岡本安晴　2001.1.29；2001.7

常微分方程式[1]

オイラー法

関数が微分方程式

$$\frac{dy}{dt} = f(t, y)$$

の形で与えられているとする。

微分を差分の形で近似すると

$$\frac{\Delta y}{\Delta t} \approx f(t, y)$$

となる。

$$y_{i+1} = y_i + \Delta y$$

とおくと、

$$y_{i+1} \approx y_i + \Delta t \cdot f(t_i, y_i) \qquad (1)$$

となる。

$t = t_0$ における値が初期値 y_0 として与えられているとき、（1）式によって関数値 y_1、y_2、・・・を求める方法はオイラー法と呼ばれている[1)]。

（1）式において、$t = t_i$、$y = y_i$、$h = \Delta t$、$t1 = t_{i+1} = t_i + \Delta t$、$y1 = y_{i+1}$ とおいたもの、すなわち t と y の値が与えられているとき、$h = \Delta t$ 後の値 t1 と y1 をオイラー法によって求める手続き Euler は次のようになる。

```
procedure Euler( h, t, y : Extended;
                 f       : TFDiff;
                 var t1, y1 : Extended );
  begin
      y1:=y+h*f(t,y);
      t1:=t+h;
  end;   {   Euler    }
```

オイラー法を手続き Euler として表したが、処理速度を優先するときは手続きとしてまとめず、プログラム中に直接組み込む。手続きとして表すと、手続き呼び出しの処理で余分な時間が掛かる。

---

[1] 本解説は、TRY！PC、2000 年５月号「Delphi による数値計算プログラミングのすすめ：第３回常微分方程式」の原稿をもとにしたものである。

手続き Euler を使った簡単な例でオイラー法の精度を見てみる。

微分方程式

$$\frac{dy}{dt} = -\sin t \qquad (2)$$

の解は

$$y = \cos t$$

で与えられる。

いま、

$$x = \sin t$$

とおくと、点$(x, y) = (\sin t, \cos t)$の軌跡は円になる。ｔの値$t_0, \cdots, t_i, \cdots$におけるｙの値を常微分方程式（２）に対してオイラー法を適用して求め、点$(\sin t_i, y_i)$と円を比べるとビジュアルにオイラー法の性能を見ることができる。

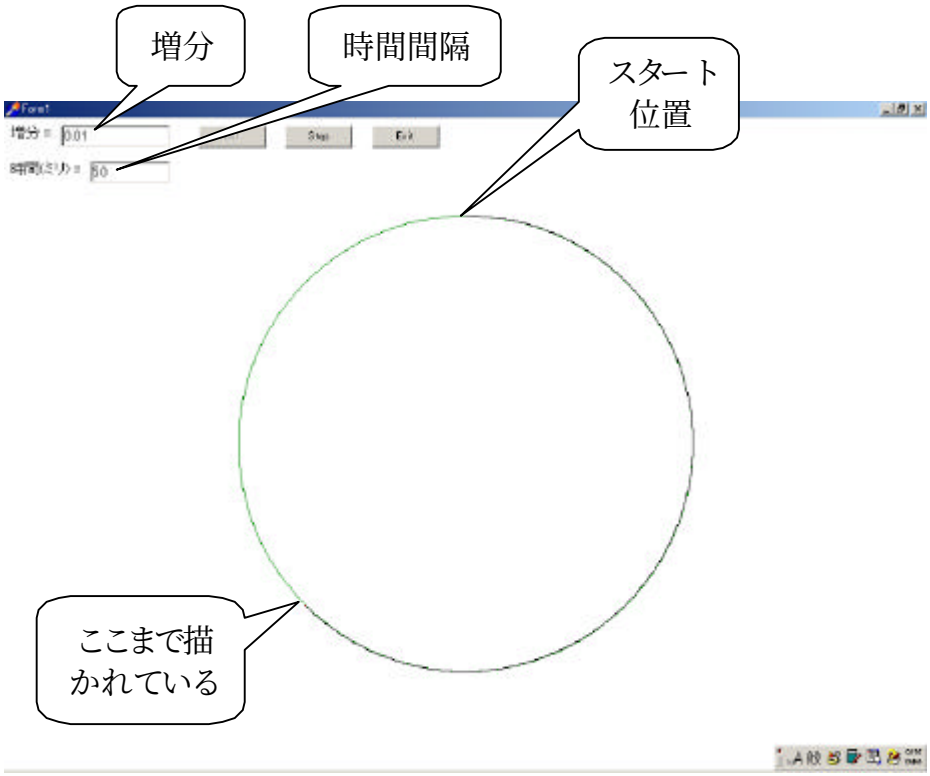

図１　オイラー法による解の描画

図１は PEuler.dpr というプログラムを実行したときのものである。このプログラムでは

```
t:=t1; y:=y1;
Euler( d, t, y, FDiff, t1, y1 );
ix1:=XPos(sin(t1)); iy1:=YPos(y1);
Pen.Color:=clRed;
LineTo(ix1,iy1);
```

という形で手続き Euler が呼出され、点$(\sin t_i, y_i)$の軌跡が点(ix1,iy1)を結ぶ線分として描かれている。黒の線が Euler 法による解の軌跡を表わし、最も新しい線分は赤で描かれている。円は緑の曲線で表わされている。

微分方程式（２）式における関数 $f(t,y)=-\sin t$ は、関数宣言

```
function FDiff( t, y : Extended ) : Extended;
    begin
        FDiff:=-sin( t );
    end;
```

