数値計算法

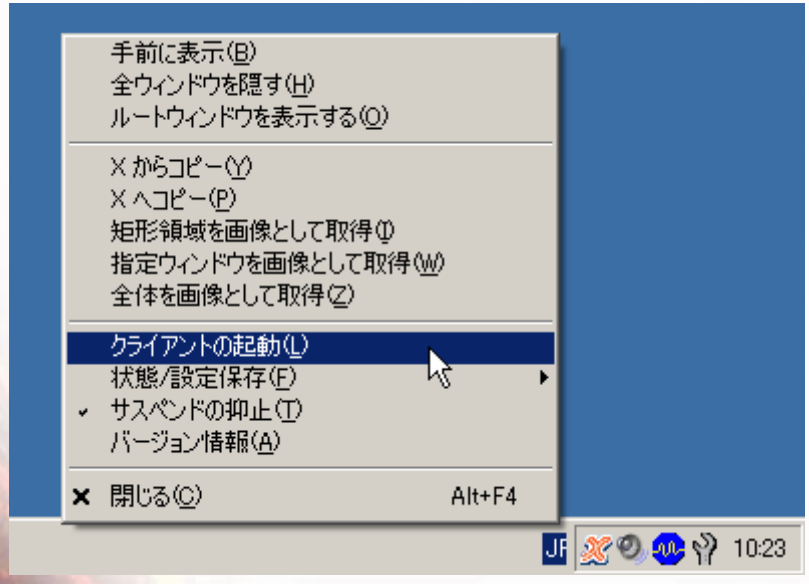
第三回
FORTRANによるプログラミング

若狭　智嗣

粒子物理学講座

Xでの接続方法（ASTEC-Xの使い方）

- Windowsで、
  スタート → プログラム → ASTEC-X → ASTEC-X
  として、ASTEC-Xを起動する
- タスクトレイ内のASTEC-Xのアイコン（X文字）上で右クリックして、「クライアントの起動」を選択
- クライアントの起動ウインドウで、ユーザー名とパスワードを入力して、OKボタンを押す



復習ーコンパイルの仕方ー

- FORTRANのプログラムは、人間は読める(理解出来る)が、計算機(CPU)は理解できない
- 計算機が理解できる形式（実行形式という）に変換することを、コンパイルするという
- コンパイルするときのコマンドは
  frt
  で、形式は、
  % frt -o　実行ファイル名　FORTRANプログラムファイル名
  である。今の場合、
  % frt -o keta_ochi keta_ochi.f
  とすると、実行ファイル keta_ochi ができる
  （lsで確かめると、keta_ochi*があるはず）

実行形式を表す

- 一般的約束
  - FORTRANプログラムは、拡張子として.fを付ける
  - 実行形式には、拡張子は付けない

数値計算の誤差（丸め誤差や桁落ち）による エラー（間違い）を防ぐには

- **アルゴリズムを工夫する**
  - 桁落ちしにくい式（アルゴリズム）を用いる
- **たった1回の計算で満足しない。**
  - 今までの例でも、aやbの値が異なる場合を計算すると、違い（桁落ち）が顕著に現れた
  - 計算機を用いた数値計算のよいところは、
    いろいろな場合（aやbの異なる計算）を瞬時にできる
    点にあるので、系統的に数値計算を行い、結果の確からしさを検証することが必要
- **倍精度の変数を使う**
  - 今までの例で、
    real → real*8
    とすると、実数を64ビットで表すことになり、仮数部の桁数（ビット数）が増え、桁落ちがしにくくなる。

8バイト割り当てる

real*8 (倍精度)を使う際の注意

- real*8（倍精度）の変数を使うと、桁落ちしにくいプログラムが安易に作成できるが、
  - 倍精度の変数は単精度の場合の倍のメモリを消費（ビット数が倍）
  - 倍精度での計算は、計算の確かさの証明にはならない

  ので、必要に応じて使うこと。
- 実習
  - 先週用いた
    keta_ochi.f
    で、realで宣言されている変数をreal*8で宣言して倍精度にして計算してみよ。
  - aやbの値を変えて実行し、どの程度まで桁落ちが起こらないか確認せよ。

