# 情報工学実験１
# C++/Octaveによる微分方程式の数値解法

学籍番号：035764C 若津 大悟

グループH
実験実施日：平成１６年6月29日
提出〆切り日：平成１６年7月6日

# 1　実験目的

微分方程式の解をコンピュータを用いて数値的に解く方法について学習する。今回は、最も直感的で分かりやすいオイラー法のアルゴリズムを、C＋＋言語を用いてプログラミングする事を目的とする。

# 2　プログラムのソースコード

オイラー法を用いて野球ボールの軌道を計算するプログラム。

```
#include "NumMeth.h"

int main() {

  /*ボールの初期位置及び初期速度を設定*/
  double y1, speed, theta;
  double r1[2+1], v1[2+1], r[2+1], v[2+1], accel[2+1];
  cout << "First height(m)  : "; cin >> y1;
  r1[1] = 0; r1[2] = y1;          /*初期位置ベクトル*/
  cout << "First speed(m/s) : "; cin >> speed;
  cout << "First degree     : "; cin >> theta;
  const double pi = 3.141592654;
  v1[1] = speed*cos(theta*pi/180);/*初期速度(x 軸)*/
  v1[2] = speed*sin(theta*pi/180);/*初期速度(y 軸)*/
  r[1] = r1[1]; r[2] = r1[2];
  v[1] = v1[1]; v[2] = v1[2];

  /*物理パラメータを設定*/
  double Cd = 0.35;       /*空気抵抗(無次元)*/
  double area = 4.3e-3;   /*投射物の横断面積(m^2)*/
  double grav = 9.81;     /*重力加速度(m/s^2)*/
  double mass = 0.145;    /*投射物の質量(kg)*/
  double airFlag, rho;
  cout << "Air resistance (Yes:1, No:0) : "; cin >> airFlag;
  if( airFlag == 0 )
    rho = 0;              /*空気抵抗なし*/
  else
    rho = 1.2;            /*空気の密度(kg/m^3)*/
  double air_const = -0.5*Cd*rho*area/mass;/*空気抵抗定数*/

```

```
31  /*ボールが地面に着くまで、あるいは最大の刻み数になるまで*/
32  double tau;                                  /*    ループ*/
33  cout << "tau(second)      : "; cin >> tau;
34  int iStep, maxStep = 1000;     /*最大の刻み数*/
35  double *xplot, *yplot, *xNoAir, *yNoAir;
36  xplot = new double [maxStep + 1];
37  yplot = new double [maxStep + 1];
38  xNoAir = new double [maxStep + 1];
39  yNoAir = new double [maxStep + 1];
40
41  double t, y_m;
42  for( iStep=1; iStep<=maxStep; iStep++ ) {
43
44    /*プロット用に位置を記録*/
45    xplot[iStep] = r[1];
46    yplot[iStep] = r[2];
47    t = ( iStep-1 )*tau;        /*現在時刻*/
48    xNoAir[iStep] = r1[1] + v1[1]*t;                /*位置(x)*/
49    yNoAir[iStep] = r1[2] + v1[2]*t - (grav*t*t)/2;/*位置(y)*/
50
51    /*ボールの加速度を計算*/
52    double normV = sqrt( v[1]*v[1] + v[2]*v[2] );/*ルート*/
53    accel[1] = air_const*normV*v[1];              /*の計算*/
54    accel[2] = air_const*normV*v[2];
55    accel[2] -= grav;
56
57    /*オイラー法を用いて、新しい位置および速度を*/
58    r[1]=r[1]+v[1]*tau;            /*計算するアルゴリズム*/
59    r[2]=r[2]+v[2]*tau;
60    v[1]=v[1]+accel[1]*tau;
61    v[2]=v[2]+accel[2]*tau;
62
63    /*ボールが地面に着いたら(y<0)ループを抜ける。if 文を使う*/
64    if( r[2] < 0 )
65      break;
66
67    /*最高到達高さを求めるアルゴリズム*/
68     if( r[2] < r[2]+v[2]*tau )
69        y_m = r[2];
```

```
70    }
71
72    /*最大到達距離*/
73    cout << "Max attainment distance is => " << r[1] << " (m)" << endl;
74    /*滞空時間の表示*/
75    cout << "Duration of flight is => " << t << " (second)" << endl;
76    /*最大到達高さ*/
77    cout << "Max attainment height is => " << y_m << "(m)" << endl;
78
79    ofstream xplotOut("xplot.txt"), yplotOut("yplot.txt"),
80      xNoAirOut("xNoAir.txt"), yNoAirOut("yNoAir.txt"),
81      xyplotOut("xyplot.txt"),xyNoAirOut("xyNoAir.txt");
82
83
84    int i;
85    for (i=1; i<=iStep+1; i++ ) {
86      xyplotOut << xplot[i] << " " <<yplot[i] << endl;
87      xplotOut << xplot[i] << endl;
88      yplotOut << yplot[i] << endl;
89    }
90    for( i=1; i<=iStep+1; i++ ) {
91      xyNoAirOut << xNoAir[i] << " " << yNoAir[i] << endl;
92      xNoAirOut << xNoAir[i] << endl;
93      yNoAirOut << yNoAir[i] << endl;
94   }
95
96    delete [] xplot, yplot, xNoAir, yNoAir;/*メモリ開放*/
97    return(0);
98  }
```

