Fortran 90/95 プログラミング II

数値計算

目次

1 プログラミング演習 3

1.1 級数 3

1.2 数値微分 8

1.3 数値積分 17

1.4 非線形代数方程式 1 30

1.5 非線形代数方程式 2 40

1.6 ベッセル関数 49

1.7 常微分方程式 64

1.8 乱数 76

1.9 ラゲルの多項式 83

1　プログラミング演習

1.1　級数

■テイラー展開　$\sin x$ を $x = 0$ においてテイラー (Taylor) 展開すると次のようになる.

$$\sin x = \sum_{n=0}^{N-1} (-1)^n \frac{x^{2n+1}}{(2n+1)!} + R_N \tag{1}$$

だたし，R_N は剰余項を示す.

■例題１＞　$\sin x$ を $x = 0$ においてテイラー展開して計算するプログラムを，次の手順に従って作成せよ．

1. プログラム作成に必要な式：　効率よく計算するため，式 (1) を次のようにする．

$$S_0 = T_0 = x \tag{2}$$
$$T_i = T_{i-1}x^2/\{2i(2i+1)\} \tag{3}$$
$$S_i = S_{i-1} + (-1)^i T_i \tag{4}$$

ただし，S_i は項数 i までで展開したときの値を示し，x は実数とする．

2. （モジュール）関数副プログラムの作成：　上式よりユーザ定義関数（テイラー展開による sin）を作成せよ．このとき，収束判定基準を設定すること（例えば T_i が 10^{-6} 以下になれば計算を終了）．
3. 主プログラムの作成：　ユーザ定義関数による値を計算，出力し，組込み関数 sin との差も出力せよ．x[deg] の値としては，例えば，$0°$ から $15°$ きざみで $90°$ まで．
4. 実行（計算）：　正しく計算できていることを確認する．
5. コメント文の確認：　関数の説明，入力データの説明等をコメント文中に記し完成させる．

■プログラム（f2_ex01.f90）　単精度の関数副プログラムによる例

```
module msubp_sin ! モジュール副プログラム
  implicit none
  real, parameter :: eps=1.0e-6 ! 収束判定条件
  integer :: n
contains
  real function my_sin(x) result(s) ! テーラ展開による sin の単精度計算
    real, intent(IN) :: x ! 入力引数
    real :: er, t
    integer :: m1, m2
    er = eps
    m2 =1
    s = x
    t = s
    do while (er>=eps)
       m1 = m2+1
       m2 = m1+1
       t = -t*(x*x/m1/m2)
       s = s+t
       er = abs(t)
    enddo
    n = m1/2
```

```
  end function my_sin
end module msubp_sin

program f2_ex01 ! 主プログラム
  use msubp_sin ! use 文
  implicit none
  real, parameter :: pi = 2.0*acos(0.0)
  real :: q = 180.0/pi, dx, x, y
  integer :: i, nx
  nx = 7 ! 計算点数
  print '(a4,1x,a8,1x,a10,1x,a12,1x,a4)','i','x[deg]','my_sin(x)','error','N'
  dx = 15.0/q
  do i = 1, nx
     x = (i-1)*dx
     y = my_sin(x)
     print '(i4,1x,f8.2,1x,f10.5,1x,f12.8,1x,i4)', i,x*q, y, y-sin(x),n
  enddo
  stop
end program f2_ex01
```

■コンパイル，リンク，実行

```
gfortran -o ex01 f2_ex01.f90; ./ex01
```

■出力結果

```
   i    x[deg]  my_sin(x)         error     N
   1      0.00     0.00000    0.00000000     1
   2     15.00     0.25882    0.00000000     3
   3     30.00     0.50000    0.00000000     4
   4     45.00     0.70711    0.00000000     4
   5     60.00     0.86603   -0.00000006     5
   6     75.00     0.96593    0.00000000     5
   7     90.00     1.00000   -0.00000006     6
```

■問題 1　関数副プログラムを ELEMENTAL FUNCTION によって作成せよ (f2_ex01_elemental.f90).

1.2　数値微分

■通常の数値微分　解析的に導関数を求めることが困難な場合，導関数を数値計算で代用して近似できる場合がある．いま，関数 $y = f(x)$ が与えられているとき，$x = x_i$ についてテイラー (Taylor) 展開すると，

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + f''(x_i)\frac{(x - x_i)^2}{2} + f'''(x_i)\frac{(x - x_i)^3}{3!} + \cdots \tag{5}$$

ここで，$x \equiv x_i + h \equiv x_{i+1}$ とおき，上式に代入すると，

$$f(x_i + h) = f(x_{i+1}) = f(x_i) + f'(x_i)h + f''(x_i)\frac{h^2}{2} + f'''(x_i)\frac{h^3}{3!} + O(h^4) \tag{6}$$

逆に，$x \equiv x_i - h \equiv x_{i-1}$ とおけば，

$$f(x_i - h) = f(x_{i-1}) = f(x_i) + f'(x_i)(-h) + f''(x_i)\frac{(-h)^2}{2} + f'''(x_i)\frac{(-h)^3}{3!} + O(h^4) \tag{7}$$

式 (6)− 式 (7) より，$f''(x_i)$ を消去すると，

$$f(x_{i+1}) - f(x_{i-1}) = 2hf'(x_i) + \frac{h^3}{3}f'''(x_i) + O(h^4) \tag{8}$$

これより，$f'(x_i)$ は次のようになる.

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} - \frac{h^2}{6} f'''(x_i) + O(h^4) \tag{9}$$

いま，$f_{i+1} = f(x_{i+1})$，$f_{i-1} = f(x_{i-1})$ が計算できれば，上式の h^2 の項および $O(h^4)$ を無視して $f'(x_i)$ は次のように近似できる.

$$f'(x_i) \simeq \frac{f(x_{i+1}) - f(x_{i-1})}{2h} \tag{10}$$

ただし，計算誤差は h^2 のオーダーとなり，これは通常の数値微分の式である.

■リチャードソンの外挿　計算誤差を小さくするため，サンプル点数を増やした方法について考えてみる．まず，式 (9) において，h の替わりに $2h$ とすると，

$$f'(x_i) = \frac{f(x_i + 2h) - f(x_i - 2h)}{2(2h)} - \frac{(2h)^2}{6} f'''(x_i) + O(h^4) \tag{11}$$

ここで，$f_{i+2} = f(x_i + 2h)$，$f_{i-2} = f(x_i - 2h)$ とおくと，

$$f'(x_i) = \frac{f_{i+2} - f_{i-2}}{4h} - \frac{4h^2}{6} f'''(x_i) + O(h^4) \tag{12}$$

式 (9)×4− 式 (12) より，$f'''(x_i)$ を消去すると，

$$4f'(x_i) - f'(x_i) = 4\,\frac{f_{i+1} - f_{i-1}}{2h} - \frac{f_{i+2} - f_{i-2}}{4h} + O(h^4) \tag{13}$$

よって，上式の $O(h^4)$ を無視すると $f'(x_i)$ は次のように近似できる．

$$f'(x_i) \simeq \frac{-f_{i+2} + 8f_{i+1} - 8f_{i-1} + f_{i-2}}{12h} \tag{14}$$

ただし，誤差は h^4 のオーダーとなり，この式はリチャードソンの外挿と呼ばれる．

■例題 2 ＞　数値微分を行うプログラムを次の要領で作成せよ.

1. 微分する関数の作成：　例えば，次のような関数 $f_1(x)$，$f_2(x)$ を作成せよ.

$$f_1(x) = 1 + x + \sin x \tag{15}$$
$$f_2(x) = x \sin x \tag{16}$$

2. 数値微分を計算するユーザ定義関数の作成：　引数に関数を用いた参照呼出しで,
   - 通常の式
   - リチャードソンの外挿の式

   を基にして，各々数値微分を行う関数副プログラムを作成せよ.
3. 主プログラムの作成：　作成した関数により計算した値と，解析的に導関数を求めた式を用いて計算した値を，出力して比較せよ.

■プログラム（f2_ex02.f90）

```
module msubp_diff ! モジュール副プログラム
  implicit none
contains
  function my_diff1(func, x, h) result(d1) ! 通常の数値微分の式
    real(8), intent(IN) :: x, h ! 入力引数
    real(8) :: func, fp, fm, d1
    external func
    fp = func(x+h)
    fm = func(x-h)
    d1 = (fp-fm)/2.0d0/h
  end function my_diff1
  function my_diff2(func, x, h) result(d2) ! リチャードソンの外挿の式
    real(8), intent(IN) :: x, h ! 入力引数
    real(8) :: func, fp, fm, d2, fpp, fmm
    external func
    fp = func(x+h)
    fm = func(x-h)
    fpp = func(x+h+h)
    fmm = func(x-h-h)
    d2 = (-fpp+8.0d0*(fp-fm)+fmm)/12.0d0/h
  end function my_diff2
```

```
end module msubp_diff

program f2_ex02 ! 主プログラム
  use msubp_diff ! use文
  implicit none
  real(8) :: dx=0.1d0, h=0.01d0, x, df1, df2, dydx
  integer :: i, nx=11

  print '(a8,1x,5(a12,1x))','x','func1(x)', 'diff1','error1','diff2','error2'
  do i = 1, nx
     x = (i-1)*dx
     df1 = my_diff1(func1, x, h)
     df2 = my_diff2(func1, x ,h)
     dydx = 1.0d0+cos(x)
     print '(f8.3,1x,5(f12.9,1x))', x, func1(x), df1,df1-dydx, df2,df2-dydx
  enddo

  print '(a8,1x,5(a12,1x))','x','func2(x)', 'diff1','error1','diff2','error2'
  do i = 1, nx
     x = (i-1)*dx
     df1 = my_diff1(func2, x, h)
     df2 = my_diff2(func2, x ,h)
     dydx = sin(x) + x*cos(x)
```

```
      print '(f8.3,1x,5(f12.9,1x))', x, func2(x), df1,df1-dydx, df2,df2-dydx
  enddo
  stop

contains
  function func1(x) result(y)
    real(8), intent(IN) :: x ! 入力引数
    real(8) :: y
    y = 1.0d0 + x + sin(x)
  end function func1
  function func2(x) result(y)
    real(8), intent(IN) :: x ! 入力引数
    real(8) :: y
    y =  x * sin(x)
  end function func2
end program f2_ex02
```