によって関数 FDiff として与えられている。

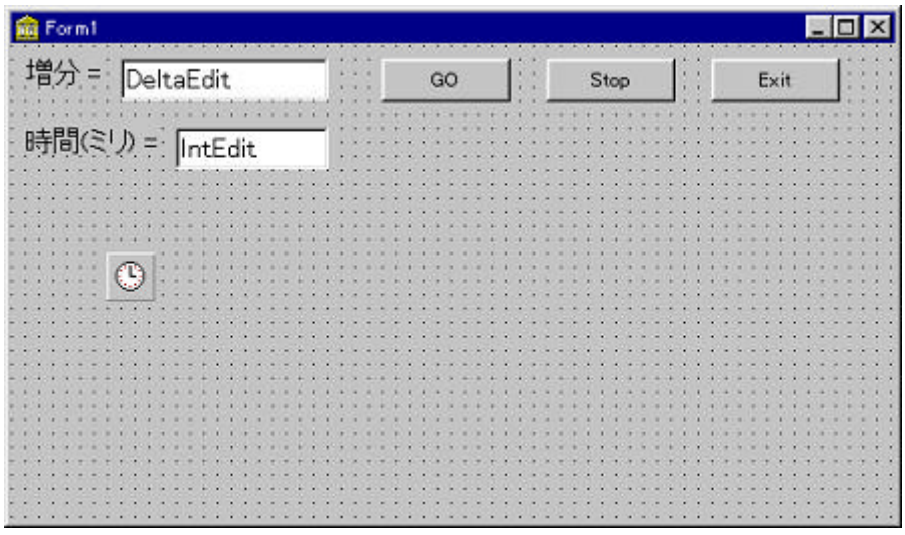

図２　UEiler.pas のフォーム

PEuler.dpr のユニット UEuler.pas のフォームは図２のように用意されていて、Timer1 コンポーネントによってオイラー法による解の計算の時間間隔をコントロールしている。GO ボタンをクリックすると、時間として設定された間隔で OnTimer イベントが生起し、オイラー法によって増分Δt後の関数値が求められている。増分は手続き Euler の第１パラメータhの実パラメータとして設定する。手続き Euler によって求められた最新の関数値

による線分、点(x_i, y_i)と点(x_{i+1}, y_{i+1})を結んだもの、は、赤い線分で示される。

プログラム PEuler.dpr は、difffiles フォルダにある。自己解凍型ファイル difffiles.EXE をダウンロードして実行すると解凍されてフォルダ difffiles が作成される。

手続き Euler はユニット UDiffEq.pas に宣言されている。このユニットはフォームを持たないユニットである。フォームなしユニットの作り方は拙著[2),3)]などに説明されている。

PEuler.dpr を増分の値 0.01 で実行すると図１のようになる。かなりよい結果といえそうである。

増分を 0.1 として実行すると図３のようになる。円からのずれが目立つ。

同じ増分に対してオイラー法より精度を上げる方法が工夫されている。

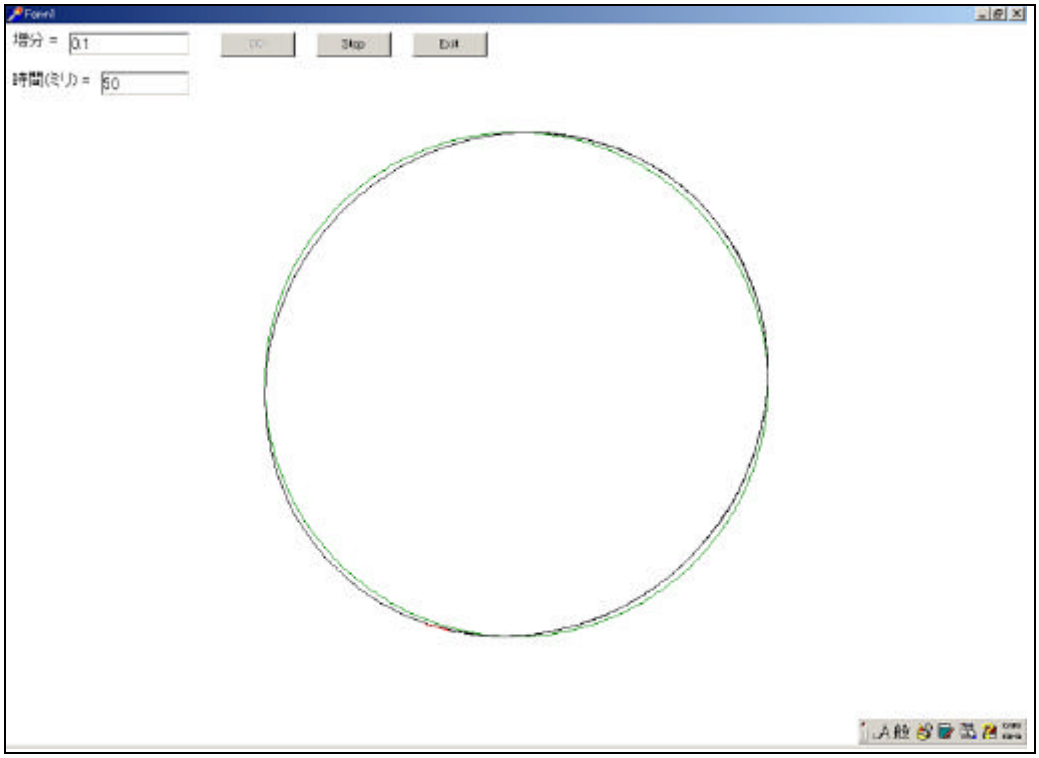

図３　増分が 0.1 のときの解

Midpoint method

オイラー法、（１）式、では、$t = t_i$における微分係数$f(t_i, y_i)$を用いて$t_{i+1} = t_i + \Delta t$における関数値y_{i+1}を求めている。オイラー法の精度を高めるために、t_iとt_{i+1}の中点での微分係数を用いるものがある。この方法は Midpoint method[4)] と呼ばれている。