# 3　問題の回答

実験テキスト中にある問題に回答せよ。

## 3.1　問題：1

実際にオイラー法を適用しようとすると、幾つかの問題のために精度が悪く、使われることはほとんどない。オイラー法の問題点を説明せよ。

オイラー法の公式は、

$$y_{n+1} = y_n + hf\left(x_n, y_n\right)$$

であり、これは $x_n$ から $x_{n+1} = x_n + h$ へと解を進展させるものである。つまり、現在の時間の微分値と、時間の積を前回の計算結果に加算する。という方法である。(公式は非対称である。) 図 1 に示すように、解を区間 h にわたって進展させるが、区間の出発点だけの微分値を用いてグラフを描いているのがわかる。

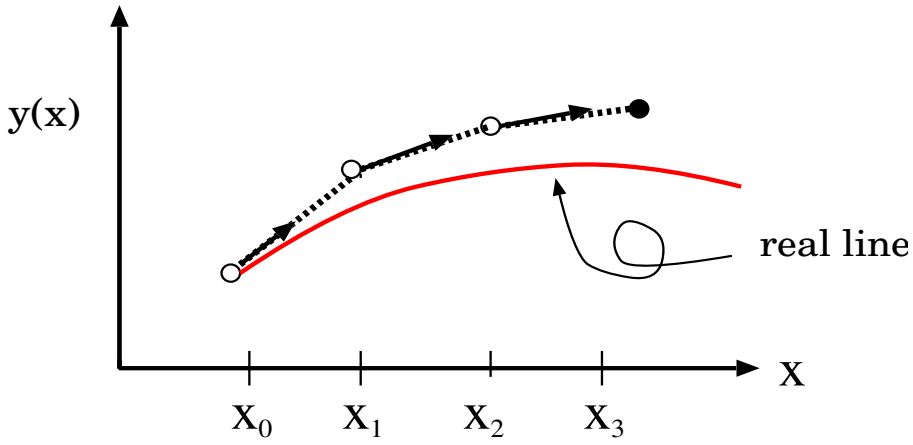


図 1: オイラー法のイメージ図

つまり、図でもわかるように、オイラー法では、出発点の導関数のみ使用しているために、精度が悪くなる、という欠点がある。

## 3.2 問題：2

微分方程式 (11) の解を導出して確かめよ。

$$\frac{d}{dt}x = -x \quad , \quad x\left(0\right) = 1$$

両辺を積分すると、

$$\begin{aligned}
\int \frac{1}{x}dx &= -\int dt \\
ln\ x + c_1 &= -t + c_2 \\
ln\ x &= -t + C \\
\text{初期値条件より} \quad ln\ 1 &= -0 + C \quad \Rightarrow \quad C = 0 \text{ となる。} \\
ln\ x &= -t \\
\therefore \quad x &= e^{-t}
\end{aligned}$$

### 3.3 問題：3

ボールの軌道のグラフを gnuplot で出力せよ。なお、時間刻み $\tau$ は $0 < t \leq 2$ の間で 5 個選ぶこと。補助的に Octave を用いてよもい。