■コンパイル，リンク，実行

```
gfortran -o ex02 f2_ex02.f90; ./ex02
```

■出力結果

```
     x      func1(x)          diff1         error1          diff2         error2
 0.000  1.000000000  1.999983333 -0.000016667  2.000000000 -0.000000000
 0.100  1.199833417  1.994987582 -0.000016583  1.995004165 -0.000000000
 0.200  1.398669331  1.980050243 -0.000016334  1.980066578 -0.000000000
 0.300  1.595520207  1.955320567 -0.000015922  1.955336489 -0.000000000
 0.400  1.789418342  1.921045643 -0.000015351  1.921060994 -0.000000000
 0.500  1.979425539  1.877567936 -0.000014626  1.877582562 -0.000000000
 0.600  2.164642473  1.825321859 -0.000013756  1.825335615 -0.000000000
 0.700  2.344217687  1.764829440 -0.000012747  1.764842187 -0.000000000
 0.800  2.517356091  1.696695098 -0.000011612  1.696706709 -0.000000000
 0.900  2.683326910  1.621599608 -0.000010360  1.621609968 -0.000000000
 1.000  2.841470985  1.540293301 -0.000009005  1.540302306 -0.000000000
     x      func2(x)          diff1         error1          diff2         error2
 0.000  0.000000000  0.000000000  0.000000000  0.000000000  0.000000000
 0.100  0.009983342  0.199327183 -0.000006650  0.199333833 -0.000000000
 0.200  0.039733866  0.394669446 -0.000013200  0.394682646 -0.000000000
```

```
 0.300  0.088656062  0.582101601 -0.000019553  0.582121153 -0.000000001
 0.400  0.155767337  0.757817129 -0.000025611  0.757842739 -0.000000001
 0.500  0.239712769  0.918185535 -0.000031284  0.918216819 -0.000000001
 0.600  0.338785484  1.059807357 -0.000036485  1.059843841 -0.000000001
 0.700  0.450952381  1.179566085 -0.000041134  1.179607217 -0.000000001
 0.800  0.573884873  1.274676301 -0.000045157  1.274721457 -0.000000001
 0.900  0.704994219  1.342727391 -0.000048490  1.342775880 -0.000000001
 1.000  0.841470985  1.381722212 -0.000051078  1.381773289 -0.000000002
```

■問題 2　単精度（f2_ex02_single.f90）と倍精度（f2_ex02.f90）の計算結果を比較せよ.

■MATLAB の関数の例

- 差分：　diff(Y)
- 近似微分値：　diff(Y)./diff(X)

ただし，X はサンプル点での変数値（ベクトル），Y はサンプル点での関数値（ベクトル）を示す.

1.3　数値積分

■ニュートン-コーツの積分公式　関数 $f(x)$ を区間 [a,b] で積分する方法として，補間多項式を用いて近似する補間型の数値積分法がある．その中でも，ニュートン-コーツ (Newton-Cotes) の数値積分は，等間隔のサンプル点で計算する方法で，与えられる積分値 $I(f)$ を，次式で近似するものである．

$$I(f) = \int_a^b f(x)dx \simeq K\Delta x_n \sum_{i=0}^{n} A_i f(a + i\Delta x_n) \tag{17}$$

ただし，

$$\Delta x_n = \frac{b-a}{n} \tag{18}$$

ここで，n は補間公式の次数を示し，KA_i は cotes 数と呼ばれる．表 1 は，K および A_i を次数 n が 1 から 6 までについて示したもので，一般に，区間 [a,b] を小区間に分割して，低次の補間公式が用いられる．

表 1　cotes 数の K，A_i の例

n	1	2	3	4	5	6	$\cdots$
K	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{3}{8}$	$\frac{2}{45}$	$\frac{5}{288}$	$\frac{1}{140}$	$\cdots$
A_0	1	1	1	7	19	41	$\cdots$
A_1	1	4	3	32	75	216	$\cdots$
A_2		1	3	12	50	27	$\cdots$
A_3			1	32	50	272	$\cdots$
A_4				7	75	27	$\cdots$
A_5					19	216	$\cdots$
A_6						41	$\cdots$
$\cdots$							$\cdots$

■台形則　次数 1 の補間公式は台形則と呼ばれ，サンプル点 x_i の間隔を h，計算できる関数値を $f_i \equiv f(x_i)$ とすると，次のようになる.

$$\int_{x_i}^{x_{i+1}} f(x)dx = \frac{h}{2}(f_i + f_{i+1}) + O(h^3 f'') \tag{19}$$

ここで，$O(\alpha)$ は真の値と数値計算による値との差を示したもので，このような誤差は正確にはわからないので α（誤差のオーダー）のある係数倍程度の値であることを意味している.

■混合型台形則　台形則を基にある程度精度よく数値積分するためには，与えられた積分区間 $[a,b]$ を細かく N 分割し，その区間毎に台形則を次のように適用すればよい．

$$\begin{aligned}
&\int_a^b f(x)dx = \int_{x_0=a}^{x_1} f(x)dx + \int_{x_1}^{x_2} f(x)dx + \cdots + \int_{x_{N-1}}^{x_N=b} f(x)dx \\
&= \frac{h}{2}(f_0+f_1) + \frac{h}{2}(f_1+f_2) + \cdots + \frac{h}{2}(f_{N-1}+f_N) + O(Nh^3f'') \\
&= \frac{h}{2}\left[f_0 + 2\left(\sum_{i=1}^{N-1} f_i\right) + f_N\right] + O\left(N\frac{(b-a)^3}{N^3}f''\right) \\
&= \frac{h}{2}(f_0+f_N) + \sum_{i=1}^{N-1} f_i + O\left(\frac{(b-a)^3}{N^2}f''\right) \qquad (20)
\end{aligned}$$

ここで，

$$h = \frac{b-a}{N} \qquad (21)$$

これを混合型台形則あるいは拡張台形則という．誤差は $1/N^2$ のオーダーであり，例えば分割数 N を 2 倍にすれば，誤差は $1/4$ に減ることがわかる．ニュートン-コーツとその拡

張について説明するため，簡単な例である最低次の台形則を取り上げたが，通常の数値積分では後述するさらに精度のよい方法を用いることが多い.

■シンプソンの 1/3 則　次数 2 の補間公式はシンプソン (Simpson) の 1/3 則と呼ばれ，次のようになる.

$$\int_{x_i}^{x_{i+2}} f(x)dx = \frac{h}{3}(f_i + 4f_{i+1} + f_{i+2}) + O(h^5 f^{(4)}) \tag{22}$$

ただし，h はサンプル間隔，$f^{(4)}$ は $f(x)$ の 4 階微分を示し，誤差は $h^5 f^{(4)}$ のオーダーである.

■混合型シンプソンの 1/3 則　精度よく数値積分するためには，与えられた積分区間 $[a, b]$ を細かく N（偶数）分割し，その 2 区間毎にシンプソンの 1/3 則を次のように適用すれば

よい.

$$\int_a^b f(x)dx = \int_{x_0=a}^{x_2} f(x)dx + \int_{x_2}^{x_4} f(x)dx + \cdots + \int_{x_{N-2}}^{x_N=b} f(x)dx$$
$$\simeq \frac{h}{3}(f_0 + 4f_1 + f_2) + \frac{h}{3}(f_2 + 4f_3 + f_4) + \cdots \frac{h}{3}(f_{N-2} + 4f_{N-1} + f_N)$$
$$= \frac{h}{3}\left[f_0 + 4\left(\sum_{i=1}^{N/2} f_{2i-1}\right) + 2\left(\sum_{i=1}^{N/2} f_{2i}\right) + f_N\right] \tag{23}$$

これを混合型シンプソンの 1/3 則あるいは拡張シンプソン則といい，誤差は $1/N^4$ のオーダーである.

■シンプソンの 3/8 則　次数 3 の補間公式はシンプソンの 3/8 則と呼ばれ，次のようになる.

$$\int_{x_i}^{x_{i+3}} f(x)dx = \frac{3h}{8}(f_i + 3f_{i+1} + 3f_{i+2} + f_{i+3}) + O(h^5 f^{(4)}) \tag{24}$$

ただし，誤差は $h^5 f^{(4)}$ のオーダーとなって，シンプソンの 1/3 則と同程度である.

■例題 3 ＞　混合型台形則，あるいは混合型シンプソンの 1/3 則による数値積分のプログラムを作成し，$f(x) = \sqrt{4 - x^2}$，$a = 0$，$b = 2$ のときの積分を数値的に行い，収束を調べよ.

1. 被積分関数 $f(x)$ の作成
2. 数値積分を実行する副プログラムの作成
   (a) 数値成分のサンプル点における被積分関数値を予め計算して 1 次元配列に格納し，これを引数とする混合型台形則の関数副プログラムを作成せよ.
   (b) 同様にして混合型シンプソンの 1/3 則および 3/8 則の関数副プログラムを作成せよ.
3. 主プログラムの作成：　数値積分により得られた値と，解析的に定積分を行なった式による計算値を，出力して比較せよ.

（ヒント）定積分より，

$$I(f) = \int_a^b f(x)dx = \int_0^2 \sqrt{4 - x^2}dx = \pi \tag{25}$$

■プログラム（f2_ex03.f90）　1次元配列を引数とする例

```
module msubp_int ! モジュール副プログラム
  implicit none
contains
  function daikei(ff, n, a, b) result(y) ! 混合型台形則
    real(8), intent(IN) :: ff(:), a, b ! 入力引数
    real(8) :: y, h
    integer, intent(IN) :: n
    h = (b-a)/(n-1)
    y = sum(ff(1:n))
    y = y-(ff(1)+ff(n))*0.5d0
    y = h*y
  end function daikei
  function simpson(ff, n, a, b) result(y) ! 混合型シンプソン 1/3 則
    real(8), intent(IN) :: ff(:), a, b ! 入力引数
    real(8) :: y, h, y4, y2
    integer, intent(IN) :: n
    h = (b-a)/(n-1)
    if(n==2) then ! 台形則
       y = (ff(1)+ff(n))*h*0.5d0
    else if(mod(n,2)==1) then ! 混合型シンプソンの 1/3 則
       y4 = sum(ff(2:n-1:2))
```

```
        y2 = sum(ff(3:n-2:2))
        y = ff(1)+4.0d0*y4+2.0d0*y2+ff(n)
        y = y*h/3.0d0
    else ! 混合型シンプソンの 1/3 則 + 3/8 則
        y = ff(1)+3.0d0*(ff(2)+ff(3))+ff(4)
        y = y*h*3.0d0/8.0d0
        if(n>=6) then
            y4 = sum(ff(5:n-1:2))
            y2 = sum(ff(6:n-2:2))
            y = y +(ff(4)+4.0d0*y4+2.0d0*y2+ff(n))*h/3.0d0
        endif
    endif
  end function simpson
end module msubp_int

program f2_ex03 ! 主プログラム
  use msubp_int ! use 文
  implicit none
  integer, parameter :: nn = 100
  real(8), parameter :: pi = 2.0d0*acos(0.0D0) ! 3.1415926535 8979323846
  real(8) :: a = 0.0d0, b = 2.0d0, h, x(nn),ff(nn), y1, y2
  integer :: i, j, n, nx=100
  print *,'definite integral : y =',pi ! 定積分による値
```

```
  print '(a4,1x,4(a12,1x))','n','daikei', 'error1','simpson','error2'
  do i = 2, nx
     n = i
     h = (b-a)/(n-1)
     x(1:n) = [ (a+(j-1)*h, j=1,n) ]
     ff(1:n) = func3(x(1:n))
     y1 = daikei(ff, n, a, b)
     y2 = simpson(ff, n, a, b)
     print '(i4,1x,4(f12.9,1x))', n, y1, y1-pi, y2, y2-pi
  enddo
  stop

contains
  elemental function func3(x) result(y) ! 被積分関数
    real(8), intent(IN) :: x ! 入力引数
    real(8) :: y
    y = sqrt(4.0d0-x**2)
  end function func3
end program f2_ex03
```