この方法では、中点$t_i + \frac{1}{2}\Delta t$における微分係数を推定するために、まず中点での$y$の値を$y_i + \frac{1}{2}\Delta t \cdot f(t_i, y_i)$で推定する。この推定値を用いて中点における微分係数を

$$f\left(t_i+\frac{1}{2}\Delta t, y_i+\frac{1}{2}\Delta t\cdot f(t_i,y_i)\right)$$

と求め、$t_{i+1}=t_i+\Delta t$ における関数値 y_{i+1} を

$$y_{i+1}=y_i+\Delta t\cdot f\left(t_i+\frac{1}{2}\Delta t, y_i+\frac{1}{2}\Delta t\cdot f(t_i,y_i)\right)$$

とおく。

Midpoint method と同じ考え方の方法に「かえる跳び法」[5] と呼ばれているものがある。この方法は Feynman の物理学の教科書[6] でも説明されているが、関数値を求める関数が２グループに分かれていて、一方のグループの中点での微分係数の値が他グループの関数によって与えられているという形になっている。拙著「Delphi プログラミング入門」[7] でも２物体の運動のシミュレーションで「かえる跳び法」を用いている。

Midpoint method はルンゲ・クッタ（Runge－Kutta）法と呼ばれている方法の特別なものですが、Midpoint method より性能のよい４次のルンゲ・クッタ法について次に説明します。

ルンゲ・クッタ法（４次）

Midpoint method では、$\Delta t=t_{i+1}-t_i$ の間の増分を中点での微分係数によって計算することにより精度の向上が図られている。（４次の）ルンゲ・クッタ法では、３点、t_i、$t_i+\Delta t/2$、$t_{i+1}=t_i+\Delta t$、の微分係数を用いて増分を４種類の方法で求め、それら４つの増分、k_1、k_2、k_3、k_4、の加重平均として $\Delta y=y_{i+1}-y_i$ を求めている[8]。

この方法では、まずオイラー法で１つ求め、k_1 とする。

$$k_1=\Delta t\cdot f(t_i,y_i)$$

２番目の増分の推定値 k_2 は、中点での微分係数を用いて計算する。微分係数 $f(t,y)$ を与える関数における中点 $t_i+\Delta t/2$ での y の推定値を、上の Δy の推定値 k_1 を用いて $y_i+\Delta y/2=y_i+k_1/2$ とおき、

$$k_2=\Delta t\cdot f(t_i+\Delta t/2, y_i+k_1/2)$$

とする。

３番目の推定値 k_3 は、k_2 と同じく中点の微分係数を用いて計算するが、中点での y の推定値は k_2 を用いて $y_i+k_2/2$ とする。したがって、

$$k_3=\Delta t\cdot f(t_i+\Delta t/2, y_i+k_2/2)$$

となる。

最後の４番目の推定値 k_4 は、$t_{i+1}=t_i+\Delta t$ での微分係数を用いて求める。微分係数 $f(t,y)$ の t_{i+1} に対する y の値は k_3 を用いて y_i+k_3 とおき、

$$k_4=\Delta t\cdot f(t_i+\Delta t, y_i+k_3)$$

とする。

最終的な y の増分 $\Delta y=y_{i+1}-y_i$ は、以上の 4 つの増分、k_1、k_2、k_3、k_4、を１：２：

２：１の比で加重平均したものとする。すなわち、

$$\Delta y = (k_1 + 2k_2 + 2k_3 + k_4)/6$$

とおく。

オイラー法の精度は$O(\Delta t)$と推定されているが、４次のルンゲ・クッタ法の精度はyが５回連続微分可能であるならば$O((\Delta t)^4)$となる[4)]。

ルンゲ・クッタ法の手順をまとめると次のようになる。

（１）$k_1 = \Delta t \cdot f(t_i, y_i)$とおく。
（２）$k_2 = \Delta t \cdot f(t_i + \Delta t/2, y_i + k_1/2)$とおく。
（３）$k_3 = \Delta t \cdot f(t_i + \Delta t/2, y_i + k_2/2)$とおく。
（４）$k_4 = \Delta t \cdot f(t_i + \Delta t, y_i + k_3)$とおく。
（５）次式によりy_{i+1}とt_{i+1}を与える。

$$y_{i+1} = y_i + (k_1 + 2k_2 + 2k_3 + k_4)/6$$
$$t_{i+1} = t_i + \Delta t$$

上の手順を、オイラー法の場合と同じように、手続き Runge_Kutta として表わすと次のようになる。

```
procedure Runge_Kutta( h, t, y : Extended;
                       f : TFDiff;
                       var t1, y1 : Extended );
  var  k1, k2, k3, k4 : Extended;
  begin
        k1:=h*f(t,y);
        k2:=h*f(t+0.5*h, y+0.5*k1);
        k3:=h*f(t+0.5*h, y+0.5*k2);
        k4:=h*f(t+h, y+k3);

        y1:=y+(k1+2*k2+2*k3+k4)/6;
        t1:=t+h;
  end;   {    Runge_Kutta     }
```

