神戸市立工業高等専門学校
電気工学科／電子工学科
専門科目「数値解析」

2017.4.28

演習1

山浦 剛（tyamaura@riken.jp）

講義資料ページ

- http://climate.aics.riken.jp/members/yamaura/numerical_analysis.html

Fortranの基礎

- Fortranは1950年代に誕生した世界初の高級プログラミング言語
  - 固定形式：FORTRAN 66, 77
  - 自由形式：Fortran 90/95, 2003, 2008
- Fortran ＝ “FORmula TRANslation”
  - 数値計算プログラム作成に適している
- Fortranがなぜ数値計算プログラム作成に適しているか
  - Fortranは、設計思想として、数値計算用プログラミング言語に特化している
  - 組込関数や複素数、配列操作など、数値計算に便利な機能が予め組み込まれている
  - 他の高級言語（C/C++/Javaなど）と比較して、数値計算を簡潔に記述することができ、一般的にパフォーマンスに優れている（言語仕様上、コンパイラにとって最適化しやすい）
  - LAPACK等、優れた数値計算ライブラリが豊富に利用可能
  - 移植性に優れ、PCからスーパーコンピュータまで様々な環境で利用可能
  - プログラミングにおいての誤りを犯しにくい

Fortranの基礎

- 大文字小文字の区別はない
- コメントは「!」以降
- 文の終了を宣言しない（;などをつけない）
  - 文を継続するには、行末に「&」をつける
- プログラムファイルの拡張子はf90
  - 例：test-program.f90
- Fortranコンパイラ（gfortran）でコンパイル
  - 例：gfortran test-program.f90
- デフォルトの出力ファイル名はa.out
  - 例：./a.out
  - -oオプションで出力ファイル名の変更も可

- 基本形
  - プログラム名、型宣言、処理を記述する

```
program [ プログラム名 ]
        implicit none

        [ 型宣言 ]

        [ 処理 ]

end program
```

Fortranの基礎

- よく使う変数の型
  - 単精度整数型: integer
  - 単精度実数型: real または real(4)
  - 倍精度実数型: double precision または real(8)
  - 単精度複素数型: complex
  - 定数: (数の型), parameter
  - 他にも文字列型や論理型などがある
- 宣言の例
  - integer :: i, j, k
  - real(8) :: x, y, z
  - real(8), parameter :: pi = 3.1415926535897932
- 演算
  - 代入: x=y
  - 加算: x+y
  - 減算: x−y
  - 乗算: x∗y
  - 除算: x/y
  - べき乗: x∗∗y
- 組み込み関数
  - 三角関数: $\sin(x)$, $\cos(x)$, $\mathrm{atan}(x)$, $\sinh(x)$
  - 対数関数: $\log(x)$, $\ln(x)$
  - 指数関数: $\exp(x)$
  - 絶対値関数: $\mathrm{abs}(x)$

Fortranの基礎

- 条件分岐
  - 論理値を使って分岐を発生させる
  - 例: if/else文による絶対値の求め方

```
a = -5
if( a < 0 ) then
        b = -a
else
        b = a
end if
```

- 反復計算
  - 例：xに1～100までの和を代入する

```
do i = 1, 100
        x = x + i
end do
```

- 出力（write文）
  - 引数1：出力先（標準出力＝*）
  - 引数2：フォーマット（自動選択＝*）

```
write(*,*) “hello”
write(*,*) “a=”,a
```

Fortranの基礎

- 型変換
  - Fortranでは、castは自動で行われる。
  - 異なる型の演算対象が含まれる演算では、強い方の型（種別も含む）が使用される。
    - 整数: 弱、 実数: 中、 複素数: 強
    - 例:
      - 整数＋実数　⇒　実数
      - 複素数＋実数　⇒　複素数
  - 演算対象が実数もしくは複素数のみで構成される場合、高い方の精度が採用される。
    - 例:
      - 単精度実数＋倍精度実数　⇒　倍精度実数
      - 倍精度実数＋単精度複素数　⇒　倍精度複素数
  - 整数の除算では切り捨てが発生する。
    - 10 / 3 ⇒ 3
    - 3 / 2 ⇒ 1

2分法

- 2分法のアルゴリズム
  1. $f(a) < 0, f(b) > 0$ となる初期値a, b を決める
  2. $c \coloneqq \frac{a+b}{2}, d \coloneqq \frac{|a-b|}{2}$ を計算
     - $\mathrm{d} < \varepsilon$ ならば3に移る
     - $\mathrm{d} \geq \varepsilon, f(c) < 0$ ならばaにcを代入し$(a \coloneqq c)$、2を繰り返す
     - $\mathrm{d} \geq \varepsilon, f(c) > 0$ ならばbにcを代入し$(b \coloneqq c)$、2を繰り返す
  3. c を数値解とする

```
program bisection_method
  implicit none

  integer :: n
  real(8) :: a, b, c, fc

  a = 1.0
  b = 2.0

  do n = 1, 20
    c = ( a + b ) / 2.0
    fc = c**3 - 5.0

    if( fc < 0.0 ) then
      a = c
    else
      b = c
    end if

    write(*,*) 'step number =',n,'; c=',c
  end do

end program
```