■コンパイル，リンク，実行

```
gfortran -o ex03 f2_ex03.f90; ./ex03
```

■出力結果　　途中は省略している.

```
 definite integral : y =   3.141592653589793
   n       daikei        error1       simpson       error2
   2  2.000000000 -1.141592654  2.000000000 -1.141592654
   3  2.732050808 -0.409541846  2.976067743 -0.165524910
   4  2.917553379 -0.224039275  3.032247551 -0.109345102
   5  2.995709068 -0.145883585  3.083595155 -0.057997499
   6  3.037048829 -0.104543825  3.100013266 -0.041579388
   7  3.061983022 -0.079609631  3.110126237 -0.031466417
   ....................
    (省略)
   ....................

  95  3.140302720 -0.001289934  3.141088507 -0.000504147
  96  3.140323029 -0.001269624  3.141096449 -0.000496205
  97  3.140342811 -0.001249842  3.141104184 -0.000488470
  98  3.140362085 -0.001230569  3.141111720 -0.000480933
```

```
 99  3.140380868 -0.001211785  3.141119065 -0.000473588
100  3.140399178 -0.001193475  3.141126224 -0.000466429
```

■問題 3　引数に上の関数 $f(x)$ を用い，参照する関数副プログラムを作成し，実行せよ (`f2_ex03_simpson.f90`).

■MATLAB の数値積分の関数の例

- 台形則：　`trapz(X,Y)`
- 適応シンプソン則：　`quad(@func,a,b,tol)`

ただし，`X` はサンプル点での変数の値（ベクトル），`Y` は被積分関数値（ベクトル），`func` は被積分関数の関数名，`a`，`b` は積分範囲の下限値，上限値，`tol` は絶対許容誤差（省略すると 10^{-6}）を示す.

■ライブラリーの例　数値積分の方法としては，他に次のような方法がある[*1]．例えば，1 次元有限区間積分に関しては，

- ガウス・ルジャンドル公式（`f2_ex03_dgausp.f90`）．
- 二重指数関数公式（`f2_ex03_defint.f90`）．

また，2 次元有限区間積分に関しては，

- ガウス・ルジャンドル公式（`f2_ex03_dgaus2.f90`）．

[*1] 渡部 力，名取 亮，小国 力，“Fortran77 による数値計算ソフトウェア,” pp. 194–220，丸善（1989）．

1.4　非線形代数方程式 1

■二分法　二分法 (bisection method) は，$f(x) = 0$ の解 x を数値的に囲い込む一つの方法で，解 x の存在範囲 $[a, b]$ を与え，その両端の関数値の符号を見ながら，繰り返し計算によって範囲を逐次，狭めていく単純な方法である（図 1 参照）．ただし，$f(x)$ は実関数，

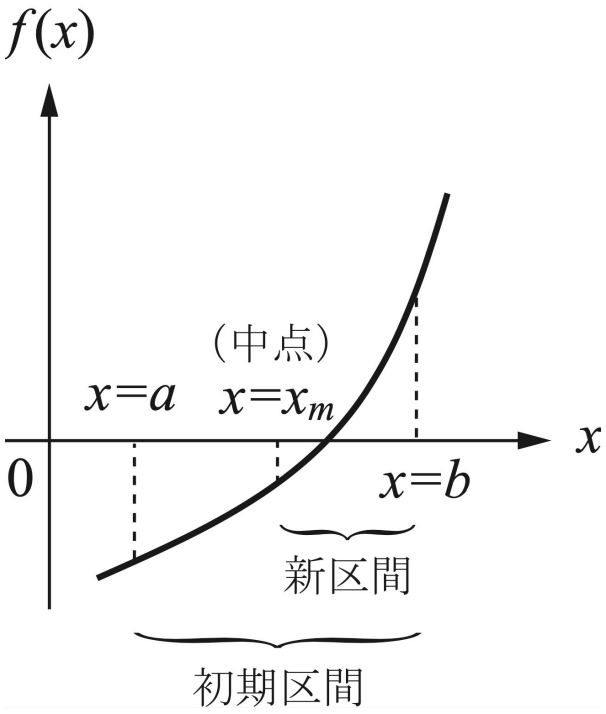

図 1　二分法

x は実変数に限られるが，導関数 df/dx の計算が不要な点で非常に有用である．計算手順は次のようになる．

1. 初期条件：　a, b を与える．
2. 解の存在について：　$f(a)f(b) < 0$ であることを確認する．$f(a)f(b) > 0$ の場合は a, b の値を修正する．
3. 逐次計算：　区間 $[a, b]$ の中点 x_m を求める．$f(a)f(x_m) < 0$ なら $[a, x_m)$ に解が存在するので $b = x_m$ と更新し，$f(a)f(x_m) > 0$ なら $(x_m, b]$ に解が存在するので $a = x_m$ と更新し，新区間 $[a, b]$ を決める．ただし，$f(a)f(x_m) = 0$ なら，$f(x_m) = 0$ 故，$x = x_m$ が解となる．
4. 収束判定：　$|a - b| < \epsilon$ まで繰り返す．例えば，$\epsilon = 10^{-9}$．
5. 解の判定：　$f(x_m)$ を計算して，$x = x_m$ が $f(x) = 0$ の解かどうかを判定する．

■例題 4 ＞　二分法を基にして，関数 $f_i(x) = 0\ (i = 1, 2)$ の解 x を求めるプログラムを，次の要領で作成せよ．

1. 零点を求める関数の作成：　例えば，次のような関数 $f_1(x)$，$f_2(x)$ を作成せよ．

$$f_1(x) = -(x + 1.5)x(x - 1.5) \tag{26}$$

$$f_2(x) = \frac{\sin x}{x} + \cos x \tag{27}$$

2. 上の関数を計算・出力するサブルーチン副プログラムの作成：　上の関数を引数とせよ．
3. 二分法の関数副プログラムの作成：　上の関数を引数とせよ．
4. 主プログラムの作成

■プログラム（f2_ex04.f90）

```
module msubp_bisect ! モジュール副プログラム
  implicit none
contains
  function bisect(func, eps, x01, x02) result(xm) ! 二分法
    real(8), intent(IN) :: eps, x01, x02 ! 収束判定条件，変数の範囲
    real(8) :: x1, x2, y, xm
    real(8), external :: func
    x1 = x01; x2 = x02
    y = func(x1)
    do while(abs(x2-x1)>eps)
       xm = (x1+x2)*0.5d0
       if(y*func(xm)>0.0d0) then
          x1 = xm
       else
          x2 = xm
       endif
    enddo
  end function bisect
  subroutine calf(func, x0, dx, nx) ! 与えられた関数計算，出力
    real(8), intent(IN) :: x0, dx ! 初期値，増分
    integer, intent(IN) :: nx ! 総数
```

```
    real(8) :: x, y
    integer :: i
    real(8), external :: func
    print '(a8,1x,a15)','x','f(x)'
    do i=1,nx
       x = x0 + dx*(i-1)
       y = func(x)
       print '(f8.3,1x,f15.11)',x,y
    enddo
  end subroutine calf
end module msubp_bisect

program f2_ex04 ! 主プログラム
  use msubp_bisect ! use文
  implicit none
  real(8), parameter :: eps = 1.0d-9
  real(8) :: xzero
  call calf(func4a, 0.0d0, 0.2d0, 21)
  xzero = bisect(func4a, eps, 0.1d0, 4.0d0)
  print *, 'zero of function: ',xzero

  call calf(func4b, 0.0d0, 0.2d0, 21)
  xzero = bisect(func4b, eps, 0.1d0, 4.0d0)
```

```
  print *, 'zero of function: ',xzero
  stop

contains
  function func4a(x) result(y)
    real(8), intent(IN) :: x ! 入力引数
    real(8) :: y
    y = -(x+1.5d0)*x*(x-1.5d0)
  end function func4a
  function func4b(x) result(y)
    real(8), intent(IN) :: x ! 入力引数
    real(8) :: y, t
    if(x==0.0d0) then
       t = 1.0d0
    else
       t = sin(x)/x
    endif
    y = t + cos(x)
  end function func4b
end program f2_ex04
```

■コンパイル，リンク，実行

```
gfortran -o ex04 f2_ex04.f90; ./ex04
```

■出力結果

```
     x              f(x)
 0.000   0.00000000000
 0.200   0.44200000000
 0.400   0.83600000000
 0.600   1.13400000000
 0.800   1.28800000000
 1.000   1.25000000000
 1.200   0.97200000000
 1.400   0.40600000000
 1.600  -0.49600000000
 1.800  -1.78200000000
 2.000  -3.50000000000
 2.200  -5.69800000000
 2.400  -8.42400000000
 2.600 -11.72600000000
 2.800 -15.65200000000
```

```
   3.000 -20.25000000000
   3.200 -25.56800000000
   3.400 -31.65400000000
   3.600 -38.55600000000
   3.800 -46.32200000000
   4.000 -55.00000000000
zero of function:    1.4999999991850927
       x             f(x)
   0.000   2.00000000000
   0.200   1.97341323182
   0.400   1.89460684977
   0.600   1.76640640390
   0.800   1.59340182297
   1.000   1.38177329068
   1.200   1.13905699278
   1.400   0.87385980718
   1.600   0.59553397960
   1.800   0.31382436691
   2.000   0.03850187687
   2.200  -0.22100275188
   2.400  -0.45595072364
   2.600  -0.65861899498
   2.800  -0.82258371561
```

```
   3.000  -0.94295249391
   3.200  -1.01653669562
   3.400  -1.04195734023
   3.600  -1.01968076169
   3.800  -0.95198294637
   4.000  -0.84284424469
 zero of function:    2.0287578385556118
```

■問題 4　主プログラムと，モジュールを別々のファイルとして，コンパイル，リンク，実行せよ．例えば，

```
gfortran -o ex04b main_ex04.f90, msubp_bisect.f90; ./ex04b
```

■ライブラリーの例　非線形方程式 $f(x) = 0$ の一つの実根を求める方法としては，他に次のような方法がある.

