Floating Point Arithmetic

The problems of representing real numbers, rounding and the errors it introduces.

The usual way to represent real numbers in a computer is by means of a binary ‘floating point’ notation. This is similar to ‘scientific notation’ with some technical alterations to represent negative numbers and zero.

Scientific Notation

- In radix 10, the scientific notation for a positive real number a takes the form $a = b \times 10^c$ where $1 \leq b < 10$ and c is an integer.

$$3279.5524 = 3.279\ 552\ 4 \times 10^3$$

$$0.000\ 000\ 000\ 000\ 197\ 704 = 1.977\ 04 \times 10^{-13}$$

- To find the scientific notation for a number, the decimal point is moved to the left or right until the result lies between 1 and 10, which gives the value of b.
- The number of places the decimal point was moved to the left is recorded as c (and if it had to be moved to the right, c is the corresponding negative integer, or if it didn't move at all, then $c = 0$).

Exercise

Express the following numbers in scientific notation:

1. $9\,800\,270 =$
2. $0.001\,010\,1 =$
3. $2.775\,13 =$

Solution

Express the following numbers in scientific notation:

1. $9\,800\,270 = 9.80027 \times 10^{6}$
2. $0.001\,010\,1 = 1.0101 \times 10^{-3}$
3. $2.775\,13 = 2.77513 \times 10^{0}$

IEEE Floating Point Standard (32 bit)

- Given a positive number a, write both its integer and fractional parts in binary form, and transform this by moving the binary decimal point to the left or right until $a = b \times 2^c$, where $1 \leq b < 2$ and c is an integer.

  1. The first bit is 0.
  2. The next eight bits store the binary form of $c + 127$ (provided $-126 \leq c \leq 128$). These are called the exponent bits.
  3. The remaining 23 bits store the first 23 places of the fractional part of b. These are called the mantissa bits.

- For negative reals, we use 1 as the first bit instead of 0, while 0 is represented by the all zero string. The significant bits are those in the mantissa.

Machine Numbers

Given a fixed number of bits for the exponent and mantissa, there is a set $A \subseteq \mathbb{R}$ of exactly representable numbers; the elements of A are known as machine numbers. Not all real numbers are machine numbers.

- There are some whose magnitude is too big, i.e. the exponent can't be represented at the positive end; in the example, this occurs if $c > 128$. This is called exponent overflow.
- There are some whose magnitude is too small, i.e. the exponent can't be represented at the negative end; in the example, this occurs if $c < -127$. This is called exponent underflow.
- There are some whose fractional part cannot be represented in the mantissa; in the example, this occurs if the fractional part is not an integer multiple of 2^{-23}. This requires rounding.

Rounding

We define the rounding function $\mathrm{rd} : \mathbb{R} \to \mathbb{R}$ as follows, where t bits are used in the mantissa. If

$$a = b \times 2^c,$$

with $1 \leq |b| < 2$,

$$|b| = 1.\beta_1 \ldots \beta_t \beta_{t+1} \ldots \qquad \text{where } \beta_1 = 1,$$

then let

$$b' = \begin{cases} 1.\beta_1 \ldots \beta_t & \text{if } \beta_{t+1} = 0 \\ 1.\beta_1 \ldots \beta_t + 2^{-t} & \text{if } \beta_{t+1} = 1. \end{cases}$$

Then we define rd by

$$\mathrm{rd}(a) := \mathrm{sign}(a) \cdot b' \times 2^c.$$

This is just a formalisation of the usual rounding up/down rule.

Machine Precision

- The machine precision is given by eps $:= 2^{-t}$, so for 32-bit numbers, machine precision is 2^{-23}. Then

$$\mathrm{rd}(a) = a(1+\epsilon), \tag{1}$$

  where $|\epsilon| \leq \mathrm{eps} = 2^{-t}$. This expresses the fact that rounding introduces an error of relative size ϵ.

- Unfortunately the rounded value $\mathrm{rd}(x)$ may not be in A (that is, may not be a machine number) because the exponent may not fit.

- The IEEE standard mandates that if there is exponent overflow the value should be rounded to a special number Inf $\notin A$.

- If there is exponent underflow, the exponent should be frozen at its smallest value, but more zeroes allowed in the mantissa. As the mantissa becomes smaller it should be allowed to tend to zero.

Floating Point Operators

- Because floating point numbers are not the same as the real numbers (due to the limited precision), we must define new arithmetic operators: $x +^* y := \mathrm{rd}(x + y)$ and similarly for subtraction, multiplication and division.
- These elementary operations are known as flops (short for 'floating point operations').
- However, these new operations do not satisfy the usual laws of arithmetic. For example, $x +^* y = x$ if $|y| < \mathrm{eps}|x|$, for $x, y \in A$.
- Another way to view eps is that it is the smallest positive machine number that, when added to 1, gives an answer that differs from one.

$$\mathrm{eps} = \min\{g \in A \mid 1 +^* g > 1 \quad \text{and} \quad g > 0\}.$$

- In fact, pretty well any flop should be thought of as introducing $\mathrm{eps} = \epsilon_m$ in relative error.

