Errors and Uncertainties

Computational Physics: R. Landau et al, Chap. 3.
Basic Concepts in Computational Physics, Chap. 1.

Problems

- Human and machines make mistakes.
- You may not be able to eliminate all problems even you are an excellent programmer.
  - Some uncontrollable situation
  - Input variables are not exact, either imperfectly represented or partially unknown.
  - Calculations can not be exactly performed
- Errors accumulates and propagates

Types of Errors

- Programmer's Error
- Random Error
- Approximation Error
- Round-off Error

Programmer’s Error

- Syntax Error, wrong data, ….
- Double check your codes.
- What if the blunder rate keeps high?

Random Errors

- Interruption of power supply;
- Unstable electric signal induced error
- Totally random and almost no way to manipulate
- Program that takes shorter time is unlikely to run into these errors.
- Double check result consistency for large programs.

Error from Finite Terms

Approximation and Round-off Error

- The computer can only present finite numbers of elements.

- Some constants, functions and series contains infinite terms.

- Approximation errors arise from using finite terms to approximate an infinite series

Approximation Error

- Expansion of exponential function

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \ldots$$

$$= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \boxed{O(x^6)}$$

$$\approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!}$$

Round-off Errors

- When rounding any real number to the floating point representation
- Floating point calculation
- Care is needed when writing the program and interpreting the result

Round-off Errors

- Even the number is in decimal representation

- Ex: 1/3 = 0.333333333……
- $\fallingdotseq$ 0.3333333

- Ex: 2/3 = 0.666666666……
- $\fallingdotseq$ 0.666667

- The number will be rounded up or down.

More about Roundoff Error in numerical methods

Where the errors are from

1. The number can not be precisely represented
2. Storage of a number is limited
3. Number conversion not exact
4. Overflow and Underflow
5. Arithmetic invoking errors

The number can not be precisely represented

- In decimal system
- Ex: 1/3 = 0.333333333……
- ≒ 0.3333333
- Ex: 2/3 = 0.666666666……
- ≒ 0.666667
- π = 3.1415926……… =3.14159
- $\sqrt{2}$= 1.414…………..
  = 1.414

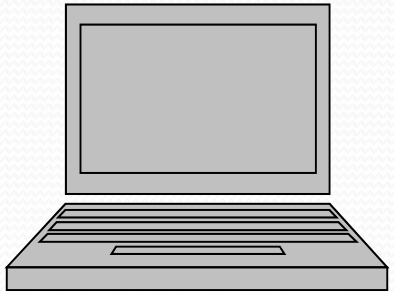
Storage of a number is limited

- Even though the number can be precisely represented
- Ex: 0.12345678987654321
- The computer limits only 10 digits to represent a number ?

Storage of a number is limited

RAJVEER MEENA : I can memorize π up to 70,000 digits

- Oh!! only 16 bits are available to store the number



http://www.guinnessworldrecords.com/world-records/most-pi-places-memorised

Where the errors are from

- The number can not be precisely represented
- Storage of a number is limited
- Number conversion not exact
- Overflow and Underflow
- Arithmetic invoking errors

Conversion

- Numbers must be converted to bit strings before processing
- The conversion is not always exact

A number $\{-\infty, \infty\}$ $\xrightarrow{\text{Rounding}}$ $\{-\infty, -0\} \cup \{-\infty, -0\} \cup NAN$

$\xrightarrow{\text{Representation specification}}$

$$sign * base^{e} * significand$$

$\xrightarrow{\text{data encoding}}$

$$0\ 100001\ 1 \ldots \ldots$$

Conversion

- Numbers must be converted to bit strings before processing
- The conversion is not always exact

