Math 311

Numerical Methods

1.2: Computer Arithmetic

Floating Point Arithmetic, Relative and Absolute Errors,Truncation and Rounding Errors.

S. K. Hyde

Burden and Faires, any ed.

Winter 2025

Introduction

- It is impossible to represent all real numbers on a computer. Why?
- Easy answer - a computer has finite storage - some numbers cannot be stored.
- For example, π is irrational, so it contains infinte, non repeating digits.
- Others are not terminating rational numbers. They have infinite digits.
- Computers use base 2. Base 10 is what we all use.
- Whether or not a fraction has a terminating expansion depends on the base:
  - In base 10, $\frac{1}{10}$ is terminating (0.1), but $\frac{1}{3}$ is not $(0.\overline{3})$.
  - In base 2, $\frac{3}{16}$ is terminating (0.0011_2), but $\frac{1}{10}$ is not $(0.0\overline{0011}_2)$.
- This means that some numbers which appear to be short and exact in decimal, are no longer that way in binary. This leads to inaccuracies.
- We also need to be able to represent large numbers.
- So, we need to invent a way to describe a lot of real numbers, but with just a finite number of digits. - Enter Floating Point Numbers.

Floating Point Standards:

- There is not a unique way to store numbers in a computer.
- Each way is called a standard. Some other examples:
  - IBM 3000 & 4300 Series computer standard (book examples use this).
  - IEEE Standard 754-1985 single precision (32 bit numbers)
  - IEEE Standard 754 double precision numbers (64 bit)
  - BigDecimal (arbitary precision) - used with monetary calculations
- How the numbers are encoded depends on the standard used.
- Our goal is to get a good range of real numbers using finite number of digits.
- Single 32-bit numbers - there are 4,294,967,296 number of choices, so we need to assign them out carefully.
- We can represent numbers as small as $\approx 10^{-45}$ and as large as $\approx 3.4 \times 10^{38}$ - pretty wide for only 4 billion numbers to choose from!
- How? Think Scientific Notation.
- Let's illustrate how using base 10 first.
- To get a bigger range of values, split every number up into three things:

$$\text{sign} \times \text{mantissa} \times \text{base}^{\text{exponent}}.$$

Base 10 Example:

- Suppose the number is -47.7531 in base 10.
- The sign is "$-$" (store 0 for positive number and 1 for negative number.)
- The mantissa (or fraction or characteristic) is .477531 (with decimal point)
- The exponent is 2 with a base of 10 $\Longrightarrow$ (10^2). This is how many decimals places to move the mantissa to the number being represented)
- So the number is $-.477531 \times 10^2$ and is stored as a sequence of digits.
- For example, suppose a 1 digit sign, a 4 digit exponent, and a 6 digit mantissa.
- Then -47.7531 would be stored as [10002477531] (it looks like about 10 billion as a number, but it is encoded as [1|0002|477531]!)
- A computer does it similarly, but with a limited number of bits for each number.
- They typically only store bits (0's or 1's) and the base is usually 2.
- Originally, every computer system had a different definition.
- It got really problematic when moving programs between systems.
- A working group of Intel, Hewlett Packard, and DEC, the main microchip producers came together and produced the IEEE 754 Standard for everyone! It is an example of cooperation for the good of everyone!! Yay for Humans!

- For the floating point definitions mentioned above, their definitions vary.
  - Single precision IBM 3000 & 4300 Series computers used 32 bits. use
    DEF 1 bit sign, a 7 bit exponent (base of 16), and a 24 bit mantissa.
  - IEEE Standard 754-1985 (now at 754-2019) (single precision (32 bit numbers))
    DEF 1 bit sign, an 8 bit exponent (base of 2), and a 24 bit mantissa (23 explicit).
  - IEEE Standard 754 double precision numbers (64 bits)
    DEF 1 bit sign, an 11 bit exponent (base of 2), and a 53 bit mantissa (52 explicit)
- How it encodes numbers depends on the standard used.
- Now to describe the Base-2 analogue of the example on previous slide:

Base-2 Analogue:

