情報工学実験
オイラー法による軌道計算

035760A:横田敏明

実験状況

共同実験者

村山正嗣
宮城大輔

1　実験概要

1.1　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．

1.2　数値解法の必要性

古くより弾道計算，構造物の振動解析，電気回路の過渡応答解析，化学反応の進行具合いの解析，近頃では宇宙ロケットの軌道計算やも加わって，偏微分あるいは (常) 微分方程式の初期値問題の形に定式化される実際問題は数知れないくらい多い．その他にも，近年の 3D(コンピュータグラフィックス) やバーチャルリアリティなどもコンピュータを使った数値計算なしでは実現不可能である．section 解答

1.3　解答 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.」

計算過程での桁落ちは，問題に依っては大変な誤差が生じる．オイラー法は連続的な関数を漸化式で表し，順序良く一定の周期で値をプロットしてゆく方法であるため，特に指数関数の計算で大きな誤差が生じる．

1.4　解答 3

以下のプログラムを使用した．

```
// 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 kidou() {

  //* ボールの初期位置及び初期速度を設定する.
  double y1, speed, theta, maxheight=0.0;
  double r1[2+1], v1[2+1], r[2+1], v[2+1], accel[2+1];
    cout << "height(m) : "; cin >> y1;
```

```
r1[1] = 0; r1[2] = y1;    // 初期位置ベクトル
cout << "velosity(m/s) : "; cin >> speed;
cout << "theta(degree) : "; 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 << "Air 1(exist) or 0(less) : "; 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 << "tau(sec) : "; cin >> tau;
int iStep, maxStep = 100000000;   // 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];
ofstream xyplotOut("xyplot.txt");
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];
  //* xyplot
  xyplotOut << xplot[iStep] << " " << yplot[iStep] << endl;
```

```
  cout << xplot [iStep] << "   " <<yplot[iStep] <<endl;
  //* xyplotend

  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;        // 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
  if( r[2] + v[2]*tau > maxheight)
      maxheight = r[2] + v[2]*tau;
  v[1] = v[1] + accel[1]*tau;
  r[1] = r[1] + v[1]*tau;
  v[2] = v[2] + accel[2]*tau;
  r[2] = r[2] + v[2]*tau;
  //* 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 << "maxheight : " << maxheight << "m" << endl;
  cout << "fly lange : " << r[1] << "m" << endl;
  cout << "fly time : " << (iStep)*tau << "sec" << endl;
  cout << "writting opelations" << endl
  << "---writting files---" << endl
  << "xyplot.txt" << endl
  << "xplot.txt" << endl
  << "yplot.txt" << endl
  << "xNoAir.txt" << endl
  << "yNoAir.txt" << endl
  << "--------------------" << endl
  << "please wait..." << 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;
  }
    cout << "finished" << endl;
  delete [] xplot, yplot, xNoAir, yNoAir;  // メモリを開放

}
```

プログラム中にステータスを逐次表示する記述を組み込み，動作を確認した．また，x プロットと y プロットを同一ファイルに書き込み，GNU プロットでの処理を容易にした．

動作確認．
初期条件と結果は以下の通り．

```
[Toshiaki-YOKODA:~/cpp] j03060% a.out
height(m) : 2
velosity(m/s) : 100
theta(degree) : 45
Air 1(exist) or 0(less) : 1
tau(sec) : 0.01

(中略)

222.42   0.639574
222.526   0.320687
```

```
222.633   0.00148584
maxheight : 93.0157m
fly lange : 222.739m
fly time : 8.55sec
writting opelations
---writting files---
xyplot.txt
xplot.txt
yplot.txt
xNoAir.txt
yNoAir.txt
--------------------
please wait...
finished
```