- 二分法とはさみうち法（$f(x)$ の符号を検査しながらセカント法を適用）を交互に行う方法[*2] （`f2_ex04_nlbit.f90`）．初期値 $x_0, x_1 (> x_0)$ の 2 点は，その根を挟むように $f(x_0)f(x_1) \leq 0$ となるように与える.
- 二分法，線型補間法，逆 2 次補間法を併用して求めるブレント法[*3]．初期値の与えた方は上と同様（`f2_ex04_dtsd1.f90`）.

[*2] 渡部 力，名取 亮，小国 力，“Fortran77 による数値計算ソフトウェア,” pp. 143–147，丸善（1989）.

[*3] 富士通 SSLII 使用手引書（科学用サブルーチンライブラリー）,pp. 593–594（1987）

1.5　非線形代数方程式 2

■ニュートン-ラフソン法　ニュートン-ラフソン法 (Newton-Raphson method) では，解 x の初期値 x_0 を与え，その点での接線より解の推定値を求め，繰り返し計算によって解の精度を上げていく．関数 $f(x)$ の $x = x_i$ における接線 $y = y(x)$ は，

$$y = f(x_i) + f'(x_i)(x - x_i) \quad (i = 0, 1, 2, \cdots) \tag{28}$$

ただし，導関数 $f'(x) = df/dx$ は解析的に求めておく必要がある．これにより，x 軸との交点 x_{i+1} を求める（図 2 参照）．このような考え方を基に，繰り返し計算を行い，解の精度を上げていくもので，具体的な計算手順は次のようになる．

1. 準備：　$f(x)$ の導関数を解析的に求める．
2. 初期条件：　解の初期値 x_0 を適切な値に決める．
3. 逐次計算：　解の推定値 x_{i+1} を次式により計算する $(i = 0, 1, 2, \cdots)$．

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{29}$$

4. 収束判定：　$|x_i - x_{i+1}| < \epsilon$ まで繰り返す．例えば，$\epsilon = 10^{-8}$．

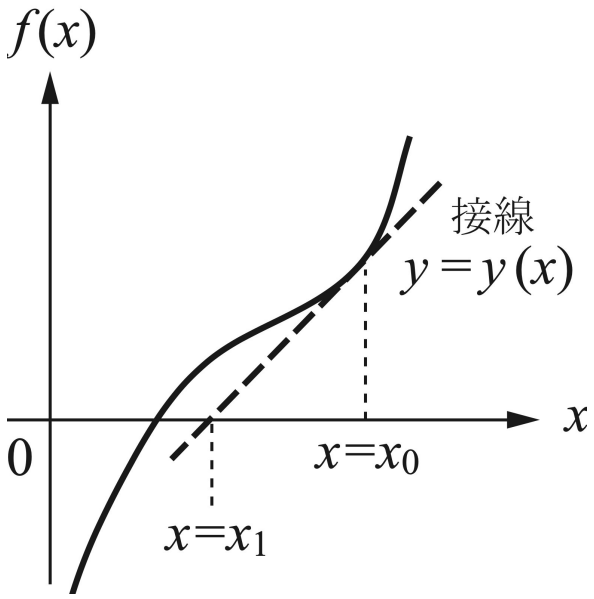

図 2　ニュートン-ラフソン法

5. 解の判定：　$f(x_{i+1})$ を計算して，$x = x_{i+1}$ が $f(x) = 0$ の解かどうかを判定する.

■例題 5 ＞　ニュートン-ラフソン法により，関数 $f(x) = 0$ の解 x を計算するプログラムを，次の要領で作成せよ.

1. 零点を求める関数の作成：　例えば，次のような関数 $f_3(x)$ を作成せよ.

$$f_3(x) = x^2 - 4 = (x + 2)(x - 2) \tag{30}$$

2. 上の関数を計算・出力するサブルーチン副プログラムの作成：　問題 4 で作成したモジュールを使用せよ.
3. 導関数 $f_3'(x)$ を計算する関数の作成
4. ニュートン-ラフソン法の関数副プログラムの作成：　上の関数を引数とせよ.
5. 主プログラムの作成

■プログラム（f2_ex05.f90）

```
module msubp_newton ! モジュール副プログラム
  implicit none
contains
  function newton(func,dfunc, eps, x00, nmax) result(xz) ! ニュートン・ラフソン法
    real(8), intent(IN) :: eps, x00 ! 収束判定条件，階の初期値
    integer, intent(IN) :: nmax ! 繰り返し回数の上限
    real(8) :: xz ! 解
    real(8), external :: func, dfunc : 関数，導関数
    integer :: icon, i
    real(8) :: x ,x0
    icon = -1
    x0 = x00
    do i=1,nmax
       x = x0-func(x0)/dfunc(x0)
       if(abs(x-x0)>eps) then
          x0 = x
       else
          icon = 0
          exit
       endif
    enddo
```

```
    xz = x
    if(icon==-1) then
       stop 'Error in newton'
    endif
  end function
end module

program f2_ex05 ! 主プログラム
  use msubp_newton
  use msubp_bisect, only: calf
  implicit none
  real(8), parameter :: eps = 1.0d-9
  real(8) :: xzero
! f3(x)=0
  call calf(func5, 0.0d0, 0.2d0, 21)
  xzero = newton(func5, dfunc5, eps, 1.0d0, 50)
  print *, 'zero of function: ',xzero
  print *, 'f(x) =',func5(xzero)
  stop

contains
  function func5(x) result(y)
    real(8), intent(IN) :: x ! 入力引数
```

```
    real(8) :: y
    y = x*x-4.0d0
  end function func5
  function dfunc5(x) result(y)
    real(8), intent(IN) :: x ! 入力引数
    real(8) :: y
    y = 2.0d0*x
  end function dfunc5
end program f2_ex05
```

■コンパイル，リンク，実行（Mac）

```
gfortran f2_ex05.f90 msubp_bisect.f90; ./a.out
```

■出力結果

```
       x              f(x)
   0.000  -4.00000000000
   0.200  -3.96000000000
   0.400  -3.84000000000
```

```
  0.600  -3.64000000000
  0.800  -3.36000000000
  1.000  -3.00000000000
  1.200  -2.56000000000
  1.400  -2.04000000000
  1.600  -1.44000000000
  1.800  -0.76000000000
  2.000   0.00000000000
  2.200   0.84000000000
  2.400   1.76000000000
  2.600   2.76000000000
  2.800   3.84000000000
  3.000   5.00000000000
  3.200   6.24000000000
  3.400   7.56000000000
  3.600   8.96000000000
  3.800  10.44000000000
  4.000  12.00000000000
zero of function:    2.0000000000000000
f(x) =   0.0000000000000000
```

■問題 5　ニュートン-ラフソン法のように 1 つの初期値を与える解法では，変数 x が複素数であってもよいので，複素解も含めて計算できる．例えば，

$$f(x) = x^2 + 4 = 0 \tag{31}$$

の解を求めるプログラム（`f2_ex05b.f90`）を作成し，実行せよ．

■ライブラリーの例　非線形方程式 $f(x) = 0$ の一つの実根を求めるにあたって，解析的な微分の式を不要とする方法として，他に次のような方法がある．

- セカント（secant）法（ニュートン・ラフソン法の微分を差分に置き換えて計算）[*4]．初期値 $x_0, x_1 (\neq x_0)$ の 2 点を与える（`f2_ex05_nlit.f90`）．
- マラー（Muller）法（セカント法を発展させ，2 次のニュートン補間多項式で近似する方法）[*5]．初期値 x_0 の 1 点を与える（`f2_ex05_dtsdm.f90`）．

なお，実係数の代数方程式 $f(x) = x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n = 0$ 場合，全ての根を求めることができる．例えば，ベアストウ・ニュートン法など（`f2_ex05_algeq.f90`）．

[*4] 渡部 力，名取 亮，小国 力，“Fortran77 による数値計算ソフトウェア,” pp. 140–143，丸善（1989）．

[*5] 富士通 SSLII 使用手引書（科学用サブルーチンライブラリー）,pp. 590–592（1987）

■MATLAB のゼロ点を求める関数の例　$f(x) = 0$ を満たす x を求める関数

- 区間 $[a, b]$ での解：　`x = fzero(@func,[a,b])`
- 初期値 x_0 近傍での解：　`x = fzero(@func,x0)`
- 一般的な形式：　`[x,fval,exitflag] = fzero(@func,x0)`

ただし，`fval` は得られた x での関数値を示す．また，`exitflag` は次のような終了条件を示す．

1: 関数が解 x に収束した．
-1: 出力関数によってアルゴリズムが終了された．
-3: 符号変化を含む区間のサーチ中に関数値 `NaN`，または `Inf` が検出された．
-4: 符号変化を含む区間のサーチ中に複素関数が検出された．
-5: `fzero` が特異点に収束した可能性がある．

1.6　ベッセル関数

■第 1 種ベッセル関数の級数展開　実数次 ν の第 1 種ベッセル関数 $J_\nu(z)$ の級数展開の公式は次のようになる.

$$J_\nu(z) = \left(\frac{z}{2}\right)^\nu \sum_{m=0}^{\infty} \frac{(-1)^m \left(\frac{z}{2}\right)^{2m}}{m!\,\Gamma(\nu+m+1)} \qquad (z \neq \text{負の実数}) \tag{32}$$

整数次 n の場合，$\nu = n$ として,

$$\Gamma(n+m+1) = (n+m)! \tag{33}$$

より，次のようになる.

$$J_n(z) = \left(\frac{z}{2}\right)^n \sum_{m=0}^{\infty} \frac{(-1)^m \left(\frac{z}{2}\right)^{2m}}{m!\,(n+m)!} \qquad (z \neq \text{負の実数}) \tag{34}$$

■プログラム作成に必要な式について　級数展開された式は，次のように変形できる.

$$J_n(x) = \left(\frac{x}{2}\right)^n \frac{1}{n!} \sum_{m=0}^{\infty} \frac{\{-(\frac{x}{2})^2\}^m}{m!\,(n+m)(n+m-1)\cdots\quad\cdot(n+1)} \tag{35}$$

いま,

$$a_i \equiv \left(\frac{x}{2}\right)^2 \frac{-1}{i(n+i)} \tag{36}$$

とおくと，次のようになる.