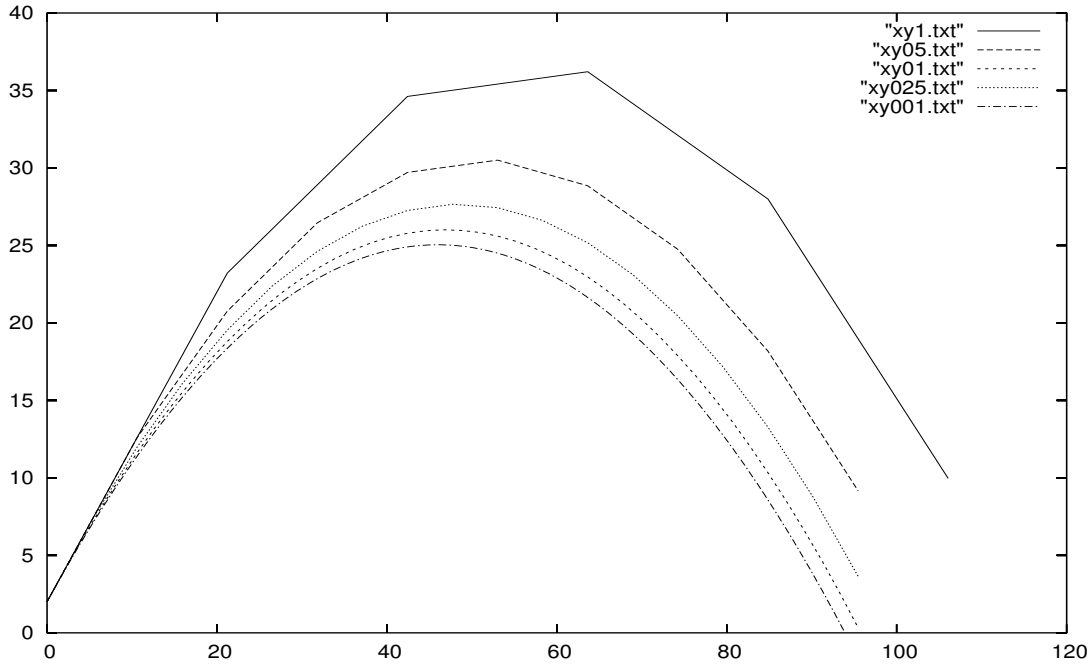


図 2: ボールの軌道グラフ

高さの初期値 $= 2$(m), 初期速度 $= 30$(m/s), 初期角度 $= 45$(度), 空気抵抗なし,$\tau = 1$ , 0.5 , 0.25 , 0.1 , 0.01 で設定

(例：tau=0.1 にした時)

```
% a.out
First height(m)  : 2
First speed(m/s) : 30
First degree     : 45
Air resistance (Yes:1, No:0) : 0
tau(second)      : 0.1
Max attainment distance is => 97.5807 (m)
Duration of flight is => 4.5 (second)
Max attainment height is => 25.9467(m)
```

## 3.4　問題：4

時間刻み$\tau$の設定によって、計算結果が大きく異なることが確認できるが、その理由を考察せよ。

◯ 図3でもわかるように、$\tau$の値を大きくすればするほど、グラフが雑になっているのがわかる。
これは、問題：1でも説明したように、オイラー法では、現在の時間の微分値と、時間の積を、前回の計算結果に加算する、という作業をしている。つまり、出発点でしか微分されない。ということなので、$\tau$の値が大きかったら、まず出発点で微分し、次の点で微分するまでの時間が長いので、あまり厳密なグラフを表すことはできない。
つまり、$\tau$を大きくすればするほど、実際の値とはかけ離れた、より雑なグラフになってしまうのである。
このことから、実際にオイラー法を用いて適用しようとする時は、なるべく$\tau$の値を小さくし、より実際の値に近づけることが必要である。

## 3.5　問題：5

オイラー法よりも高精度な数値計算アルゴリズムについて調べよ。余力のある人は、アルゴリズムを実装しその結果をオイラー法 と比較してみよ。

◯ まず、常微分方程式は、オイラー法やルンゲクッタ法で、数値解を求めることができる。
そこで、オイラー法よりも高精度な数値計算アルゴリズムとして知られる、ルンゲクッタ法という方法を考える。ルンゲクッタ法の基本的な手順はオイラー法と同じだが、オイラー法との違いは、傾きの求め方を調整・工夫するということである。

