1 常微分方程式

1.1 全般的注意

1年生で習った力学は見方を変えれば常微分方程式をいかに解くかというテクニックを講述されたと言える。従って微分方程式を解くことがニュートン力学を数百年に渡って支えてきたと見做せる。それを裏付ける様に哲学者であり数学者でもあるラプラスは「私が充分な能力の計算機を持ったならば万物は初期値問題として解く事ができる。」と言いはなった。一方、ポアンカレ以来、ラプラスの素朴な（楽観的な）力学観は誤りで単に微分方程式を解くという作業が2重の意味で微妙で難しい問題を含む事が認識されるようになってきた。一つは非線形微分方程式系のカオスの問題であり、もう一つは差分近似の難しさ（とそれに伴うカオス、離散数学の発展）である。

ここでは標準的な差分近似 (Euler 法) で基の微分方程式の性質を保つには何に注意をすべきかを解説する。最後に実際的な応用として初期値問題に対する古典的な Runge-Kutta 法との応用を説明する。

1.2 微分と差分

一般に厳密な数値微分は計算機の中に存在しない。計算機には無限小という概念が理解できないからである。そこで近似的に微分を扱うことになる。その方法は大別して2つに分類できる。(1) 差分法、(2) 関数近似法である。(2) としてはガレルキン法、スペクトル法、更には有限要素法等があるがここでは例えばフーリエ展開等のような完全系をなす基底関数で展開するというアイデアを紹介するのにとどめておく。従って本講義では専ら差分法を紹介する。

既に第2講の桁落ちの項で触れた通り、微分と差分の関係は微妙なものがある。理論的には差分間隔を小さくするとより厳密な解析に近付きそうだが、桁落ちその他のために期待通りの結果は得られない。また刻みを小さくすれば計算時間が余計にかかるため、現実的では小さな刻みを取れない場合もある。

高校で習う様に

$$f'(a) = \lim_{h\to 0} \frac{f(a+h) - f(a)}{h} \tag{1}$$

で微分を定義するが既に述べた通り、計算機の中では $h \to 0$ という極限は意味を持たない。h を有限にすれば当然誤差がある。$D_h \equiv (f(a+h) - f(a))/h$ とするならば

$$D_h - f'(a) = \frac{h}{2} f''(\xi) = O(h); \quad a < \xi < a + h \tag{2}$$

である。

(1) で定義される D_h は差分法の言葉では前進差分と呼ばれる。他にも

$$\frac{f(a) - f(a-h)}{h} = f'(a) - \frac{h}{2} f''(\xi); \quad a - h < \xi < a \tag{3}$$

という後退差分や

$$\frac{f(a+h/2)-f(a-h/2)}{h} = f'(a) + \frac{h^2}{48}(f'''(\xi_1)+f'''(\xi_2)),$$
$$\text{for} \quad a-h/2 < \xi_1, \xi_2 < a+h/2 \tag{4}$$

という中心差分が良く用いられる。勿論、より複雑な組み合わせで近似度の高い差分式をつくり出すことは可能である。2階微分についても全く同様であり、例えば中心差分を用いると

$$f''(a) = \frac{1}{h^2}\{f(a+h)+f(a-h)-2f(a)\} + O(h^2) \tag{5}$$

となる。

1.3 差分方程式における初期値問題

初期値問題を考えてみよう。

$$y'(t) = f(t, y(t)), \quad y(0) = a \tag{6}$$

を考えるべき方程式であるとする。時間について刻み h の前進差分を取るならば $y(t) \to Y(t)$ として（差分方程式と微分方程式の解は異なるので）、

$$Y(t+h) = Y(t) + hf(t, Y(t)) \tag{7}$$

となる。従って $Y(h) = a + hf(0,a), Y(2h) = Y(h) + hf(h, Y(h)), \cdots$ という具合に順に計算可能である。従って (7) は $Y(jh)$ と $Y((j+1)h)$ の間の漸化式を与えるものと解釈できる。

(7) はオイラー (Euler) 法と呼ばれる最も単純な差分スキームである。このスキームは後に説明する様に h を大きくすることができないので標準的な方法となっていない。最も標準的な手法は (古典的 4 次の) ルンゲ＝クッタ (Runge-Kutta) 法と呼ばれるものである。これは次の意味でオイラー法の一般化と言える。

$$\begin{aligned} Y_{n+1} &= Y_n + h\sum_i^s b_i k_i \\ k_i &= f(t_n + c_i h, Y_n + h\sum_{j=1}^s a_{ij}k_j) \end{aligned} \tag{8}$$

という一般の Runge-Kutta 法を考えよう。ここで $c_i = \sum_j^s a_{ij}$ が成り立っている。自然数 s は段差と呼ばれ、a_{ij},b_i,c_i は公式を定めるパラメータである。ここで (8) が逐次的に解けるために $a_{ij} = 0$ for $j \geq i$ を仮定する。

