情報工学実験２
C++/Octaveによる微分方程式の数値解法

学籍番号 045718D：翁長絵美

実地日：平成17年12月19日
提出日：平成17年12月26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

```
// baseball.cpp: 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する.
  double y1, speed, theta;
  double r1[2+1], v1[2+1], r[2+1], v[2+1], accel[2+1];
    cout << "高さの初期値 (メートル) : "; cin >> y1;
  r1[1] = 0; r1[2] = y1;    // 初期位置ベクトル
  cout << "初期速度 (m/s) : "; cin >> speed;
  cout << "初期角度 (度) : "; cin >> theta;
  const double pi = 3.141592654;
  v1[1] = speed*cos(theta*pi/180);   // 初期速度 (x)
  v1[2] = speed*sin(theta*pi/180);   // 初期速度 (y)
  r[1] = r1[1]; r[2] = r1[2];   // 初期位置および初期速度を設定
  v[1] = v1[1]; v[2] = v1[2];

  //* 物理パラメータを設定 (質量, Cd 値など)
  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 << "空気抵抗 (あり:1, なし: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
```

```
//* 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ループ
double tau;
cout << "時間刻みτ（秒）: "; cin >> tau;
int iStep, maxStep = 1000;   // 最大の刻み数
double *xplot, *yplot, *xNoAir, *yNoAir;
xplot = new double [maxStep + 1];
yplot = new double [maxStep + 1];
xNoAir = new double [maxStep + 1];
yNoAir = new double [maxStep + 1];
for( iStep=1; iStep<=maxStep; iStep++ ) {

  //* プロット用に位置（計算値および理論値）を記録する
  xplot[iStep] = r[1];   // プロット用に軌道を記録
  yplot[iStep] = r[2];
  double t = ( iStep-1 )*tau;   // 現在時刻
  xNoAir[iStep] = r1[1] + v1[1]*t; // 位置（x）
  yNoAir[iStep] = r1[2] + v1[2]*t - 0.5*grav*t*t; // 位置（y）

  //* ボールの加速度を計算する
  double normV = sqrt( v[1]*v[1] + v[2]*v[2] );
  accel[1] = air_const*normV*v[1];   // 空気抵抗
  accel[2] = air_const*normV*v[2];   // 空気抵抗
  accel[2] -= grav;                  // 重力

  //* オイラー法を用いて，新しい位置および速度を計算する
  r[1] = r[1] + tau * v[1];
  r[2] = r[2] + tau * v[2];
  v[1] = v[1] + tau * accel[1];
  v[2] = v[2] + tau * accel[2];

  //* ボールが地面に着いたら（y < 0）ループを抜ける
  if(r[2] < 0) break;
}

//* 最大到達高さと滞空時間を表示する
cout << "最大到達高さは" << r[1] << "メートル" << endl;
cout << "滞空時間は" << (iStep-1)*tau << "秒" << endl;
```

```
  //* プロットする変数を出力する
  //   xplot, yplot xNoAir, yNoAir
  ofstream xplotOut("xplot.txt"), yplotOut("yplot.txt"),
    xNoAirOut("xNoAir.txt"), yNoAirOut("yNoAir.txt");

  int i;
  for( i=1; i<=iStep+1; i++ ) {
    xplotOut << xplot[i] << endl;
    yplotOut << yplot[i] << endl;
  }
  for( i=1; i<=iStep+1; i++ ) {
    xNoAirOut << xNoAir[i] << endl;
    yNoAirOut << yNoAir[i] << endl;
  }

  delete [] xplot, yplot, xNoAir, yNoAir;  // メモリを開放

}
```

3　実験テキスト中にある問題に回答せよ。

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。

オイラー法は、「刻み幅」τ を定めて、独立変数のとびとびの値、

$$t_n = a + n\tau \tag{1}$$

における未知関数 $x^i(t_n)$ の近似値 x_n^i を

$$x_{n+1}^i = x_n^i + \tau f_n^i$$
$$f_n^i \equiv f^i(x_n^1, \cdots, x_n^m, t_n)$$

という式において次々と定めていく方法である。$(n = 0, 1, 2, \cdots)$ 式 (1) から見てとれるように、オイラー法は傾きに依存している。そのため、刻み幅が

大きいと誤差が大きくなり、また、それを解消しようと刻み幅を小さくすると計算時間が長くなってしまう。他にも、オイラー法の式は対称でないため、逆から計算しても同じ値にならない。そのため精度が悪く、実際にはあまり使われない。

