Numerical Analysis: Computer Representation of Numbers

Computer Science, Ben-Gurion University
(slides based on Prof. Ben-Shahar’s notes and notes by Steve Hollasch on the IEEE754 Standard)

2019/2020, Fall Semester
Computer representation of data is often only an approximation.

Example

// filename: prog.c
#include<stdio.h>
int main()
{
    float a=0.0;
    int i;
    for (i=0;i<1000000;i++){
        a+=0.1;
    }
    printf("a=%f",a); // on my laptop: a=100958.343750
    return 0;
}
We tried to compute
\[ \sum_{i=1}^{10^6} 0.1 = 0.1 \sum_{i=1}^{10^6} 1 = 0.1 \times 10^6 \]
and instead of getting
\[ 100,000 \]
we got
\[ 100958.343750 \].

The relative error is
\[ \frac{958.343750}{10^5} \approx 1\% . \]

Why does this happen? And what can be the consequences?
There are several ways to represent real numbers on computers.

We will discuss fixed-point and floating-point representations.
Fixed-point Representation

- **Fixed point** places a radix point somewhere in the middle of the digits, and is equivalent to using integers that represent portions of some unit.

**Example**

If we have 4 decimal digits, we may choose to place the radix point between the second and third. Then:

- digit #1 represents tens;
- digit #2 represents units;
- digit #3 represents tenths;
- digit #4 represents hundredths.

Thus numbers such as 10.82 or 00.01 are represented exactly.
Problems with a Fixed-point Representation

Consider the last example.

- What if $x = 123$?
- Or $x = -3000$?
- Or $x = 12.3456789$?
Scientific Notation

A standard method for representing numbers, known as Scientific Notation, is based on shifting the decimal point (or decimal comma) and multiplying by the suitable power of 10.

Example

0.0000123 ⇒ 1.23 \cdot 10^{-5} = 1.23E - 5

1,230,000 ⇒ 1.23 \cdot 10^{6} = 1.23E6
Representing numbers in a **computer** is based directly on the Scientific Notation.

The range of positive floating point numbers can be split into **normalized** numbers and **denormalized** numbers.
Floating point solves a number of representation problems.

- Fixed point has a fixed window of representation, which limits it from representing very large or very small numbers.
- Fixed point is prone to a loss of precision when two large numbers are divided.
Floating point, on the other hand, employs a sort of "sliding window" of precision appropriate to the scale of the number. This allows it to represent numbers from, say,

\[ 10^{-12} \]

to

\[ 10^{12} \]

with ease, and while maximizing precision (the number of digits) at both ends of the scale.
A floating-point system captures the following numbers:

\[ F(B, n, m, M) \triangleq \left\{ x \mid x = \pm b_1.b_2\ldots b_n \cdot B^e, \begin{align*} 0 < b_1 &\leq B - 1 \\ 0 &\leq b_{i>1} \leq B - 1 \\ m &\leq e \leq M \end{align*} \right\} \]

- base: \( B \)
- mantissa (AKA significand): \( b_1 b_2 \ldots b_n \)
- exponent: \( e \) (plus a bias, to be explained soon)
- lower and upper bound on \( e \): \( m, M \).
- The decimal point “floats” till the place where the leading digit is greater than zero.
- The term **normalized** refers to the fact that \( b_1 > 0 \). We will discuss **denormalized** later on.
- On a computer, \( B \) is usually 2 (so \( 0 < b_1 \leq B - 1 \) implies \( b_1 = 1 \)). Thus, numbers are typically represented as

\[ x = \pm1.b_2b_3\ldots b_n2^e \]
IEEE Standard 754 for Floating Point

- For a more detailed explanation of the IEEE Standard 754 for FP, see “Explaining the IEEE754 Standard for Floating Point” (by Steve Hollasch) in our course website.

