微分方程式の数値解法

学籍番号 045713C:大城和也

提出日:平成 17 年 12 月 26 日 (月)

実験共同者
045709E：上原直久
045734B：知念栄作
045739G：友寄雄一朗

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．

2 課題項目

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事はほとんどない．オイラー法の問題点を説明せよ．

問題 2

下記の微分方程式の解を導出し，例題の通りになるか確かめよ．

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

問題 3

実験において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 を用いても良い．

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できるが，その理由を考察せよ．

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．

3 報告項目

3.1 オイラー法の問題点について

オイラー法は以下の式によって微分方程式を求める数値解法である．

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

この一般化された式は以下のように求められている．

$$\begin{aligned}\frac{d}{dt}x &= \frac{x_{n+1}-x_n}{t_{n+1}-t_n} = \frac{x_{n+1}-x_n}{\tau} \\ x_{n+1} &= x_n + \tau\frac{d}{dt}x_n\end{aligned}$$

これを見ると式は微分の定義式を変形させたものだということが分かる．つまり，τ の値を限りなく 0 に近づけることをすることによりこのオイラー法は完全な精度の値を取ることが出来る．しかし，コンピュータでこのように計算することは計算時間の問題で不可能となる．よって，通常は τ の値を大きくし，現実的に計算可能な時間に押さえるようにするのだが，この値を大きくすると，１つ１つの間の誤差がでることとなる．さらに，オイラー法では前の値から次の値を求める形になるため，誤差は蓄積されていき，最終的には無視できない誤差となる．

さらにオイラー法の式は対称性が無いため，逆算を行っても元の式に戻ることが出来ない．これらの理由のため，オイラー法は精度が悪く，あまり扱われない解法になっている．

3.2 微分方程式の解法

以下の微分方程式の解法を示す．

$$\begin{aligned}\frac{d}{dt}x &= -x & (1) \\ x(0) &= 1 & (2)\end{aligned}$$

初めに (1) の変形を行う．

$$\begin{aligned}\frac{1}{x}dx &= -dt \\ \int \frac{1}{x}dx &= -\int dt \\ \ln x &= -t + C\end{aligned}$$

指数対数をとって

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

式 (2) の初期条件を代入して

$$\begin{aligned}1 &= Ce^0 \\ C &= 1\end{aligned}$$

上記より C を代入する．

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

となり，例題通りの値となっていることが確認できる．

3.3　実験におけるボールの軌道

今回のプログラムは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を用いた．そのプログラムと実行結果，及びその時の軌道のグラフを示す．

3.3.1　baseball プログラム

```
-------------------------------------------------------------------------
//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( yplot[iStep] < 0 && yNoAir[iStep] < 0 ) break;

    //ボールの最大到達高さの表示
    if( yplot[iStep-1]-yplot[iStep-2] > 0 && yplot[iStep-1]-yplot[iStep] > 0 )
      cout << "最大到達高さ予測値:" << yplot[iStep-1] << endl;
    if( yNoAir[iStep-1]-yNoAir[iStep-2]>0 && yNoAir[iStep-1]-yNoAir[iStep]>0 )
      cout << "最大到達高さ理想値:" << yNoAir[iStep-1] << endl;
  }

  //* 到達距離と滞空時間を表示する
  cout << "到達距離は" << r[1] << "メートル" << endl;
  cout << "滞空時間：" <<  ( iStep-1 )*tau << "秒" << endl;
  // 滞空時間の表示をここに書く

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

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

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

}
-----------------------------------------------------------------------------
```