ニュートン法

- ニュートン法のアルゴリズム
  1. 初期値x_0と許容する誤差εを決める
  2. $c \coloneqq x - \frac{f(x)}{f'(x)}$ を計算
     - $\left|\frac{c-x}{c}\right| < \varepsilon$ ならば3に移り、そうでなければxにcを代入し、2を繰り返す
  3. c を数値解とする

```
program newton_method
  implicit none

  integer :: n
  real(8) :: x, fx, dfx

  x = 2.0

  do n = 1, 10
    fx = x**3 - 5.0
    dfx = 3.0 * x**2

    x = x - fx / dfx

    write(*,*) 'step number =',n,'; x=',x
  end do

end program
```

課題

1. 2分法のサンプルプログラムを参考に、誤差$\varepsilon = 10^{-8}$とし、誤差未満まで計算できたときに終了するように修正せよ。（ヒント：doループはexit文で抜けることができる）
2. ニュートン法のサンプルプログラムを参考に、有効数字8桁まで計算できたときに終了するように修正せよ。
3. 次の式の根を求めるプログラムを作成し、有効数字8桁以上で根を示せ。ただし、xの初期値を1とし、反復計算の反復回数は10回を上限とする。
   - $f(x) = \sin(x) + 2x + 1$
4. 次の式の根を求めるプログラムを作成し、有効数字8桁以上で根を示せ。ただし、xの初期値を2とし、反復計算の反復回数は10回を上限とする。
   - $f(x) = \exp\left(-\frac{x}{4}\right) - 2$
5. 次の式の根を求めるプログラムを作成し、有効数字8桁以上で根を示せ。ただし、xの初期値を4または-4とし、反復計算の反復回数は10回を上限とする。（ヒント：$\frac{d(\arctan(x))}{dx} = \frac{1}{1+x^2}$）
   - $f(x) = \arctan(x) - \frac{1}{2}$

提出方法

- 〆切：2017/05/12（金） 講義開始前まで
- メールにプログラムを添付
  - 主題：演習1レポート（学籍番号）
  - 宛先：tyamaura@riken.jp
  - 本文：なくてもOK
  - 添付：学籍番号_課題番号.f90 を5ファイル
    - 課題1−1：r???????_kadai01−1.f90
    - 課題1−2：r???????_kadai01−2.f90
    - 課題1−3：r???????_kadai01−3.f90
    - 課題1−4：r???????_kadai01−4.f90
    - 課題1−5：r???????_kadai01−5.f90

課題1：解答例

- ループの終了条件を意識
  - 誤差ε　1.0E-8 (=0.00000001)

```
program bisection_method
  implicit none

  ! definition
  integer :: n
  real(8) :: a, b, c, d, fc
  real(8) :: eps
  ! ----------

  ! initial value
  a = 1.0
  b = 2.0
  eps = 1.0e-8

  ! bisection method loop
  do n = 1, 100
    c = ( a + b ) / 2.0
    d = abs( a - b ) / 2.0
    fc = c**3 - 5.0

    if( fc < 0.0 ) then
      a = c
    else
      b = c
    end if

    ! output
    write(*,*) 'step number =',n,'; c=',c

    if( d < eps ) then
      exit
    end if

  end do

end program
```

```
step number =           1 ; c=   1.5000000000000000
step number =           2 ; c=   1.7500000000000000
step number =           3 ; c=   1.6250000000000000
step number =           4 ; c=   1.6875000000000000
step number =           5 ; c=   1.7187500000000000
step number =           6 ; c=   1.7031250000000000
step number =           7 ; c=   1.7109375000000000
step number =           8 ; c=   1.7070312500000000
step number =           9 ; c=   1.7089843750000000
step number =          10 ; c=   1.7099609375000000
step number =          11 ; c=   1.7104492187500000
step number =          12 ; c=   1.7102050781250000
step number =          13 ; c=   1.7100830078125000
step number =          14 ; c=   1.7100219726562500
step number =          15 ; c=   1.7099914550781250
step number =          16 ; c=   1.7099761962890625
step number =          17 ; c=   1.7099685668945312
step number =          18 ; c=   1.7099723815917969
step number =          19 ; c=   1.7099742889404297
step number =          20 ; c=   1.7099752426147461
step number =          21 ; c=   1.7099757194519043
step number =          22 ; c=   1.7099759578704834
step number =          23 ; c=   1.7099758386611938
step number =          24 ; c=   1.7099758982658386
step number =          25 ; c=   1.7099759280681610
step number =          26 ; c=   1.7099759429693222
step number =          27 ; c=   1.7099759504199028
```

