# Numerical Analysis

## Chapter Two: Computer Arithmetic

主讲人 朱亚菲
幻灯片制作 朱亚菲

中国海洋大学 信息科学与工程学院

2013 年 9 月 13 日

## Contents

1. 2.0 Introduction
2. 2.1 Floating-point Numbers and Roundoff Errors
   - 2.1.1 Rounding
   - 2.1.2 Normalized Scientific Notation
   - 2.1.3 Hypothetical Computer Marc-32
   - 2.1.4 Zero,Infinity,NaN
   - 2.1.5 Machine Rounding
   - 2.1.6 Intrinsic Procedures in Fortran 90
   - 2.1.7 IEEE Standard Floating-Point Arithmetic
   - 2.1.8 Nearby Machine Numbers
   - 2.1.9 Floating-Point Error Analysis
   - 2.1.10 Relative Error Analysis

## Contents

1 2.0 Introduction

2 2.1 Floating-point Numbers and Roundoff Errors
- 2.1.1 Rounding
- 2.1.2 Normalized Scientific Notation
- 2.1.3 Hypothetical Computer Marc-32
- 2.1.4 Zero,Infinity,NaN
- 2.1.5 Machine Rounding
- 2.1.6 Intrinsic Procedures in Fortran 90
- 2.1.7 IEEE Standard Floating-Point Arithmetic
- 2.1.8 Nearby Machine Numbers
- 2.1.9 Floating-Point Error Analysis
- 2.1.10 Relative Error Analysis

## 2.0 Introduction

- In this chapter, we are going to:
  - Describe the floating-point number system;
  - Develop basic facts about roundoff errors, which may contaminate computer calculations;
  - Discuss other types of errors and loss of significance;
  - Survey some stable/unstable algorithms and ill-conditioned problems.

## 2.0 Introduction

Most computers deal with real numbers in the binary number system, in contrast to the decimal number system that humans prefer to use. The binary system uses 2 as the base in the same way that the decimal system uses 10.
In the binary system, only the two digits 0 and 1 are used. A typical number in the binary system can be written out in detail; for example,

$$(1001.11101)_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5}$$

## 2.0 Introduction

Since the typical computer works internally in the binary system but communicates with its human users in the decimal system, conversion procedures must be executed by the computer. These come into use at input and output time. Ordinarily the user need not be concerned with these conversions, but they do involve small roundoff errors.

## 2.0 Introduction

Computers are not able to operate using real numbers expressed with more than a fixed number of digits. Even a simple number like 1/10 cannot be stored exactly in any binary machines. It requires an infinite binary expression:

$$\frac{1}{10} = (0.00011001100110011\ldots)_2$$

## 2.0 Introduction

For example, if we read 0.1 into a 32-bit computer workstation and then print it out to 40 decimal places, we obtain the following results:

$$0.1000000014901161193847656250000000000000$$

Usually, we won't notice this conversion error since printing using the default fotmat would show us 0.1.

## Contents

1 2.0 Introduction

2 2.1 Floating-point Numbers and Roundoff Errors

- 2.1.1 Rounding
- 2.1.2 Normalized Scientific Notation
- 2.1.3 Hypothetical Computer Marc-32
- 2.1.4 Zero,Infinity,NaN
- 2.1.5 Machine Rounding
- 2.1.6 Intrinsic Procedures in Fortran 90
- 2.1.7 IEEE Standard Floating-Point Arithmetic
- 2.1.8 Nearby Machine Numbers
- 2.1.9 Floating-Point Error Analysis
- 2.1.10 Relative Error Analysis

## Contents

1 2.0 Introduction

2 2.1 Floating-point Numbers and Roundoff Errors
- 2.1.1 Rounding
- 2.1.2 Normalized Scientific Notation
- 2.1.3 Hypothetical Computer Marc-32
- 2.1.4 Zero,Infinity,NaN
- 2.1.5 Machine Rounding
- 2.1.6 Intrinsic Procedures in Fortran 90
- 2.1.7 IEEE Standard Floating-Point Arithmetic
- 2.1.8 Nearby Machine Numbers
- 2.1.9 Floating-Point Error Analysis
- 2.1.10 Relative Error Analysis