$$\sqrt{2} \xrightarrow{\text{Rounding}} 1.41421356(2)$$

$$\xrightarrow{\text{Representation specification}} 1.41421356 \times 10^1$$

$$\xrightarrow{\text{data encoding}}$$

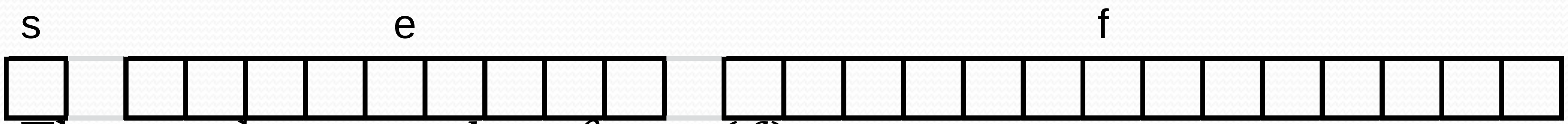
$$0\ 100001\ 1 \ldots \ldots$$

Data Types

- For integers:
  - (u)int16, (u)int32, (u)int64.
- For rational numbers:
  - float32, float64, decimal64, decimal128
  - These are actually floating numbers
- For real number:
  - Not Applicable

Representation of Floating numbers

- IEEE 754:



The number = $s * base^{e} * m(f)$

e.g(1): -32000 = (-1) * 10^{4}*3.2;
s = (-), base = 10, c = 4, f = 3.2=m(f)

e.g(2): 4.5 = 4*1.25 = (+1) *2^{2}*(1+0.125);
s = +1, base = 2, c = 2, f = 0.125, m(f) = 1+0.125;

Double Type representation

- For every floating number should be presented with 64 binary digits. (double precision)

- s: 1 bit (sign)
- c: 11 bits (exponent)
- f: 52 bits (mantissa)

$$(-1)^s \, 2^{c-1023} \, (1+f)$$

- Current version: IEEE754-2008

Float Type representation

- For every floating number should be presented with 32 binary digits. (single precision)

- s: 1 bit (sign)
- c: 8 bits (exponent)
- f: 23 bits (mantissa)

$$(-1)^s \, 2^{c-127} \, (1+f)$$

Floating-point example (double precision)

Consider for example, the machine number

$$\boxed{0}\ \boxed{10000000011}\ \boxed{1011100100010000000000000000000000000000000000000000}.$$

$+$

$$e = 1 \cdot 2^{10} + 0 \cdot 2^{9} + \cdots + 0 \cdot 2^{2} + 1 \cdot 2^{1} + 1 \cdot 2^{0} = 1024 + 2 + 1 = 1027.$$

$$m = 1 \cdot \left(\frac{1}{2}\right)^{1} + 1 \cdot \left(\frac{1}{2}\right)^{3} + 1 \cdot \left(\frac{1}{2}\right)^{4} + 1 \cdot \left(\frac{1}{2}\right)^{5} + 1 \cdot \left(\frac{1}{2}\right)^{8} + 1 \cdot \left(\frac{1}{2}\right)^{12}.$$

Number conversion

$$(-1)^s 2^{c-127}(1+f)$$

$$a = (0\,10.....0\;0...00)_2 = 2^{128-127}(1+0) = 2$$

$$b = (0\,10.....0\;0...01)_2 = 2(1+2^{-23}) = 2+2^{-22}$$

$$c = (0\,10.....0\;0...10)_2 = 2(1+2^{-22}) = 2+2^{-21}$$

$$d = (0\,10.....0\;0...11)_2 = 2(1+2^{-22}+2^{-23})$$

Note

- Floating point format can only represent rational numbers

- Not every rational numbers can be represented by floating points

Machine Epsilon

- Unit roundoff error or maximum relative error

- The minimum increment to change the mantissa from 1.

- In this 32 digit binary system

$$\varepsilon : f = (00...01)_2 = 2^{-23}$$

Number conversion

- Covert 1.125 to 32bit floating point representation

$$2^{c-127} = 1 = 2^{127-127} \Rightarrow c = 127 = (011...11)_2$$

$$f = 0.125 = 0*(\frac{1}{2})^1 + 0*(\frac{1}{2})^2 + 1*(\frac{1}{2})^3 + 0*(\frac{1}{2})^4$$

$$\Rightarrow f = (0010...0)_{fl}$$

- Floating number presentation of 1.125

0 011..11 00100.....0

Conversion to floating point number Example II

- Covert 2.4 to floating point representation

$$2.4 = 2(1+0.2)$$

$$2 = 2^{1} = 2^{128-127} \Rightarrow c = 128 = (100\ldots00)_2$$

$$0.2 = (00110011\underline{0011}\ldots)_2$$

- 32 bit fl. pt. representation of 2.4

  0 100..00 00110011...001(1)

  or 0 100..00 00110011...010

- The value after chopping is 2.39999...