Definition. *Any non-zero real number, x, can be written as*

$$x = \pm q \times 2^m, \text{ where } \frac{1}{2} < \boldsymbol{q} < 1.$$

Then $\boldsymbol{q}$ has a possible infinite binary representation as:

$$\boldsymbol{q} = .1d_1d_2d_3..., \text{ where "." is called the "binary point"}$$

Examples:

- Suppose $x = \frac{1}{16}$. Then we can write it as $0.0001_2 = .1 \times 2^{-3}$
  - Note that this number is a good finite length case and is exact in binary!!
- Not all numbers can be represented as a finite length binary number.
- For example, $\frac{1}{10}$ can be written in base 10 with finite number of digits, but it is infinite in binary.
- It is actually repeating binary $0.0001100110011001100110011001100110011\overline{0011}$
- We can only use a finite number of bits (52 in double precision).
- That means that instead of storing "0.1", the calculator stores the closest binary equivalent: 0.0001100110011001100110011001100110011001100110011001100
- If you use R, it converts the base 10 number 0.1 into: 0.10000000000000000055511
- Because we can't store infinite digits, we are forced to truncate the number, which can cause a loss of precision when adding and subtracting a lot of them.
- Let move on and introduce the IEEE 754-1985 standard.

IEEE 754-1985 single precision (32 bits) numbers

Definition. *They are saved as a sequence of 32 bits, like below*

s	(base 2 expon.)								f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}	f_{21}	f_{22}	f_{23}
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

The floating point number it represents is

$$x = (-1)^s \times 1.f_1 f_2 \cdots f_{23} \times 2^{e-b},$$

where $b = 127$ *and is called the bias. The exponent* e *uses 8 bits and is* $0 \leqslant e \leqslant 127$. *The bias splits the exponents in half, so the range of values of exponents is between* $2^{-127} \leqslant 2^{e-b} \leqslant 2^{128}$. *The mantissa is assumed to start with 1 (but is not stored).*

- A mantissa of 1 (stored as 0) and exponent of 0 gives the range of $2^{-127} \leqslant 2^{e-127} \leqslant 2^{128}$ for the exponent. So the smallest number is

$$1. \times 2^{-127} \approx 5.9 \times 10^{-39}.$$

- Similarly, the largest number is

$$1.11111111111111111111111 \times 2^{128} \approx 2 \times 2^{128} = 2^{129} \approx 6.8 \times 10^{38}$$

IBM 3000 & 4300 series

Definition. *The book uses the floating point standard of the IBM 3000 and 4300 series computer. For this, it uses:*

s	(base 16 expon.)							f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}	f_{21}	f_{22}	f_{23}	f_{24}
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

The floating point number it represents is

$$x = (-1)^s \times 0.f_1 f_2 \cdots f_{24} \times 16^{e-b},$$

where the bias is $b = 64$. *The exponent* e *uses 7 bits and is* $0 \leqslant e \leqslant 127$. *The bias gives the range of* $16^{-64} \leqslant 16^{e-b} \leqslant 16^{63}$. *The mantissa must have at least one of* f_1, f_2, f_3, *and* f_4) *be a one.*

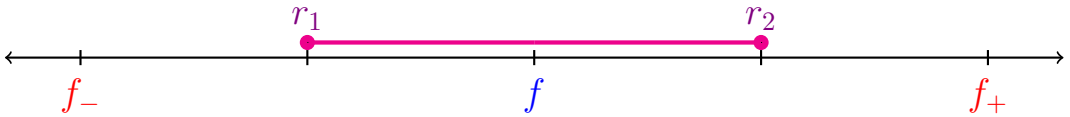
- A mantissa of 0001 followed by 19 zeros, an exponent of 0 gives the range of $16^{-65} < 16^{e-b} < 16^{63}$ for the exponents. So an estimate of the smallest number is

$$1. \times 16^{-65} \approx 5.4 \times 10^{-79}.$$

- Similarly, the largest is

$$0.111111111111111111111111 \times 16^{63} \approx \times 16^{63} \approx 7.237 \times 10^{75}$$

- Not all rational numbers are possible! Suppose that f_-, f, f_+ represents three consecutive machine representable (floating point) numbers.