FORTRANのプログラム

- 第5章には以下のような、FORTRANでプログラムを作成する上で必要な事項をまとめてある
  - 扱えるデータの型(integer, real, etc.)
  - 扱える構造(do…end do, if…end if, etc)
  - サブルーチン・関数
  - 組み込み算術関数
  - 配列・複素数・文字列の扱い
- 全てを今すぐ理解する必要はない(必要な時に戻って参照すればよい)
- 以下では、例題のプログラムについて解説する。プログラムは全て
  /home/teacher/z6wt01in/SAMPLE/
  に置いてあるので、各人コピーしてmuleで開いて見るなり、実行するなり、変更するなりすること。

circle.fの解説

左は6文字空いている

1行目はプログラム名を宣言

全ての変数は定義する事を宣言

```
      program circle
      implicit none
c constant:
      real PI                                  ! 円周率PIの値を入れる変数
      parameter (PI=3.141593)                  ! 円周率PIを定義
c local variables:
      real radius                              ! 半径の値を入れる変数
      real area,volume                         ! 面積、体積の値を入れる変数
c begin:
      radius = 10.0                            ! 半径として10.0を入れる
      area   = PI * radius**2                  ! *は乗算の、**はべき乗の記号
      volume = 4.0 * PI * radius**3 / 3.0      ! /は除算の記号
      write(*,*) 'radius = ',radius            ! 半径を画面に表示
      write(*,*) 'area   = ',area              ! 面積を画面に表示
      write(*,*) 'volume = ',volume            ! 体積を画面に表示
      stop
      end
```

1文字目がcの場合、その行はコメントとみなす

文字列はカンマ(,)で括る。表示は、

volume = 体積値

となる

実習

- circle.fをコンパイルし、実行しなさい。
- 半径の値を変えて、実行しなさい。
- 円の周りの長さ、及び球の表面積を計算して表示するようにプログラムを変更して、実行しなさい。

ネイピア数（自然対数の底）を求める

- マクローリン展開

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \quad \Rightarrow \quad e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

- 階乗（factorial = n!）を　Fortran の ループ文で計算
  - n=0の場合
    factorial = 1
  - n≧1の場合
    factorial = 1
    do i=1,n
      factorial = factorial*n
    end do

do から end do までの間を
I を 1 から n まで、
1づつ増やしながら
繰り返しn回実行

do ループという

do ループ終了後は、factorial = n! になる

ネイピア数を計算するプログラム

- mule で新しいファイルを開く
  - % mule napier.f &(リターン)
- 以下の内容を入力

```
      program calc_napier
      implicit none
c const:
      integer    LOOP    ! 計算する次数
      parameter(LOOP=10)
c variable:
      integer    i,n,factorial
      real       napier
```

整数を入れる変数を定義

LOOP という変数を定数 10 に定義

実数を入れる変数を定義

変数はカンマで区切る

ネイピア数を計算するプログラムーつづきー

```
c begin:
      napier = 1.0        ! 0次の項の値
      do n=1,LOOP         ! 1次の項以降を計算して加える
         factorial =1
         do i=1,n         ! factorial = n!の計算
            factorial = factorial*i
         end do
c 計算したn次の項を加える
         napier = napier + 1.0/float(factorial)
         write(*,*) n,napier   ! 結果を画面に出力
      end do
      stop
      end
```

factorial=n!

整数型を実数型に変換

$$napier = \sum_{i=0}^{n} \frac{1}{i!}$$

コンパイルして実行してみる

- コンパイル
  ```
  % frt -o napier napier.f
  ```
- 実行
  ```
  % napier
  ```
- ネイピア数（e）の値
  e = 2.71828 18284 59045 23536 02874 71352 …
- 問題
  - 計算結果は上記の値と一致するか？
    - 一致しない場合、何桁までは一致するか？
  - 級数の何次の項まで計算すると収束するか？
  - real → real*8にして倍精度にするとどうなるか？
  - 高い精度で求められるか？