- Most of the following slides are adapted from there.
IEEE Standard 754 for Floating Point

\[ x = \pm 1.b_2 b_3 \ldots b_n 2^e \]

- **single (32 bits):**
  - 1 bit for sign
  - 8 for the exponent
  - 23 bits for the fractional part \((b_2 \ldots, b_{24})\)

- **double (64 bits):**
  - 1 bit for sign
  - 11 for the exponent
  - 52 bits for the fractional part \((b_2 \ldots, b_{53})\)

- In both cases, the sign is encoded via \((-1)^s\) where \(s \in \{0, 1\}\) so \(s = 0 \Rightarrow \text{“plus”} \) and \(s = 1 \Rightarrow \text{“minus”}\).
In the exponent, we exclude the cases of all ones and all zeros – these cases are regarded as special.
single precision:

(0 excluded) \[ 1 \leq \text{exponent} \leq 2^8 - 2 = 254 \] (255 excluded)

\[ e \triangleq \text{exponent} - 127 \]

so

\[ -126 = 1 - 127 \leq e \leq 254 - 127 = 127. \]

\[ -126 \leq e \leq 127 \]
double precision:

(0 excluded) \[ 1 \leq \text{exponent} \leq 2^{11} - 2 = 2048 - 2 = 2046 \] (2047 excluded)

\[ e \triangleq \text{exponent} - 1023 \]

so

\[ -1022 = 1 - 1023 \leq e \leq 2046 - 1023 = 1023 . \]

\[ -1222 \leq e \leq 1023 \]
Do We Lose Something?

- Consider single precision.
- Essentially, we take 32 bits and reinterpret the fields to cover a much broader range than what a 32bit integer covers.
- Something has to give, and it’s precision.
- For example, regular 32bit integers, with all precision centered around zero, can precisely store integers with 32bits of resolution.
Single precision floating point, on the other hand, is unable to match this resolution with its 24 bits. It does, however, approximate this value by effectively truncating from the lower end and rounding up.

**Example**

\[
\begin{align*}
11110000 11001100 10101010 10101111 & \quad // 32\text{-bit integer} \\
\approx +1. \underbrace{1110000 11001100 10101011}_{23 \text{ bits}} \times 2^{31} & \quad // \text{Single-Precision Float} \\
= \underbrace{11110000 11001100 10101011}_{24 \text{ bits}} \times 2^8 \\
= \underbrace{11110000 11001100 10101011}_{24 \text{ bits}} \times 100000000_{\text{2 in binary}} \\
= \underbrace{11110000 11001100 10101011}_{24 \text{ bits}} 00000000 & \quad // \text{Corresponding Value}
\end{align*}
\]
This approximates the 32bit value, but yields an inexact representation.

On the other hand, besides the ability to represent fractional components, the floating-point value can also represent numbers around $2^{127}$, compared with 32bit integers’ maximal value around $2^{32}$.
- Since the leading figure of $x$ is 1, the number 0 is not represented in this scheme.

- Thus, by convention, if both the exponent and the mantissa are zero, then this means $x = 0$
  (recall: the case $\text{exponent} = 0$ – i.e., all bits are zero – was excluded)
- We represent 0 as
  \[ +1.00\ldots00 \cdot 2^0 \]
  (where the “1” is implicit, i.e., we don’t waste a bit on it) even though this number is not zero of course.

- Moreover, note we also have
  \[ -1.00\ldots00 \cdot 2^0 \]

  So we have a negative zero and a positive zero.

  By convention, they are equal to each other.
As with any finite representation, in FP representation not all real numbers are represented exactly.

Observation: the spacing of those reals that can be represented exactly is not uniform.

The advantage of this non-uniform spacing is that we can have a large range while maintaining small relative errors in the representation.
Given a real number $x$, there are two standard ways to approximate using a FP representation: rounding and chopping.
Let $\text{fl}(x)$ denote the FP representation of $x$ (using either rounding or chopping). The absolute error of this representation,

$$|\text{fl}(x) - x|$$

is called a \textit{round-off error}. 
Recall:

\[ F(B, n, m, M) \triangleq \left\{ x \mid x = \pm b_1.b_2 \ldots b_n \cdot B^e, \begin{array}{l}
0 < b_1 \leq B - 1 \\
0 \leq b_{i>1} \leq B - 1 \\
m \leq e \leq M
\end{array} \right\} \]