$$J_n(x) = \left(\frac{x}{2}\right)^n \frac{1}{n!}[1 + a_1\{1 + a_2(1 + a_3(1 + \cdots\quad\cdots))\}] \tag{37}$$

これより，効率よくプログラムで計算するため，次のような式を用いて $J_n(x)$ の近似値 S_i を求める.

$$S_0 = T_0 = \left(\frac{x}{2}\right)^n \frac{1}{n!} \tag{38}$$

$$T_i = T_{i-1} \left(\frac{x}{2}\right)^2 \frac{-1}{i(n+i)} \tag{39}$$

$$S_i = S_{i-1} + T_i \tag{40}$$

■第 1 種ベッセル関数の漸近展開　整数次 n の第 1 種ベッセル関数 $J_n(x)$ の漸近展開の公式は次のようになる.

$$J_n(x) = \sqrt{\frac{2}{\pi x}} \left\{ A_n(x) \cos\left(x - \frac{2n+1}{4}\pi \right) - B_n(x) \sin\left(x - \frac{2n+1}{4}\pi \right) \right\} \tag{41}$$

ここで,

$$A_n(x) \sim \sum_{r=0}^{\infty} (-1)^r \frac{(4n^2 - 1^2)(4n^2 - 3^2) \cdots \quad (4n^2 - (4r-1)^2)}{(2r)!\ (8x)^{2r}} \tag{42}$$

$$B_n(x) \sim \sum_{r=0}^{\infty} (-1)^r \frac{(4n^2 - 1^2)(4n^2 - 3^2) \cdots \quad (4n^2 - (4r+1)^2)}{(2r+1)!\ (8x)^{2r+1}} \tag{43}$$

■第 1 種ベッセル関数の漸近展開の補足　式 (42) の $A_n(x)$ および式 (43) の $B_n(x)$ は，低次の項から書いていくと次のようになる．

$$\begin{aligned} A_n(x) &\sim 1 + (-1)^1 \frac{(4n^2-1^2)(4n^2-3^2)}{2!\cdot(8x)^2} \\ &\quad + (-1)^2 \frac{(4n^2-1^2)(4n^2-3^2)(4n^2-5^2)(4n^2-7^2)}{4!\cdot(8x)^4} + \cdots \\ &\equiv A_{n,0} + A_{n,1} + A_{n,2} + \cdots \end{aligned} \tag{44}$$

$$\begin{aligned} B_n(x) &\sim \frac{4n^2-1^2}{8x} + (-1)^1 \frac{(4n^2-1^2)(4n^2-3^2)(4n^2-5^2)}{3!\cdot(8x)^3} \\ &\quad + (-1)^2 \frac{(4n^2-1^2)(4n^2-3^2)(4n^2-5^2)(4n^2-7^2)(4n^2-9^2)}{5!\cdot(8x)^5} + \cdots \\ &\equiv B_{n,0} + B_{n,1} + B_{n,2} + \cdots \end{aligned} \tag{45}$$

これより，低次の項から順次計算していく場合，次のように式を変形すればよい．

$$A_n(x) \sim A_{n,0} - \frac{4n^2-3^2}{2\cdot 8x} B_{n,0} - \frac{4n^2-7^2}{4\cdot 8x} B_{n,1} + \cdots \tag{46}$$

$$B_n(x) \sim B_{n,0} + \frac{4n^2-5^2}{3\cdot 8x}A_{n,1} + \frac{4n^2-9^2}{5\cdot 8x}A_{n,2} + \cdots \tag{47}$$

つまり,

$$A_n(x) \sim \sum_{r=0}^{\infty} A_{n,r}, \qquad B_n(x) \sim \sum_{r=0}^{\infty} B_{n,r} \tag{48}$$

$$A_{n,0} = 1 \tag{49}$$

$$B_{n,0} = \frac{4n^2-1}{8x} \tag{50}$$

$$A_{n,r} = -\frac{4n^2-(4r-1)^2}{2r\cdot 8x}B_{n,r-1} \quad (r=1,2,\cdots) \tag{51}$$

$$B_{n,r} = \frac{4n^2-(4r+1)^2}{(2r+1)\cdot 8x}A_{n,r} \quad (r=1,2,\cdots) \tag{52}$$

■例題 6 ＞　整数次 n の第 1 種ベッセル関数 $J_n(x)$ を，級数展開のみを用いる関数副プログラム，変数が小さい場合は級数展開，変数が大きい場合は漸近展開を用いる関数副プログラムを作成し，実行して計算値を比較せよ．ただし，x は負でない実数とする．

■プログラム（f2_ex06.f90）

```
module msubp_bessel ! モジュール関数副プログラム
  implicit none
  real(8), parameter :: eps=1.0e-12 ! 収束判定条件
  real(8), parameter :: pi = 2.0d0*acos(0.0D0) ! 3.1415926535 8979323846
contains
  function my_besselj(x, n, m) result(y) ! 第１種ベッセル関数の級数展開
    real(8), intent(IN) :: x ! 変数
    integer, intent(IN) :: n ! ベッセル関数の次数
    integer, intent(OUT) :: m ! 展開項数
    real(8) :: y, a, z
    integer :: i
    a = 1.0d0
    z = x/2.0d0
    do i=1,n
       a = a*z/i
    enddo
    y = a
    i = 0
    do while(abs(a)>=eps)
       i = i+1
       a = -a*z*z/i/(i+n)
```

```
      y = y+a
    enddo
    m = i
end function my_besselj

function besselj(x, n, m) result(y) ! 第1種ベッセル関数の級数展開と漸近展開
    real(8), intent(IN) :: x ! 変数
    integer, intent(IN) :: n ! ベッセル関数の次数
    integer, intent(OUT) :: m ! 展開項数
    real(8) :: y, a, z, za, zb, an, bn, th
    integer :: i, i1, i2
    a = 1.0
    if(abs(x)<14.0d0) then ! 級数展開
       z = x/2.0d0
       do i=1,n
          a = a*z/dble(i)
       enddo
       y = a
       i = 0
       do while(abs(a)>=eps)
          i = i+1
          a = -a*z*z/i/(i+n)
          y = y+a
```

```
      enddo
      m = i
    else ! 漸近展開
      za = 1.0d0
      zb = (4.0d0*n*n-1.0d0)/8.0d0/x
      an = za
      bn = zb
      i = 0
      i2 = 1
      do while(abs(zb)>=eps)
        i = i+1
        i1 = i2+1
        i2 = i1+1
        za = -zb*(4.0d0*n*n-(2.0d0*i1-1.0d0)*(2.0d0*i1-1.0d0))/i1/8.0d0/x
        zb =  za*(4.0d0*n*n-(2.0d0*i2-1.0d0)*(2.0d0*i2-1.0d0))/i2/8.0d0/x
        an = an+za
        bn = bn+zb
      enddo
      m = i
      th = x-(2.0d0*n+1.0d0)*pi/4.0d0
      y = sqrt(2.0d0/pi/x)*(an*cos(th)-bn*sin(th))
    endif
  end function besselj
```

```
end module

program f2_ex06 ! 主プログラム
  use msubp_bessel ! use 文
  implicit none
  real(8) :: x, x0=0.0d0, dx=1.0d0, y1, y2
  integer :: i, nx=31, m1, m2
  print '(a8,1x,a4,1x,a15,1x,a4,1x,a15)','x','m1','my_besselj','m2','besselj'
  do i = 1, nx
     x = (i-1)*dx
     y1 = my_besselj(x, 0, m1)
     y2 = besselj(x, 0, m2)
     print '(f8.2,1x,i4,1x,f15.11,1x,i4,1x,f15.11)', x,m1,y1,m2,y2
  enddo
  stop
end program f2_ex06
```

■コンパイル，リンク，実行

```
gfortran f2_ex06.f90; ./a.out
```

■出力結果

```
      x    m1        my_besselj    m2          besselj
   0.00     1    1.00000000000     1    1.00000000000
   1.00     8    0.76519768656     8    0.76519768656
   2.00    10    0.22389077914    10    0.22389077914
   3.00    12   -0.26005195490    12   -0.26005195490
   4.00    14   -0.39714980986    14   -0.39714980986
   5.00    15   -0.17759677131    15   -0.17759677131
   6.00    17    0.15064525725    17    0.15064525725
   7.00    18    0.30007927052    18    0.30007927052
   8.00    20    0.17165080714    20    0.17165080714
   9.00    22   -0.09033361118    22   -0.09033361118
  10.00    23   -0.24593576445    23   -0.24593576445
  11.00    24   -0.17119030041    24   -0.17119030041
  12.00    26    0.04768931080    26    0.04768931080
  13.00    27    0.20692610238    27    0.20692610238
  14.00    29    0.17107347611     9    0.17107347651
  15.00    30   -0.01422447282     8   -0.01422446814
  16.00    32   -0.17489907397     7   -0.17489906946
  17.00    33   -0.16985425216     7   -0.16985425181
  18.00    34   -0.01335580590     6   -0.01335580964
  19.00    36    0.14662943987     6    0.14662943523
```

```
  20.00   37   0.16702466460    6   0.16702466339
  21.00   39   0.03657906881    6   0.03657907421
  22.00   40  -0.12065146429    5  -0.12065147141
  23.00   41  -0.16241277065    5  -0.16241277984
  24.00   43  -0.05623024426    5  -0.05623027672
  25.00   44   0.09626663715    5   0.09626677916
  26.00   46   0.15599937257    5   0.15599931362
  27.00   47   0.07274168414    5   0.07274191995
  28.00   48  -0.07315711454    5  -0.07315700665
  29.00   50  -0.14784115220    5  -0.14784876242
  30.00   51  -0.08636246307    5  -0.08636798494
```

■ライブラリーの例　ベッセル関数の計算法としては，上で示したテイラー展開や漸近展開による方法の他に，漸化式による方法[*6]がある．第 1 種ベッセル関数 $J_0(x)$，$J_1(x)$，$J_2(x)$，$J_3(x)$，第 2 種ベッセル関数 $Y_0(x)$，$Y_1(x)$，$Y_2(x)$，$Y_3(x)$ について，変数を $x \leq 12$ まで計算してみよう（`f2_ex06_djbes.f90`）．