課題2：解答例

- ループの終了条件を意識
  - 誤差ε　1.0E-8 (=0.00000001)
- 鍵となる変数(x)の扱いに注意

```
program newton_method
  implicit none

  ! definition
  integer :: n
  real(8) :: x, nx, fx, dfx
  real(8) :: eps
  ! ----------

  ! initial value
  x = 2.0
  eps = 1.0e-8

  ! newton method loop
  do n = 1, 10
    fx = x**3 - 5.0
    dfx = 3.0 * x**2

    nx = x - fx / dfx

    ! output
    write(*,*) 'step number =',n,'; x=',nx

    if( abs( ( nx - x ) / nx ) < eps ) then
      exit
    else
      x = nx
    end if

  end do

end program
```

```
step number =           1 ; x=   1.7500000000000000
step number =           2 ; x=   1.7108843537414966
step number =           3 ; x=   1.7099764289169748
step number =           4 ; x=   1.7099759466768329
step number =           5 ; x=   1.7099759466766971
```

課題3：解答例

- 素早く反復計算を収束させるにはニュートン法を使うこと

```
program newton_method
  implicit none

  ! definition
  integer :: n
  real(8) :: x, nx, fx, dfx
  real(8) :: eps
  ! ----------

  ! initial value
  x = 1.0
  eps = 1.0e-8

  ! newton method loop
  do n = 1, 10
    fx = sin(x) + 2.0 * x + 1
    dfx = cos(x) + 2.0

    nx = x - fx / dfx

    ! output
    write(*,*) 'step number =',n,'; x=',nx

    if( abs( ( nx - x ) / nx ) < eps ) then
      exit
    else
      x = nx
    end if

  end do

end program
```

```
step number =         1 ; x= -0.51221017118082202
step number =         2 ; x= -0.33303690657719909
step number =         3 ; x= -0.33541771697828504
step number =         4 ; x= -0.33541803238493451
step number =         5 ; x= -0.33541803238494011
```

課題4：解答例

- 式が異なるだけで、課題3同様
- 素早く反復計算を収束させるにはニュートン法を使うこと

```
program newton_method
  implicit none

  ! definition
  integer :: n
  real(8) :: x, nx, fx, dfx
  real(8) :: eps
  ! ----------

  ! initial value
  x = 2.0
  eps = 1.0e-8

  ! newton method loop
  do n = 1, 10
    fx = exp(-x/4.0) - 2.0
    dfx = - exp(-x/4.0) / 4.0

    nx = x - fx / dfx

    ! output
    write(*,*) 'step number =',n,'; x=',nx

    if( abs( ( nx - x ) / nx ) < eps ) then
      exit
    else
      x = nx
    end if

  end do

end program
```

```
step number =           1 ; x=  -7.1897701656010256
step number =           2 ; x=  -4.5155475601804209
step number =           3 ; x=  -3.1026917853911273
step number =           4 ; x=  -2.7858426363123630
step number =           5 ; x=  -2.7726106562868758
step number =           6 ; x=  -2.7725887222999188
step number =           7 ; x=  -2.7725887222397811
```

課題5：解答例

- 式$y = \mathrm{atan}(x)$は、ニュートン法で単純に解くと、初期値によっては発散してしまう関数
- 二分法で力業で解くこともできるが、ループ回転数を制限しているのでニュートン法と組み合わせる必要がある
- 始めの数回を二分法、その値を初期値としてニュートン法を行うのが実践的な解き方

```
program newton_method
  implicit none

  ! definition
  integer :: n
  real(8) :: x1, x2, x, nx, fx, dfx
  real(8) :: eps
  ! ----------

  ! initial value
  x1 = -4.0
  x2 = 4.0
  eps = 1.0e-8

  ! bisection method loop
  do n = 1, 3
    x = ( x1 + x2 ) / 2.0
    fx = atan(x) - 0.5

    if( fx < 0.0 ) then
      x1 = x
    else
      x2 = x
    end if

    ! output
    write(*,*) 'step number =',n,'; x=',x
  end do

  ! newton method loop
  do n = 4, 10
    fx = atan(x) - 0.5
    dfx = 1.0 / ( 1.0 + x**2 )

    nx = x - fx / dfx

    ! output
    write(*,*) 'step number =',n,'; x=',nx

    if( abs( ( nx - x ) / nx ) < eps ) then
      exit
    else
      x = nx
    end if

  end do

end program
```

```
step number =           1 ; x=   0.0000000000000000
step number =           2 ; x=   2.0000000000000000
step number =           3 ; x=   1.0000000000000000
step number =           4 ; x=  0.42920367320510344
step number =           5 ; x=  0.54119993245423292
step number =           6 ; x=  0.54629158818074863
step number =           7 ; x=  0.54630248979378815
step number =           8 ; x=  0.54630248984379048
```