If \( \tilde{x} = \text{fl}(x) \) in an \( F(B, n, m, M) \) system then

\[
\delta \tilde{x} = \frac{|x - \text{fl}(x)|}{|x|} \leq \left\{ \begin{array}{l}
\frac{1}{2} B^{1-n} \quad \text{for rounding} \\
B^{1-n} \quad \text{for chopping}
\end{array} \right. 
\]
Example

Suppose \( n = 3 \), \( B = 2 \), and

\[
1.b_20 \cdot 2^e \leq x = 1.b_2b_3b_4b_5 \ldots \cdot 2^e \leq 1.b_21 \cdot 2^e
\]

Suppose we do chopping.

- If \( x = 1.b_20 \cdot 2^e \) or \( x = 1.b_21 \cdot 2^e \) then \( \text{fl}(x) = x \) and \( \delta\tilde{x} \) is zero.
- If \( b_3 = 0 \) and \( \exists i > 3 \) such that \( b_i > 0 \), then \( \text{fl}(x) = 1.b_20 \cdot 2^e < x \) and:

\[
\frac{x - \text{fl}(x)}{x} = \frac{(1.b_20b_4b_5 \ldots - 1.b_20) \cdot 2^e}{1.b_2b_3b_4b_5 \ldots \cdot 2^e} = \frac{0.00b_4b_5 \ldots}{1.b_2b_3b_4b_5 \ldots}
\]

\[
\leq \frac{0.00b_4b_5 \ldots}{1.00000 \ldots} = 0.00b_4b_5 \ldots \leq 0.0011 \ldots \text{just as } 0.999 \ldots = 1
\]

\[
= \frac{0.01}{2^{-2}} = 2^{1-3} = B^{1-n}
\]
Thus, by definition, an $F(B, n, m, M)$ system provides an approximation upto $n$ significant figures.
Denormalized Numbers

- If the exponent is all zeros, but the mantissa is nonzero, then the value is a **denormalized** number, which has an assumed leading 0 before the binary point. Thus, in single precision, we have

\[
(\-1)^s \times 0.f \times 2^{-126}
\]

where \(s\) is the sign bit and \(f\) is the fraction.

- For double precision, denormalized numbers are of the form

\[
(\-1)^s \times 0.f \times 2^{-1022}
\]

- Zero can be interpreted as a special type of denormalized number.
Example

The minimal value the positive reals in single precision (using denormalized numbers) is

\[ (-1)^0 \times 0.\overline{0000000000000000000000000000000} \times 2^{-126} = 2^{-23} \times 2^{-126} = 2^{-149}. \]

Example

The corresponding maximal value (using denormalized numbers) is

\[ (-1)^0 \times 0.\overline{1111111111111111111111111111111} \times 2^{-126} = (1 - 2^{-23}) \times 2^{-126}. \]
As denormalized numbers get smaller, they gradually lose precision as the left bits of the fraction become zeros. At the smallest denormalized value (only the least significant fraction bit is one), a 32bit floating-point number has only a single bit of precision, compared with the standard 24bits for normalized values.
<table>
<thead>
<tr>
<th>Precision</th>
<th>Denormalized</th>
<th>Normalized</th>
<th>Approximate Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>$\pm 2^{-149}$ to $(1-2^{-23}) \times 2^{-126}$</td>
<td>$\pm 2^{-126}$ to $(2-2^{-23}) \times 2^{127}$</td>
<td>$\pm \approx 10^{-44.85}$ to $\approx 10^{38.53}$</td>
</tr>
<tr>
<td>Double</td>
<td>$\pm 2^{-1074}$ to $(1-2^{-52}) \times 2^{-1022}$</td>
<td>$\pm 2^{-1022}$ to $(2-2^{-52}) \times 2^{1023}$</td>
<td>$\pm \approx 10^{-323.3}$ to $\approx 10^{308.3}$</td>
</tr>
</tbody>
</table>
Ranges

There are five distinct numerical ranges that single-precision floating-point numbers are unable to represent:

