第1章　誤差 (Error)

1.1　打ち切り誤差と丸め誤差 (Truncation and round off errors)

数値計算のねらいは，できるだけ精確・高速に解を得ることである．誤差 (精度) と収束性 (安定性, 速度) が数値計算のキモとなる．前回に説明した収束判定条件による誤差は打ち切り誤差 (truncation error) と呼ばれる．ここでは，誤差のもう一つの代表例である，計算機に特有の丸め誤差 (roundoff error) について見ておこう．

1.1.1　整数型と実数型の内部表現

計算機は一般に無限精度の計算をおこなっているわけではない．CPU で足し算をおこなう以上，一般的な計算においては CPU が扱う一番効率のいい数の大きさが存在する．これが,32bit の CPU では 1 ワード,4byte(4x8bits) である．1 ワードで表現できる最大整数は, 符号に 1bit 必要なので,2^(31)-1 となる．実数は以下のような仮数部と指数を取る浮動小数点数で表わされる．

表 1.1: 浮動小数点数の内部表現 (IEEE754).

	$s \times f \times B^{e-E}$
s	sign bit(符号ビット:正負の区別を表す)
e	biased exponent(指数部)
f	fraction portion of the number(仮数部)
B	base(基底) で通常は 2
E	bias(下駄) と呼ばれる
real(単精度)	s=1, e=8, f=23 E=127
double precision(倍精度)	s=1, e=11, f=52 E=1023

E は指数が負となる小数点以下の数を表現するためのもの．演算結果は実際の値から浮動小数点数に変換するための操作「丸め (round-off)」が常に行われる．それに伴って現れる誤差を丸め誤差と呼ぶ．

1.2　有効桁数 (Significant digits)

1 ワードの整数の最大値とその 2 進数表示.

```
> restart;
> 2^(4*8-1)-1;#res: 2147483647
```

この整数を 2 進数で表示するように変換するには，convert(n,binary) を用いて，

```
> convert(2^(4*8-1)-1,binary);  #res: 11111111111111111111111111111111
```

となり，31 個の 1 が並んでいることが分かる．1 ワードの整数の最大桁は，n の桁数を戻すコマンド length(n) を使って，

```
> length(2^(4*8-1)-1);  #res: 10
```

となり，たかだか 10 桁程度であることが分かる．一方，64bit の場合の整数の最大桁．

```
> length(2^(8*8-1)-1);  #res: 19
```

である．

Maple では多倍長計算するので，通常のプログラミング言語で起こる int の最大数あたりでの奇妙な振る舞いは示さない．

```
> 2147483647+100;  #res: 2147483747
```

単精度の浮動小数点数は, 仮数部 2 進数 23bit,2 倍長実数で 52bit である．この有効桁数は以下の通り．

```
> length(2^(23)); #res: 7
> length(2^(52)); #res: 16
```

1.3 浮動小数点演算による過ち (FloatingPointArithmetic)

「丸め」にともなって誤差が生じる．C や Fortran 等の通常のプログラミング言語では「丸める」仕様なのでプログラマーが気をつけなければならない．

プログラムリスト ： 実数のケタ落ち

```
#include <stdio.h>

int main(void){
  float a,b,c;
  double x,y,z;

  a=1.23456789;
  printf(" a= %17.10f\n",a);

  b=100.0;
  c=a+b;
  printf("%20.10f %20.10f %20.10f\n",a,b,c);

  x=(float)1.23456789;
  y=(double)100;
  z=x+y;
  printf("%20.12e %20.12e %20.12e\n",x,y,z);

  x=(double)1.23456789;
  y=(double)100;
  z=x+y;
  printf("%20.12e %20.12e %20.12e\n",x,y,z);

  return 0;
}
```

分かっているつもりでも，よくやる間違い．

プログラムリスト ： 丸め誤差

```
#include <stdio.h>
```

```
int main(void){
  float x=77777,y=7,y1,z,z1;
  y1=1/y;
  z=x/y;
  z1=x*y1;
  printf("%10.2f %10.2f\n",z,z1);
  if (z!=z1){
    printf("z is not equal to z1.\n");
  }
  printf("Surprising?? \n\n\n\n\n%10.5f %10.5f\n",z,z1);
  return 0;
}
```

これを避けるには，EPSILON という小さな数字を定義しておいて，値の差の絶対値を求める fabs を使って

とすべき．このときは数学関数である fabs を使っているので，

```
> gcc -lm test.c
```

と math library を明示的に呼ぶのを忘れないように．

1.4 機械精度 (Machine epsilon)

