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
http://www.ec-securehost.com/SIAM/CS07.html
Goals of this chapter

- To describe how numbers are stored in a floating point system;
- 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.
Outline

- The essentials (We will do only this)
- Floating point systems
- Roundoff error accumulation
- The IEEE standard
Roundoff errors are 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 two options for studying roundoff errors: The essentials: just enough to know what issues to expect. In this course we will take this option. The fuller version.
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 two options for studying roundoff errors:
  - The essentials: just enough to know what issues to expect.
  - In this course we will take this option.
  - The fuller version.
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 two options for studying roundoff errors:
  - **The essentials**: just enough to know what issues to expect.
    - **In this course we will take this option**.
  - The fuller version.
We will consider:

- Real number representation – floating point system
- Rounding unit
- IEEE standard
- Roundoff error accumulation
- Rough appearance of roundoff errors
Real number representation: decimal

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

An instance of the floating point representation

\[ \mathfrak{f}(x) = \pm d_0.d_1 \cdots d_{t-1} \times 10^e \]

\[ = \pm \left( \frac{d_0}{10^0} + \frac{d_1}{10^1} + \cdots + \frac{d_{t-2}}{10^{t-2}} + \frac{d_{t-1}}{10^{t-1}} \right) \times 10^e \]

for \( t = 4, \ e = 0. \)

Note that \( d_0 > 0: \) normalized floating point representation.
Real number representation: decimal

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

An instance of the floating point representation

\[ \Phi(x) = \pm d_0.d_1 \cdots d_{t-1} \times 10^e \]

\[ = \pm \left( \frac{d_0}{10^0} + \frac{d_1}{10^1} + \cdots + \frac{d_{t-2}}{10^{t-2}} + \frac{d_{t-1}}{10^{t-1}} \right) \times 10^e \]

for \( t = 4, \ e = 0. \)

Note that \( d_0 > 0: \) normalized floating point representation.
Real number representation: decimal

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

An instance of the floating point representation

\[ \Phi(x) = \pm d_0.d_1 \cdots d_{t-1} \times 10^e \]

\[ = \pm \left( \frac{d_0}{10^0} + \frac{d_1}{10^1} + \cdots + \frac{d_{t-2}}{10^{t-2}} + \frac{d_{t-1}}{10^{t-1}} \right) \times 10^e \]

for \( t = 4, \ e = 0. \)

Note that \( d_0 > 0: \) normalized floating point representation.
The decimal system is convenient for humans; but computers prefer binary.

- **In binary** the (normalized) representation of a real number $x$ is
  \[
  x = \pm (1.d_1d_2d_3\cdots d_{t-1}d_t d_{t+1}\cdots) \times 2^e
  = \pm (1 + \frac{d_1}{2} + \frac{d_2}{4} + \frac{d_3}{8} + \cdots) \times 2^e,
  \]
  with binary digits $d_i = 0$ or $1$ and exponent $e$.

- **Floating point representation**: with a fixed number of digits $t$
  \[
  \text{fl}(x) = \pm (1.\tilde{d}_1\tilde{d}_2\tilde{d}_3\cdots \tilde{d}_{t-1}\tilde{d}_t) \times 2^e
  \]
The decimal system is convenient for humans; but computers prefer binary.

- In **binary** the (normalized) representation of a real number $x$ is
  \[
  x = \pm (1.d_1d_2d_3 \cdots d_{t-1}d_t d_{t+1} \cdots) \times 2^e
  = \pm (1 + \frac{d_1}{2} + \frac{d_2}{4} + \frac{d_3}{8} + \cdots) \times 2^e,
  \]
  with binary digits $d_i = 0$ or $1$ and exponent $e$.

- **Floating point representation**: with a fixed number of digits $t$
  \[
  \text{fl}(x) = \pm (1.\tilde{d}_1\tilde{d}_2\tilde{d}_3 \cdots \tilde{d}_{t-1}\tilde{d}_t) \times 2^e
  \]
Determining digits

How to determine digits $\tilde{d}_i$?

**Rounding:**

$$\text{fl}(x) = \begin{cases} 
\pm 1.d_1d_2d_3 \cdots d_t \times 2^e & d_{t+1} = 0 \\
\text{to nearest even} & \text{otherwise}
\end{cases}.$$

Then the relative floating point error is bounded by rounding unit

$$\frac{|\text{fl}(x) - x|}{|x|} \leq \frac{1}{2} \cdot 2^{-t}.$$ 

Recommendation: prove this important bound!
How to determine digits $\tilde{d}_i$?

**Rounding:**

\[
fl(x) = \begin{cases} 
\pm 1.d_1d_2d_3 \cdots d_t \times 2^e & d_{t+1} = 0 \\
\text{to nearest even} & \text{otherwise}
\end{cases}
\]

Then the relative floating point error is bounded by rounding unit

\[
\frac{|fl(x) - x|}{|x|} \leq \frac{1}{2} \cdot 2^{-t}.
\]

Recommendation: prove this important bound!
Determining digits

How to determine digits $\tilde{d}_i$?