Effect of Rounding Error

- An error of size eps may not look very big.
- For 32-bit floating point numbers this is $2^{-23} \approx 1.19 \times 10^{-7}$, while for double precision numbers this is $2^{-52} \approx 2.22 \times 10^{-16}$.
- However, during a long computation rounding errors can accumulate until the answer is so inaccurate as to be useless.

Now read the example.

Comment on Example

- Ironically, the fact that some programmers had noticed this problem and had 'improved' their part of the code, while others had not, meant that the inaccuracies did not cancel and made the problem significantly worse.
- Using the time since booting the system makes the problem much worse, since the rounding error on 1/10 is multiplied by a large value. There seems to be no good reason for this choice.

Effect of Rounding Error

- We shall write u for the true real number, and $\tilde{u}$ for the approximation introduced by rounding (and similarly for v).
- The absolute error in u is

$$\Delta u := \tilde{u} - u \tag{2}$$

- The relative error in u is

$$\epsilon_u := \frac{\Delta u}{u}, \tag{3}$$

  if $u \neq 0$.
- We are interested in the relationship between the relative errors of the inputs (u and v) and the relative error of the output.

Exercise: Calculation of Errors

Suppose that our computer uses floating point arithmetic to five decimal places. If $u = 0.37214\,48693$, what is $\tilde{u}$, the rounded version of u? What are the absolute and relative error in u?

Exercise: Calculation of Errors

Suppose that our computer uses floating point arithmetic to five decimal places. If $u = 0.37214\,48693$, what is $\tilde{u}$, the rounded version of u? What are the absolute and relative error in u?

Answer:

$$\tilde{u} = 0.37214$$

$$\Delta u := \tilde{u} - u = -0.00000\,48693 = -4.8693 \times 10^{-6}$$

$$\epsilon_u := \frac{\Delta u}{u} = \frac{-4.8693 \times 10^{-6}}{0.37214\,48693} \approx -1.3084 \times 10^{-5}.$$

Rounding Error Analysis of Multiplication

- $y = f(u, v) := u \times v$. Then

$$\epsilon_y \doteq \epsilon_u + \epsilon_v.$$

- This means that the relative error in the result is the sum of the relative errors on the inputs. Thus if we do a lot of multiplications, the error increases slowly.
- The technical term for this is that multiplication is well-conditioned.

Rounding Error Analysis of Addition

- $y = f(u, v) := u + v$. Then

$$\epsilon_y \doteq \frac{u}{u+v}\epsilon_u + \frac{v}{u+v}\epsilon_v,$$

  if $u + v \neq 0$.

- This is OK if u and v have the same sign. In this case, $|\epsilon_{u+v}| \leq \max\{|\epsilon_u|, |\epsilon_v|\}$ and the function is well-conditioned (the errors are damped).

- If u and v have opposite signs, at least one of

$$\left|\frac{u}{u+v}\right| \qquad \text{and} \qquad \left|\frac{v}{u+v}\right| > 1,$$

  and so errors are magnified; we say that the function is ill-conditioned. The closer to 0 the value of $u + v$ lies, the worse conditioned the computation.

Subtractive Cancellation

- We can create catastrophically bad errors with addition; this effect is called subtractive cancellation.
- Consider a machine with floating point arithmetic accurate to 5 decimal places. Let $u = 0.37214\,48693$ and $v = 0.37202\,14371$. Then the machine numbers are $\tilde{u} = 0.37214$ and $\tilde{v} = 0.37202$.
- It follows that

$$\tilde{u} - \tilde{v} = 0.00012 \qquad u - v = 0.00012\,34322.$$

We can see that the computed result only contains 2 significant figures, not the full five. This is confirmed by the size of the relative error

$$\frac{|\tilde{u} - \tilde{v}|}{|u - v|} \approx 3 \times 10^{-2}.$$

Mathematics Support Office

- MB154
- Staffed by 2 tutors
- Open 10:00 until 17:00, Monday to Friday
- Closed some lunchtimes

Summary

1. The IEEE standard floating point notation for a real number a is $a = b \times 2^c$, where $1 \leq b < 2$ and c is an integer. b is the mantissa and c is the exponent.
2. Given a fixed number of bits for the exponent and mantissa, there is a set a of exactly representable machine numbers; the elements of A are known as machine numbers.
3. Rounding is used to convert general real numbers to machine numbers.
4. Machine precision is given by 2^{-t}, where t bits are used in the mantissa. For 32-bit numbers, machine precision is 2^{-23}.
5. Floating point operations (flops) are defined by rounding mathematical operations, e.g. $x +^* y := \mathrm{rd}(x + y)$.
6. Multiplication is well-conditioned: relative errors of the arguments are summed.
7. Addition is ill-conditioned: subtractive cancellation can cause catastrophic errors.