プログラム PRK.dpr は、オイラー法を用いたプログラム PEuler.dpr をルンゲ・クッタ法を用いたものに書き改めたものである。手続き Euler が Runge_Kutta に置き換えられている。オイラー法による増分が 0.1 のときの解（図３）と比べると、同じ増分であるがよい結果が得られている（図４）。

増分を１として実行したときも良い結果が得られている（図５）。図５において、緑色の曲線で円があらわされ、ルンゲ・クッタ法による解の軌跡は黒の線分の端点で表わされている。

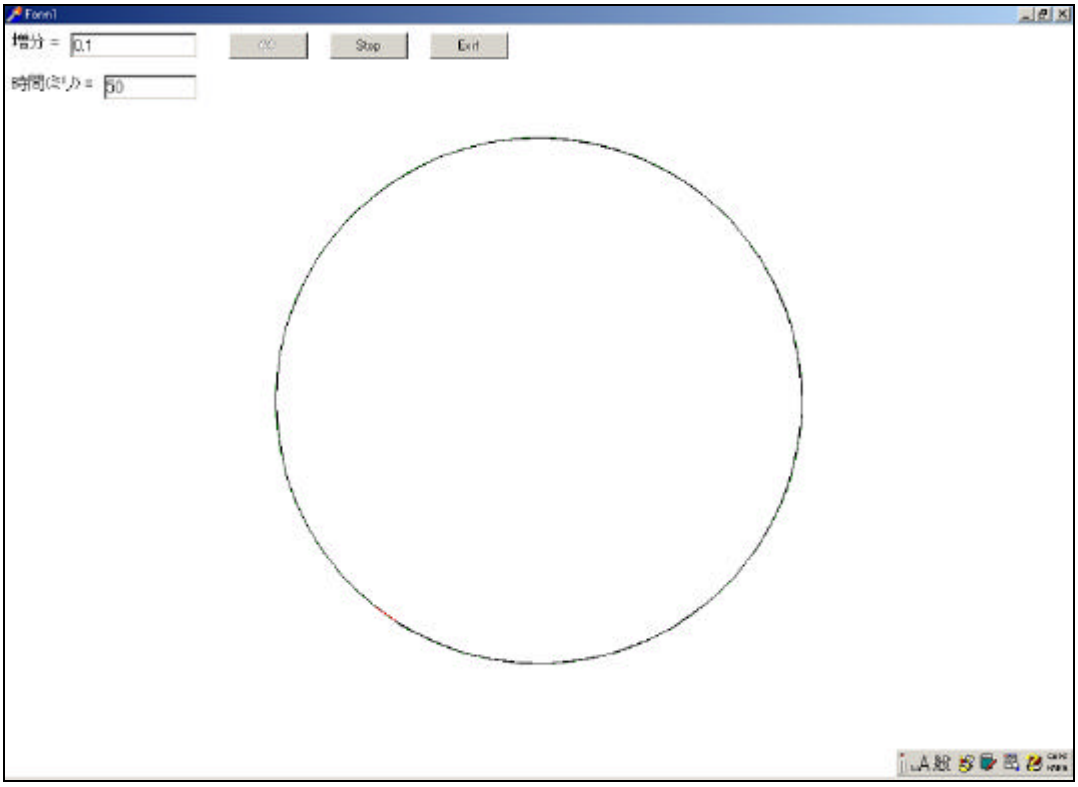

図４　ルンゲ・クッタ法を用いた場合

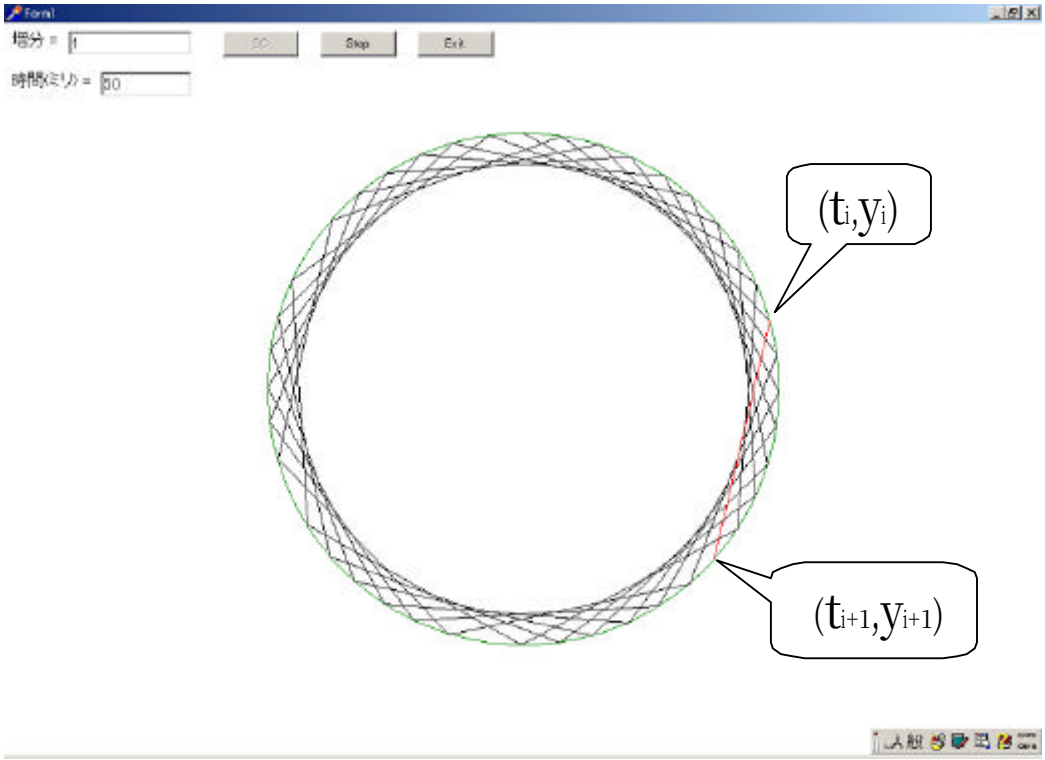

図５　増分を１としたときのルンゲ・クッタ法の解

OnTimer イベントの生起による最新の解の軌跡は赤い線分で表わされている。増分 Δt の値が大きいので軌跡を描いている線分の長さが長くなっているが、解に対応している線分の端点は円周上にある。
ルンゲ･クッタ法の使用例のプロジェクト PRK.dpr もフォルダ difffiles にある。

連立常微分方程式

微分方程式が次の連立方程式

$$\frac{dy_1}{dt} = f_1(t, y_1, \cdots, y_n)$$
$$\vdots$$
$$\frac{dy_n}{dt} = f_n(t, y_1, \cdots, y_n)$$

で与えられているときのルンゲ・クッタ法は次のようになる[4)]。

まず、オイラー法で増分を求める。y_j に対するオイラー法による増分 $K_{1,j}$ は次式で与えられる。

$$K_{1,j} = \Delta t \cdot f_j(t_i, y_{1,i}, \cdots, y_{n,i})$$

次に、$K_{1,j}$ を用いて、各 y_j の t_i と $t_{i+1} = t_i + \Delta t$ の間の中点における値を $y_{j,i} + K_{1,j}/2$ と推定して、中点における微分係数を

$$f_j(t_i + \Delta t/2, y_{1,i} + K_{1,1}/2, \cdots, y_{n,i} + K_{1,n}/2)$$

で推定する。この中点における微分係数の推定値を用いて y_j 増分を推定したものを $K_{2,j}$ とおく。

$$K_{2,j} = \Delta t \cdot f_j(t_i + \Delta t/2, y_{1,i} + K_{1,1}/2, \cdots, y_{n,i} + K_{1,n}/2)$$

となる。

上の中点における微分係数を用いて増分を求める方法を、$K_{1,j}$ の代わりに $K_{2,j}$ を用いて行ったときの増分の推定値を $K_{3,j}$ とおく。

