 1.80 is obtained by rounding 1.7977, rather than a truncation).

Exercises

(1) Run the MATLAB code

```
for n=50:55, n, 2^(1023) * (2 - 2^(-n) ), end,
```

Describe the results, and explain what this has to do with (1).

(2) (a) Run the MATLAB code

```
clear
x{1}=1
x{2}=3/4

for i=3:250, x{i}=(7/4)*x{i-1} - (3/4)*x{i-2}; end

x
```

(which is an implementation of (2) with $x_0 = 1$ and $x_1 = r$ for the value $r = 3/4$). Assuming that for small $n = 1$ you have $x_n = C_2(3/4)^n$ (to within a negligible term)and for large $n = 249$ you have $x_{249} = C_1$ (in the above MATLAB code, $x\{250\}$ represents x_{249}), what are the values of C_1 and C_2?

(b) Add the following MATLAB code:

```
C2 = 1
C1 = x{250}

for i=1:250, y{i}=C1 + C2 * (3/4)^(i-1) ; end

y

for i=1:250, ratio{i} = x{i}/y{i}; ratio_versus_one{i} = 1 -ratio{i} ; end
ratio
ratio_versus_one
```

Describe what you see; in particular describe which values of `ratio` are reported by MATLAB as 1, which are reported as 1.0000, which values of `ratio_versus_one` are reported as 0, which values are reported as something else.

(c) Based on this experiment, what is the difference between the value 1 and 1.0000 that MATLAB reports?

(d) Perform the experiment in the above parts with 250 replaced by 400. Describe which values MATLAB reports as 1 versus 1.0000 for `ratio`, and as 0 versus non-zero for `ratio_versus_one`.

(3) (a) Run the MATLAB code

```
clear
r = 3/4
x{1}=1
x{2}= r

for i=3:200, x{i}=(1+r)*x{i-1} - r*x{i-2}; end

[x{1},x{2},x{3},x{4}]
[x{197},x{198},x{199},x{200}]
```

(which is an implementation of (2) with $x_0 = 1$ and $x_1 = r$ for the value $r = 3/4$). What does MATLAB report for $x\{200\}$?

(b) Run the same experiment with $r = 3/8$, then $r = 3/16$, and then $r = 3/32$. What values do you get for $x\{200\}$? What is the rough pattern (e.g., the order of magnitude of $x\{200\}$) that you observe for this value for $r = 3/4, 3/8, 3/16, 3/32$?

(c) Given that the general solution of (2) is given by (3), and in view of the way that double precision works, explain **in 15-60 words** why you see this pattern in the $x\{200\}$ values?

Department of Computer Science, University of British Columbia, Vancouver, BC V6T 1Z4, CANADA.

E-mail address: jf@cs.ubc.ca

URL: http://www.cs.ubc.ca/~jf