Fortran プログラミング入門

Fortran プログラムの作成と実行　その４

１．放射性元素の崩壊: If 文と条件判定

放射性元素の崩壊など、生成や消滅を表す常微分方程式

$$\frac{du}{dt} = -au$$

の解

$$u = u_0 e^{-at}$$

を求めるプログラムである。ここで、u は放射性元素の量を表す。a は壊変定数と呼ばれ、半減期と

$$a = \frac{\log 2}{\tau}$$

の関係がある。

このプログラムでは if 文が用いられていることに注意しよう。if 文によって、画面やファイルに全タイムステップの計算値を出力するのではなく、画面に 100 タイムステップ毎、ファイルに 10 タイムステップ毎に計算値を出力するようになっている。

また、設定が必要なパラメータはキー入力でなく、ファイルから読み込むようになっている。

問題 F13

(1) プログラムとパラメータのデータファイルを入力し、実行せよ。
(2) 結果のグラフを作成せよ。

２．２次方程式の解

２次方程式

$$ax^2 + bx + c = 0$$

の解を解の公式

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

から求める。このとき、判別式

$$D = b^2 - 4ac$$

に従い解を分類し、２つの実数解あるいは１つの重解を持つときのみ解を求めて表示する。

問題 F14

(1) プログラムとパラメータのデータファイルを入力し、実行せよ。

(1) 虚数解を持つ場合でも解を表示できるようにプログラムを改良せよ。このとき、実数変数だけを用いたプログラムとするにはどのようなプログラムにすれば良いか考えよ。

問題 F13 プログラム

```
!
!   Decay of radio active isotope
!

      program decay

      implicit none
      real(8):: u,u0,t
      real(8):: a,dt
      integer:: n,nmax
      character(40):: Apara
      character(20):: fmt1,fmt2

!   open a parameter file
      open (10,file='decayp.dat')
!   open a output file
      open (11,file='decayu.dat',status='new')
!
!   read input parameters a,dt,u at t=0,nmax
!
      read (10,*) Apara
      read (10,*) a
      write (6,*) Apara,' = ',a
      read (10,*) Apara
      read (10,*) u0
      write (6,*) Apara,' = ',u0
      read (10,*) Apara
      read (10,*) dt
      write (6,*) Apara,' = ',dt
      read (10,*) Apara
      read (10,*) nmax
```

```
      write (6,*) Apara,' = ',nmax
      write (6,*)
!
!   initialize
!
      n = 0
      t = 0.0d0
      u = u0
!
!   define output format
!
      fmt1 = '(I5,f10.5,f10.6)'
      fmt2 = '(f10.5,f10.6)'
!
!   write initial condition to file
!
      write (6, fmt1) n,t,u
      write (11,fmt2) t,u
!
!   time marching
!
      t = 0.0
      do n = 1,nmax

        t = t + dt
        u = u0  * exp( -a*t )
!  write results
        if ( mod(n,100)==0 ) write (6, fmt1) n,t,u
        if ( mod(n,10)==0 )  write (11,fmt2) t,u

      end do
```

```
      stop
      end program decay
```

`if (mod(it,100).eq.0) write …`：条件判定文。( )の中には論理式が入る。論理式の結果が真(.true.)の時だけ write 以下を実行する。ここでは“it を 100 で割った余りが 0 ”が論理式で、この値が真であるとき、wirte 文を実行して画面に n,t,u の値を書く。

なお、実行させたい部分が多数の行に渡る場合はブロック if 文を使用する。

`mod(it,100)`：剰余関数。it を 100 で割った余りを計算する。

`fmt1 = (I5,f10.5,f10.6)` ：フォーマットの指定。fmt1 は文字変数を使う。read や write のとき、桁数などをそろえて、見やすく表示したいときに使用する。文字変数の文字数には( )も含めておく。( )の( )の中にフォーマットが書かれる。I5 は 5 桁の整数、f10.5 は全体が 10 桁で、小数点以下が 5 桁の固定小数点の小数を表す。正負符号なども 1 桁に含まれるので、全体の桁数に注意が必要である。まとめの所にフォーマットについてまとめた。

`write (6, fmt1)` ：フォーマット指定のある write 文。

問題 F13 入力データ

```
'decay constant a'
0.5d0
'initial value u0'
5.0d0
'time interval dt'
1.0d-2
'maximum time step'
1000
```

