September 2, 2014

Chapter 2: Roundoff Errors

Uri M. Ascher and Chen Greif
Department of Computer Science
The University of British Columbia
{ascher,greif}@cs.ubc.ca

Slides for the book
A First Course in Numerical Methods (published by SIAM, 2011)
`http://bookstore.siam.org/cs07/`

Goals of this chapter

- To describe how numbers are stored in a floating point system;
- to understand how standard floating point systems are designed and implemented;
- to get a feeling for the almost random nature of rounding error;
- to identify different sources of roundoff error growth and explain how to dampen their cumulative effect.

Roundoff errors

- Roundoff error is generally inevitable in numerical algorithms involving real numbers.
- People often like to pretend they work with exact real numbers, ignoring roundoff errors, which may allow concentration on other algorithmic aspects.
- However, carelessness may lead to disaster!
- This chapter provides an overview of roundoff errors, including *floating point number representation, rounding error and arithmetic, the IEEE standard, and roundoff error accumulation*.

Outline

- Floating point systems
- The IEEE standard
- Roundoff error accumulation

Real number representation: decimal

$$\frac{8}{3} \simeq \left(\frac{2}{10^0} + \frac{6}{10^1} + \frac{6}{10^2} + \frac{6}{10^3} \right) \times 10^0 = 2.666 \times 10^0.$$

An instance of the floating point representation

$$\begin{aligned} \mathrm{fl}(x) &= \pm d_0.d_1 \cdots d_{t-1} \times 10^e \\ &= \pm \left(\frac{d_0}{10^0} + \frac{d_1}{10^1} + \cdots + \frac{d_{t-2}}{10^{t-2}} + \frac{d_{t-1}}{10^{t-1}} \right) \times 10^e \end{aligned}$$

for $t = 4, \ e = 0$.
Note that $d_0 > 0$: normalized floating point representation.

Real number representation: binary

The decimal system is convenient for humans; but computers prefer binary.

- In binary the (normalized) representation of a real number x is

$$\begin{aligned} x &= \pm(1.d_1 d_2 d_3 \cdots d_{t-1} d_t d_{t+1} \cdots) \times 2^e \\ &= \pm(1 + \frac{d_1}{2} + \frac{d_2}{4} + \frac{d_3}{8} + \cdots) \times 2^e, \end{aligned}$$

  with binary digits $d_i = 0$ or 1 and exponent e.

- Floating point representation: with a fixed number of digits t

$$\mathrm{fl}(x) = \pm(1.\tilde{d}_1 \tilde{d}_2 \tilde{d}_3 \cdots \tilde{d}_{t-1} \tilde{d}_t) \times 2^e$$

- How to determine digits $\tilde{d}_i$?
  A popular strategy is **Rounding**:

$$\mathrm{fl}(x) = \begin{cases} \pm\, 1.d_1 d_2 d_3 \cdots d_t \times 2^e & d_{t+1} = 0 \\ \text{to nearest even} & \text{otherwise} \end{cases}.$$

  Alternatively, **Chopping** simply sets $\tilde{d}_i = d_i, \; i = 1, \ldots, t$.

General floating point system

defined by (β, t, L, U), where:
β: base of the number system (for binary, $\beta = 2$; for decimal, $\beta = 10$);
t: precision (number of digits);
L: lower bound on exponent e;
U: upper bound on exponent e.

For each $x \in \mathbb{R}$ corresponds a normalized floating point representation

$$\mathrm{fl}(x) = \pm \left(\frac{d_0}{\beta^0} + \frac{d_1}{\beta^1} + \cdots + \frac{d_{t-1}}{\beta^{t-1}} \right) \times \beta^e,$$

where $0 \leq d_i \leq \beta - 1$, $d_0 > 0$, and $L \leq e \leq U$.