$$\begin{array}{r} 0.2 \\ \times\ 2 \\ \hline \underline{0}.4 \\ \times\ 2 \\ \hline \underline{0}.8 \\ \times\ 2 \\ \hline \underline{1}.6 \\ \times\ 2 \\ \hline \underline{1}.2 \\ \times\ 2 \\ \hline 0.4 \end{array}$$

Conversion to floating point number Example II

- Covert 2.4 to floating point representation
- 32 bit fl. pt. representation of 2.4

  0 100..00 00110011...001(1)

  or 0 100..00 00110011...010

- The value after chopping is

  2.39999985...

  or 2.400000094...

Conversion

- Converting a decimal number to binary or hexadecimal system, approximation is needed again.
    - The infinite series of digits
    - Finite storage
    - Chopping or rounding

Round-off error

- The room to store a number is limited
- Not all the numbers perfectly fit the conversion rule.
- These approximations need to chop or round the input number to the nearest digits.
- Errors occur when rounding the number to the nearest representation number

Example

- One loop goes from 1 to 2 with step size 1/3
- Skip the loop when the value is equal to or greater than 2
- Expected: 1, 4/3, 5/3, 2

```
a = 1; %% number to start loop
b = 2; %% number to end loop
n = 3; %% number of iterations
h = (b-a)/n;
text = sprintf('h=%.20f',h);
disp(text);
x = a;
text = sprintf('x=%.20f',x);
disp(text);

while(1)
    if(x>=b)
        break;
    else
    x = x + h;
    text = sprintf('x=%.20f',x);
    disp(text);
    end
end
```

Example: output

- Ahead of the loop

```
Command Window
h=0.3333333333333331000
x=1.0000000000000000000
```

- In the loop

```
x=1.3333333333333330000
x=1.6666666666666650000
x=1.9999999999999980000
x=2.3333333333333300000
```

Example

- One loop goes from 1 to 1.1 with step size 1/30

- Skip the loop when the value is equal to or greater than 1.1

- Expected:
  1, 31/30, 32/30, 33/30

```
a = 1; %% number to start loop
b = 1.1; %% number to end loop
n = 3; %% number of iterations
h = (b-a)/n;

text = sprintf('h=%.20f',h);
disp(text);
x = a;
text = sprintf('x=%.20f',x);
disp(text);

while(1)
    if(x>=b)
        break;
    else
    x = x + h;
    text = sprintf('x=%.20f',x);
    disp(text);
    end
end
```

Example: output

- Ahead of the loop
- In the loop

```
Command Window
h=0.03333333333333336100
x=1.00000000000000000000
```

```
x=1.03333333333333340000
x=1.06666666666666690000
x=1.10000000000000030000
>>
```

Modification

- Roundoff error shall be smaller than half a step size.

- Change the condition to a better criteria such as 'bigger than "stop value – half step"'

```
a = 1; %% number to start loop
b = 2; %% number to end loop
n = 3; %% number of iterations
h = (b-a)/n;

fprintf('h=%.20f\n',h);
x = a;
fprintf('x=%.20f\n',x);


while(1)
    if(x>b-h/2)
        break;
    else
    x = x + h;
    fprintf('x=%.20f\n',x);
    end
end
```

Where the errors are from

- The number can not be precisely represented
- Storage of a number is limited
- Number conversion not exact
- Overflow and Underflow
- Arithmetic invoking errors

Data representation

- S: 1 bit (sign)
- C: 11 bits (exponent)
- f: 52 bits (mantissa)

$$(-1)^s 2^{c-1023}(1+f)$$

- Maximum:
  - C=2047, f=($1-2^{-52}$) :
- Minimum:
  - C=-1023, f=2^{-52} :