  Then the number f represents **ALL real numbers** between r_1 and r_2 (the midpoints). The other numbers besides f_-, f, f_+ are not machine representable!

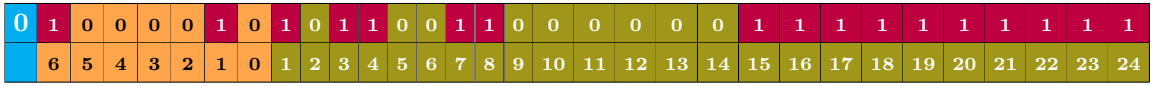
- For example, suppose we are analyzing the IBM 3000 floating point (where I modified the bit number to be the placevalue instead.

0	1	0	0	0	0	1	0	1	0	1	1	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	6	5	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

  - The sign bit is zero, so it's positive.
  - The exponent is $2^6 + 2^1 = 64 + 2 = 66$. With the bias we have $16^{66-64} = 16^2$.
  - The mantissa part is $\overbrace{\left(\frac{1}{2}\right)^1 + \left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\right)^4 + \left(\frac{1}{2}\right)^7 + \left(\frac{1}{2}\right)^8}^{\frac{179}{256}=\frac{179}{16^2}} + \left(\frac{1}{2}\right)^{14} = \frac{179}{256} + \left(\frac{1}{2}\right)^{14}$
  - So the real value of this floating point number is

$$= 16^2 \times \left[\frac{179}{16^2} + \left(\frac{1}{2}\right)^{14}\right] = \mathbf{179 + \frac{1024}{2^{16}}} = 179.015625$$

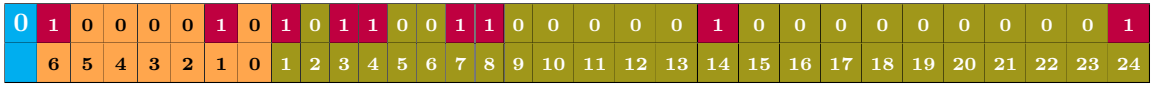
- The next smallest machine representable number is



with the value of $16^2\left[\frac{179}{256}+\sum_{i=15}^{24}\left(\frac{1}{2}\right)^i\right]$. The sum part is a finite geometric sum.

$$= 179 + 16^2\left[\left(\frac{1}{2}\right)^{14} - \left(\frac{1}{2}\right)^{24}\right] = 179 + \frac{1}{64} - \left(\frac{1}{2}\right)^{16} = \mathbf{179 + \frac{1023}{2^{16}}}$$

- Similarly, the next largest machine representable number is



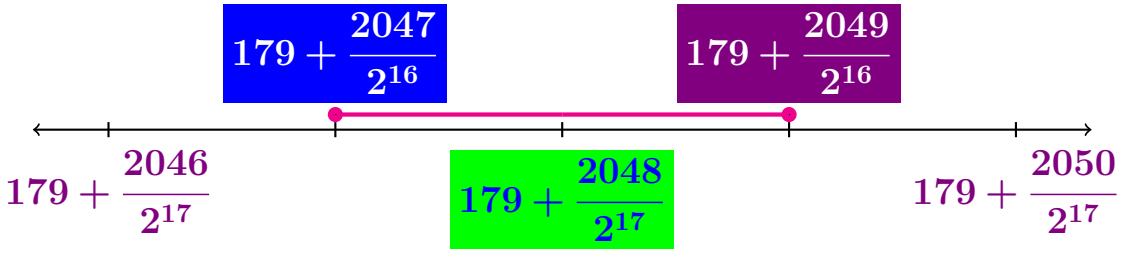
with the value of

$$= 179 + 16^2\left[\left(\frac{1}{2}\right)^{14} + \left(\frac{1}{2}\right)^{24}\right] = 179 + \frac{1}{64} + \left(\frac{1}{2}\right)^{16} = \mathbf{179 + \frac{1025}{2^{16}}}$$