[*6] 渡部 力，名取 亮，小国 力，“Fortran77 による数値計算ソフトウェア,” pp. 12–37，丸善（1989）．

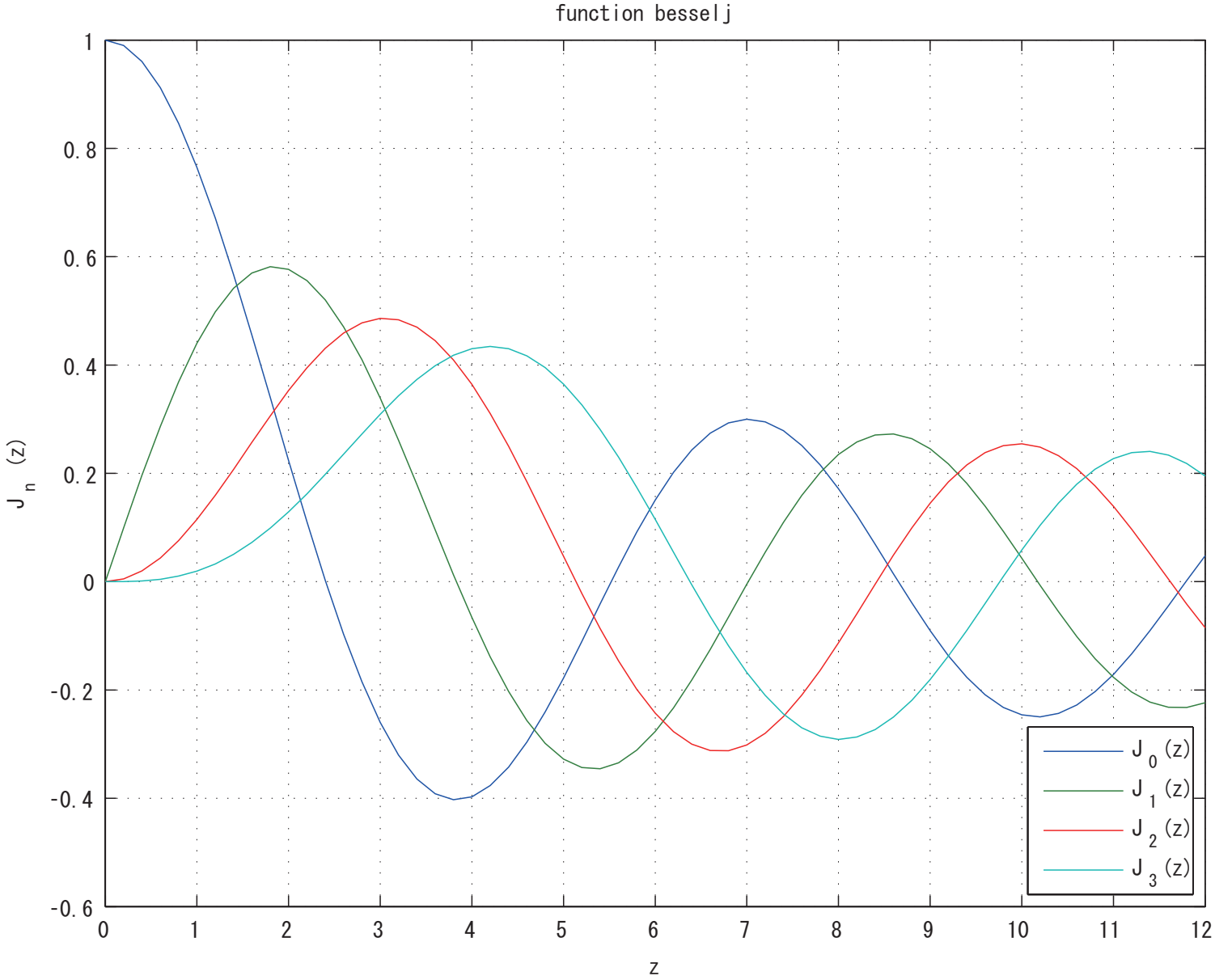

図 3　第 1 種ベッセル関数 $J_0(x)$, $J_1(x)$, $J_2(x)$, $J_3(x)$

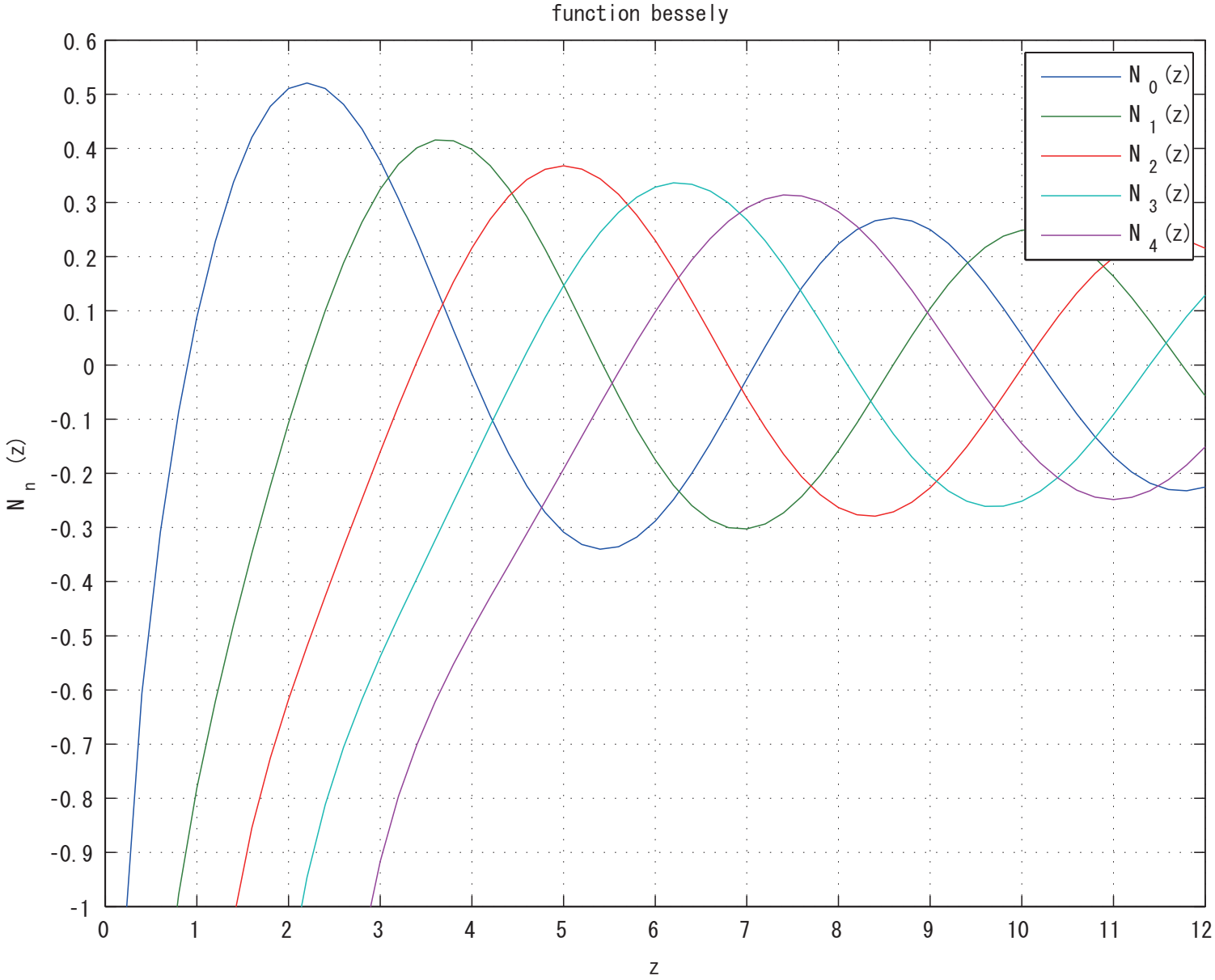

図 4　第 2 種ベッセル関数 $Y_0(x)$, $Y_1(x)$, $Y_2(x)$, $Y_3(x)$

■問題 6　第一種ベッセル関数 $J_0(x)$ の零点 x_i（$J_0(x_i) = 0$）を求めるプログラムを作成し，零点 x_i $(i = 1, 2, 3, 4, 5)$ を 5 つ求めよ.

■MATLAB のベッセル関数の例

- 第 1 種ベッセル関数：　`besselj(nu,Z)`
- 第 2 種ベッセル関数：　`bessely(nu,Z)`
- 第 1 種変形ベッセル関数：　`besseli(nu,Z)`
- 第 2 種変形ベッセル関数：　`besselk(nu,Z)`

ただし，`nu` は次数（実数），`Z` は変数（実数，あるいは実部が正でない複素数）を示す.

1.7　常微分方程式

次の常微分方程式

$$y' = \frac{dy}{dx} = f(x, y) \tag{53}$$

が与えられたとき，初期条件 $y_0 = y(x_0)$ のもとで，数値的に $y = y(x)$ を計算することを考える．いま，両辺を x について微少区間 $[x_i, x_{i+1}]$ で積分すると，

$$\int_{x_i}^{x_{i+1}} \frac{dy}{dx} dx = \int_{x_i}^{x_{i+1}} f(x, y) dx$$

$$y(x_{i+1}) - y(x_i) = \int_{x_i}^{x_{i+1}} f(x, y) dx$$

よって，$y_i = y(x_i)$ とおくと，$x = x_{i+1}$ のときの y_{i+1} は初期条件をもとに次式で計算できることになる．

$$y_{i+1} = y_i + \int_{x_i}^{x_{i+1}} f(x, y) dx \qquad (i = 0, 1, 2, \cdots) \tag{54}$$

計算精度は，上式の積分項をいかに精度よく計算できるかにかかっている．

■オイラー (Euler) 法　最も簡単な方法は，積分項を $(x_{i+1}-x_i)f(x_i,y_i)$ で近似するもので，オイラー法という．いま，微少区間を等間隔 h にとると，y_{i+1} は与えられた初期値 x_0，y_0 から次式により繰り返し計算で求めることができる．

$$y_{i+1}=y_i+hf(x_i,y_i) \qquad (i=0,1,2,\cdots) \tag{55}$$

ただし，打切り誤差は h^2 のオーダーになる．

■修正オイラー法　積分項を台形公式で近似すると，次のようになる．

$$y_{i+1}=y_i+\frac{h}{2}\{f(x_i,y_i)+f(x_{i+1},y_{i+1})\} \qquad (i=0,1,2,\cdots) \tag{56}$$

上式の右辺をみると，未知数 y_{i+1} があるので，このままでは計算できない．そこで，右辺の y_{i+1} についてオイラー法の式を用いて求めたものを修正オイラー法と呼び，次のようになる．

$$y_{i+1}=y_i+\frac{h}{2}\{f(x_i,y_i)+f(x_{i+1},y_i+hf(x_i,y_i))\} \tag{57}$$

いま,

$$k_1 = hf(x_i, y_i) \tag{58}$$

$$k_2 = hf(x_{i+1}, y_i + k_1) = hf(x_i + h, y_i + k_1) \tag{59}$$

とおくと,

$$y_{i+1} = y_i + \frac{1}{2}(k_1 + k_2) \qquad (i = 0, 1, 2, \cdots) \tag{60}$$

ただし，打切り誤差は h^3 のオーダーになる．この方法は，改良オイラー法あるいは２次のルンゲ-クッタ法ともいう.