## 2.1.1 Rounding

Consider a positive decimal number x of the form $0.\square\square\square\ldots\square\square\square$ with m digits to the right of the decimal point.
One rounds x to n desimal places (n<m) in a manner that depends on the value of the (n+1)st digit.
If this digit is 0, 1, 2, 3, or 4, then the nth digit is not changed and all following digits are discarded.
If it is a 5, 6, 7, 8, or 9,then the nth digit is increased by one unit and the remaining digits are discarded.

## 2.1.1 Rounding

Here are some examples of seven-digit numbers being correctly rounded to four digits:

$$0.1735 \longleftarrow 0.1735499$$

$$1.000 \longleftarrow 0.9999500$$

$$0.4322 \longleftarrow 0.4321609$$

## 2.1.1 Rounding

If x is rounded so that $\widetilde{x}$ is the n-digit approximation to it, then

$$|x - \widetilde{x}| \leqslant \frac{1}{2} \times 10^{-n}$$

Proof:

- If the (n+1)st digit of x is 0, 1, 2, 3, or 4, then $x = \widetilde{x} + \varepsilon$ with $\varepsilon < \frac{1}{2} \times 10^{-n}$ and the Inequality follows.
- If it is 5, 6, 7, 8, or 9, then $\widetilde{x} = \widehat{x} + 10^{-n}$ where $\widehat{x}$ is a number with the same n digits as x and all digits beyond the nth are 0. Now $x = \widehat{x} + \delta \times 10^{-n}$ with $\delta \geqslant \frac{1}{2}$ and $\widetilde{x} - x = (1 - \delta) \times 10^{-n}$. Since $1 - \delta \leqslant \frac{1}{2}$, the Inequality follows.

## 2.1.1 Rounding

If $x$ is a decimal number, the chopped or truncated $n$-digit approximation to it is the number $\widehat{x}$ obtained by simply discarding all digits beyond the $n$th. For it, we have

$$|x - \widehat{x}| < 10^{-n}$$

The relationship between $x$ and $\widehat{x}$ is that $x - \widehat{x}$ has 0 in the first $n$ places and $x = \widehat{x} + \delta \times 10^{-n}$ and the Inequality follows.

## Contents

1 2.0 Introduction

2 2.1 Floating-point Numbers and Roundoff Errors
- 2.1.1 Rounding
- 2.1.2 Normalized Scientific Notation
- 2.1.3 Hypothetical Computer Marc-32
- 2.1.4 Zero,Infinity,NaN
- 2.1.5 Machine Rounding
- 2.1.6 Intrinsic Procedures in Fortran 90
- 2.1.7 IEEE Standard Floating-Point Arithmetic
- 2.1.8 Nearby Machine Numbers
- 2.1.9 Floating-Point Error Analysis
- 2.1.10 Relative Error Analysis

## 2.1.2 Normalized Scientific Notation

We can use scientific notation in the binary system. Now we have

$$x = \pm q \times 2^m$$

where $\frac{1}{2} \leqslant q < 1$(if $x \neq 0$) and m is an integer. The number $q$ is called the mantissa and the integer $m$ the exponent. Both $q$ and $m$ are base 2 numbers.