$$K_{3,j} = \Delta t \cdot f_j(t_i + \Delta t/2, y_{1,i} + K_{2,1}/2, \cdots, y_{n,i} + K_{2,n}/2)$$

となる。

増分の４番目の推定値$K_{4,j}$は、における微分係数の推定値を用いて算出する。微分係数を与える関数$f_k(t, y_1, \cdots, y_n)$における各y_jの$t_i + \Delta t$における値を$y_{j,i} + K_{3,j}$で与えて、４番目の増分の推定値$K_{4,j}$を

$$K_{4,j} = \Delta t \cdot f_j(t_i + \Delta t, y_{1,i} + K_{3,1}, \cdots, y_{n,i} + K_{3,n})$$

とする。

これら４つの推定値、$K_{1,j}$、$K_{2,j}$、$K_{3,j}$、$K_{4,j}$、を１：２：２：１の重みで加重平均したものでy_jの増分$\Delta y_j = y_{j,i+1} - y_{j,i}$を与える。すなわち、

$$y_{j,i+1} = y_{j,i} + (K_{1,j} + 2K_{2,j} + 2K_{3,j} + K_{4,j})/6$$

とする。

上の手順をまとめると以下のようになる。

（１）$K_{1,j} = \Delta t \cdot f_j(t_i, y_{1,i}, \cdots, y_{n,i})$; $j = 1, \cdots, n$、とおく。

（２）$K_{2,j} = \Delta t \cdot f_j(t_i + \Delta t/2, y_{1,i} + K_{1,1}/2, \cdots, y_{n,i} + K_{1,n}/2)$; $j = 1, \cdots, n$、とおく。

（３）$K_{3,j} = \Delta t \cdot f_j(t_i + \Delta t/2, y_{1,i} + K_{2,1}/2, \cdots, y_{n,i} + K_{2,n}/2)$; $j = 1, \cdots, n$、とおく。

（４）$K_{4,j} = \Delta t \cdot f_j(t_i + \Delta t, y_{1,i} + K_{3,1}, \cdots, y_{n,i} + K_{3,n})$; $j = 1, \cdots, n$、とおく。

（５）$K_{1,j}$、・・・、$K_{4,j}$の値から$y_{1,i+1}$、・・・、$y_{n,i+1}$を次式により与える。

$$y_{j,i+1} = y_{j,i} + (K_{1,j} + 2K_{2,j} + 2K_{3,j} + K_{4,j})/6$$

上の連立常微分方程式に対するルンゲ・クッタ法の手順を RungeKuttaSystem 手続きとしてまとめた。この手続きはユニット USystemRK.pas に宣言されている（リスト１）。