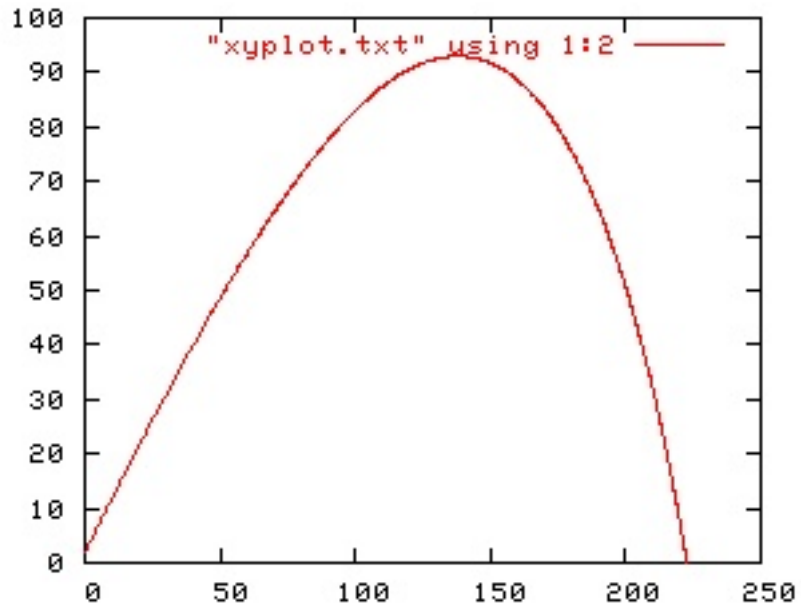

図 1: GNUplot によるグラフ描画 (tau=0.01)

```
[Toshiaki-YOKODA:~/cpp] j03060% a.out
height(m) : 1
velosity(m/s) : 200
```

```
theta(degree) : 45
Air 1(exist) or 0(less) : 1
tau(sec) : 0.5
0   1   TIME = 0
26.675   25.2225   TIME = 0.5
47.3644   41.5572   TIME = 1
64.6573   52.7578   TIME = 1.5
79.7315   60.0689   TIME = 2
93.2328   64.1646   TIME = 2.5
105.548   65.4479   TIME = 3
116.913   64.1798   TIME = 3.5
127.469   60.5495   TIME = 4
137.292   54.7191   TIME = 4.5
146.415   46.8509   TIME = 5
154.854   37.1206   TIME = 5.5
162.616   25.7182   TIME = 6
169.712   12.8428   TIME = 6.5
maxheight : 71.7107m
fly lange : 176.157m
fly time : 7sec
writting opelations
---writting files---
xyplot.txt
xplot.txt
yplot.txt
xNoAir.txt
yNoAir.txt
--------------------
please wait...

xplot.txt  finished

yplot.txt finished
all finished.
```

これ以上τを大きくすると計算できなくなってしまう．さらに飛距離や滞空時間を見ると，大きな誤差が生じてしまう．

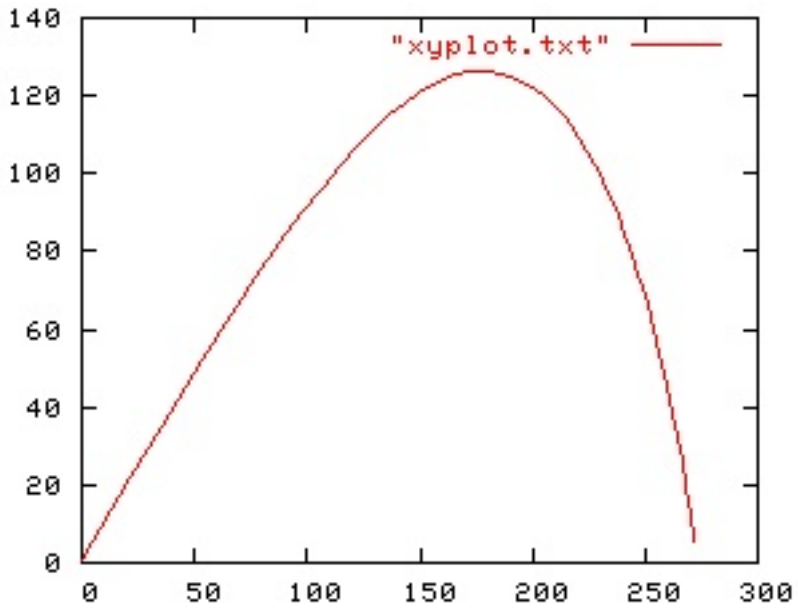

図 2: GNUplot によるグラフ描画 (tau=0.2)

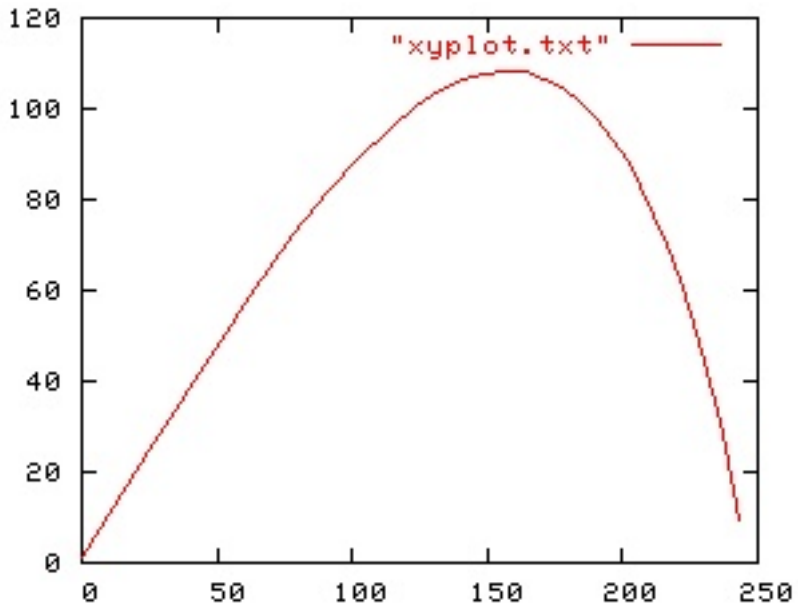

図 3: GNUplot によるグラフ描画 (tau=0.3)