次に適合条件を与えよう。局所離散誤差を

$$T_{n+1} \equiv \frac{1}{h}\left(Y_{n+1} - Y_n - h\sum_{i=1}^s b_i k_i\right) \tag{9}$$

で定義する。$T_{n+1} = O(h^p)$ という時、p 次と言い、$p \geq 1$ の時適合であると言う。k_i の h について 0 次項を取り出すと (8) から

$$k_i = f(t_n, Y_n) + O(h) = Y_n' + O(h) \tag{10}$$

である。従って (9) から Runge-Kutta 法の適合条件は

$$\sum_{i=1}^{s} b_i = 1 \tag{11}$$

である。

$s = 1$ の時を考えてみる。条件より $a_{11} = c_1 = 0$ となる。一方、適合条件 (11) から $b_1 = 1$ となる。これはオイラー法に他ならない。$s = 2$ とする 2 段 RK 法ではパラメータは $a_{21}, b_1, b_2, c_1, c_2$ であるが、$c_i = \sum_j^s a_{ij}$ から $c_1 = 0$,$a_{21} = c_2$ となる。適合性から $b_1 + b_2 = 1$ となる。もう一つ次数条件を加えると $b_2 c_2 = 1/2$ と決まり、2 段 2 次の RK 公式が得られる。

この様な方法を 4 段 4 次に適用したのがいわゆる古典的 Runge-Kutta 法である。ここでは天下り的であるが結果を書くと

$$Y_{n+1} - Y_n = \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{12}$$

但し、

$$\begin{aligned} k_1 &= f(t_n, Y_n), k_2 = f(t_n + \frac{h}{2}, Y_n + \frac{h}{2}k_1) \\ k_3 &= f(t_n + \frac{h}{2}, Y_n + \frac{h}{2}k_2), \quad k_4 = f(t_n + h, Y_n + hk_3) \end{aligned} \tag{13}$$

である。この公式は 4 次収束する（つまり誤差が $O(h^4)$ である）ことが知られている。

1.4 減衰振動の計算例

減衰振動の方程式

$$y'' + 10y' + 16y = 0, \quad y(0) = 1, y'(0) = 0 \tag{14}$$

を考えてみよう。まず (14) を連立一次方程式に変形する。

$$y_1' = y_2, \quad y_2' = -16y_1 - 10y_2, \quad y_1(0) = 1, y_2(0) = 0 \tag{15}$$

この解析解は

$$y(t) = \frac{1}{3}(4e^{-2t} - e^{-8t}) \tag{16}$$

である。一方、オイラー法を用いた差分方程式も解くことが出来て

$$Y_n^{(1)} = \frac{1}{3}\{4(1-2h)^n - (1-8h)^n\} \tag{17}$$

となる。勿論、(16) と (17) は $h \to 0$ で一致する。しかし注意して欲しいのは差分解が収束するための必要条件として $|1-2h| < 1$ かつ $|1-8h| < 1$ が課されることである。つまり $h > 1/4$ ではスキームは不安定になる。また $1/8 < h < 1/4$ の場合でも $(1-8h)^n$ は絶対値は小さくなるが符合は変化する。従ってこの場合も解析解とは異なった区分線形の解しか得ることができない。

このことはプログラミングをして実際にオイラー法に従って方程式を解くことで理解できる。アルゴリズムは極めて簡単で初期設定の後、$j = 0, 1, ...N-1$ の順に

$t_j = jh$
$Y_{1n} = Y_1 + hf_1(t, Y_1, Y_2)$
$Y_{2n} = Y_2 + hf_2(t, Y_1, Y_2)$
$Y_1 = Y_{1n}, Y_2 = Y_{2n}$
を繰り返すだけである。

```
      external f,g

      ti=0.0
      tf=2.0
      y0=0.0
      x0=1.0
      n=10

      call euler(f,g,ti,tf,n,x0,y0)
      stop
      end

      function f(t,x,y)
         f=y
      end

      function g(t,x,y)
         g=-16.0*x-10.0*y
      end

ccccccccccccc subroutine CCCCCCCCCCCCCCCCCCCCCCCccccccccccccccccc

      subroutine euler(f,g,ti,tf,n,x0,y0)
      external f,g

      dt=(tf-ti)/real(n)

      t=ti
      x=x0
      y=y0

      do 10 i=1,n

        if (mod(i,1).eq.0) then
          write (6,100) t,x
100       format(5x,2f10.4)
        endif

        t=t+dt
        x=x+dt*f(t,x,y)
        y=y+dt*g(t,x,y)

10    continue
```

```
        return
        end
```