まず、2次のルンゲクッタ法について説明する。2次のルンゲクッタ法には、ホイン法と中点法などがある。
オイラー法の精度は1次だったのに対し、2次のルンゲクッタ法の精度は2次である。

- ホイン法
  2次の精度ということは、テイラー展開より、

$$y(x_0+h) = y(x_0) + y'(x_0)h + \frac{1}{2}y''(x_0)h^2 + O\left(h^3\right)$$

  即ち、計算アルゴリズムは

$$\Delta y = y'(x_0)h + \frac{1}{2}y''(x_0)h^2 + O\left(h^3\right)$$

  となっている。

y の増分 Δ を計算するためには、1 階微分と 2 階微分の 2 項を満たす式が必要。そうすると少なくとも 2 点の値が必要となる。つまり、2 点として、計算区間の両端の導関数の値を使う。

まとめると、数値計算で近似値を求めるには次式を使うことになる。

$$\begin{aligned} k_1 &= hf\left(x_n, y_n\right) \\ k_2 &= hf\left(x_n + h\ ,\ y_n + k_1\right) \\ y_{n+1} &= y_n + \frac{1}{2}\left(k_1 + k_2\right) \end{aligned}$$

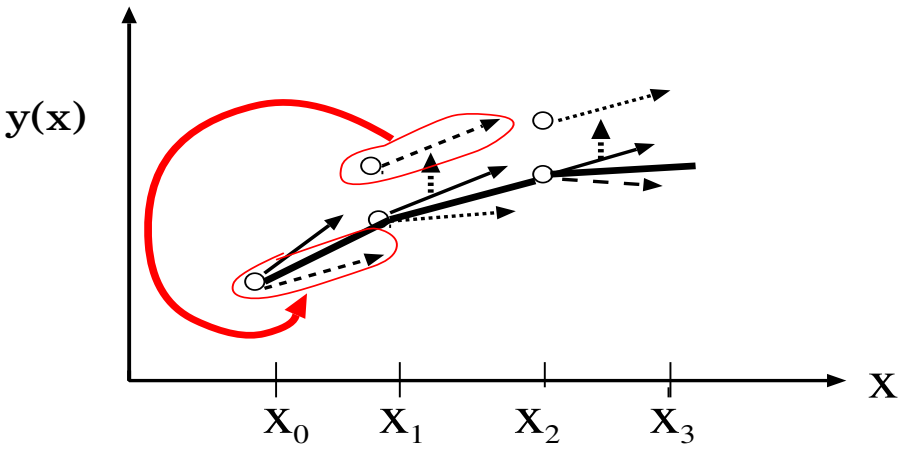


図 3: ホイン法のイメージ図

図 3 より、ホイン法は、ある区間での $y$ の変化 $\Delta y$ は、 計算の始めと終わりの点付近の平均傾きに区間の幅 $\Delta x$ を乗じて求めている。

オイラー法の図 1 と比較してみても、ホイン法の方が精度がいいことがわかる。

- 中点法

  これも、ホイン法と同じく 2 次の精度で、ホイン法は区間の両端の点の導関数を使ったが、中点方は、始点と中点を使う。2 点あるから 2 次の精度となる。

  ホイン法との違いは、ホイン法が、区間の増分 $\alpha$ ，$\beta$ で、

$$\Delta y = h\{\alpha y'\left(x_0\right) + \beta y'\left(x_0 + h\right)\}$$

だったのに対して、中点法は、

$$\Delta y = h\{\alpha y'\left(x_0\right) + \beta y'\left(x_0 + \frac{h}{2}\right)\}$$

と表せるところだ。

よって、まとめると、中点法の公式は、

$$\begin{aligned} k_1 &= hf\left(x_n, y_n\right) \\ k_2 &= hf\left(x_n + \frac{h}{2}\ ,\ y_n + \frac{k_1}{2}\right) \\ y_{n+1} &= y_n + k_2 \end{aligned}$$