3.3.2 実行結果

今回は τ の値による変化を見るために τ 以外の値は初期高さ:0，初期速度:20[m/s]，初期角度:45 度，空気抵抗は無しで統一している．

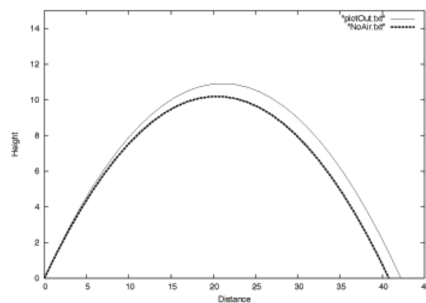

図 1: τ=0.1

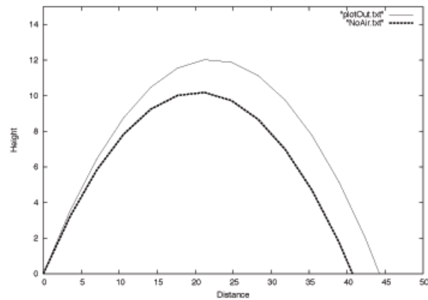

図 2: τ=0.25

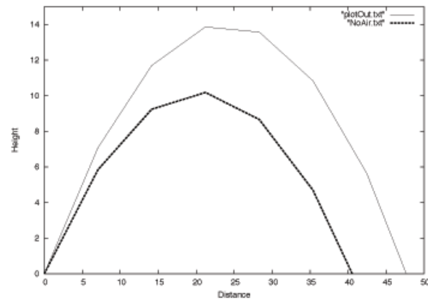

図 3: τ=0.5

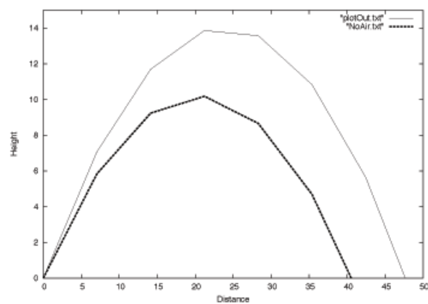

図 4: τ=1.0

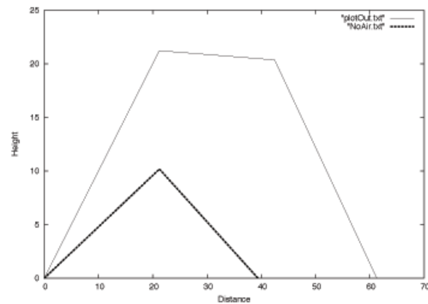

図 5: τ=1.5

3.4 τ の値の考察

上記の図 1〜5 を見れば分かるように τ の値が小さければ小さいほどオイラーで導きだされた値は理想値に近くなることが分かる．これはオイラー法の欠点でも述べたように，τ という刻み幅は即ち次の値へ動くための時間となっていることに関連している．

オイラー法は初期値とその時の微分した値を足すことにより次の値を導く．この時の微分した値 (今回は速度と加速度になっている) は即ちその際の傾きとなっている．τ は時間の幅なので τ を掛けることによりその点の傾きで τ 秒だけ進んだ値が次の値となる．そして次の値でまた微分し，傾きを求めて $\cdots$. ということを繰り返す．よって τ が大きいと次の微分までの距離が延びるので正確さが失われることとなる．微分定義ではこの τ を微小時間とすることで微分できるとなっている，よって τ の値が小さいほど正確な値が選ばれるようになり，大きくすることで誤差が大幅に増えて行くのである．

3.5　オイラー法以外の数値計算アルゴリズム

オイラー法以外の数値計算を行う方法としてここでは代表的なルンゲクッタ法について述べる．

3.5.1　ルンゲクッタ法

ルンゲクッタ法はオイラー法と同じく現時点から微分などを行い，傾きを求めてその値に刻み値を掛けて次の値を予測している．ただ，オイラー法と異なるのは傾きを求める際の決め方が変化する．以下が 4 次ルンゲクッタ法と呼ばれるルンゲクッタ法の公式である．

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

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

ちなみにこの公式はテイラー展開から導かれている．テイラー展開の結果，第 2 項目のを取ったのがオイラー法，第 2 項と 3 項を扱うのが 2 次のルンゲクッタ法，4 項まで扱うのが 3 次のルンゲクッタ法と呼ばれ，以下項数が増えればそれに応じて n 次のルンゲクッタ法と言われる．しかし，4 次以降は刻み値を変化させた方が有効なためあまり使われることは無い．

3.6　ホイン法

では，ここから 2 次のルンゲクッタ法の１つであるホイン法 (修正オイラー法) と呼ばれている方法について示す．ホイン法は 2 次であるため，テイラー展開の第 3 項までを用いる．つまり，

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

を扱う．このとき y の増加分を求めるには計算区間の両端の導関数の値を扱うことを考えると

$$y(x_0 + h) = h\alpha y'(x_0) + \beta y'(x_0 + h)$$

となるのでこれを x_0 周りでテイラー展開すると

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

これを先ほどのテイラー展開と比べることで α と β の値を決めるつまり，

$$
\begin{aligned}
\alpha &= \frac{1}{2} \\
\beta &= \frac{1}{2}
\end{aligned}
$$

