OLD DOMINION UNIVERSITY

Numbers and errors

A. Godunov

1. Computers and numbers
2. Errors and uncertainties

updated 31 August 2022

1

Part 1:

Computers and numbers

2

Computers and numbers

Computers have limited amount of memory (internal, external) used to represent numbers.

A problem in computer design is how to represent a general number in a finite amount of space, and then how to deal with the approximate representation that results.

Every computer has a limit how small or large a number can be.

Therefore, there are various types of data (integer, float, character, …)

3

3

Number representation

Number representation:

Base 10 (0-9): decimal, base 2 (0,1): binary, base 8 (0-7): octal,
base 16 (0-15): hexadecimal

Example 1000 in decimal:
1111101000 (binary), 1750 (octal), 3E8 (hexadecimal)

A computer represent ALL numbers in the binary form as a combination of the digits 0 and 1.

4

4

Finite number representation

Numbers are represented as **words**.

Word length: number of bytes used to store a number

units: 1 bit is either 0 or 1, 1 byte = 8 bits

note: 1 kilobyte = 1 KB = 2^{10} bites = 1024 bites (not 1000 bites).

Most common computer architecture:

Word length = 4 bytes = 32 bites

Word length = 8 bytes = 64 bites

5

5

Integer numbers

Since computers represent numbers in the binary form, then for N-bit computers there are only 2^N integers that can be represented with N-bits.

Because the sign of the integer is represented by the first bit (a zero bit for positive numbers), this leaves the remaining N - 1 bits to represent the value of the integer.

Therefore N-bit integers will be in the range [0-2^{N-1}].

Example: the highest number

for 8-bit computer is $2^{8-1} = 128$

for 32-bit computer is $2^{32-1} = 2{,}147{,}483{,}648$

for 64-bit computer is $2^{64-1} = 9{,}223{,}372{,}036{,}854{,}775{,}808$

6

6

Floating point numbers – single precision

In scientific calculations we mainly use floating-point numbers.
In floating-point notation, a number is stored as a sign, a mantissa, and an exponential field. The number is reconstituted as

$$x_{float} = (-1)^s \times mantissa \times 2^{exponent}$$

For 32-bit computer (single precision)

8-bit range of exponent [-127,128] ($2^{128} \sim 10^{+38}$)

23-bit mantissa: 6-7 decimal places $1/2^{23} \sim 1.2 * 10^{-7}$

range: max – about $\pm 3.402923 \times 10^{+38}$

range: min – about $\pm 1.401298 \times 10^{-45}$

machine precision ε: $1.0 + \varepsilon = 1.0$

7

7

Trouble with single precision (example)

It is so easy to run into troubles with single precision

Example: Bohr radius

$$a_0 = \frac{4\pi\varepsilon_0\hbar^2}{m_e e^2} \approx 5.3 \cdot 10^{-11} m$$

Where the numerator $1.24 \cdot 10^{-78}$, the denominator $2.33 \cdot 10^{-68}$

Remember that the single precision is $\sim 10^{-38}$

What can we do?

- restructure the equation
- change units (e.g. use atomic units in this case)
- increase precision

8

8

Floating point – double precision

As a rule, in physics, we always use double precision

For 64-bit computer (double precision)

1-bit sign

11-bit range of exponent [-1023,1024] ($2^{1024} \sim 10^{+308}$)

52-bit mantissa: 15-16 decimal places $1/2^{52} \sim 1.2 * 10^{-15}$

range: max – about $\pm 1.7976931348623157 \times 10^{+308}$

range: min – about $\pm 4.94065645841246544 \times 10^{-324}$

machine precision ε: $1.0 + \varepsilon = 1.0$

9

9

Floating point – double precision

```
% Part 1: - find the number of decimal places for the given precision
% Method: 1.0 + small = 1.0 then the exponent of small is the answer
small = 1.0;
for j=1:100
    small = small/2.0;
    one = 1.0 + small;
         if one == 1.0
          break
      end
end
Ndecimal = abs(floor(log10(small)));
fprintf('Decimal places in floats = %3i  \n',Ndecimal)
Decimal places in floats =  16
```

So, we have 16 decimal places!
Note: by default, MatLab uses double precision.
Note: "vpa" function provides variable precision which can be increased without limit

10

10

Floating point – double precision

```
% Part 2: Find find zero for given precision
% Method: when 0.0 + eps = 0.0 (print eps before the last iteration)
eps(1) = 1.0;
for j=2:2000
    eps(j) = eps(j-1)/2.0;
    zero = 0.0 + eps(j);
    if zero == 0.0
        break
    end
end
fprintf(' Machine zero = %13.7e \n',eps(j-1))
Machine zero  = 4.9406565e-324
```

11

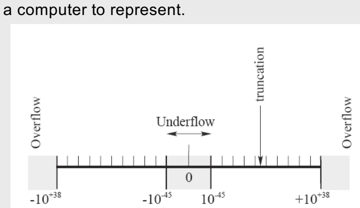
11

Overflow and underflow

from "A Survey of Computational Physics. Introductory Computational Science" by R. Landau et al (2008)

Overflow is an error that occurs when there are not enough bits to express a value in a computer.

Underflow is an error when the result of a computation is too small for a computer to represent.



12

12

Part 2:

Errors and uncertainties

13

Errors

Main reasons

1. Limited precision (number representation) – round-off errors, subtractive cancellation, …
2. Computer errors (example – compilation with optimization)
3. Random errors: electronic fluctuations (e.g. cosmic rays) – very rare but for many steps …
4. Approximation errors (algorithms, truncating series, …)
5. Human errors or blunders: typos, wrong program, wrong data, …

For 1 to 3 reasons: let there are n steps to compute, let p is the probability that the step is correct. Then after n steps the probability that the whole calculation is correct is $P = p^n$.
Example (from R. Landau: for n=1000 and p=0.9993 P=1/2!

14

14