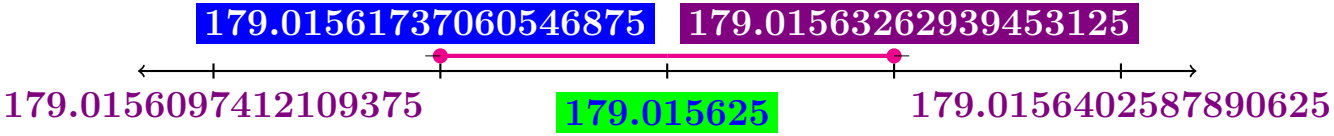
- Compare to the real floating point number:

$$= 179 + 16^2\left[\left(\frac{1}{2}\right)^{14} + \left(\frac{1}{2}\right)^{24}\right] = 179 + \frac{1}{64} \qquad = \mathbf{179 + \frac{1024}{2^{16}}}$$

- Place these (and their midpoints) on a number line:



- The decimal value of these (and with the midpoints on the right):



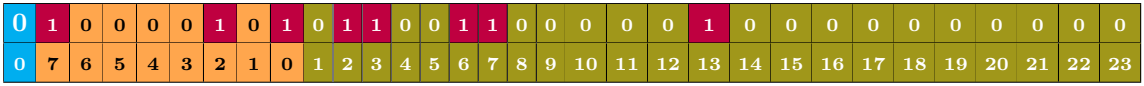
- Implicit in this is that the the midpoints ARE NOT machine representable)! This means that the number 179.015625 represents ALL real numbers in between:

$$[\ \ 179.01561737060546875, 179.01563262939453125\ \ ]$$

- Remember that rationals are infinite but countable! The reals are uncountable, which means we are leaving out A LOT of real numbers (an infinite amount). This feels like we are bound to make mistakes!

Example:

- Today, the standard is IEEE 754-1985 single floating point numbers. Let's take the exact sequence of 32 bits and interpret it.



- In this case, the exponent has 8 bits, and each mantissa starts with 1. The sign is positive, and $e = 1 + 2^2 + 2^7 = 133$ with a bias of 127. So the value of the exponent is $2^{133-127} = 2^6 = 64$.

- The mantissa is
$$1 + \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\right)^6 + \left(\frac{1}{2}\right)^7 + \left(\frac{1}{2}\right)^{13} = \frac{11457}{2^{13}}$$

- Thus, the actual number is
$$1 \times 2^6 \times \frac{11457}{2^{13}} = \frac{11457}{2^7} = 89.5078125$$

Absolute Error and Relative Error

Definition.

- *If p^* is an approximation to p, then the* ***absolute error*** *is defined as*

$$|p - p^*|.$$

- *If p^* is an approximation to p, then the* ***relative error*** *is defined as*

$$\frac{|p - p^*|}{p}, \text{ provided } p \neq 0$$

Truncation and Rounding Errors

Definition. *Because we have only finite digits to work with, then we either should "chop" (or truncate) the part past the last bit, or we can "round" the number.*

Floating Point Errors

Theorem. *For floating point numbers, the relative error can be controlled. In particular,*

$$\frac{|fl(x) - x|}{|x|} \leqslant \varepsilon_M.$$

- *If* $x \neq 0$, $fl(x) = x(1 + r)$, *where* $|r| \leqslant \varepsilon_M$.
- *We can get the relative accuracy of the operations* $(+, -, \times, /)$ *to be just as accurate as above if we use "extended precision" for operations, but not for storage. The error only results from storage. If* $x \odot y$ *represents a perfect operation from* $(+, -, \times, /)$, *then*

$$\frac{|fl(x \odot y) - (x \odot y)|}{|x \odot y|} \leqslant \varepsilon_M$$

Practical Implications

- **Minimize the number of subtractions of nearly equal numbers in your calculations as you can lose a lot of digits of accuracy.**
- **Polynomials should always be expressed in nested form before performing an evaluation. This form minimizes the number of required arithmetic calculations.**
- **For the polynomial**