Reservation for special case

- C = 2047 is reserved for special cases
  - Inf
  - Nan

- Max C = 2046

Underflow

- The minimum normalized magnitude

$$2^{-1023}(1+2^{-52}) \approx 10^{-308}$$

- Any number of which magnitude is smaller than the minimum magnitude is considered under flow

Overflow

- The maximum normalized magnitude

$$2^{1023}(1+(1-2^{-52}))\approx 10^{308}$$

- Any number of which magnitude is bigger than the maximum magnitude is considered overflow

For single precision floating point

- Data representation (32 bit)

$$(-1)^s 2^{c-127}(1+f)$$

- Maximum normalized value:

$$2^{127}(2-2^{-23}) \approx 10^{38}$$

- Minimum normalized value:

$$2^{-127}(1+2^{-23}) \approx 10^{-38}$$

In case of overflow or underflow

- A more naïve but better outcome
  - Halt
- A more favorable outcome
  - Halt and return an error message indicating the case of overflow or underflow
- What we don’t want
  - Return an error result without any message
  - Typically takes place in the case of integer

Example : RMS

- Calculate the root of

$$x^2 - 10^{300} x - 2 \times 10^{600} = 0$$

$$=> x = \frac{10^{300} \pm \sqrt{(10^{300})^2 + 8 \times 10^{600}}}{2}$$

$$\Rightarrow x = -10^{300}, 2 \times 10^{300}$$

In MATLAB

Command Window

```
>> 10^300+sqrt(((10^300)^2)+8*10^600)/2

ans =

   Inf
```

To avoid the overflow: RMS

$$x = \frac{10^{300} \pm \sqrt{(10^{300})^2 + 8\times 10^{600}}}{2}$$

$$= 10^{300}\left(\frac{1 \pm \sqrt{(1)^2 + 8\times 1}}{2}\right)$$

$$\Rightarrow x = -10^{300}, 2\times 10^{300}$$

In MATLAB

```
>> 10^300*(1+sqrt(((1)^2)+8*1))/2

ans =

  2.0000e+300

>>
```

Overflow: integer

```
clc
clear all
close all

a=int8(2);
disp(a)
for ii=1:8
    a=a*2;
    disp(a)
end
```

Command Window

```
   2

   4

   8

  16

  32

  64

 127

 127
```

Where the errors are from

- The number can not be precisely represented
- Storage of a number is limited
- Number conversion not exact
- Overflow and Underflow
- Arithmetic invoking errors

Commonly used arithmetic is not correct? - About precision

Back to Decimal System

$$(-1)^s 10^{c-B}(1+f)$$

Source of ERROR

- Arithmetic invoking errors
  - The number can not be precisely represented
  - Storage of a number is limited
  - Number conversion not exact

```
#include<stdio.h>
void main()
{
    int a,b,c;
    a = 1;
    b = 3;
    c = a/b;
    printf("a = %d, b = %d, c=%d\n",a,b,c);
    return;
}
```

```
#include<stdio.h>
void main()
{
    double a,b,c;
    a = 1;
    b = 10^-300;
    c = a+b;
    printf("a = %lf, b = %lf, c=%lf\n",a,b,c);
    return;
}
```

Back to Decimal System

- Focus on the mantissa part

$$\pm a \times 10^{b}$$

$$0 \le a < 1$$

Addition

- $0.120 \times 10^5 + 0.130 \times 10^6$
- Shift to have identical exponent
- $0.012 \times 10^6 + 0.13 \times 10^6$
- Mantissa addition
- $0.012 + 0.13 = 0.142$
- Shift if necessary

Multiplication

- Problem
- $0.120 \times 10^5 \times 0.130 \times 10^6$
- Exponent addition
- $5+6 = 11$
- Mantissa multiplication
- $0.120 \times 0.130 = 0.0156$
- Shift

Estimate errors

- Absolute error

$$|y_{ref} - y_{app}|$$

- Relative Error

$$\left|\frac{y_{ref} - y_{app}}{y_{ref}}\right|$$

Relative Error and Significant Digit

- The number of y_{app} is said to approximate y_{ref} to k significant digits if k is the largest non-negative integer for which

$$\frac{|y_{ref} - y_{app}|}{y_{ref}} < \frac{1}{0.1} \times 10^{-k} < 10 \times 10^{-k}$$