リスト１　連立常微分方程式に対するルンゲ・クッタ法

```
unit USystemRK;  //  連立常微分方程式に対するルンゲ・クッタ法
                 //      R.L.Burden & J.D.Faires (1985)
                 //  Numerical Analysis, 3rd ed., Algorithm 5.7
interface

uses UTypeDefRK;


procedure RungeKuttaSystem( t, h : Extended;
                            y    : TVectorRK;
                            f    : TFuncRK;
                            m    : integer;
                            var t1 : Extended;
                            var y1 : TVectorRK );


implementation


procedure RungeKuttaSystem( t, h : Extended;
                            y    : TVectorRK;
                            f    : TFuncRK;
                            m    : integer;
                            var t1 : Extended;
                            var y1 : TVectorRK );
  var  y0, v, k1, k2, k3, k4 : TVectorRK;
       i : integer;
  begin
        v:=f(t, y);
        for i:=1 to m do
          k1[i]:=h*v[i];

        for i:=1 to m do
          y0[i]:=y[i]+k1[i]*0.5;
        v:=f(t+h*0.5, y0);
        for i:=1 to m do
          k2[i]:=h*v[i];

        for i:=1 to m do
          y0[i]:=y[i]+k2[i]*0.5;
        v:=f(t+h*0.5, y0);
        for i:=1 to m do
          k3[i]:=h*v[i];
```

```
        for i:=1 to m do
          y0[i]:=y[i]+k3[i];
        v:=f(t+h, y0);
        for i:=1 to m do
          k4[i]:=h*v[i];

        for i:=1 to m do
          y1[i]:=y[i]+(k1[i]+2*k2[i]+2*k3[i]+k4[i])/6;

        t1:=t+h;
  end;    {    RungeKuttaSystem    }

end.
```

手続き RungeKuttaSystem の頭部は次のようになっている。

```
procedure RungeKuttaSystem( t, h : Extended;
                            y    : TVectorRK;
                            f    : TFuncRK;
                            m    : integer;
                            var t1 : Extended;
                            var y1 : TVectorRK );
```

手続き RungeKuttaSystem を呼び出すときは、第１, 第２パラメータ t と h に t_i と Δt を設定する。第３パラメータの配列 y の要素には $y_{1,i}$、・・・、 $y_{n,i}$ の値を設定する。第４パラメータの関数 f は連立微分方程式の関数 $f_j(t, y_1, \cdots, y_n)$ を指定するもので、関数値の型は TVectorRK という配列型である。配列の j 番目の要素が、j 番目の関数 $f_j(t, y_1, \cdots, y_n)$ の値を表わす。

これらの配列や関数をパラメータとするための型はユニット UTypeDefRK で宣言しておく（リスト２）。ユニット USystemRK の uses 節で UTypeDefRK の使用が宣言されている。

リスト２　ユニット USystemRK.pas および UDiffEqSysObj.pas で用いられている型の宣言のためのユニット

```
unit UTypeDefRK;

interface


const MaxNFunc  = 10;

type  TVectorRK = array[1..MaxNFunc] of Extended;
      TFuncRK   = function( t : Extended;
                           y : TVectorRK ) : TVectorRK;

implementation

end.
```

ユニット UTypeDefRK において、型 TFuncRK は

```
TFuncRK   = function( t : Extended;
                      y : TVectorRK ) : TVectorRK;
```

と宣言されている。これは $f_j(t, y_1, \cdots, y_n);\ \mathrm{j}=1, \cdots, n$ に対応した形になっている。

手続き RungeKuttaSystem の第５パラメータｍには y_1、・・・、y_n の総数ｎを設定する。ｎは配列 TVectorRK の大きさ MaxNFunc 以下の値でなければならない。

以上の設定で手続き RungeKuttaSystem を呼出すと、ルンゲ・クッタ法が実行されて t_{i+1} の値が第６パラメータ t1 に、$y_{1,i+1}$、・・・、$y_{n,i+1}$ の値が第７パラメータである配列 y1 に返される。

手続き RungeKuttaSystem を使用したプログラム例がプロジェクト PRKSystem.dpr である。

このプログラムでは、

$$y_1 = \sin t$$
$$y_2 = \cos t$$

に対する連立常微分方程式

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = -y_1$$

の解を求めています。点$(y_1, y_2) = (\sin t, \cos t)$の軌跡は円を描くので、この円とルンゲ・クッタ法によって求めた解$(y_{1,i}, y_{2,i})$の軌跡を比較することにより解の精度を見ることができる。

手続き RungeKuttaSystem の第４パラメータである連立常微分方程式の関数、

$$f_1(t, y_1, y_2) = y_2$$
$$f_2(t, y_1, y_2) = -y_1$$

は関数 FDiff として次のように宣言されている。

```
function FDiff( t : Extended;
                y : TVectorRK ) : TVectorRK;
begin
        Result[1]:= y[2];
        Result[2]:=-y[1];
end;
```

