Eigen を使って常微分方程式の初期値問題を解く

桂田 祐史

2018 年 7 月 24 日

1 Eigen の入手・インストール

Eigen は、C++ で行列・ベクトルを扱うための、フリーのクラスライブラリィである。
Eigen[1] から Eigen 3.3.5[2] を入手する。
ターミナルで

```
tar xjf eigen-eigen-b3f3d4950030.tar.bz2
cd eigen-eigen-b3f3d4950030
tar cf - Eigen | (cd /usr/local/include;sudo tar xpf -)
```

(“b3f3d4950030” の部分は、いつファイルをダウンロードしたかによって異なる。その部分を実際に自分が入手したものに置き換えること。)

とする。パスワードを尋ねられる。これでインストール終了する。
Eigen Documentation[3]
特に Quick reference guide[4]

2 例題: 空気抵抗のあるボールの運動

バウンドの処理はいいかげんです。

2.1 計算だけするプログラム

計算だけして、軌跡は gnuplot で、というプログラム

[1] http://eigen.tuxfamily.org/index.php?title=Main_Page
[2] http://bitbucket.org/eigen/eigen/get/3.3.5.tar.bz2
[3] http://eigen.tuxfamily.org/dox/
[4] http://eigen.tuxfamily.org/dox/group__QuickRefPage.html

ball-bound.cpp

```
/*
 * ball-bound.cpp --- Eigen を使ってはずむボールのシミュレーション
 *
 *   c++ ball-bound.cpp
 *     もしかしたら -I/usr/local/include のような指定が必要かも。
 *   ./a.out > ball-bound.dat
 *   gnuplot
 *   gnuplot> plot "ball-bound.dat"
 */

#include <iostream>
#include <cmath>
#include <Eigen/Dense>
using namespace Eigen;

double m, g, Gamma, e;

VectorXd f(double t, VectorXd x)
{
  VectorXd y(4);
  y(0) = x(2);
  y(1) = x(3);
  y(2) = - Gamma / m * x(2);
  y(3) = - g - Gamma / m * x(3);
  return y;
}

int main(void)
{
  int n, N;
  double tau, Tmax, t,pi;
  VectorXd x(4),k1(4),k2(4),k3(4),k4(4);

  pi = 4 * atan(1.0);
  m = 100;
  g = 9.8;
  Gamma = 1.0;
  e = 1.0;

  Tmax = 20;
  N = 1000;
  tau = Tmax / N;
  x << 0,0,50*cos(pi*50/180),50*sin(pi*50/180);
  for (n = 0; n < N; n++) {
    t = n * tau;
    k1 = tau * f(t, x);
    k2 = tau * f(t+tau/2, x+k1/2);
    k3 = tau * f(t+tau/2, x+k2/2);
    k4 = tau * f(t+tau, x+k3);
    x = x + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
    if (x(1)<0) {
      x(1) = - x(1);
      x(3) = - x(3);
    }
    std::cout << x(0) << " " << x(1) << std::endl;
  }
  return 0;
}
```

```
curl -O http://nalab.mind.meiji.ac.jp/~mk/ball-bound.cpp
head ball-bound.cpp
c++ ball-bound.cpp
./a.out > ball-bound.dat
gnuplot
```
(curl でファイルを入手。head で先頭 10 行を表示して中身を確認。c++ でコンパイル。実行形式 a.out の出力を `ball-bound.dat` に保存。gnuplot を起動。)

gnuplot のプロンプトに `gnuplot>` に対して、`plot "ball-bound.dat"` とすると軌跡が描かれる (はず)。gnuplot を終了するには、`quit` とすれば良い。

```
gnuplot> plot "ball-bound.dat"
gnuplot> quit
$
```

2.2 GLSC で軌跡を描くプログラム

GLSC で描画するには、コンパイル用のコマンド `c++glsc` を準備する必要がある。

```
curl -O http://nalab.mind.meiji.ac.jp/~mk/c++glsc
chmod +x c++glsc
cp c++glsc ~/bin
```

各自の Mac 用の調整が必要かもしれない。

それが出来れば

```
curl -O http://nalab.mind.meiji.ac.jp/~mk/ball-bound-glsc.cpp
head ball-bound-glsc.cpp
c++glsc ball-bound-glsc.cpp
./ball-bound-glsc
```

(curl でファイルを入手。head で先頭 10 行を表示して中身を確認。`c++glsc` でコンパイル。実行形式 `ball-bound-glsc` を実行する (画面に図が表示されるはず)。

2.3 プログラムの解説

2.3.1 微分方程式と初期条件

質点の質量は m とする。質点の位置は (y 成分は考えないことにして) x, z 座標で定まる。そこで

$$\boldsymbol{x} = \begin{pmatrix} x \\ z \end{pmatrix}$$

とおく。

質点に働く力は、重力と空気抵抗のみとする。

重力は $m\boldsymbol{g}$ ($\boldsymbol{g} = \begin{pmatrix} 0 \\ -g \end{pmatrix}$).