■ 4次のルンゲ-クッタ (Runge-Kutta) 法　さらに精度のよいものとして，４次のルンゲ-クッタ法があり，微少区間の中点での値を巧みに使ったもので，次式で与えられる.

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad (i = 0, 1, 2, \cdots) \tag{61}$$

ここで,

$$k_1 = hf(x_i, y_i) \tag{62}$$

$$k_2 = hf\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right) \tag{63}$$

$$k_3 = hf\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}\right) \tag{64}$$

$$k_4 = hf(x_i + h, y_i + k_3) \tag{65}$$

ただし，打切り誤差は h^4 のオーダーになる.

■例題 7 ＞　オイラー法，修正オイラー法（2 次のルンゲ-クッタ法），4 次のルンゲ-クッタ法による関数副プログラムを各々作成し，次のような微分方程式，初期条件に対して，$0 < x \leq 2$ の範囲で，数値的に y の値を求める主プログラムを作成，実行せよ．

$$y' = \frac{dy}{dx} = -4(x-1)y \tag{66}$$

ここで，初期条件は，

$$x_0 = 0 \tag{67}$$

$$y_0 = e^{-2} \tag{68}$$

ヒント）微分方程式を解析に解くと，

$$y = e^{-2(x-1)^2} \tag{69}$$

■プログラム（f2_ex07.f90）

```
module msubp_rk ! モジュール副プログラム
  implicit none
contains
  function euler(dfunc, x0, y0, xs, n) result(y) ! オイラー法
    real(8), intent(IN) :: x0, y0 ! 初期条件, y0 = y'(x0)
    real(8), intent(IN) :: xs ! y を求める x の値
    real(8), external :: dfunc
    integer, intent(IN) :: n ! 区間数
    real(8) :: y ! 数値解, y = y(xs)
    real(8) :: h, x
    integer :: i
    h = (xs-x0)/n
    y = y0
    do i=0,n-1
       x = x0+h*i
       y = y+h*dfunc(x,y)
    enddo
  end function euler
  function rk2(dfunc, x0, y0, xs, n) result(y) ! 修正オイラー法
    real(8), intent(IN) :: x0, y0 ! 初期条件, y0 = y'(x0)
    real(8), intent(IN) :: xs ! y を求める x の値
```

```
  real(8), external :: dfunc
  integer, intent(IN) :: n ! 区間数
  real(8) :: y ! 数値解, y = y(xs)
  real(8) :: h, x, k1, k2
  integer :: i
  h = (xs-x0)/n
  y = y0
  do i=0,n-1
     x = x0+h*i
     k1 = h*dfunc(x,y)
     k2 = h*dfunc(x+h,y+k1)
     y = y+(k1+k2)*0.5d0
  enddo
end function rk2
function rk4(dfunc, x0, y0, xs, n) result(y) ! ４次のルンゲ・クッタ法
  real(8), intent(IN) :: x0, y0 ! 初期条件, y0 = y'(x0)
  real(8), intent(IN) :: xs ! y を求める x の値
  real(8), external :: dfunc
  integer, intent(IN) :: n ! 区間数
  real(8) :: y ! 数値解, y = y(xs)
  real(8) :: h, x, k1, k2, h2, k3, k4
  integer :: i
  h = (xs-x0)/n
```

```
    h2 = h*0.5d0
    y = y0
    do i=0,n-1
       x = x0+h*i
       k1 = h*dfunc(x,y)
       k2 = h*dfunc(x+h2,y+k1*0.5d0)
       k3 = h*dfunc(x+h2,y+k2*0.5d0)
       k4 = h*dfunc(x+h,y+k3)
       y = y+(k1+2.0d0*(k2+k3)+k4)/6.0d0
    enddo
  end function rk4
end module msubp_rk

program f2_ex07 ! 主プログラム
  use msubp_rk
  implicit none
  real(8), parameter :: eps = 1.0d-9
  integer :: n = 5, m = 11, i
  real(8) :: x00 = 0.0d0, y00 = exp(-2.0d0) ! 初期条件
  real(8) :: xe = 2.0d0, dx
  real(8) :: x0, y0, x, y, y1, y2, y4, y0a, y0b, y0c
  dx = (xe-x00)/(m-1)
  x0 = x00
```

```
  y0a = y00; y0b = y00; y0c = y00
  print '(a8,1x,6(a10,1x))','x', 'euler', 'er1', 'rk2', 'er2','rk4','er4'
  do i=1,m
     x = x00+dx*(i-1)
     y = func7(x)
     y1 = euler(dfunc7, x0, y0a, x, n)
     y2 = rk2(dfunc7, x0, y0b, x, n)
     y4 = rk4(dfunc7, x0, y0c, x, n)
     print '(f8.3,1x,6(f10.6,1x))',x, y1, y1-y, y2, y2-y, y4, y4-y
     x0 = x
     y0a = y1; y0b = y2; y0c = y4
  enddo
  stop

contains
  function func7(x) result(y) ! 微分方程式を解析的に解いて得られた関数
    real(8), intent(IN) :: x
    real(8) :: y
    y = exp(-2.0d0*(x-1.0d0)*(x-1.0d0))
  end function func7
  function dfunc7(x,y) result(dydx) ! 微分方程式 y' = dy/dx = f(x,y)
    real(8), intent(IN) :: x, y
    real(8) :: dydx
```

```
    dydx = -4.0d0*(x-1.0d0)*y
  end function dfunc7
end program f2_ex07
```

■コンパイル，リンク，実行

```
gfortran f2_ex07.f90; ./a.out
```

■出力結果

```
     x       euler         er1         rk2         er2         rk4         er4
 0.000    0.135335    0.000000    0.135335    0.000000    0.135335    0.000000
 0.200    0.268868   -0.009169    0.277404   -0.000633    0.278037   -0.000001
 0.400    0.463694   -0.023058    0.485100   -0.001652    0.486751   -0.000002
 0.600    0.691351   -0.034798    0.723366   -0.002783    0.726147   -0.000002
 0.800    0.887248   -0.035868    0.919466   -0.003651    0.923113   -0.000003
 1.000    0.975566   -0.024434    0.996014   -0.003986    0.999997   -0.000003
 1.200    0.914515   -0.008601    0.919419   -0.003697    0.923113   -0.000003
 1.400    0.727051    0.000902    0.723298   -0.002851    0.726147   -0.000002
 1.600    0.487461    0.000709    0.485056   -0.001696    0.486751   -0.000002
 1.800    0.273970   -0.004067    0.277421   -0.000616    0.278037   -0.000001
 2.000    0.128249   -0.007086    0.135411    0.000076    0.135335    0.000000
```

■問題 7　次のような微分方程式，初期条件に対して，$0 < x \leq 2$ の範囲で，数値的に y の値を計算せよ．

$$y' = \frac{dy}{dx} = y \cos x \tag{70}$$

ここで，初期条件は，

$$x_0 = 0 \tag{71}$$

$$y_0 = 1 \tag{72}$$

■MATLAB の常微分方程式（数値計算）の関数の例

- 2 次のルンゲ-クッタ：　`[x,y]=ode23(@odefun,xspan,y0)`
- 4 次のルンゲ-クッタ：　`[x,y]=ode45(@odefun,xspan,y0)`

ただし，`@odefun` は $y' = f(x, y)$ の関数ポインタ，`xspan` は変数 x の範囲，`y0` は y の初期値を示す．また，`x` は x の計算値，`y` は y の計算値を示す．

1.8　乱数

■乱数の発生　ある確率法則に従うでたらめな，予測できない数を**乱数**といい，通常，物理的な確率現象によって得られる．その中でも，ある区間で一様に分布する乱数を特に**一様乱数**といい，例えばさいころの目等がある．また，正規分布に従う乱数は，**正規乱数**という．このような乱数をここでは計算機でつくり出そう（**擬似乱数**）．

■線形合同法　乱数の初期値 x_0（整数）を与え，順次，一様乱数を求めていく一つの方法である線形合同法について説明する．この方法は，整数型の変数がとり得る正の範囲を，一様に擬似的にでたらめな（正）整数をとるようにしたものであり，初期値 x_0 を同じ値にすれば常に同じ乱数が得られることになる．予測できるという点で擬似乱数といわれるわけであるが，逆に考えれば再現性があるのでプログラムの動作確認には適しているといえる．

　この方法では，ある乱数 x_i（整数）から $ax_i + c$（a，c は整定数）を求め，整定数 m（$x_0 < m$）で割ったときの余りを次の乱数 x_{i+1}（整数）とする．

$$x_{i+1} = ax_i + c \quad (\mod m) \tag{73}$$

ただし，m は合同式の法，a は乗数（正整数），c は増分（正整数）を示す．より一様に分布する乱数を得るためには，整数の範囲を十分大きくとる必要があるが，漸化式はいつかは元にもどり，周期は m 以下となる．したがって，整定数 a，c は乱数の周期が十分大きくなるよう注意して選ぶことが重要である．例えば，

- $m = 2^{31} - 1 = 2147483647$
- $a = 7^5 = 16807$
- $c = 0$

なお，$c = 0$ としたのものは，乗算合同法という．

■例題 8 ＞　一様に発生する乱数を用いたプログラムを，次の要領で作成せよ.

1. 一様乱数を計算するユーザ定義関数の作成：
   (a) 0 以上，1 未満の一様乱数 $y_i = x_i/m$ を求める関数副プログラムを作成せよ. ただし，x_i は区間 $[0, m]$ の一様乱数を示す.
   (b) b_{min} 以上，b_{max} 未満の一様乱数 $z_i = (b_{max} - b_{min})y_i + b_{min}$ 求める関数副プログラムを作成せよ.
2. 主プログラムの作成：
   (a) $b_{min} = 0.0$，$b_{max} = 1.0$ とおいて 12 の乱数を生成し，6 点の 2 次元座標 (x, y) として出力せよ.
   (b) $b_{min} = 0.0$，$b_{max} = 6.0$ とおいて 6000 の乱数を生成し，区間を 6 等分して各々の区間に含まれる乱数の数を出力せよ.