PRKSystem.dpr を実行すると図６のような画面になる。

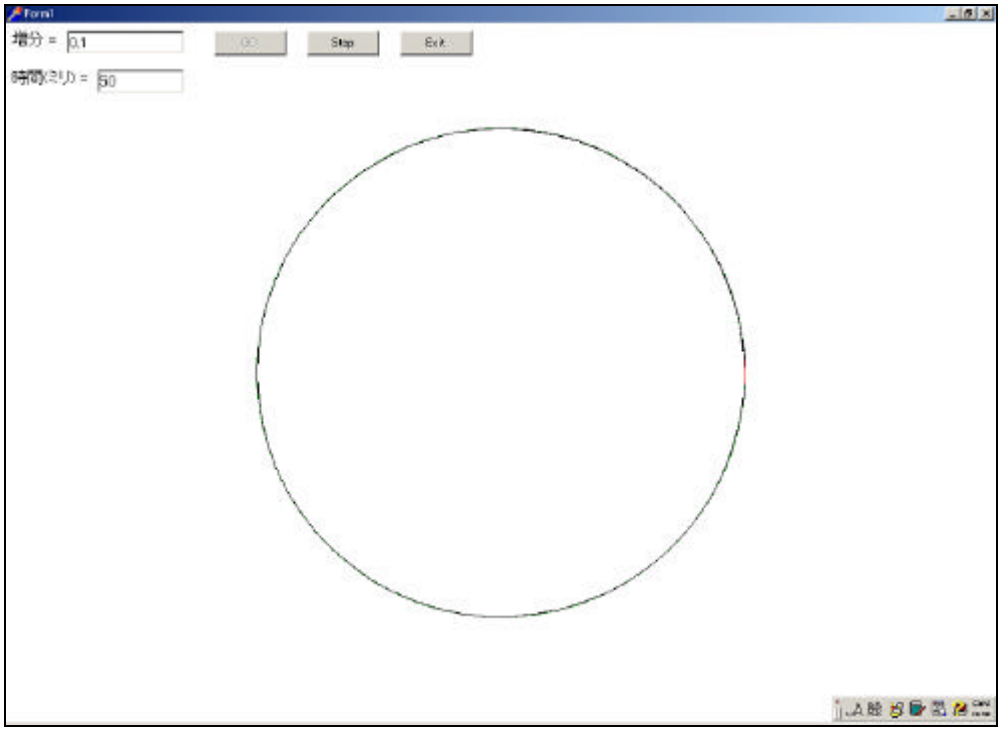

図６　連立常微分方程式の解をルンゲ・クッタ法で求めた場合

プログラム PRKSystem.dpr もフォルダ difffiles にある。

ユニットファイル UTypeDefRK.pas において配列の型 TVectorRK と手続き型 TFuncRK が宣言されている。これらの型は、USystemRK.pas では手続き RungeKuttaSystem のパラメータの宣言において、URKSystem.pas では連立常微分方程式の関数 $f_j(t, y_1, \cdots, y_n)$ を与える関数 FDiff の宣言において用いられているので、それぞれの uses 節で使用を宣言しておく。これらのユニットの uses 節による関係を図示すると図７のようになる。図中の矢印は、元の方のユニットの uses において先の方のユニットの使用が宣言されていることを表わす。

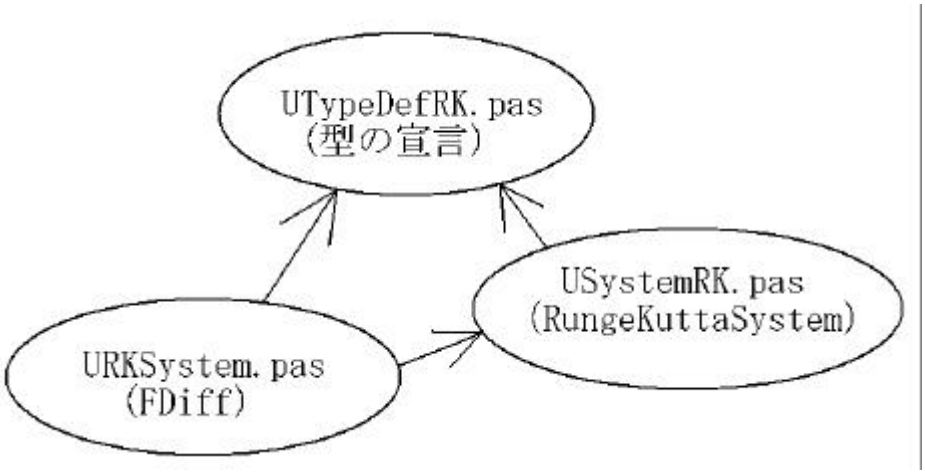

図７ ユニットの関係

オブジェクト（クラス型）化

微分方程式とその解法を１つのまとまりとして考えると、オブジェクトとして表わすことが考えられる。「オブジェクト」とは、Object Pascal（Delphi）ではクラス型として生成されたもの（インスタンス）をいう[9]、[10]。しかし、C++では、構造型として用意されたもの[11]、あるいは、あらゆるデータ型に対してそのメモリー上にとられたもの[12]をオブジェクトといっている。C++では、オブジェクトがクラス型以外のものを表わしていることがあるので注意が必要である。

ルンゲ・クッタ法のクラス型を次のように用意した。

```
type TRKObj =
        class
            h, t, y : Extended;
            f       : TFDiff;
```

```
        //  初期化を伴うオブジェクトの生成
        constructor CreateRK( h0, t0, y0 : Extended;
                              f0         : TFDiff   );
        //  初期化用メソッド
        procedure   InitRK( h0, t0, y0 : Extended;
                            f0         : TFDiff   );
        //  1ステップ分の計算
        procedure   CalcRK( var t1, y1 : Extended );
    end;
```