Example

- y_{ref} =0.123456789123456789

- 3 significant digit: $y_{app} = 0.123$

$$\left|\frac{y_{ref} - y_{app}}{y_{ref}}\right| = \left|\frac{0.000456789123456789}{0.123456789123456789}\right|$$

$$=0.0037 < 10^{-2} = 10 \times 10^{-3}$$

- 6 significant digit $y_{app} = 0.123456$

$$\left|\frac{y_{ref} - y_{app}}{y_{ref}}\right| = \left|\frac{0.000000789123456789}{0.123456789123456789}\right|$$

$$= 6.4 \times 10^{-6} < 10 \times 10^{-6}$$

Relative Error and Significant Digit

$$y = 0.d_1d_2d_3\ldots d_k d_{k+1}\ldots \times 10^c$$

$$\Rightarrow \; fl(y) = 0.d_1d_2d_3\ldots d_k \times 10^c$$

K-significant digits

$$\Rightarrow \left|\frac{y - fl(y)}{y}\right| = \left|\frac{0.000\ldots d_{k+1}d_{k+2}\ldots}{0.d_1d_2d_3\ldots d_k d_{k+1}\ldots}\right|$$

$$= \left|\frac{0.d_{k+1}d_{k+2}\ldots \times 10^{c-k}}{0.d_1d_2d_3\ldots d_k d_{k+1}\ldots \times 10^c}\right| \le \frac{1}{0.1}\times 10^{-k} = 10^{-k+1}$$

Propagation of significant digits

- The last significant digit is usually an estimated number – NOT EXACT
- Basic rules in arithmatics
  - Exact digit (+,-,*,/) Exact digit = Exact digit
  - Non-Exact digit (+,-,*,/) Exact digit = Non-Exact digit
  - Non-Exact digit (+,-,*,/) Non-Exact digit = Non-Exact digit

Example

- At a 5-digit decimal system
- $X = 5/7$ $fl(X)=0.71428x10^0$
- $Y = 1/3$ $fl(Y)=0.33333x10^0$

Operation	Result	Actual value	Absolute error	Relative error
$x \oplus y$	0.10476×10^1	$22/21$	0.190×10^{-4}	0.182×10^{-4}
$x \ominus y$	0.38095×10^0	$8/21$	0.238×10^{-5}	0.625×10^{-5}
$x \otimes y$	0.23809×10^0	$5/21$	0.524×10^{-5}	0.220×10^{-4}
$y \oplus x$	0.21428×10^1	$15/7$	0.571×10^{-4}	0.267×10^{-4}

Operation	Result	Actual value
$x \oplus y$	0.10476×10^1	$22/21$
$x \ominus y$	0.38095×10^0	$8/21$
$x \otimes y$	0.23809×10^0	$5/21$
$y \oplus x$	0.21428×10^1	$15/7$

Actual value	Absolute error	Relative error
$22/21$	0.190×10^{-4}	0.182×10^{-4}
$8/21$	0.238×10^{-5}	0.625×10^{-5}
$5/21$	0.524×10^{-5}	0.220×10^{-4}
$15/7$	0.571×10^{-4}	0.267×10^{-4}

Example I :a 5-digit decimal system

- $X = 5/7$ $fl(X)=0.71428 \times 10^0$
- $Y = 1/3$ $fl(Y)=0.33333 \times 10^0$
- $U = 0.714251$ $fl(U) = 0.71425 \times 10^0$
- $V = 98765.9$ $fl(V) = 0.98765 \times 10^5$
- $W = 0.111111 \times 10^{-4}$ $fl(W)= 0.11111 \times 10^{-4}$