となる。

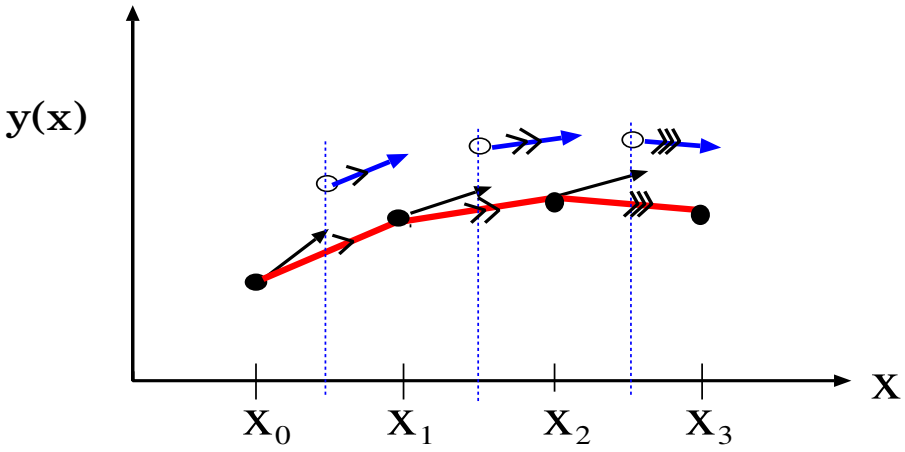


図 4: 中点法のイメージ図

図４より、中点法は、ある区間での $y$ の変化 $\Delta y$ は、中点付近の傾きに区間の幅 $\Delta x$ を乗じて求めている。

オイラー法の図１と比較してみても、この中点法の方が精度がいいことが明らかである。

◯ 次に、４次のルンゲクッタ法について説明する。４次のルンゲクッタ法の特徴は、まず、出発点 $y_n$ で１回、中点で２回、終点で１回の、計４回微分するということである。これらの微分の計算を経て、最後に関数値 $y_{n+1}$ が得られる。これを式で表すと以下のようになる。

$$\begin{aligned} k_1 &= hf\left(x_n, y_n\right) \\ k_2 &= hf\left(x_n + \frac{h}{2}\ ,\ y_n + \frac{k_1}{2}\right) \\ k_2 &= hf\left(x_n + \frac{h}{2}\ ,\ y_n + \frac{k_2}{2}\right) \\ k_2 &= hf\left(x_n + h\ ,\ y_n + k_3\right) \\ y_{n+1} &= y_n + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \end{aligned}$$

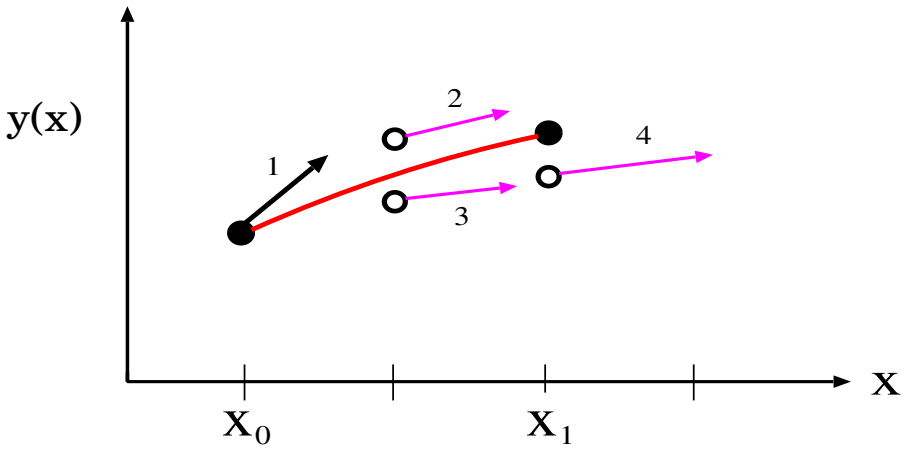


図 5: ４次のルンゲクッタ法のイメージ図

図５より、４次のルンゲクッタ法は、ある区間での $y$ の変化 $\Delta y$ は、区間内の４点の傾きのある種の加重平均にに幅 $\Delta x$ を乗じて求めている。

この方法も、図１に示したオイラー法よりも、精度がいいことが明らかである。