1. Negative numbers less than \(- (2 - 2^{-23}) \times 2^{127}\) (negative overflow)
2. Negative numbers greater than \(-2^{-149}\) (negative underflow)
3. Zero
4. Positive numbers less than \(2^{-149}\) (positive underflow)
5. Positive numbers greater than \((2 - 2^{-23}) \times 2^{127}\) (positive overflow)
Underflow is of course a problem, but arguably less severe than overflow since it is “just” loss of precision, and can be closely approximated by zero. Thus, the effective ranges (excluding infinite values) of IEEE floating-point numbers are:

- **Single**: Binary $\pm (2 - 2^{-23}) \times 2^{127}$ (in decimal: $\approx \pm 10^{38.53}$).
- **Double**: Binary $\pm (2 - 2^{-52}) \times 2^{1023}$ (in decimal: $\approx \pm 10^{308.2553}$).
## Float Values ($b = \text{bias}$)

<table>
<thead>
<tr>
<th>Sign</th>
<th>Exponent ($e$)</th>
<th>Fraction ($f$)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00⋯00</td>
<td>00⋯00</td>
<td>+0</td>
</tr>
<tr>
<td>0</td>
<td>00⋯00</td>
<td>00⋯01</td>
<td>Positive Denormalized Real $0.f \times 2^{(-b+1)}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11⋯11</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>00⋯01</td>
<td>XX⋯XX</td>
<td>Positive Normalized Real $1.f \times 2^{(e-b)}$</td>
</tr>
<tr>
<td></td>
<td>11⋯10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>11⋯11</td>
<td>00⋯00</td>
<td>$+\infty$</td>
</tr>
<tr>
<td>0</td>
<td>11⋯11</td>
<td>00⋯01</td>
<td>SNaN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01⋯11</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>11⋯11</td>
<td>1X⋯XX</td>
<td>QNaN</td>
</tr>
<tr>
<td>Digit</td>
<td>Significand</td>
<td>Exponent</td>
<td>Representation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>----------</td>
<td>----------------</td>
</tr>
<tr>
<td>1</td>
<td>00...00</td>
<td></td>
<td>−0</td>
</tr>
<tr>
<td>1</td>
<td>00...00</td>
<td>:</td>
<td>−0.f × 2^{(-b+1)}</td>
</tr>
<tr>
<td>1</td>
<td>00...01</td>
<td>:</td>
<td>−1.f × 2^{(e-b)}</td>
</tr>
<tr>
<td>1</td>
<td>11...11</td>
<td></td>
<td>−∞</td>
</tr>
<tr>
<td>1</td>
<td>11...11</td>
<td></td>
<td>SNaN</td>
</tr>
<tr>
<td>1</td>
<td>11...11</td>
<td>:</td>
<td>QNaN</td>
</tr>
<tr>
<td>1</td>
<td>11...11</td>
<td>1X...XX</td>
<td></td>
</tr>
</tbody>
</table>

Negative Denormalized Real

Negative Normalized Real
Anomalies in FP Representations

- We already saw $\sum_{i=1}^{10^6} 0.1$ might fail.
- We mentioned that we can have $(x + y) + z \neq x + (y + z)$.
- Likewise, we can have $xz + yz \neq (x + y)z$.
- Also, summing a sequence from small to large can give a different result from when summing from large to small (and both results can be wrong).
- Why do we get all these?
Suppose we use rounding. Let $\text{fl}(\cdot)$ denote the floating-point representation of its argument.

We can have situations such as:

$$0 < B < \text{fl}(B) < \text{fl}(A) < A < \text{fl}(\text{fl}(B) + \text{fl}(A)) < \text{fl}(A) + \text{fl}(B) < A + B$$
Or, we may be adding numbers of different orders (we can have problems even if $A$ and $B$ are accurately represented). E.g. if $A \ll B$, then even if $A = \text{fl}(A)$ and $B = \text{fl}(B)$, we may get:

$$0 < \text{fl}(A) \ll \text{fl}(B) = \text{fl}(\text{fl}(A) + \text{fl}(B)) < \text{fl}(A) + \text{fl}(B) (= A + B)$$

So got that “$A+B$”=“$B$” even if $A > 0$! And if now added another $A$, it will again have no effect, and so on.
Loss of Significance

One of the fundamental problems in arithmetics based on a FP representation is a “loss of significance” which happens when subtracting almost equal numbers.
Let $p$ and $k$ be two positive integers such that $p < k$.