$$f(x) = ax^3 + bx^2 + cx + d$$

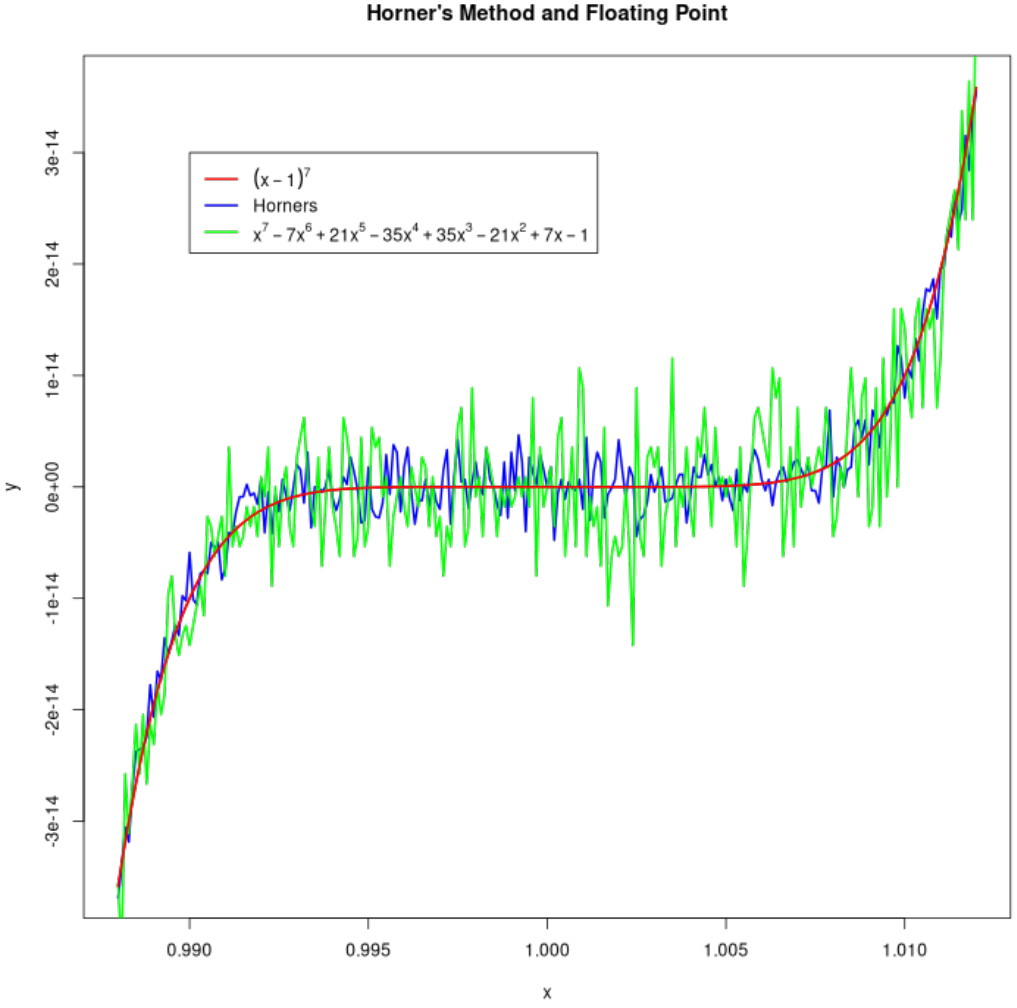
- Use `f(x) = ((a*x + b)*x + c)*x + d` **(3 multiplies and 3 adds)**
instead of `f(x) = a*x^3 + b*x^2 + c*x + d` **(6 multiplies and 3 adds)**

- or you may prefer it backwards:
Use `f(x) = d + x*(c + x*(b + a*x))`
instead of `f(x) = d + c*x + b*x^2 + a*x^3`

- The method of function evaluation is also known as "Horner's Method" (or "synthetic division").

For example, when $f(x) = x^4 + 2x^2 - x + 1$, then $f(2)$ is

$$\begin{array}{c|ccccc} 2 & 1 & 0 & 2 & -1 & 1 \\ & & 2 & 4 & 12 & 22 \\ \hline & 1 & 2 & 6 & 11 & 23 \end{array} \Longrightarrow f(2) = 23$$


Horner's Method and Floating Point
$(x-1)^7$
Horners
$x^7-7x^6+21x^5-35x^4+35x^3-21x^2+7x-1$
y
3e-14
2e-14
1e-14
0e+00
-1e-14
-2e-14
-3e-14
0.990
0.995
1.000
1.005
1.010
x