この式に従って解いたものを図２にプロットしてある。

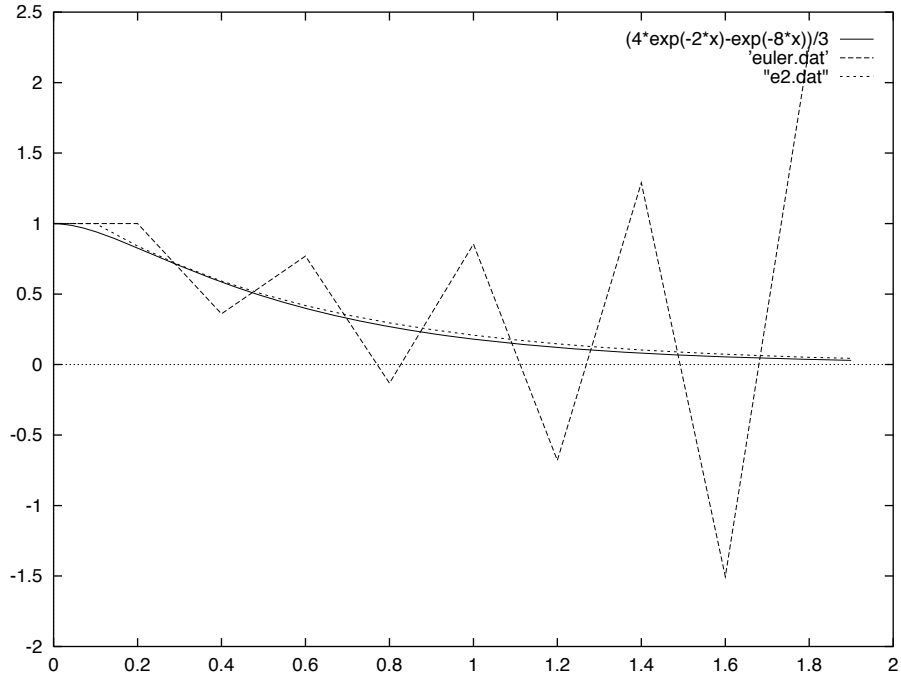

図 1: 実線は厳密解、euler.dat は刻みの個数 10, e2.dat は刻みを 20 にしたもの。

```
real(8),external::f,g
real(8)::ti,tf,y0,x0
integer::n

ti=0.0_8
tf=2.0_8
y0=0.0_8
x0=1.0_8
n=40

call euler(f,g,ti,tf,n,x0,y0)

end


real(8) function f(t,x,y)
real(8)::t,x,y
```

```
  f=y
end

real(8) function g(t,x,y)
real(8)::t,x,y
  g=-16.0_8*x-10.0_8*y
end



!!!!!!subroutine!!!!!!


subroutine euler(f,g,ti,tf,n,x0,y0)
real(8),external::f,g
real(8)::ti,tf,y0,x0,t,dt,x,y
integer::n

dt=(tf-ti)/n
t=ti
x=x0
y=y0

do i=1,n

   print'(5x,2f20.8)',t,x


t=t+dt
x=x+dt*f(t,x,y)
y=y+dt*g(t,x,y)

enddo

return

end subroutine
```

が Fortran90 の自由形式のプログラムであり、

```
#include <stdio.h>
#include <math.h>
```

```
double f(double, double, double);
double g(double, double, double);
void euler(double (*)(double,double,double), double (*)(double,double,double),
   double, double, int, double, double);

main()
{
  double ti, tf, y0, x0;
  int n;

  ti = 0.0;
  tf = 2.0;
  y0 = 0.0;
  x0 = 1.0;
  n = 10;

  euler(f,g,ti,tf,n,x0,y0);

  return 0;
}

double f(double t, double x, double y)
{
  return y;
}

double g(double t, double x, double y)
{
  return -16.0 * x - 10.0 * y;
}

void euler(double (*f)(double,double,double), double (*g)(double,double,double),
   double ti, double tf, int n, double x0, double y0)
{
  int i;
  double dt, t, x, y;

  dt = (tf - ti)/(double)n;

  t = ti;
  x = x0;
  y = y0;
```

```
  for(i = 1; i <= n; i++){

    if((i % 1) == 0){
      printf("     %10.4g %10.4g\n", t,x);
    }

    t += dt;
    x += dt * f(t,x,y);
    y += dt * g(t,x,y);

  }

  return;
}
```

が C の対応するプログラムである。

一方、ルンゲ・クッタ法は収束の良い方法であることがよくわかる。例として

$$y'' = -0.1y' - y, \quad y(0) = 1, y'(0) = 0 \tag{18}$$

という減衰振動の方程式を解いてみよう。解析解は $y(t) = \exp(-0.05t)\cos(t\sqrt{3.99}/2)$ であるが RK 法が非常にいい結果を与えていることが図 3からわかる。