各コンポーネントおよびパラメータの意味は、手続き Runge_Kutta のものに対応している。このクラス型によるルンゲ･クッタ法は、ユニット UDiffEqObj.pas に用意した。プログラム PRK.dpr をこのクラス型を用いたものに変更したものが PRKObj.dpr である。このプログラムもフォルダ difffiles に含まれている。

常連立微分方程式に対するルンゲ・クッタ法のクラス型は次のように宣言されている。

```
type TRKSystemObj =
        class
            h, t : Extended;
            y    : TVectorRK;
            f    : TFuncRK;
            m    : Longint;

            //  初期化を伴うオブジェクトの生成
            constructor CreateRKSys( h0, t0 : Extended;
                                     y0     : TVectorRK;
                                     f0     : TFuncRK;
                                     m0     : Longint  );
            //  初期化用メソッド
            procedure   InitRKSys(  h0, t0 : Extended;
                                    y0     : TVectorRK;
                                    f0     : TFuncRK;
                                    m0     : Longint  );
            //  1ステップ分の計算
            procedure   CalcRKSys(  var t1 : Extended;
                                    var y1 : TVectorRK );
        end;
```

コンポーネントおよびメソッドのパラメータの意味は、手続き RungeKuttaSystem におけるものに対応している。

このクラス型 TRKSysObj は、ユニット UDiffEqSysObj.pas に用意されている。TVectorRK などの UDiffEqSysObj.pas で用いられている型は、ユニット UTypeDefRK.pas（リスト２）に宣言されている。この UTypeDefRK.pas は、ユニット UDiffEqSysObj.pas、およびクラス

型 TRKSystemObj を使用するユニットの両ユニットの uses 節において、その使用を宣言する必要がある。

クラス型 TRKSysObj を用いてプロジェクト PRKSystem.dpr を書き改めたものがプロジェクト PRKSystemObj.dpr として用意されている。

独立な２つのオブジェクト

オブジェクトを用いると複数のお互いに独立な微分方程式が簡単に扱える。プログラム PRK2Obj.dpr は PRKObj.dpr を２つのオブジェクトを用いたものに改めたものである。単位円の軌跡 $(\sin t, \cos t)$ を表わす２つの関数

$$x = \sin t\ 、\quad y = \cos t$$

を２つの微分方程式

$$\frac{dx}{dt} = \cos t\ 、\quad \frac{dy}{dt} = -\sin t$$

の解として与えている。すなわち、２つの関数宣言

```
function FDiffC( t, y : Extended ) : Extended;
  begin
        FDiffC:=-sin( t );
  end;
```

および

```
function FDiffS( t, y : Extended ) : Extended;
  begin
        FDiffS:=cos( t );
  end;
```

と２つのオブジェクト

```
RKC:=TRKObj.Create;        RKS:=TRKObj.Create;
```

を用意して、初期化を

```
RKS.InitRK( d, t, vs, FDiffS );
RKC.InitRK( d, t, vc, FDiffC );
```

というように行い、$x = \sin t$、$y = \cos t$ の値を求めるオブジェクトを設定している。

２つのオブジェクトを用いたプログラム PRK2Obj.dpr もフォルダ difffiles に含まれている。

簡単な使用例

微分方程式の解を単に求めるだけのときは、

```
for i:=1 to n do
   Runge_Kutta( dt, t[i-1], RKC[i-1], DCos, t[i], RKC[i] );
```

というようにして簡単に配列に解の値を設定していくことができます。

プログラム PSimple.dpr は、上の様にして解の値を配列に求める例である。このプログラムの場合は、使用例ということでオイラー法からクラス型による連立常微分方程式の解法までが同時に使用されている。コメントを詳しく付けたので参照されたい。このプログラムもフォルダ difffiles にある。実行開始時には計算結果を書き出すためのテキストファイル名の設定を求めるダイアログボックスが表示される。ファイル名の設定後、フォームのＯＫボタンをクリックすると計算が始まる。計算結果はテキストファイルに書き出されているので、プログラムの実行終了後、エディタで開いて見ることができる。

参 考 文 献

（1）川上一郎 「数値計算」Pp.201、岩波書店、1989

（2）岡本安晴 「Delphi で学ぶデータ分析法」Pp.274、CQ 出版株式会社、1998.。

（3）岡本安晴 「Delphi でエンジョイプログラミング」Pp.158、CQ 出版株式会社、1999.

（4）R.L.Burden & J.D.Faires. *Numerical Analysis, 3rd ed.* Pp.676, Prindle, Weber & Schmidt, 1985.

（5）吉澤純夫 「力学シミュレーション入門」Pp.161、CQ 出版株式会社、1998.

（6）R.P.Feynman, R.B.Leighton & M.Sands. *Lectures on Physics,I.* Addison - Wesley Publishing company, 1963.

（7）岡本安晴「Delphi プログラミング入門」Pp.207、CQ 出版株式会社、1997.

（8）長野三郎・長島忍・吉村伸「Pascal プログラミング、第２版」、Pp.170、東京大学出版会、1992.

（9）インプライズ株式会社（編訳）「Borland Delphi ５ Object Pascal 言語ガイド」、1999.

（１０） M. Cantù , Mastering Delphi 5. Pp.1085, SYBEX, 1999.

（１１）S.R.ディビス（著）・瀬谷啓介（訳）「改訂Ｃ++のからくり」、Pp.439、ソフトバンクパブリッシング株式会社、1999.

（１２）柴田望洋「プログラミング講義 C++」、Pp.505、ソフトバンクパブリッシング株式会社、1996.