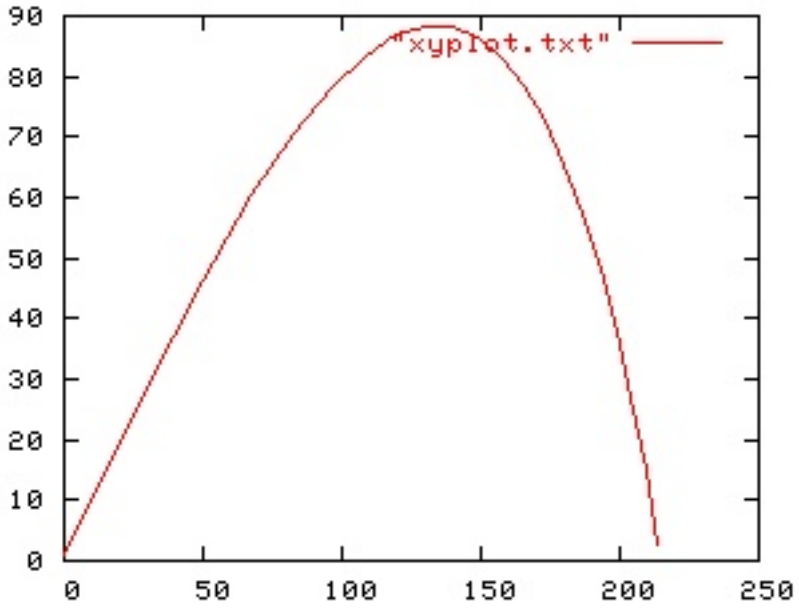

図 4: GNUplot によるグラフ描画 (tau=0.4)

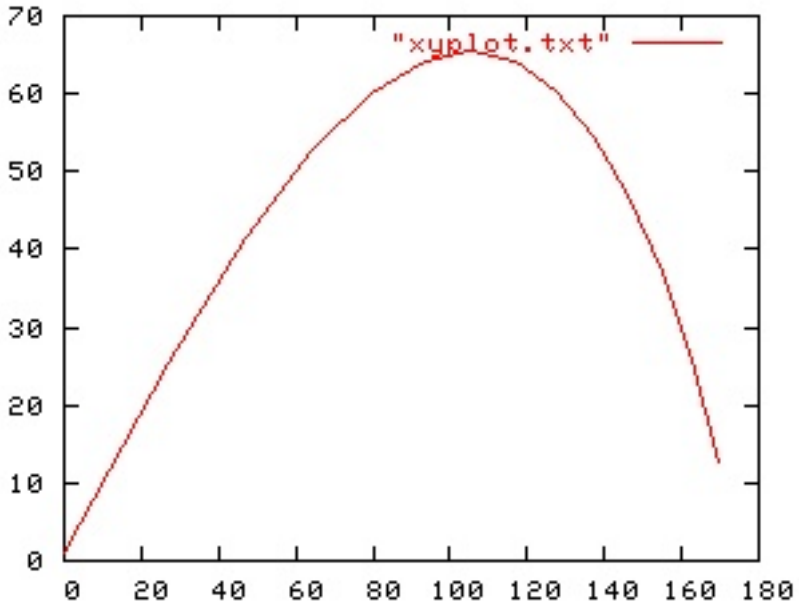

図 5: GNUplot によるグラフ描画 (tau=0.5)

2　時間刻みτの設定によって，計算結果が大きく異なる理由を考察せよ．

空気抵抗による影響は，最後に計算した時点でのステータスが，次回の計算まで保持されるため，刻み幅が大き過ぎると誤差が許容誤差を大きく上回る．空気抵抗は場合によっては速度の自乗に比例するため，速度があまりに速すぎるとき，急激なブレーキ力が加わる．そのブレーキ力は，刻み幅τの間は一定であるため，速度が速く，刻み幅が大きいほど余分に減速されてしまう．すなわち，指数関数となる計算はその時点での値が微分係数を決定付けるために，オイラー法の精度を悪くする原因になっている．

3　オイラー法以外の高精度な測定法

例として，ルンゲクッタ法が挙げられる．

オイラー法の漸化式に対して，次項までの間に中間的な関数を用いて，誤差を減少させる方法をルンゲクッタ法という．すなわち，h を刻み幅とすると，

$$y_{n+1} = y_n + k_2 + O(h^3) \tag{1}$$

とすれば，刻み幅はさらに減少したことになる．