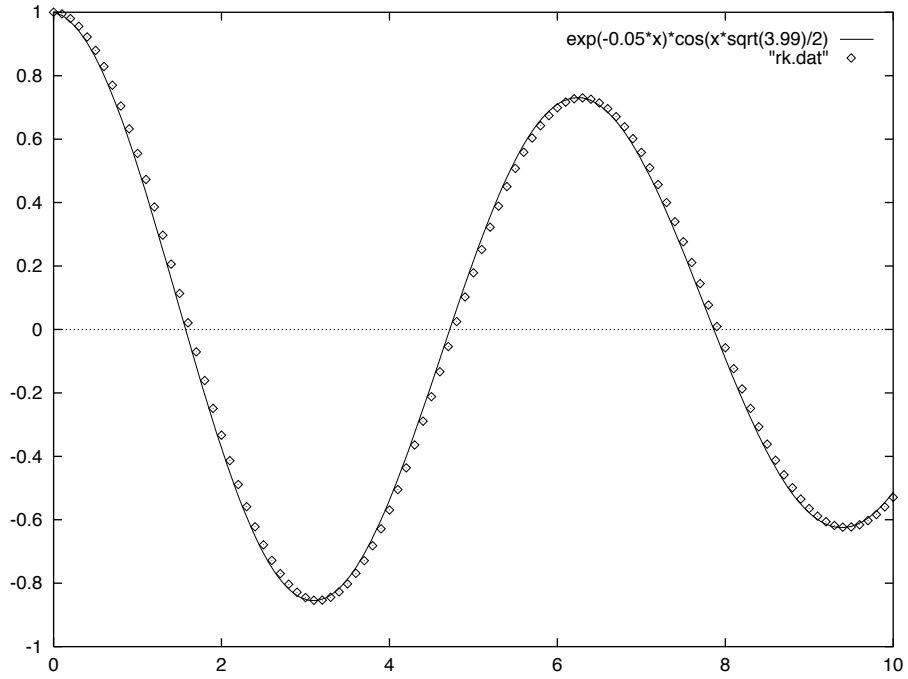

図 2: 解析解と Runge-Kutta 法で得られた数値解。両者の一致は良い。

サンプルプログラムは以下の通り。(アルゴリズムは省略)。

```
      external f,g

      ti=0.0
      tf=10.0
      y0=0.0
      x0=1.0
      n=100

      call rk4(f,g,ti,tf,n,x0,y0)
      stop
      end

      function f(t,x,y)
         f=y
      end

      function g(t,x,y)
         g=-x-0.1*y
      end

ccccccccccccccc subroutineCCCCCCCCCCCCCCCCCCCCCCCCCCC

      subroutine rk4(f,g,ti,tf,n,x0,y0)
      external f,g

      dt=(tf-ti)/real(n)

      t=ti
      x=x0
      y=y0
      dt2=dt/2.0


      do 10 i=0,n

        if (mod(i,1).eq.0) then
          write (6,100) t,x
100       format(5x,2f10.4)
        endif

      dx1=dt*f(t,x,y)
      dy1=dt*g(t,x,y)
      dx2=dt*f(t+dt2,x+dx1/2.0,y+dy1/2.0)
```

```
          dy2=dt*g(t+dt2,x+dx1/2.0,y+dy1/2.0)
          dx3=dt*f(t+dt2,x+dx2/2.0,y+dy2/2.0)
          dy3=dt*g(t+dt2,x+dx2/2.0,y+dy2/2.0)
          dx4=dt*f(t+dt,x+dx3,y+dy3)
          dy4=dt*g(t+dt,x+dx3,y+dy3)

          t=t+dt
          x=x+(dx1+2.0*dx2+2.0*dx3+dx4)/6.0
          y=y+(dy1+2.0*dy2+2.0*dy3+dy4)/6.0

 10     continue

        end
```

である。Fortran 90 自由形式は

```
real(8),external::f,g
real(8)::ti,tf,y0,x0
integer::n

ti=0.0_8
tf=10.0_8
y0=0.0_8
x0=1.0_8
n=200

call rk4(f,g,ti,tf,n,x0,y0)

end

real(8) function f(t,x,y)
real(8)::t,x,y
  f=y
end

real(8) function g(t,x,y)
real(8)::t,x,y
  g=-x-0.1_8*y
end


!!!!!subroutine!!!!!

subroutine rk4(f,g,ti,tf,n,x0,y0)
```

```
real(8),external::f,g
real(8)::ti,tf,y0,x0,t,dt,x,y,dt2,dx1,dx2,dx3,dx4,dy1,dy2,dy3,dy4
integer::n

dt=(tf-ti)/n
t=ti
x=x0
y=y0
dt2=dt/2.0_8

do i=0,n

print'(5x,2f20.8)',t,x

dx1=dt*f(t,x,y)
dy1=dt*g(t,x,y)
dx2=dt*f(t+dt2,x+dx1/2.0_8,y+dy1/2.0_8)
dy2=dt*g(t+dt2,x+dx1/2.0_8,y+dy1/2.0_8)
dx3=dt*f(t+dt2,x+dx2/2.0_8,y+dy2/2.0_8)
dy3=dt*g(t+dt2,x+dx2/2.0_8,y+dy2/2.0_8)
dx4=dt*f(t+dt,x+dx3,y+dy3)
dy4=dt*g(t+dt,x+dx3,y+dy3)

t=t+dt

x=x+(dx1+2.0_8*dx2+2.0_8*dx3+dx4)/6.0_8
y=y+(dy1+2.0_8*dy2+2.0_8*dy3+dy4)/6.0_8

enddo

end subroutine
```