■プログラム（f2_ex08.f90）　一様乱数を用いた計算例

```
module msubp_rand ! モジュール副プログラム
  implicit none
contains
  function my_rand(s) result(y) ! 擬似一様乱数（線形合同法）
    integer(4), intent(INOUT) :: s !  整数の乱数の初期値 -> 次の乱数の初期値
    integer(4), parameter :: m  =2147483647 ! =2**31-1（整数の最大値）
    integer(4), parameter :: t30=1073741824 ! =2**30
    integer(4) :: a=7**5, c=0 ! Park and Miller (e.g. s=1)
!    integer(4) :: a=843314861, c=453816693 ! FORTRAN77 text (s=1095665669)
    real(8) :: y ! 範囲 [0,1] の一様乱数
    s = a*s+c
    if(s.lt.0) then
       s = s+t30
       s = s+t30
    endif
    y = dble(s)/m
  end function my_rand
  function my_rand2(s, bmin, bmax) result(y)
    integer(4), intent(INOUT) :: s !  整数の乱数の初期値 -> 次の乱数の初期値
    real(8), intent(IN) :: bmin, bmax
    real(8) :: y ! 範囲 [bmin,bmax] の一様乱数
```

```
    y = (bmax-bmin)*my_rand(s)+bmin
  end function my_rand2
end module msubp_rand

program f2_ex08 ! 主プログラム
  use msubp_rand ! use文
  implicit none
  integer(4) ::  s=1 ! Park and Miller
!  integer(4) :: s=1095665669 ! FORTRAN77 text
  integer(4) :: n=6, nn=6000, i, iz, j, k(6)
  real(8) ::  bmin=0.0d0, bmax=6.0d0, z

  print '(2(a15,1x))','x','y'
  do i=1,n
     print '(2(f15.10,1x))', my_rand(s), my_rand(s)
  enddo

  print '(7(a6,1x))','#1','#2','#3','#4','#5','#6','total'
  k(:) = 0
  do i=1,nn
     z = my_rand2(s, bmin, bmax)
     iz = ceiling(z)
     do j=1,6
```

```
        if(iz==j) then
           k(j) = k(j)+1
        endif
     enddo
  enddo
  print '(7(i6,1x))',k(:),sum(k)
  stop
end program f2_ex08
```

■コンパイル，リンク，実行

```
gfortran f2_ex08.f90; ./a.out
```

■出力結果

```
             x                y
0.0000078264    0.1315377881
0.7556042931    0.4413479424
0.7348649240    0.8747715735
0.2858293463    0.9338209340
0.7284304983    0.7313786381
0.2807641585    0.8032093802
 #1     #2     #3     #4     #5     #6  total
983   1062   1014   1014    983    944   6000
```

■問題 8　　投げ矢を，一辺の長さが 2 の正方形に向けて多数回投げた場合を一様乱数（線形合同法）によって模擬せよ．このとき，中心 $(0, 0)$ で半径 1 の円の中に当たる確率を計算し，π の近似値を求めよ．

(ヒント)　確率の推定値は，$\pi \cdot 1^2/2^2 (= \pi/4)$ となる．

1.9　ラゲルの多項式

ラゲル（Laguerre）の多項式 $L_{n,l}(x)$ は次式で与えられる.

$$L_{n,l}(x) = \sum_{i=0}^{n} \binom{n+l}{n-i} \frac{(-x)^i}{i!} \qquad (l > -1) \tag{74}$$

ここで,

$$\binom{n+l}{n-i} =_{n+l} C_{n-i} = \frac{(n+l)!}{(n-i)!\{(n+l)-(n-i)\}!} = \frac{(n+l)!}{(n-i)!(l+i)!} \tag{75}$$

なお,

$$\binom{n}{k} =_{n} C_{k} = \frac{n(n-1)\cdots(n-k+1)}{1\cdot 2\cdots k} = \frac{n!}{k!(n-k)!} \tag{76}$$

■正規直交系　密度関数を $e^{-x}x^l$ とすると，次のような直交性が得られる.

$$\int_0^\infty e^{-x}x^l L_{n,l}(x)L_{n',l}(x)dx = \frac{(n+l)!}{n!}\delta_{n,n'} \tag{77}$$

従って，重み係数を $L_{n,l}(x)$ にかければ直交関数系が得られ，次のようになる.

$$e^{-\frac{x}{2}} x^{\frac{l}{2}} L_{n,l}(x)$$

さらに正規直交系は，式 (77) の直交性より

$$\sqrt{\frac{n!}{(n+l)!}}\, e^{-\frac{x}{2}} x^{\frac{l}{2}} L_{n,l}(x)$$

によって得られる.

■ビームモード関数（重みをつけて正規直交系にしたラゲルの多項式）

$$F_{\bar{m},n}(t) = \sqrt{\frac{n!}{(n+\bar{m})!}} \sqrt{2^{\bar{m}}} t^{\bar{m}} L_{n,\bar{m}}(2t^2) e^{-t^2} \tag{78}$$

ただし,

$$t \equiv \frac{\rho}{\omega} \tag{79}$$

■例題 9 ＞　ラゲルの多項式，およびビームモード関数を計算するプログラムを，次の要領で作成せよ.

1. ラゲルの多項式を求める関数副プログラムの作成
2. ビームモード関数を求める関数副プログラムの作成
3. 主プログラムの作成：
   (a) 低次のラゲルの多項式を計算，出力せよ.
   (b) 低次のビームモード関数を計算，出力せよ.

■プログラム（f2_ex09.f90）

```
module msubp_lague ! モジュール関数副プログラム
  implicit none
contains
  function lague(n, m, x) result(fla) ! ラゲルの多項式
    integer, intent(IN) :: n, m ! 次数
    real(8), intent(IN) :: x ! 変数
    real(8) :: fla, f1, f2, f3, c1, c2
    integer :: j, je, n1

    f1 = 1.0d0
    fla = f1
    if(n.ne.0) then
       f2 = dble(1+m)-x
       fla = f2
       if(n.ne.1) then
          je = n-1
          do j=1,je
             n1 = j+1
             c2 = x-dble(2*n1+m-1)
             c1 = dble(n1+m-1)
             f3 = (-c2*f2-c1*f1)/dble(n1)
```

```
            f1 = f2
            f2 = f3
         enddo
         fla = f3
      endif
   endif
   return
end function lague

recursive function dfactorial(n) result(k) ! 階乗
   implicit none
   integer, intent(IN) :: n
   real(8) :: k
   if(n==1 .or. n==0) then
      k = 1.0d0
   else
      k = n*dfactorial(n-1)
   endif
   return
end function dfactorial

recursive function dfactorial2(n,m) result(k) ! n!/m! (n>m)
   implicit none
```

```
  integer, intent(IN) :: n, m
  real(8) :: k
  if(n==1 .or. n==0 .or. n==m) then
     k = 1
  else
     k = n*dfactorial2(n-1,m)
  endif
  return
end function dfactorial2

function bm_func(m ,n, t) result(f) ! ビームモード
  integer, intent(IN) :: m, n ! ビームモードの次数
  real(8), intent(IN) :: t  ! 変数
  real(8) :: f, x
  if(m.eq.0 .and. n.eq.0) then
     f = 1.0d0
  else
     x = 2.0d0*t*t
     f = 1.0d0/dfactorial2(n+m,n)
     f = sqrt(f)*(sqrt(2.0d0)*t)**m
     f = f*lague(n,m,x)
  endif
  f = f*exp(-t*t)
```

```
  end function bm_func
end module msubp_lague

program f2_ex09 ! 主プログラム
  use msubp_lague ! use文
  implicit none
  real(8) :: x, x0=0.0d0, dx=0.1d0
  integer :: i, nx=41, m1, m2
  print '(4(a8,1x))','x','L0,0','L0,1','L0,2'
  do i = 1, nx
     x = x0+(i-1)*dx
     print '(4(f8.3,1x))', x, lague(0,0,x), lague(0,1,x), lague(0,2,x)
  enddo
  print '(4(a8,1x))','x','L1,0','L1,1','L1,2'
  do i = 1, nx
     x = x0+(i-1)*dx
     print '(4(f8.3,1x))', x, lague(1,0,x), lague(1,1,x), lague(1,2,x)
  enddo
  print '(4(a8,1x))','x','L2,0','L2,1','L2,2'
  do i = 1, nx
     x = x0+(i-1)*dx
     print '(4(f8.3,1x))', x, lague(2,0,x), lague(2,1,x), lague(2,2,x)
  enddo
```

```
print '(7(a8,1x))','x','F0,0','F0,1','F0,2','F0,3','F0,4','F0,5'
do i = 1, nx
   x = x0+(i-1)*dx
   print '(7(f8.3,1x))', x,bm_func(0,0,x),bm_func(0,1,x),bm_func(0,2,x),&
        bm_func(0,3,x),bm_func(0,4,x),bm_func(0,5,x)
enddo
print '(7(a8,1x))','x','F1,0','F1,1','F1,2','F1,3','F1,4','F1,5'
do i = 1, nx
   x = x0+(i-1)*dx
   print '(7(f8.3,1x))', x,bm_func(1,0,x),bm_func(1,1,x),bm_func(1,2,x),&
        bm_func(1,3,x),bm_func(1,4,x),bm_func(1,5,x)
enddo
  print '(7(a8,1x))','x','F2,0','F2,1','F2,2','F2,3','F2,4','F2,5'
do i = 1, nx
   x = x0+(i-1)*dx
   print '(7(f8.3,1x))', x,bm_func(2,0,x),bm_func(2,1,x),bm_func(2,2,x),&
        bm_func(2,3,x),bm_func(2,4,x),bm_func(2,5,x)
enddo
  print '(7(a8,1x))','x','F0,0','F0,1','F0,2','F0,3','F0,4','F0,5'
do i = 1, nx
   x = x0+(i-1)*dx
   print '(7(f8.3,1x))', x,bm_func(3,0,x),bm_func(3,1,x),bm_func(3,2,x),&
```

```
            bm_func(3,3,x),bm_func(3,4,x),bm_func(3,5,x)
  enddo
  stop
end program f2_ex09
```

■コンパイル，リンク，実行

```
gfortran f2_ex09.f90; ./a.out
```

■出力結果

```
     x      L0,0      L0,1      L0,2
 0.000     1.000     1.000     1.000
 0.100     1.000     1.000     1.000
 0.200     1.000     1.000     1.000
 ....................
 (省略)
 ....................

     x      F0,0      F0,1      F0,2      F0,3      F0,4      F0,5
 0.000     1.000     1.000     1.000     1.000     1.000     1.000
 0.100     0.990     0.970     0.951     0.931     0.912     0.893
 0.200     0.961     0.884     0.810     0.739     0.671     0.606
 ....................
 (省略)
 ....................
```

■問題 9　　計算した結果を作図せよ.