Let $x$ and $y$ be two positive real numbers where $x$ and $y$ are almost equal and have FP representations of $k$ figures each.

Particularly, suppose the first $p$ figures of (the FP representation of) $x$ are the same as the first $p$ figures of (the FP representation of) $y$, and that they differ in the $(p+1)^{th}$ figure.

Let us first consider the real $x$ and $y$ (as opposed to their FP representations):

\[
x = 0.\overbrace{d_1 d_2 \ldots d_p}^{p} \overbrace{\alpha_{p+1} \alpha_{p+2} \ldots}^{k-p} \overbrace{\alpha_k \alpha_{k+1} \alpha_{k+2} \ldots}^{\infty} \cdot 10^n
\]

\[
y = 0.\overbrace{d_1 d_2 \ldots d_p}^{p} \overbrace{\beta_{p+1} \beta_{p+2} \ldots}^{k-p} \overbrace{\beta_k \beta_{k+1} \beta_{k+2} \ldots}^{\infty} \cdot 10^n
\]

where $\alpha_{p+1} \neq \beta_{p+1}$.
WLOG, suppose \( x > y \). Thus, \( \alpha_{p+1} > \beta_{p+1} \).

Particularly, the first \( p \) figures, after the period, of

\[
0.d_1d_2 \ldots d_p \alpha_{p+1} - 0.d_1d_2 \ldots d_p \beta_{p+1}
\]

are all zeros, and

\[
\gamma_{p+1} \triangleq \alpha_{p-1} - \beta_{p-1} > 0.
\]

The difference between \( x \) and \( y \) satisfies

\[
x - y = 0.0 \ldots 0_{\text{p}} \underbrace{\gamma_{p+1} \ldots \gamma_k}_{\text{p}} \underbrace{\gamma_{k+1} \gamma_{k+2} \ldots}_{\text{k-p}} 10^n \\
= 0. \underbrace{\gamma_{p+1} \ldots \gamma_k}_{\text{k-p}} \underbrace{\gamma_{k+1} \gamma_{k+2} \ldots}_{\text{inf}} 10^{n-p}.
\]

for some (possibly non-terminating) sequence \( \gamma_{p+1} \gamma_{p+2} \ldots \).
Now let’s shift our discussion to the FP representation. In FP representation:

\[
\tilde{x} = \text{fl}(x) = 0.\overbrace{d_1 d_2 \ldots d_p}^{p} \alpha_{p+1} \alpha_{p+2} \ldots \alpha_k \cdot 10^n
\]

\[
\tilde{y} = \text{fl}(y) = 0.\overbrace{d_1 d_2 \ldots d_p}^{p} \beta_{p+1} \beta_{p+2} \ldots \beta_k \cdot 10^n
\]

Both \(\text{fl}(x)\) and \(\text{fl}(y)\) have \(k\) significant figures by definition. The FP representation of \(\text{fl}(y) - \text{fl}(x)\) is

\[
\text{fl}(\text{fl}(x) - \text{fl}(y)) = 0.\underbrace{0 \ldots 0}_{p} \gamma_{p+1} \ldots \gamma_k \cdot 10^n
\]

\[
= 0.\underbrace{\gamma_{p+1} \ldots \gamma_k}_{k-p} 00 \ldots 00 \cdot 10^{n-p}.
\]