である。また C は

```
#include <stdio.h>
#include <math.h>

double f(double, double, double);
double g(double, double, double);
void rk4(double (*)(double,double,double), double (*)(double,double,double),
   double, double, int, double, double);

main()
{
```

```
  double ti, tf, y0, x0;
  int n;

  ti = 0.0;
  tf = 10.0;
  y0 = 0.0;
  x0 = 1.0;
  n = 100;

  rk4(f,g,ti,tf,n,x0,y0);

  return 0;
}

double f(double t, double x, double y)
{
  return y;
}

double g(double t, double x, double y)
{
  return -x - 0.1 * y;
}

void rk4(double (*f)(double,double,double), double (*g)(double,double,double),
   double ti, double tf, int n, double x0, double y0)
{
  int i;
  double dt, dt2, t, x, y;
  double dx1, dx2, dx3, dx4;
  double dy1, dy2, dy3, dy4;

  dt = (tf - ti)/(double)n;

  t = ti;
  x = x0;
  y = y0;
  dt2 = dt / 2.0;

  for(i = 0; i <= n; i++){

    if((i % 1) == 0){
      printf("     %10.4g %10.4g\n", t,x);
```

```
    }

    dx1 = dt * f(t,x,y);
    dy1 = dt * g(t,x,y);
    dx2 = dt * f(t + dt2, x + dx1 / 2.0, y + dy1 / 2.0);
    dy2 = dt * g(t + dt2, x + dx1 / 2.0, y + dy1 / 2.0);
    dx3 = dt * f(t + dt2, x + dx2 / 2.0, y + dy2 / 2.0);
    dy3 = dt * g(t + dt2, x + dx2 / 2.0, y + dy2 / 2.0);
    dx4 = dt * f(t + dt, x + dx3, y + dy3);
    dy4 = dt * g(t + dt, x + dx3, y + dy3);

    t += dt;
    x += (dx1 + 2.0 * dx2 + 2.0 * dx3 + dx4) / 6.0;
    y += (dy1 + 2.0 * dy2 + 2.0 * dy3 + dy4) / 6.0;

  }

  return;

}
```

となる。

1.5 多自由度系に対する Runge-Kutta 法

実際に数値計算をする場合は前節までのような単独の常微分方程式を解く事は稀であり、大抵の場合は連立させた方程式を一挙に解く。その場合に前節のアルゴリズムでプログラムを組むとサブルーチンが膨大になって対応できない。その意味で単純に組むと汎用性のないものになる。この場合はサブルーチンを使わずにべたにメインルーチンに書き込む事が可能であるがやや式が繁雑になるのと、同時に異なった問題になった場合に大幅に書き換えが必要になって間違いを誘発する。そのために一般的に勧められない方法である。ここでは 98 年度のレポート試験を参考にしてそのプログラムの組み方を考えてみよう。

ここでは戸田格子と呼ばれる次の系を考えよう：

$$\ddot{r}_n = 2\exp(-r_n) - \exp(-r_{n-1}) - \exp(-r_{n+1}) \tag{19}$$

ここで r_n は格子間隔である。(19) を解くためには直接解くよりも

$$\psi_n\ddot{\psi}_n - \dot{\psi}_n^2 = \psi_{n-1}\psi_{n+1} - \psi_n^2 \tag{20}$$

を解いた方がいいことが知られている。但し

$$S_n = \ln\psi_n, \quad r_n = 2S_n - S_{n+1} - S_{n-1} \tag{21}$$

という関係で結ばれている。ここで初期条件

$$\psi_n = 2 + \sin(2\pi n/N), \quad \dot{\psi}_n = 0 \tag{22}$$

(N 格子点の総数。ここでは $N = 50$ とした）として

$$q_n = \exp(-r_n) - 1 \tag{23}$$

の時間発展を図で示す。但しサンプルの図では (t, n, q_n) のデータを $t = 0$ から $t = 100$ まで 2 刻みで保存してある。また gnuplot で絵を描く場合にはまず set parametric とタイプして set view 10,80 として sp 'toda.dat' w l としている。view をいじったり、初期条件を変えたりしていろいろ試してみると面白い。