Example

$(\beta, t, L, U) = (10, 4, -2, 1)$.
Largest number is $9.999 \times 10^U = 99.99 \lessapprox 10^{U+1} = 100$
Smallest positive number is $1.000 \times 10^L = 10^L = 0.01$
Total different fractions: $(\beta - 1) \times \beta^{t-1} = 9 \times 10 \times 10 \times 10 = 9,000$
Total different exponents: $U - L + 1 = 4$
Total different positive numbers: $4 \times 9,000 = 36,000$
Total different numbers in system: 72,001

Error in floating point number representation

For the real number $x = \pm\, d_0.d_1d_2d_3\cdots d_{t-1}d_td_{t+1}\cdots \times \beta^e$,

- Chopping:

$$\mathrm{fl}(x) = \pm\, d_0.d_1d_2d_3\cdots d_{t-1} \times \beta^e$$

  Then absolute error is clearly bounded by $\beta^{1-t}\cdot\beta^e$.

- Rounding:

$$\mathrm{fl}(x) = \begin{cases} \pm\, d_0.d_1d_2d_3\cdots d_{t-1} \times \beta^e & d_t < \beta/2 \\ \pm\left(d_0.d_1d_2d_3\cdots d_{t-1} + \beta^{1-t}\right) \times \beta^e & d_t > \beta/2 \end{cases},$$

  round to even in case of a tie.
  Then absolute error is bounded by half the above, $\frac{1}{2}\cdot\beta^{1-t}\cdot\beta^e$.
  So relative error is bounded by **rounding unit**

$$\eta = \frac{1}{2}\cdot\beta^{1-t}.$$

Floating point arithmetic

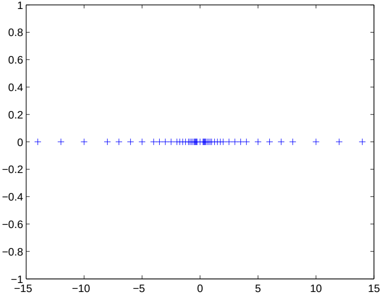
- Important to use **exact rounding**: if $\mathrm{fl}(x)$ and $\mathrm{fl}(y)$ are machine numbers, then

$$\begin{aligned} fl(fl(x) \pm fl(y)) &= (fl(x) \pm fl(y))(1+\epsilon_1), \\ fl(fl(x) \times fl(y)) &= (fl(x) \times fl(y))(1+\epsilon_2), \\ fl(fl(x)/fl(y)) &= (fl(x)/fl(y))(1+\epsilon_3), \end{aligned}$$

  where $|\epsilon_i| \le \eta$.
- Thus, the *relative errors* remain small after each such operation.
- This is achieved using guard digits (see Example 2.6 in text).

Spacing of floating point numbers

Run program Example2_8Figure2_3



Note the **uneven** distribution, both for large exponents and near 0.

Overflow, underflow, NaN

- Overflow: when $e > U$. (fatal)
- Underflow: when $e < L$. (non-fatal: set to 0 by default)
- NaN: Not-a-number. (e.g., $0/0$)

Outline

- Floating point systems
- The IEEE standard
- Roundoff error accumulation

IEEE standard

- Used by everyone today.
- Binary: use $\beta = 2$.
- Exact rounding: use guard digits to ensure that relative error in each elementary arithmetic operation is bounded by η.
- `NaN`
- Overflow and underflow
- Subnormal numbers near 0.
- Many other features...

IEEE standard word

Use base $\beta = 2$.
Recall that with this base, in normalized numbers the first digit is always $d_0 = 1$ and thus it need not be stored.

Double precision (64 bit word)

$s = \pm$	$b =11$-bit exponent	$f =52$-bit fraction

Rounding unit:

$$\eta = \frac{1}{2} \cdot 2^{-52} \approx 1.1 \times 10^{-16}$$

Can have also single precision (32 bit word).
Then $t = 23$ and $\eta = 2^{-24} \approx 6.0 \times 10^{-8}$.