上の例では，浮動小数点数で計算した場合に小さい数の差を区別することができなくなるということを示している．これは，CPU に固有の精度で，機械精度 (Machine epsilon) と呼ばれる．つまり，小さい数を足したときにその計算機がその差を認識できなくなる限界ということで，以下のようにして求めることができる．

```
> Digits:=7;
> e:=evalf(1.0);
> w:=evalf(1.0+e);
> while (w>1.0) do
    printf("%-15.10e %-15.10e %-15.10e\n",e,w,evalf(w-1.0));
    e:=evalf(e/2.0);
    w:=evalf(1.0+e);
  end do:
```

7

1.0

2.0

```
1.0000000000e+00 2.0000000000e+00 1.0000000000e+00
5.0000000000e-01 1.5000000000e+00 5.0000000000e-01
2.5000000000e-01 1.2500000000e+00 2.5000000000e-01
1.2500000000e-01 1.1250000000e+00 1.2500000000e-01
6.2500000000e-02 1.0625000000e+00 6.2500000000e-02
```

```
3.1250000000e-02 1.0312500000e+00 3.1250000000e-02
1.5625000000e-02 1.0156250000e+00 1.5625000000e-02
7.8125000000e-03 1.0078120000e+00 7.8120000000e-03
3.9062500000e-03 1.0039060000e+00 3.9060000000e-03
1.9531250000e-03 1.0019530000e+00 1.9530000000e-03
9.7656250000e-04 1.0009770000e+00 9.7700000000e-04
4.8828120000e-04 1.0004880000e+00 4.8800000000e-04
2.4414060000e-04 1.0002440000e+00 2.4400000000e-04
1.2207030000e-04 1.0001220000e+00 1.2200000000e-04
6.1035150000e-05 1.0000610000e+00 6.1000000000e-05
3.0517580000e-05 1.0000310000e+00 3.1000000000e-05
1.5258790000e-05 1.0000150000e+00 1.5000000000e-05
7.6293950000e-06 1.0000080000e+00 8.0000000000e-06
3.8146980000e-06 1.0000040000e+00 4.0000000000e-06
1.9073490000e-06 1.0000020000e+00 2.0000000000e-06
9.5367450000e-07 1.0000010000e+00 1.0000000000e-06
```

1.5 桁落ち，情報落ち，積み残し (Cancellation)

桁落ち (Cancellation)

$$\begin{array}{rr} & 0.723657 \\ - & 0.723649 \\ \hline \end{array}$$

情報落ち (Loss of Information)

$$\begin{array}{rr} & 72365.7 \\ + & 1.23659 \\ \hline \end{array}$$

積み残し

$$\begin{array}{rr} & 72365.7 \\ + & 0.001 \\ \hline \end{array}$$

1.6 課題

1. 次の項目について答えよ．(2004, 05, 06 年度期末試験)
   (a) 数値計算の精度を制約するデータ形式とその特徴は何か．
   (b) 丸め誤差とは何か．
   (c) 打ち切り誤差とは何か．
   (d) 安定性とは何か．
2. 10 進数 4 桁の有効桁数をもった計算機になったつもりで，以下の計算をおこなえ．
   (a) 2718-0.5818 (b) 2718+0.5818 (c) 2718/0.5818 (d) 2718*0.5818
3. 自分の計算機で機械精度がどの位かを確かめよ．Maple スクリプトを参照して, C あるいは Fortran で作成し, 適当に調べよ．

4. (2147483647 + 100) を C あるいは Fortran で試せ.

5. 係数を a = 1, b = 10000000, c = 1 としたときに, 通常の解の公式を使った解と, 解と係数の関係 (下記の記述を参照) を使った解とを出力するプログラムを作成し, 解を比べよ.

   2 次方程式 $ax^2 + bx + c = 0$ の係数 a, b, c が特殊な値をもつ場合，通常の解の公式

   $$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

   にしたがって計算するとケタ落ちによる間違った答えを出す．その特殊な値とは

   $$\sqrt{b^2 - 4ac} \approx |b|$$

   となる場合である.

   ケタ落ちを防ぐには, $b > 0$ の場合は,

   $$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

   として，ケタ落ちを起こさずに求め, この解を使って, 解と係数の関係より

   $$x_2 = \frac{c}{a\, x_1}$$

   で求める．$b < 0$ の場合は，解の公式の足し算の方を使って同様に求める.