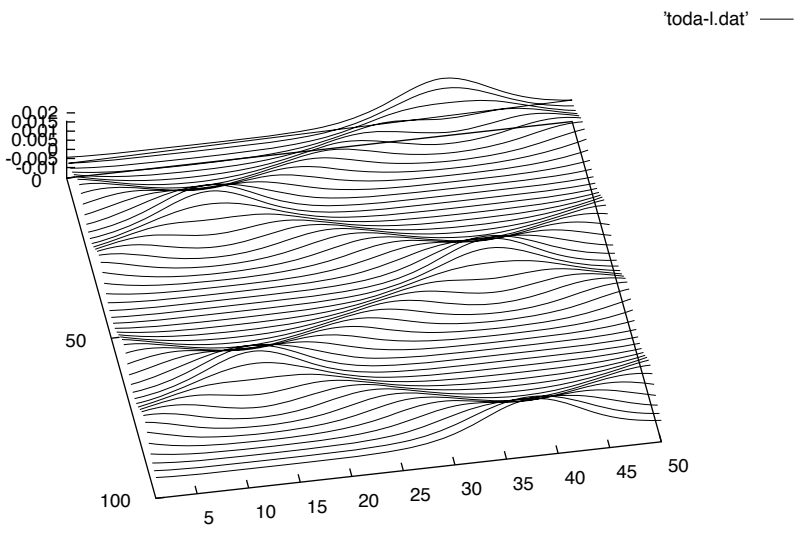

図 3: 戸田格子の時間発展

このプログラムを前節までの考え方に基づいて作成すると繁雑になって手がつかない。自由度の数が $100(= 50 \times 2)$ もあるのでいちいち外部関数を呼び込むのは難しいからだ。今までとは異なりここではプログラムに直接コメント行を挿入することで内容を説明しよう。但し本ページには Fortran 77 のプログラムのみを載せる。[1]

但し q(0:n+1),p(0:n+1),s(0:n+1),pi はそれぞれ $\psi_n, p_n = \dot{\psi}_n, q_n, \pi$ に対応している。(ちょっとややこしい対応関係になっているから注意して下さい)。

ここで示した様にサブルーチン rk4 だけでなく、更にそのなかにサブルーチンを用意しておくと多自由度 4 段階法に対処可能なプログラムを作成出来る。このようにしておけば非常に多くの問題をとりあえず解いてみる事が可能になり、やっていることがシミュレーションらしくなる。またこういう解法は偏微分方程式の陽解法に他ならないので有用性は高い。

[1] 本プログラムを実行させるのには相当の時間がかかります。

```
      parameter(n=50,mmm=50)
c       外部パラメータとして格子点の数 n と時間発展のサイクルの終了 mmm を
c     与える。

      implicit real*8 (a-h,o-z)
c      a-h,o-z で始まる文字列を倍精度の実数型であることを宣言。

      real*8 q(0:n+1),p(0:n+1),qd(0:n+1),pd(0:n+1)
      real*8 s(0:n+1)
      real*8 t,pi
      integer i,j,l,ntime,ncycle

      open(1,file='toda.dat')

c       toda.dat というファイルを開く。

      dt=5.d-5
c       時間刻み

      tend=1.d2
c         終了時刻

      ntime=int(tend/dt)/mmm
c         内側の時間ループ

      ncycle=mmm
c           外側の時間ループ

c       initial condition



       t=0.d0
      pi=4.d0*datan(1.d0)

      do j=0,n+1
       q(j)=dsin(2.d0*pi*dble(j)/dble(n))+2.d0
       p(j)=0.0d0
c      初期条件は q(i) に正弦波をいれ、その速度は 0 とする。
      enddo

       q(0)=q(n)
       q(n+1)=q(1)
       p(0)=p(n)
```

```
          p(n+1)=p(1)
c         周期境界条件

          do j=1,n
           s(j)=dexp(-2.d0*dlog(q(j))+dlog(q(j+1))+dlog(q(j-1)))-1.d0
c          s(j) は q_j であることに注意。
           write(1,*) t,dble(j),s(j)
         enddo



c        Main routine

         do l=1,ncycle
          write(*,*) l
c         外回りの時間発展を書かせます。外回りは 1,2,,,ncycle ステップまで。

          do i=1,ntime
c          内回りの時間発展を書かせます。内回りは 1,2,,,ntime ステップまで。

          t=t+dt
c           時間発展

            call rk4(q,p,qd,pd,dt,n)
c            Runge-Kutta のサブルーチンの読み込み。

         do j=0,n+1
            q(j)=qd(j)
            p(j)=pd(j)
c         ダミー配列に書き込んだ情報をメインの配列に移します。
         enddo
        enddo
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c        内回りの計算の終了
c        output of the data (データの書き出しは外回りで行います)
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

          do j=1,n
           s(j)=dexp(-2.d0*dlog(q(j))+dlog(q(j+1))+dlog(q(j-1)))-1.d0
           write(1,*) t,dble(j),s(j)
          enddo
           write(1,*)

         do j=0,n+1
```

```
          q(j)=qd(j)
          p(j)=pd(j)
        enddo

      enddo

c      Main routine is over. ほぼ計算の終了です。
 1    close(1)
c      File toda.dat を閉じます。

       end

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
       subroutine rk4(q,p,qm,pm,dt)
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c サブルーチン：引数がパラメータで配列も渡せます。
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

       implicit real*8 (a-h,o-z)
       real*8 q(0:n+1),p(0:n+1),qm(0:n+1),pm(0:n+1)
       real*8 qd(0:n+1),pd(0:n+1)
c       メインから渡した配列も定義して下さい。

       real*8 k1q(0:n+1),k1p(0:n+1),k2q(0:n+1),k2p(0:n+1)
       real*8 k3q(0:n+1),k3p(0:n+1),k4q(0:n+1),k4p(0:n+1)
c         k1p,k1q 等は変数 p,q 等に対するルンゲ・クッタの k1 等を表します。

       call onestep(q,p,k1q,k1p,dt,n)
c         1 ステップ進むサブルーチンです。配列 q,p を入れ、k1q,k1p がでて来
c ます。

       do j=0,n+1
         qd(j)=q(j)+k1q(j)/2.d0
         pd(j)=p(j)+k1p(j)/2.d0
c         k1 に対する引数を見て下さい。
       enddo

       call onestep(qd,pd,k2q,k2p,dt,n)
c         1 ステップ進むサブルーチンです。
       do j=0,n+1
         qd(j)=q(j)+k2q(j)/2.d0
         pd(j)=p(j)+k2p(j)/2.d0
c         k2 に対する引数を見て下さい。
```

```
        enddo

        call onestep(qd,pd,k3q,k3p,dt,n)
c          1 ステップ進むサブルーチンです。
       do j=0,n+1
          qd(j)=q(j)+k3q(j)
          pd(j)=p(j)+k3p(j)
c          k3 に対する引数を見て下さい。
        enddo

        call onestep(qd,pd,k4q,k4p,dt,n)
c          1 ステップ進むサブルーチンです。
        do j=0,n+1
         qm(j)=q(j)+(k1q(j)+2.d0*k2q(j)+2.0d0*k3q(j)+k4q(j))/6.d0
         pm(j)=p(j)+(k1p(j)+2.d0*k2p(j)+2.0d0*k3p(j)+k4p(j))/6.d0
c         Runge-Kutta の 4 段公式 (12) です。
        enddo

        end

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        subroutine onestep(q,p,qq,pp,dt,n)
        real*8 q(0:n+1),p(0:n+1),qq(0:n+1),pp(0:n+1)
        real*8 dt

c        変数 q,p=>qq,pp はその変位増分です。

        do j=1,n
         qq(j)=p(j)*dt
         pp(j)=dt*((p(j)*p(j)+q(j-1)*q(j+1))/q(j)-q(j))
c        qq は速度に時間刻みをかけたもの。pp は加速度に時間刻みをかけたもの。
        enddo
         qq(n+1)=qq(1)
         qq(0)=qq(n)
         pp(n+1)=pp(1)
         pp(0)=pp(n)
c        周期境界
        end
```