Outline

- Floating point systems
- The IEEE standard
- Roundoff error accumulation

Roundoff error accumulation

- In general, if E_n is error after n elementary operations, cannot avoid linear roundoff error accumulation

$$E_n \simeq c_0 n E_0.$$

- Will not tolerate an **exponential** error growth such as

$$E_n \simeq c_1^n E_0 \quad \text{for some constant} \quad c_1 > 1$$

  – an **unstable algorithm**.
- In some situations an individual error contribution is particularly large and occasionally can be made smaller.

Cancellation error

When two nearby numbers are subtracted, the relative error is large. That is, if $x \simeq y$, then $x - y$ has a large relative error.
This occurs in practice consistently and naturally, as we will see.

Function evaluation at nearby arguments:

- If $g(\cdot)$ is a smooth function then $g(t)$ and $g(t+h)$ are close for h small.
- But rounding errors in $g(t)$ and $g(t+h)$ are unrelated, so they can be of opposing signs!
- For numerical differentiation, e.g.

$$f'(x_0) \approx \frac{f(x_0+h) - f(x_0)}{h}, \quad 0 < h \ll 1,$$

  if the relative rounding error in the representation is bounded by η then in $|g(t+h) - g(t)|/h$ it is bounded by $2\eta/h$. This (tight) bound is much larger than η when h is small.

Example

Compute $y = \sinh(x) = \frac{1}{2}(e^x - e^{-x})$.

- Naively computing y at an x near 0 may result in a (meaningless) 0.
- Instead use Taylor's expansion

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \dots$$

to obtain

$$\sinh(x) = x + \frac{x^3}{6} + \dots.$$

- If x is near 0, can use $x + \frac{x^3}{6}$, or even just x, for an effective approximation to $\sinh(x)$.

So, a good library function would compute $\sinh(x)$ by the regular formula (using exponentials) for $|x|$ not very small, and by taking a term or two of the Taylor expansion for $|x|$ very small.

Illustration

Compute $y = \sqrt{x+1} - \sqrt{x}$ for $x = 100,000$ in a 5-digit decimal arithmetic. (So $\beta = 10,\ t = 5$.)

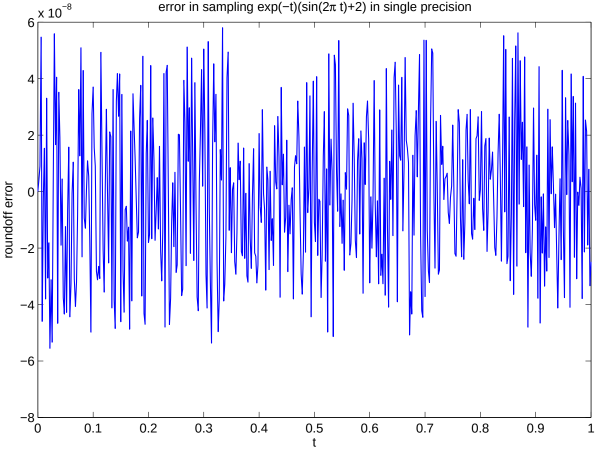
- Naively computing $\sqrt{x+1} - \sqrt{x}$ results in the value 0.
- Instead use the identity

$$\frac{(\sqrt{x+1} - \sqrt{x})(\sqrt{x+1} + \sqrt{x})}{(\sqrt{x+1} + \sqrt{x})} = \frac{1}{\sqrt{x+1} + \sqrt{x}}.$$

- In 5-digit decimal arithmetic calculating the right hand side expression yields 1.5811×10^{-3}: correct in the given accuracy.

The rough appearance of roundoff errors

Run program Example2_2Figure2_2.m



Note how the sign of the floating point representation error at nearby arguments t fluctuates as if randomly: as a function of t it is a "non-smooth" error.

Avoiding overflow

- 2-norm computation: for a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ with n components,

$$\|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}.$$

  The vector $\mathbf{x}$ may have many components: accumulating damage may be fatal.

- Suppose $a \gg b$ and we wish to compute $c = \sqrt{a^2 + b^2}$. Then it may be better to compute

$$c = a\sqrt{1 + (b/a)^2}.$$

- The norm calculation can be scaled in a similar manner.