Operation	Result	Actual value	Absolute error	Relative error
$x \ominus u$	0.30000×10^{-4}	0.34714×10^{-4}	0.471×10^{-5}	0.136
$(x \ominus u) \oplus w$	0.29629×10^{1}	0.34285×10^{1}	0.465	0.136
$(x \ominus u) \otimes v$	0.29629×10^{1}	0.34285×10^{1}	0.465	0.136
$u \oplus v$	0.98765×10^{5}	0.98766×10^{5}	0.161×10^{1}	0.163×10^{-4}

Operation	Result	Actual value
$x \ominus u$	0.30000×10^{-4}	0.34714×10^{-4}
$(x \ominus u) \oplus w$	0.29629×10^{1}	0.34285×10^{1}
$(x \ominus u) \otimes v$	0.29629×10^{1}	0.34285×10^{1}
$u \oplus v$	0.98765×10^{5}	0.98766×10^{5}

Actual value	Absolute error	Relative error
0.34714×10^{-4}	0.471×10^{-5}	0.136
0.34285×10^{1}	0.465	0.136
0.34285×10^{1}	0.465	0.136
0.98766×10^{5}	0.161×10^{1}	0.163×10^{-4}

Example II :a 4-digit decimal system

$$x^2 + 62.10x + 1 = 0$$

$$x = \frac{-62.10 \pm \sqrt{(62.10)^2 - 4}}{2}$$

Original Form

$$x = \frac{-62.10 - \sqrt{(62.10)^2 - 4}}{2} = -62.0839$$

$$approximation \Rightarrow \frac{-62.10 - 62.06}{2} = -62.08$$

$$x = \frac{-62.10 + \sqrt{(62.10)^2 - 4}}{2} = -0.01610723$$

$$approximation \Rightarrow \frac{-62.10 + 62.06}{2} = -0.02$$

Example II :a 4-digit decimal system

- An alternative: Rationalized Numerator Form

$$x^2 + 62.10x + 1 = 0$$

$$x = \frac{-62.10 \pm \sqrt{(62.10)^2 - 4}}{2}$$

$$= \frac{62.10^2 - (62.10^2 - 4)}{2(-62.10 \mp \sqrt{(62.10)^2 - 4})}$$

Rationalized Numerator

$$x = \frac{62.10^2 - (62.10^2 - 4)}{2(-62.10 + \sqrt{(62.10)^2 - 4})} = -62.0839$$

$$approximation \Rightarrow \frac{4}{2(-62.10 + 62.06)} = -50$$

$$x = \frac{62.10^2 - (62.10^2 - 4)}{2(-62.10 - \sqrt{(62.10)^2 - 4})} = -0.01610723$$

$$approximation \Rightarrow \frac{4}{2(-62.10 - 62.06)} = -0.016$$

Example III

- Find f(4.71) with

$$f(x) = x^3 - 6x^2 + 3x - 0.149$$

- at a 3-digit system

	x	x^2	x^3	3x	$6x^2$
Exact	4.71	22.18	104.5	14.13	133.1
3-digit	4.71	22.2	104	14.1	132

Example III

	x	x^2	x^3	3x	$6x^2$
Exact	4.71	22.18	104.5	14.13	133.1
3-digit	4.71	22.2	104	14.1	132

$$f(4.71) = -14.636489$$

$$f(4.71)_{3-digit} = 104 - 132 + 14.1 - 0.149 = -14.0$$

Example III

	x	x^2	x^3	3x	$6x^2$
Exact	4.71	22.18	104.5	14.13	133.1
3-digit	4.71	22.2	104	14.1	132

$$
\begin{aligned}
f(x) &= x^3 - 6x^2 + 3x - 0.149 \\
&= ((x-6)x+3)x - 0.149 \\
&= ((4.71-6)4.71+3)4.71) - 0.149 = -14.5
\end{aligned}
$$

Arithmetic invoking errors

- Use double precision in the case of huge arithmetic is needed.

- In real experiment, the error propagation due to measurement uncertainty may increase faster than round-off error. (Ref: error propagation in your statistics textbook)

- Use a smarter code to avoid the problem.

Brief Summary

- The approximation and roundoff errors can become Blunders leading to disasters.

- Watch out every step that may cause blunders

- Approximation error is generally more significant than actual round-off error

- Use a smarter way to code your program to minimize round-off errors.