Fortran90 自由形式では

```
module com

integer,parameter::n=50,mmm=50
real(8),parameter::dt=5.d-5,tend=1.d0

end module


!!!main program!!!

program todakoushi

use com

real(8)::q(0:n+1),p(0:n+1),qd(0:n+1),pd(0:n+1),s(0:n+1),t,pi

integer::ntime,ncycle


open(1,file='toda.dat')

ntime=int(tend/dt)/mmm
ncycle=mmm

t=0.0_8
pi=4.0_8*datan(1.0_8)


do j=0,n+1

    q(j)=dsin(2.0_8*pi*dble(j)/dble(n))+2.0_8
    p(j)=0.0_8

enddo

q(0)=q(n)
q(n+1)=q(1)
p(0)=p(n)
p(n+1)=p(1)

do j=1,n
```

```
   s(j)=dexp(-2.0_8*dlog(q(j))+dlog(q(j+1)+dlog(q(j-1))))-1.0_8
   write(1,*)t,dble(j),s(j)

enddo


do l=1,ncycle



   do i=1,ntime

      t=t+dt
      call rk4(q,p,qd,pd)

      do j=0,n+1

         q(j)=qd(j)
         p(j)=pd(j)

      enddo

   enddo


   do j=1,n

      s(j)=dexp(-2.0_8*dlog(q(j))+dlog(q(j+1)+dlog(q(j-1))))-1.0_8
      write(1,*)t,dble(j),s(j)

   enddo

   write(1,*)


enddo


close(1)

end
```

```
!!!subroutine rk4!!!


subroutine rk4(q,p,qm,pm)

use com

real(8),intent(in)::q(0:n+1),p(0:n+1)
real(8),intent(out)::qm(0:n+1),pm(0:n+1)
real(8)::qd(0:n+1),pd(0:n+1)
real(8)::k1q(0:n+1),k1p(0:n+1),k2q(0:n+1),k2p(0:n+1)
real(8)::k3q(0:n+1),k3p(0:n+1),k4q(0:n+1),k4p(0:n+1)

call onestep(q,p,k1q,k1p)

do j=0,n+1

   qd(j)=q(j)+k1q(j)/2.0_8
   pd(j)=p(j)+k1p(j)/2.0_8

enddo

call onestep(qd,pd,k2q,k2p)

do j=0,n+1

   qd(j)=q(j)+k2q(j)/2.0_8
   pd(j)=p(j)+k2p(j)/2.0_8

enddo

call onestep(qd,pd,k3q,k3p)

do j=0,n+1

   qd(j)=q(j)+k3q(j)/2.0_8
   pd(j)=p(j)+k3p(j)/2.0_8

enddo

call onestep(qd,pd,k4q,k4p)

do j=0,n+1
```

```
   qm(j)=q(j)+(k1q(j)+2.0_8*k2q(j)+2.0_8*k3q(j)+k4q(j))/6.0_8
   pm(j)=p(j)+(k1p(j)+2.0_8*k2p(j)+2.0_8*k3p(j)+k4p(j))/6.0_8

enddo


end


!!!subroutine onestep!!!

subroutine onestep(q,p,qq,pp)

use com

real(8),intent(in)::q(0:n+1),p(0:n+1)
real(8),intent(out)::qq(0:n+1),pp(0:n+1)

do j=1,n

   qq(j)=p(j)*dt
   pp(j)=dt*((p(j)*p(j)+q(j-1)*q(j+1))/q(j)-q(j))

enddo

qq(n+1)=qq(1)
qq(0)=qq(n)
pp(n+1)=pp(1)
pp(0)=pp(n)

end
```