decayp.dat という名前で保存する。数値は適当に変えて入れること。

問題 F14 プログラム

```
!
!   solve second-order equation by solution's formula
!
      program so_equation

      implicit none
      real(8):: a,  b,  c
      real(8):: D,  x1, x2

      write (6,*) 'Input coefficient a,b,c for ax^2+bx+c=0'
      read (5,*) a,b,c
      write (6,*) 'Your equation is ',a,'x**2+',b,'x+',c,'=0'

      if ( dabs(a)<1.0d-30 ) then
!  ************  This would be a first-order equation  *************

        if ( dabs(b)<1.0d-30 ) then
!  ***  This is not an equation  ***

          write (6,*) 'Your input value is not correct.'
         else
!  ***  This is a first-order equation  ***
          x1 = -c / b
          write (6,*) ' The solution is ',x1
        end if

       else
!  ************  This is second-order equation  ************

!  ***  Examine solution by denominator  ***
        D = b**2 - 4.0d0*a*c
```

```
        if ( D>1.0d-30 ) then
!  ***  D is larger than 0; 2 solutions  ***

          x1 =  ( -b + sqrt( D ) ) / ( 2.0d0*a )
          x2 =  ( -b - sqrt( D ) ) / ( 2.0d0*a )
          write (6,*) 'The solutions are ',x1,x2

         else if ( D<1.0d-30 .and. D>-1.0d-30 ) then
!  ***  D is equal to 0; 1 degenerated solution  ***

          x1 =  -b / ( 2.0*a )
          write (6,*) 'The degenerated solution is ',x1

         else
!  ***  D is less than 0; imaginary solutions  ****

          write (6,*) 'There are no real solutions.'
          write (6,*) 'There exist 2 imaginary solutions.'

        end if

! ************  bottom of outer if block  ************
      end if

      stop
      end program so_equation
```

if (dabs(a).lt.1.0d-30) then：ブロック if 文。( )の中は a が 0 という条件式である。a==0.0d0 としないのは以下のような理由である。実数は 10 進数とぴったりの値になるとは限らない(3.0 が 0.2999998e1 のように表されてしまう)のと、2a で割ったとき、0 での割り算となり、エラーが出るのを防ぐためである。

`dabs(a)`: 絶対値(absolute)を取る組み込み関数の倍精度型である。

`else if ( D<1.0d-30 .and. D>-1.0d-30 ) then`：ブロック if 文。条件式は `dabs(a).lt.1.0d-30` と同じ意味である。

まとめ

条件判定

単純 if 文

`if ( ) …`

( )の中には論理式が入る。実行文…は論理式が真の時だけ実行される。論理式の記号は以下の通り。

関係演算子

`==` **または** `.eq.` ：等しい (be equal to)

`>` **または** `.gt.` ：大きい (be greater than)

`>=` **または** `.ge.`：大きいあるいは等しい (be greater than or equal to)

`<` **または** `.lt.` ：小さい (be less than)

`<=` **または** `.le.`：小さいあるいは等しい (be less than or equal to)

論理演算子

`.not.` ：論理否定、後ろのみに論理式

`.and.` ：論理積(且つ)、前後に論理式

`.or.` ：論理和(または)、前後に論理式

実行優先順位は

算術演算子＞**関係演算子**＞`.not.`＞`.and.`＞`.or.`

となる。

ブロック if 文

```
if ( ) then
  …
  …
 else if ( ) then
  …
 else
  …
  …
end if
```

実行文が複数の場合や条件がいくつか続くような場合に使用する。else if の文は複数書くことが出来る。

フォーマット指定

i9 ：整数９桁

f11.6 ：固定小数点小数 11 桁、小数点以下を７桁。つまり、_-98.123456。先頭のアンダースコアのところは実際には空白１文字が入る。write のときには符号や小数点が１桁使うから全体の桁数に注意すべし。

e14.5 ：浮動小数点小数(0.4×10^{7} のような小数) 14 桁、小数点以下を５桁。つまり、__-0.12345e+34 である。先頭のアンダースコアのところは実際には空白２文字が入る。符号、０、小数点、e、指数とその符号でだいたい７桁を使うので全体の桁数に注意すべし。

4e13.6 ：e13.6 を４回繰り返すという意味。e13.6,e13.6,e13.6,e13.6 と書いたのと同じ。

a12 ：文字 12 文字

3x ：空白３文字

Gnuplot のコマンド（再録）

plot "*datafile*" ：点でグラフのプロット

plot "*datafile*" with line ：線でグラフをプロット

plot "*datafile*" using 1:2 ：データの１コラム目をｘ軸、２コラム目をｙ軸としてグラフをプロット

plot "*datafile*" using 1:3 title "*example*" ：データの１コラム目をｘ軸、３コラム目をｙ軸としてグラフをプロット、右上の注釈に example と名前を付ける。

replot "*datafile2*" ：別のファイルのグラフを重ねてプロットする

set title "*Graph1*"：グラフに Graph1 とタイトルを付ける

xlabel "*axis1*"： ｘ軸に axis1 とラベルを付ける

set yrange [-1.0:2.0]： ｙ軸の範囲を-1.0 から 2.0 までに指定する

set xtics 1.0： ｘ軸に 1.0 ごとに目盛りを付け、値を入れる。

set mxtics 5：目盛りの間を５等分して数値のない小目盛りを入れる。

set terminal postscript：グラフの出力先をポストスクリプトファイルに指定する

set output "*graph1.ps*"：グラフの出力先のファイル名を graph1.ps に指定する

save "*graph1.plt*"：Gnuplot で作成したグラフを保存する