## Contents

1 2.0 Introduction

2 2.1 Floating-point Numbers and Roundoff Errors
- 2.1.1 Rounding
- 2.1.2 Normalized Scientific Notation
- 2.1.3 Hypothetical Computer Marc-32
- 2.1.4 Zero,Infinity,NaN
- 2.1.5 Machine Rounding
- 2.1.6 Intrinsic Procedures in Fortran 90
- 2.1.7 IEEE Standard Floating-Point Arithmetic
- 2.1.8 Nearby Machine Numbers
- 2.1.9 Floating-Point Error Analysis
- 2.1.10 Relative Error Analysis

## 2.1.3 Hypothetical Computer Marc-32

The bits composing a word in the Marc-32 are allocated in the following way when representing a nonzero real number $x = \pm q \times 2^m$:

$$sign\ of\ the\ real\ number\ x \quad 1bit$$

$$biased\ exponent\ (integer\ e) \quad 8bits$$

$$mantissa\ part\ (real\ number\ f) \quad 23bits$$

## 2.1.3 Hypothetical Computer Marc-32

Nonzero normalized machine numbers are bit strings whose values are decoded as follows:

$$x = (-1)^s q \times 2^m$$

where

$$q = (1.f)_2 \quad and \quad m = e - 127$$

Here $1 \leqslant q < 2$ and the most significant bit in $q$ is 1 and is not explicitly stored. Also, here s is the bit representing the sign of $x$ (positive: bit 0, negative: bit 1), $m = e - 127$ is the 8-bit biased exponent, and f is the 23-bit fractional part of the real number $x$ that, together with an implicit leading bit 1, yields the significant digit field $(1.\square\square\square\ldots\square\square\square)_2$.

## 2.1.3 Hypothetical Computer Marc-32

The restriction that $|m|$ require no more than 8 bits means that

$$0 < e < (11111111)_2 = 2^8 - 1 = 255$$

and the values e=0 and e=255 are reserved for special cases such as $\pm 0, \pm\infty$, and NaN (Not a Number). Since $m = e - 127$, we take $-126 \leqslant m \leqslant 127$ and the Marc-32 can handle numbers as small as $2^{-126} \approx 1.2 \times 10^{-38}$ and as large as $(2 - 2^{-23})2^{127} \approx 3.4 \times 10^{38}$. This is not a sufficiently generous range of magnitudes for some scientific calculations, and for this reason and others, we occasionally must write a program in double-precision or extended-precision arithmatic.

## Contents

1 2.0 Introduction

2 2.1 Floating-point Numbers and Roundoff Errors
- 2.1.1 Rounding
- 2.1.2 Normalized Scientific Notation
- 2.1.3 Hypothetical Computer Marc-32
- 2.1.4 Zero,Infinity,NaN
- 2.1.5 Machine Rounding
- 2.1.6 Intrinsic Procedures in Fortran 90
- 2.1.7 IEEE Standard Floating-Point Arithmetic
- 2.1.8 Nearby Machine Numbers
- 2.1.9 Floating-Point Error Analysis
- 2.1.10 Relative Error Analysis

## 2.1.4 Zero,Infinity,NaN

In IEEE standard arithmetic, there are two forms of 0,+0 and -0, in single precisions, represented in the computer by the words $[00000000]_{16}$ and $[80000000]_{16}$, respectively.
Similarly, there are two forms of infinity, $+\infty$ and $-\infty$, in single precision, represented by the computer words $[7F800000]_{16}$ and $[FF800000]_{16}$, respectively.
NaN means Not a Number and results from an indeterminate operation such as $0/0, \infty - \infty, x + NaN$, and so on. NaN's are represented by computer words with e=255 and $f \neq 0$.

## Contents

1 2.0 Introduction

2 2.1 Floating-point Numbers and Roundoff Errors
- 2.1.1 Rounding
- 2.1.2 Normalized Scientific Notation
- 2.1.3 Hypothetical Computer Marc-32
- 2.1.4 Zero,Infinity,NaN
- 2.1.5 Machine Rounding
- 2.1.6 Intrinsic Procedures in Fortran 90
- 2.1.7 IEEE Standard Floating-Point Arithmetic
- 2.1.8 Nearby Machine Numbers
- 2.1.9 Floating-Point Error Analysis
- 2.1.10 Relative Error Analysis

## 2.1.5 Machine Rounding

The usual (default) rounding mode is round to nearest: The closer of the two machine numbers on the left and right of the real number is selected. In the case of a tie, round to even is uesd: If the real number is exactly halfway between the machine numbers to its left and right, then the even machine number is chosen.
With this default rounding scheme (round to nearest plus round to even), the maximum error is half a unit in the least significant place.