となる。同じ問題を C でプログラムを書くと

```
#include
#include

#define N 50
#define MMM 50

double q[N+2],p[N+2],qd[N+2],pd[N+2];
```

```
double s[N+2];
double t,pi;
int i,j,l,ntime,ncycle;

void rk4(double [],double [],double [],double [],double,int);
void onestep(double [],double [],double [],double [],double,int);

int main()
{
  double dt,tend;

  dt=5.0e-5;
  tend=100;

  ntime=(int)(tend/dt)/MMM;
  ncycle=MMM;

  /*** 初期条件 ***/
  t=0.0;
  pi=4.0*atan(1.0);

  for(j = 0;j <= N+1;j++){
    q[j] = sin(2.0*pi*(double)j/(double)N)+2.0;
    p[j] = 0.0;
  }

  /***  周期境界条件 ***/
  q[0] = q[N];
  q[N+1] = q[1];
  p[0] = p[N];
  p[N+1] = p[1];

  for(j = 1;j <= N;j++){
    s[j] = exp(-2.0*log(q[j])+log(q[j+1])+log(q[j-1]))-1.0;
    printf("%f %f %f\n",t,(double)j,s[j]);
  }
  printf("\n");

  for(l = 1;l <= ncycle;l++){

    printf("\n");

    for(i = 1;i <= ntime;i++){
```

```
      t += dt;
      rk4(q,p,qd,pd,dt,N);

      for(j = 0;j <= N+1;j++){
        q[j] = qd[j];
        p[j] = pd[j];
      }
    }

    for(j = 1;j <= N;j++){
      s[j] = exp(-2.0*log(q[j])+log(q[j+1])+log(q[j-1]))-1.0;
      printf("%f %f %f\n",t,(double)j,s[j]);
    }

    for(j = 0;j <= N+1;j++){
        q[j] = qd[j];
        p[j] = pd[j];
    }

  }

  return 0;
}

void rk4(double q[],double p[],double qm[],double pm[],double dt,int n)
{
  double k1q[N+2],k1p[N+2],k2q[N+2],k2p[N+2];
  double k3q[N+2],k3p[N+2],k4q[N+2],k4p[N+2];
  double qd[N+2],pd[N+2];


  onestep(q,p,k1q,k1p,dt,N);

  for(j = 0;j <= N+1;j++){
    qd[j] = q[j]+k1q[j]/2.0;
    pd[j] = p[j]+k1p[j]/2.0;
  }

   onestep(qd,pd,k2q,k2p,dt,N);

  for(j = 0;j <= N+1;j++){
    qd[j] = q[j]+k2q[j]/2.0;
    pd[j] = p[j]+k2p[j]/2.0;
```

```
  }

   onestep(qd,pd,k3q,k3p,dt,N);

  for(j = 0;j <= N+1;j++){
    qd[j] = q[j]+k3q[j];
    pd[j] = p[j]+k3p[j];
  }

   onestep(qd,pd,k4q,k4p,dt,N);

  for(j = 0;j <= N+1;j++){
    qm[j] = q[j]+(k1q[j]+2.0*k2q[j]+2.0*k3q[j]+k4q[j])/6.0;
    pm[j] = p[j]+(k1p[j]+2.0*k2p[j]+2.0*k3p[j]+k4p[j])/6.0;
  }
}

void onestep(double q[],double p[],double qq[],double pp[],double dt,int n)
{
  for(j = 1;j <= N;j++){
    qq[j] = p[j]*dt;
    pp[j] = dt*((p[j]*p[j]+q[j-1]*q[j+1])/q[j]-q[j]);
  }

  qq[N+1] = qq[1];
  qq[0] = qq[N];
  pp[N+1] = pp[1];
  pp[0] = pp[N];

}
```