This result has, at most, \(k - p\) significance figures.
In other words, the last $p$ figures in the mantissa are “wasted” as they all became zero. Indeed:

$$
\delta(\widetilde{x} - \widetilde{y}) = \frac{(x - y) - (\widetilde{x} - \widetilde{y})}{x - y}
$$

$$
= 0.\underbrace{\gamma_{p+1} \cdots \gamma_{k}}_{k-p} \underbrace{\gamma_{k+1} \gamma_{k+2} \cdots}_{\infty} \cdot 10^{n-p} - 0.\underbrace{\gamma_{p+1} \cdots \gamma_{k}}_{k-p} \underbrace{00 \cdots 00}_{p} \cdot 10^{n-p}
$$

$$
= \frac{0.\underbrace{\gamma_{p+1} \cdots \gamma_{k}}_{k-p} \underbrace{\gamma_{k+1} \gamma_{k+2} \cdots}_{\infty}}{x - y}
$$

$$
\Rightarrow \delta(\widetilde{x} - \widetilde{y}) \leq 0.\underbrace{0 \cdots 0}_{k-p-1} \underbrace{\gamma_{k+1} \gamma_{k+2} \cdots}_{\infty}
$$
Similarly:

\[ \delta(\tilde{x} - \tilde{y}) = \frac{0.0 \ldots 0}{k-p} \frac{\gamma_k+1 \gamma_k+2 \ldots}{\infty} \geq \frac{0.0 \ldots 0}{k-p} \frac{\gamma_k+1 \gamma_k+2 \ldots}{1.0} \].

Together:

\[ 0.0 \ldots 0 \frac{\gamma_k+1 \gamma_k+2 \ldots}{k-p} \frac{\infty}{\infty} \leq \delta(\tilde{x} - \tilde{y}) \leq 0.0 \ldots 0 \frac{\gamma_k+1 \gamma_k+2 \ldots}{k-p-1} \frac{\infty}{\infty} \]

This expresses, at best, \( k - p \) significant figures. The practical implication is that the relative error might increase substantially.
Example

Consider the following two real numbers:

\[ p = 3.145957365 \quad q = 3.145925124 \]

and their difference using 7 decimal-digit mantissa with rounding:

\[ \tilde{p} = 0.314596 \cdot 10^1 \quad \tilde{q} = 0.314593 \cdot 10^1 \]

Then:

\[ \Delta \tilde{p} = 0.000000264 \quad \Delta \tilde{q} = 0.000000488 \]

\[ \delta \tilde{p} = 0.00000008403\ldots \quad \delta \tilde{q} = 0.00000015533\ldots \]

Thus the relative error \( \leq \frac{1}{2} 10^{-6} \) as expected.
Example (continued)

The true difference is

\[ p - q = 0.000000 \underbrace{32241}_{6} \]

The approximated difference is

\[ \tilde{p} - \tilde{q} = 0.000000 \underbrace{3}_{6} \]

So in terms of errors:

\[ \Delta(\tilde{p} - \tilde{q}) = (p - q) - (\tilde{p} - \tilde{q}) = 0.000000 \underbrace{2241}_{7} \]

\[ \delta(\tilde{p} - \tilde{q}) = \frac{\Delta(\tilde{p} - \tilde{q})}{|p - q|} \approx 0.07 = 7\% \]

This is a fairly large relative error.
Being aware to the issue, many cases of significance loss can be avoided.

**Example**

Suppose we want to find the roots of \( ax^2 + bx + c = 0 \):

\[
x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
\]

Unfortunately, suppose that \( b^2 \gg 4ac \). Thus \( \sqrt{b^2 - 4ac} \approx b \); i.e.,
\(-b + \sqrt{b^2 - 4ac} \) means subtracting numbers that are too close to each other, causing loss of significance in \( x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \). A possible remedy is to use an algebraic identity obviating the need for subtracting close numbers:

\[
x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \cdot \frac{b + \sqrt{b^2 - 4ac}}{b + \sqrt{b^2 - 4ac}} = \frac{-4ac + b^2 - b^2}{2ab + 2a\sqrt{b^2 - 4ac}}
\]

\[
x_1 = \frac{-4ac - 2b}{2ab + 2a\sqrt{b^2 - 4ac}} = \frac{-2c}{b + \sqrt{b^2 - 4ac}}
\]
Version Log

- 30/10/2019, ver 1.00.