となる．

これらをまとめるとホイン法の公式は以下のようになる．

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

この式はつまり，現時点で微分して求めた傾きと次の点での傾きを平均したものを次の値を求めるための傾きとするものである．図 6 はこの関係を表している．丸で囲まれているのが同じ傾きであり，現在の傾きとの間を通っている．これを繰り返していることが分かる．

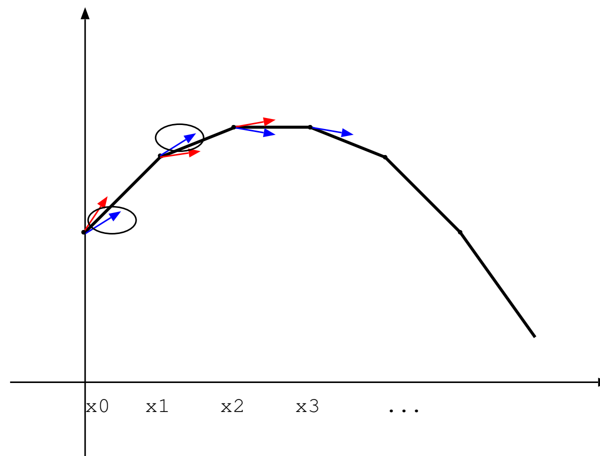

図 6: τ=0.1

このように，次の点の傾きを用いることで精度を高めている．基本的には 3 次や 4 次のルンゲクッタも同様であり，たくさんとることでさらに精度を上げている．

3.6.1 実装

上記で説明したホイン法を実装したプログラムを示す．プログラムは前出の baseball.cpp のオイラー法の記述の辺りだけを書き換えたものであるのでそれ以外は省略する．

```
--------------------------------------------------------
...
// ホイン法を用いて，新しい位置および速度を計算する
 v[3] = v[1] + tau*accel[1];
 v[4] = v[2] + tau*accel[2]; // 次の点での速度

 r[1] = r[1] + 0.5*(tau*v[1]+tau*v[3]);
 r[2] = r[2] + 0.5*(tau*v[2]+tau*v[4]);

 accel[3] = air_const*normV*v[3];
 accel[4] = air_const*normV*v[4];
 accel[4] -= grav; // 次の点での加速度

 v[1] = v[1] + 0.5*(tau*accel[1]+tau*accel[3]);
 v[2] = v[2] + 0.5*(tau*accel[2]+tau*accel[4]);
...
--------------------------------------------------------
```

このプログラムを動かした時のグラフは図 7 のようになる．パラメータは問題 3 の時と同様であり，τ の値は 0.5 としている．グラフを見て分かるように NoAir の理想の値と全く同じである．この時，τ の値をどんな値にしても同様なことが起こる．これは放物線の関数が 2 次の式で得られるため，2 次の精度であるホイン法で全て予測できたからである．

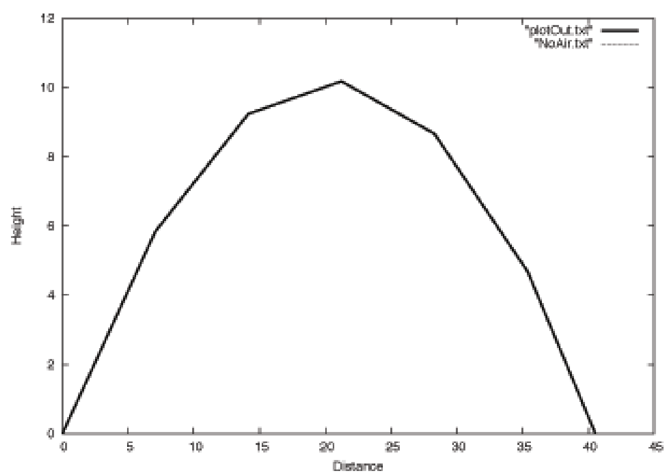

図 7: ホイン法による予測

参考文献

[1] "http://www.sskato.jp/process/joubibun.htm", 8 微分方程式

[2] "http://www.ny.airnet.ne.jp/satoh/index.htm", 佐藤 元　青空ページ

[3] "http://www.ipc.akita-nct.ac.jp/ỹamamoto/lecture/2003/5E/lecture_5E/diff_eq/", 常備分方程式の数値計算法

[4] "http://maya.phys.kyushu-u.ac.jp/k̃nomura/education/numerical-physics/", 計算物理学