**Rounding:**

$$\text{fl}(x) = \begin{cases} 
\pm 1.d_1d_2d_3 \cdots d_t \times 2^e & d_{t+1} = 0 \\
to nearest even & otherwise
\end{cases}.$$ 

Then the relative floating point error is bounded by rounding unit

$$\frac{|\text{fl}(x) - x|}{|x|} \leq \frac{1}{2} \cdot 2^{-t}.$$ 

Recommendation: prove this important bound!
Determining digits

How to determine digits $\tilde{d}_i$?

**Rounding:**

$$\text{fl}(x) = \begin{cases} 
\pm 1.d_1d_2d_3 \cdots d_t \times 2^e & d_{t+1} = 0 \\
to \text{ nearest even} & \text{otherwise}
\end{cases}.$$

Then the relative floating point error is bounded by rounding unit

$$\frac{|\text{fl}(x) - x|}{|x|} \leq \frac{1}{2} \cdot 2^{-t}.$$ 

Recommendation: prove this important bound!
IEEE standard word

Double precision (64 bit word)

\[
\begin{array}{|c|c|c|}
\hline
s &= \pm \\
\hline
b &= 11 \text{-bit exponent} \\
\hline
f &= 52 \text{-bit fraction} \\
\hline
\end{array}
\]

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 \( \eta = 2^{-24} \approx 6.0 \times 10^{-8} \).
IEEE standard word

Double precision (64 bit word)

\[ s = \pm \quad b = 11 \text{-bit exponent} \quad f = 52 \text{-bit fraction} \]

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 \( \eta = 2^{-24} \approx 6.0 \times 10^{-8} \).
IEEE standard word

Double precision (64 bit word)

\[ s = \pm \quad b = \text{11-bit exponent} \quad f = \text{52-bit fraction} \]

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 \( \eta = 2^{-24} \approx 6.0 \times 10^{-8} \).
Comparing single and double precision

If we represent the number 1/3 in IEEE single precision (32 bits), the error will be approximately how many times larger than if we represent the same number in IEEE double precision (64 bits)?

- $2^{29} \approx 5.37 \times 10^8$
- 32
- a little over 2
- 1 (the error will be the same)
IEEE standard

- Used by everyone today.
- **Exact rounding**: use guard digits to ensure that relative error in each elementary arithmetic operation is bounded by $\eta$.
- NaN
- Overflow and underflow
- Subnormal numbers near 0.
- Many other features...
Roundoff error accumulation

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

  $$E_n \approx c_0 n E_0.$$ 

- Will not tolerate an exponential error growth such as

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

  - an unstable algorithm.

- In some situations an individual error contribution is particularly large and occasionally can be made smaller.
Roundoff error accumulation

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

$$E_n \sim c_0 n E_0.$$  

• Will not tolerate an exponential error growth such as

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

– 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.
Naturally occurs in practice.

Instance:

- 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!
- Recall numerical differentiation example from Chapter 1: if the relative error in the representation is bounded by \( \eta \) then in \( |g(t + h) - g(t)/h| \) it is bounded by \( 2\eta/h \). This (tight) bound is much larger than \( \eta \) when \( h \) is small.
Cancellation error

When two nearby numbers are subtracted, the relative error is large. Naturally occurs in practice.

Instance:
- 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!
- Recall numerical differentiation example from Chapter 1: if the relative error in the representation is bounded by $\eta$ then in $|g(t + h) - g(t)/h$ it is bounded by $2\eta/h$. This (tight) bound is much larger than $\eta$ when $h$ is small.
Cancellation error

When two nearby numbers are subtracted, the relative error is large. Naturally occurs in practice.

Instance:

- 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!
- Recall numerical differentiation example from Chapter 1: if the relative error in the representation is bounded by $\eta$ then in $|g(t+h) - g(t)/h|$ it is bounded by $2\eta/h$. This (tight) bound is much larger than $\eta$ 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

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

...to obtain

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

- 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.
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

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

to obtain

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

- 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.
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

\[
e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \ldots
\]


to obtain

\[
\sinh(x) = x + \frac{x^3}{6} + \ldots.
\]

- 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.
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

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

to obtain

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

- 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.
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

\[
e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \ldots
\]

to obtain

\[
\sinh(x) = x + \frac{x^3}{6} + \ldots
\]

- 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.
Limiting roundoff error accumulation

We are supposed to calculate $\sqrt{x+1} - \sqrt{x}$ for $x \gg 1$. We realize that $\sqrt{x+1} - \sqrt{x} = \frac{1}{\sqrt{x+1}+\sqrt{x}}$. Which formula should we use for the computation?

- $\sqrt{x+1} - \sqrt{x}$.
- $\frac{1}{\sqrt{x+1}+\sqrt{x}}$.
- Neither.
- It does not matter which one: the error will be the same.
Example: 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.