問題:2

$\frac{d}{dt}x = -x, x(0) = 1$ の解は $x(t) = e^{-t}$ となる。微分方程式の解を導出して確かめよ。

$$\frac{d}{dt}x = -x$$

の式を変形すると

$$\frac{-1}{x}dx = dt$$

となり、両辺を次のように積分する。

$$\int \frac{-1}{x}dx = \int dt$$
$$logx = -t + c$$

また、$logx = -t + c$ を計算する。

$$\begin{aligned} logx &= -t + c \\ x &= e^{-t+c} \\ x &= e^{-t} * e^{c} \end{aligned}$$

ここで、$x(0) = 1$ が与えられているので、それを使うと、

$$\begin{aligned} x(0) &= e^{0} * e^{c} \\ &= 1 * e^{c} = 1 \end{aligned}$$

これより、$c = 1$ ということがわかる。よって答えは

$$x = e^{-t}$$

になり確認できる。

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 τ は $0 < t \leq 2$ の間で 5 個選ぶこと。補助的に Octave を用いてもよい。

セクション 2 で表示したプログラムのソースを τ の値を変えて実行を 5 回行った。図 1 にそれを示す。値はそれぞれ、高さの初期値 $= 23$(m)、初期速度 $=$ 30(m/s)、初期角度 $= 45$(度)、空気抵抗:あり、τ= 0.1、1.5、0.5、0.05、0.2 と設定した。

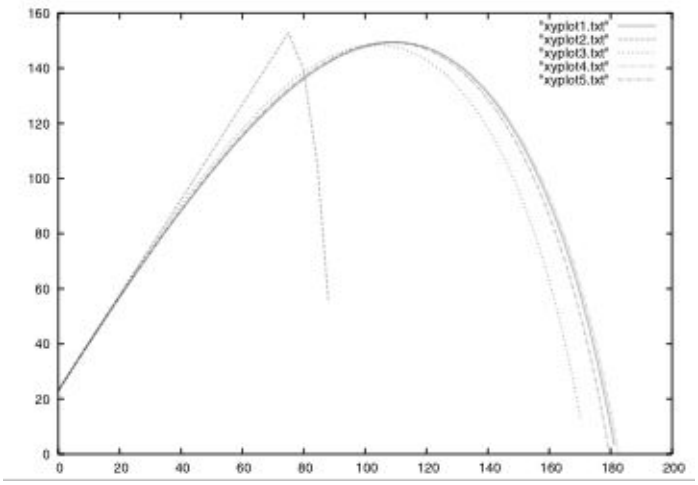

図 1: ボールの軌道グラフ

グラフから見えるように、時間刻みの値を細かくすればするほど、正確な値に近づいている。

問題:4

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

問題 3 の図 1 でもわかるように、刻み時間 τ が大きい値のものはグラフが大雑把になってることが見てとれる。問題 1 で述べたように、オイラー法は、傾きと刻み幅によって答えが左右されるため、より刻み幅が小さい方が実際の値に近くなる。

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てよ。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。

オイラー法の欠点である、区間が先に進むにつれて誤差が蓄積されるということを解消するための方法として、変形オイラー法がある。変形オイラー法は、初期値とは別にもう１つ出発点を設け、

$$y_{i+1} = y_{i-1} + 2h * f(x_i, y_i)$$

という計算式を使って近似値解を求めていく。しかし、問題によっては、解が振動して求まらない場合がる。

また、その他にもルンゲ・クッタ法というものがある。式は

$$x_{i+1} = x_i + h$$
$$y_{i+1} = y_i + k$$

と表す。ルンゲ・クッタ法は１階の常微分方程式の解法としてよく用いられている方法である。具体的に解を求めていく方法はオイラー法と同様であるがこの方法は傾きの求め方を工夫することで精度を高く実現する。傾きの求め方は、式の k の部分であるが、k は

$$k = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

とあたえられ、それぞれ

$$\begin{aligned} k_1 &= f(x, y) * h \\ k_2 &= hf(x + \frac{h}{2}, y + \frac{k_1}{2}) \\ k_3 &= hf(x + \frac{h}{2}, y + \frac{k_2}{2}) \\ k_4 &= hf(x_0 + h, y + k_3) \end{aligned}$$

となる。k_1, k_2, k_3, k_4 の係数比を 1:2:2:1 の割合で平均した値で修正して行く。このように傾きの求め方を細かくしていき精度を高める。

参考文献

[1] オイラー法について
http://www5c.biglobe.ne.jp/ uso-ats/robbt/robo-no2/r-no2-1.html
http://yosirin9.hp.infoseek.co.jp/contents/no3/paper_no3.htm