数値計算法概論：No.4 誤差

1 プログラム

1.1 誤差の評価

数値計算から信頼できる結果を得るには誤差の評価が重要である。誤差には演算が有限回であることによるもの (打ち切り誤差) と、計算機が有限の桁数 (実数は無限桁！) しか扱えない為に起こる誤差がある。ここでは後者について考える。例として次のプログラムを扱う。

課題 1

```
      integer i,n
      real eps,sum
c
      read(*,*) eps,n
      sum=0.0
      do i=1,n
         sum=sum+eps
      end do
      write(*,1000) "n*eps=",n*eps,"sum=",sum
c
 1000 format(a,f12.7,a,f12.7)
c
      end
```

無限精度の計算機ならば、$N * eps = sum$ であるが、実際にはそうはならない。$\epsilon = 1.0e-7, N = 10000000$ として見よ。N を変えるとどうなるか試してみよ。

1.1.1 機械精度

実数 1.0 に加えた時、1.0 と異なる結果になるような最小の正の浮動小数点数 ϵ_m を**機械精度**または**機械イプシロン**という。4byte 実数では、ほぼ $\epsilon_m \approx 3 \times 10^{-8}$ である。これより小さな相対精度は数値計算では無意味になる。これによる誤差を丸め誤差という。

計算量の増加に伴い丸め誤差も累積する。計算量が N ならば、全体の丸め誤差は、$\sqrt{N} \times \epsilon_m$ の程度で増加する (誤差が無相関に生じるとランダムウォークの理論が使える)。しかしこれより早く誤差が増大することがある。

1.1.2 桁落ち、積み残し

課題 2

1)a, b がほぼ等しい場合、$a+b, a-b$ を数値計算して誤差を評価しなさい。これによる誤差を桁落ちと呼ぶ。

2)$a \gg b$ の場合、$x = a + b$ とし次に $y = x - a$ を数値計算して誤差を評価しなさい。これによる誤差を積み残しと呼ぶ。

1.2 型宣言と機械精度

1.2.1 実数型と整数型

Fortran では (一般にプログラム言語では) 実数と整数の扱いは大きく違う。例えば整数の割算は切捨てとなる。また実数型で 0.001 を 1000 回足しても必ずしも 1 にならない (計算機内部では 2 進数表示であるのと、誤差の関係から)。奇妙な結果が出た場合、変数の型を確認すること。

1.2.2 計算機内部での数値表現

これは一般に機種により異なるが、現在標準的な ANSI/IEEE-754 規格を元に説明する。ディジタル計算機内部では、全ての情報が 0,1 の 2 進数で表示される。2 進数の 1 桁を**ビット (bit)** と呼ぶ。最近の電子計算機では、情報を 8 ビットづつまとめて扱い、この単位を**バイト (byte)** とよぶ。ちなみに文字情報を扱う場合、英数字は 1 バイト、漢字は 2 バイトで表される。

- 整数型では最初の 1bit は符号に使われるので、
  1. 2byte 整数では -32768 から +32767 まで
  2. 4byte 整数では -2^{31} から $2^{31}-1$ まで (± 約 20 億)
  3. 8byte 整数では -2^{63} から $2^{63}-1$ まで (± 約 8×10^{18})
- 実数型は $f = a \times 2^n$ と表される。ただし、$1/2 \leq |a| < 1$,n は符号付き整数。a を仮数部、n を指数部と呼ぶ。
  1. 4byte 実数 (単精度) では指数部 8 ビット、仮数部 24 ビット。10 進数での精度約 7 桁、数値の範囲は最小約 10^{-38}〜 最大 10^{38}
  2. 8byte 実数 (倍精度) では指数部 11 ビット、仮数部 53 ビット。10 進数での精度 約 16 桁。数値の範囲は最小約 10^{-308}〜 最大 10^{308}

1.2.3 倍精度の宣言

倍精度 (8byte) 実数を使うには以下のようにする。宣言文で、

```
real xy,uv
```

とするかわりに、

```
double precision xy,uv
```

か、

```
real*8 xy,uv
```

または

```
real(8) xy,uv
```

(F90 での形式)、とする。
課題 3：倍精度で課題 1,2 をして見よ。

1.2.4 倍精度の暗黙の宣言

暗黙の型宣言で単精度になっているものを倍精度に変えるには、次の宣言文を使う。

```
implicit real*8 (a-h,o-z)
```

こうすると、一括して変数を倍精度にできるので便利である。

1.3 出力書式 (format 文)

出力の書式を整えるには write(*,*) の 2 番目の *印 (実は書式識別子) を文番号に置き換え、文番号で指定した format 文により書式様式を定義する (課題 1 のプログラム例参照)。format() の中の a,f12.7 の記号は文字、実変数の出力の形式に対応し、その文法は以下の通り。

i) F 型編集子
実数型の表示に使う。f20.16 のように、f$w.d$ と書く。w は全体の文字数、d は小数点以下の桁数。$w \geq d+3$ でなければならない。2f20.16 は f20.16,f20.16 と同じ。

ii) I 型編集子
整数型の表示に使う。i4 のように、iw と書く。w は全体の文字数。3i4 は i4,i4,i4 と同じ。

ii) A 型編集子
文字を出力する時に必要。

2 一般的注意

2.1 ディレクトリ及びファイルシステム (利用の手引 8.4.1)

ファイルを沢山作ると整理が大変になる。通常の事務作業と同様に、目的ごとに 1 連のファイルをまとめておくと整理し易くなる。これはディレクトリ (フォルダ と呼ぶ OS も

ある) を使うと容易にできる。ディレクトリを作るには mkdir, ディレクトリを移動するには cd というコマンドを使う。また、現在どのディレクトリにいるのか確認するためには pwd というコマンドを使う。ディレクトリの中にサブディレクトリを作るように、多重にディレクトリを作成することもできる。

例：

```
# ls
test1.f test2.f test20.f test21.f
# mkdir task1
# cp test20.f task1
# cp test21.f task1
# pwd
# cd task1
# ls
test20.f test21.f
# cd ..
```