## Contents

1 2.0 Introduction

2 2.1 Floating-point Numbers and Roundoff Errors
- 2.1.1 Rounding
- 2.1.2 Normalized Scientific Notation
- 2.1.3 Hypothetical Computer Marc-32
- 2.1.4 Zero,Infinity,NaN
- 2.1.5 Machine Rounding
- 2.1.6 Intrinsic Procedures in Fortran 90
- 2.1.7 IEEE Standard Floating-Point Arithmetic
- 2.1.8 Nearby Machine Numbers
- 2.1.9 Floating-Point Error Analysis
- 2.1.10 Relative Error Analysis

## 2.1.6 Intrinsic Procedures in Fortran 90

| | |
|---|---|
| digits | number of significant (binary) digits. |
| epsilon | a positive number that is almost negligible compared to unity. |
| huge | largest number. |
| maxexponent | maximum (binary) exponent. |
| minexponent | minimum (most negative, binary) exponent. |
| precision | decimal precion. |
| radix | the base for the computer floating-point number system. |
| range | decimal exponent range. |
| tiny | smallest positive number. |

## Contents

1 2.0 Introduction

2 2.1 Floating-point Numbers and Roundoff Errors
- 2.1.1 Rounding
- 2.1.2 Normalized Scientific Notation
- 2.1.3 Hypothetical Computer Marc-32
- 2.1.4 Zero,Infinity,NaN
- 2.1.5 Machine Rounding
- 2.1.6 Intrinsic Procedures in Fortran 90
- 2.1.7 IEEE Standard Floating-Point Arithmetic
- 2.1.8 Nearby Machine Numbers
- 2.1.9 Floating-Point Error Analysis
- 2.1.10 Relative Error Analysis

## 2.1.7 IEEE Standard Floating-Point Arithmetic

The Marc-32 representation for real numbers is patterned after the usual floating-point representation in 32-bit machines, which is the IEEE standard floating-point representation. We have chosen to give only a brief description here. For example, computers that implement floating-point arithmetic according to the current officail standard use 80 bits for internal calculations. There are many additional concerpts —guard bit, denormalized numbers, unnormalized numbers, double rounding, and others—that enter into any detailed discussion of this subjects.

## Contents

1 2.0 Introduction
2 2.1 Floating-point Numbers and Roundoff Errors
- 2.1.1 Rounding
- 2.1.2 Normalized Scientific Notation
- 2.1.3 Hypothetical Computer Marc-32
- 2.1.4 Zero,Infinity,NaN
- 2.1.5 Machine Rounding
- 2.1.6 Intrinsic Procedures in Fortran 90
- 2.1.7 IEEE Standard Floating-Point Arithmetic
- 2.1.8 Nearby Machine Numbers
- 2.1.9 Floating-Point Error Analysis
- 2.1.10 Relative Error Analysis

## 2.1.8 Nearby Machine Numbers

We now want to assess the error involved in approximating a given positive real number x by a nearby machine number in the Marc-32. We assume that

$$x = q \times 2^m \quad 1 \leqslant q < 2 \quad -126 \leqslant m \leqslant 127$$

We ask, What is the machine number closest to x ? First, we write

$$x = (1.a_1 a_2 \ldots a_{23} a_{24} a_{25} \ldots)_2 \times 2^m$$

in which each $a_i$ is either 0 or 1.

## 2.1.8 Nearby Machine Numbers

One nearby machine number is obtained by simply discarding the excess bits $a_{24}a_{25}\ldots$This procedure is usually called chopping. The resulting number is

$$x_{-} = (1.a_1 a_2 \ldots a_{23})_2 \times 2^m$$

Another nearby machine number lies to the right of x. It is obtained by rounding up; that is, we drop the excess bits as before but increase the last remaining bit $a_{23}$ by one unit. This number is

$$x_{+} = \left((1.a_1 a_2 \ldots a_{23})_2 + 2^{-23}\right) \times 2^m$$

## 2.1.8 Nearby Machine Numbers

There are two situations, the closer of $x_-$ and $x_+$ is chosen to represent $x$ in the computer.
If $x$ is represented better by $x_-$, then we have

$$|x - x_-| \leqslant \frac{1}{2}|x_+ - x_-| = \frac{1}{2} \times 2^{m-23} = 2^{m-24}$$

In this case, the relative error is bounded as follows:

$$\left|\frac{x - x_-}{x}\right| \leqslant \frac{2^{m-24}}{q \times 2^m} = \frac{1}{q} \times 2^{-24} \leqslant 2^{-24}$$

In the second case, $x$ is closer to $x_+$ than to $x_-$ and we have

$$|x - x_+| \leqslant \frac{1}{2}|x_+ - x_-| = 2^{m-24}$$

The same analysis then shows the relative error to be no greater than $2^{-24}$.

## 2.1.8 Nearby Machine Numbers

What has just been said about the Marc-32 can be summarized: If $x$ is a nonzero real number within the range of the machine, then the machine number $x^*$ closest to $x$ satisfies the inequality

$$\left| \frac{x - x_*}{x} \right| \leqslant 2^{-24}$$

By letting $\delta = (x^* - x)/x$, we can write this inequality in the form

$$fl(x) = x(1 + \delta) \quad |\delta| \leqslant 2^{-24}$$

The notation $fl(x)$ is used to denote the floating-point machine number $x^*$ closest to $x$. The number $2^{-24}$ that occurs in the preceding inequalities is called the unit roundoff error for the Marc-32.

## Contents

1 2.0 Introduction

2 2.1 Floating-point Numbers and Roundoff Errors
- 2.1.1 Rounding
- 2.1.2 Normalized Scientific Notation
- 2.1.3 Hypothetical Computer Marc-32
- 2.1.4 Zero,Infinity,NaN
- 2.1.5 Machine Rounding
- 2.1.6 Intrinsic Procedures in Fortran 90
- 2.1.7 IEEE Standard Floating-Point Arithmetic
- 2.1.8 Nearby Machine Numbers
- 2.1.9 Floating-Point Error Analysis
- 2.1.10 Relative Error Analysis

## 2.1.9 Floating-Point Error Analysis

We assume that the design of this machine is such that whenever two machine numbers are to be combined arithmetically, the combination is first correctly formed, then normalized, rounded off, and finally stored in memory in a machine word.

## Contents

1 2.0 Introduction

2 2.1 Floating-point Numbers and Roundoff Errors
- 2.1.1 Rounding
- 2.1.2 Normalized Scientific Notation
- 2.1.3 Hypothetical Computer Marc-32
- 2.1.4 Zero,Infinity,NaN
- 2.1.5 Machine Rounding
- 2.1.6 Intrinsic Procedures in Fortran 90
- 2.1.7 IEEE Standard Floating-Point Arithmetic
- 2.1.8 Nearby Machine Numbers
- 2.1.9 Floating-Point Error Analysis
- 2.1.10 Relative Error Analysis

## 2.1.10 Relative Error Analysis

THEOREM 1 Theorem on Relative Roundoff Error in Adding
Let $x_0, x_1, \ldots, x_n$ be positive machine numbers in a computer whose unit roundoff error is $\varepsilon$. Then the relative roundoff error in computing

$$\sum_{i=0}^{n} x_i$$

in the usual way is at most $(1+\varepsilon)^n - 1 \approx n\varepsilon$.