空気抵抗は速さに比例した大きさで、速度と逆向きと仮定すると、$-\gamma\dfrac{d\boldsymbol{x}}{dt}$ (γ は正の定数) とおける。

運動方程式は

$$m\frac{d^2\boldsymbol{x}}{dt^2} = m\boldsymbol{g} - \gamma\frac{d\boldsymbol{x}}{dt}. \tag{1}$$

初期条件は、秒速 50 m で、仰角 50° で投げる。

$$x(0) = 0, \quad z(0) = 0, \quad \frac{dx}{dt}(0) = 50\cos 50^\circ, \quad \frac{dz}{dt}(0) = 50\sin 50^\circ \tag{2}$$

とする。

$$X_1 = x, \quad X_2 = z, \quad X_3 = \frac{dx}{dt}, \quad X_4 = \frac{dz}{dt}, \quad \boldsymbol{X} = \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{pmatrix}$$

とおくと、

$$\frac{d\boldsymbol{X}}{dt} = \begin{pmatrix} \dfrac{dX_1}{dt} \\ \dfrac{dX_2}{dt} \\ \dfrac{dX_3}{dt} \\ \dfrac{dX_4}{dt} \end{pmatrix} = \begin{pmatrix} \dfrac{dx}{dt} \\ \dfrac{dz}{dt} \\ \dfrac{d^2x}{dt^2} \\ \dfrac{d^2z}{dt^2} \end{pmatrix} = \begin{pmatrix} X_3 \\ X_4 \\ -\gamma\dfrac{dx}{dt} \\ -g - \dfrac{\gamma}{m}\dfrac{dz}{dt} \end{pmatrix} = \begin{pmatrix} X_3 \\ X_4 \\ -\gamma X_3 \\ -g - \frac{\gamma}{m}X_4 \end{pmatrix}.$$

そこで

$$\boldsymbol{f}(t, \boldsymbol{X}) := \begin{pmatrix} X_3 \\ X_4 \\ -\gamma X_3 \\ -g - \frac{\gamma}{m}X_4 \end{pmatrix} \tag{3}$$

とおくと、

$$\frac{d\boldsymbol{X}}{dt} = \boldsymbol{f}(t, \boldsymbol{X}). \tag{4}$$

$$\boldsymbol{X}(0) = \begin{pmatrix} 0 \\ 0 \\ 50\cos 50^\circ \\ 50\sin 50^\circ \end{pmatrix}. \tag{5}$$

2.3.2 プログラムの解読

- 11 行目 これは C++ のプログラムにはいつでも書く。
- 12 行目 C の `#include <math.h>` に相当する。代わりに `#include <math.h>` としても OK.

- 13 行目 これで Eigen の密行列関係の機能が使えるようになる。
- 14 行目 `Eigen::` を省略するため。同様に `using namespace std;` とすることも考えられるが、このプログラムでは、55 行目に `std::cout`, `std::endl` としているだけなので、しなかった。
- 18 行目 `VectorXd` は成分が double で、サイズが動的に選べるベクトル
- 18〜26 行 関数定義。`f()` は、常微分方程式 $\dfrac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(t, \boldsymbol{x})$ の右辺の $\boldsymbol{f}(t, \boldsymbol{x})$ を計算するための関数である。
- 43 行目。これは

```
x(0) = 0;
x(1) = 0;
x(2) = 50 * cos(pi*50/180);
x(3) = 50 * sin(pi*50/180);
```

  としても良い。
- 46〜50 行はいわゆる Runge-Kutta 法である。時間刻みを τ としたとき、

$$\begin{aligned}
\boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \frac{1}{6}\left(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4\right), \\
\boldsymbol{k}_1 &= \tau f(t, \boldsymbol{x}) \\
\boldsymbol{k}_2 &= \tau f(t + \frac{\tau}{2}, \boldsymbol{x} + \boldsymbol{k}_1/2) \\
\boldsymbol{k}_3 &= \tau f(t + \frac{\tau}{2}, \boldsymbol{x} + \boldsymbol{k}_2/2) \\
\boldsymbol{k}_4 &= \tau f(t + \tau, \boldsymbol{x} + \boldsymbol{k}_3)
\end{aligned}$$

  というのが Runge-Kutta 法の公式であるが、それをほぼそのまま書けるところが、ベクトルを使える利点である。

A　C++とつきあう

A.1　C++ ってどんなもの

C++は C(言語) を拡張した言語である[5]。

C++ は誕生間もない頃は better C とか言われていたが、その後も拡張を続けて、今ではお化けのようになってしまった嫌いがなくもない。

しかし色々便利な仕掛けが用意されているため、多くのソフトウェア・ライブラリィが C++ のクラス・ライブラリィとして実現されるのが普通になってきている。すでに C を知っているならば、少しの手間で使えるようになるので、自分の必要に応じて C++ も使うようにすることを勧めたい。

大事なことは

C で出来ることは、おおむねそのままのやり方で C++ でも出来る
(C のプログラムは C++ のプログラムでもある、は大体正しい)

[5] C 言語のプログラムで、変数に `++` がつくと、値を 1 だけ増やす (インクリメント) ことになるのは知っているだろう。`for (i = 0; i < n; i++)` の `i++` とか。C++ は C (言語) をインクリメントするもの、と考えて命名されたらしい。

A.2 コンパイラーの使い方

C のコンパイラーのコマンド名は cc や gcc であるが、C++ のコンパイラーのコマンド名は c++ や g++ である。

C のプログラムはファイル名末尾が .c であるが、C のプログラムはファイル名末尾が .cpp, .cxx, .cc, .C (大文字), …である。自分でプログラムを作るときは、とりあえず .cpp を使うことを勧める。

ヘッダーファイル .h というのもあるが、これについては .hpp を使う。

例えば、C の場合

```
cc nantoka.c
./a.out

cc -O -o nantoka nantoka.c
./nantoka
```

C++ では

```
c++ nantoka.cpp
./a.out

c++ -O -o nantoka nantoka.c
./nantoka
```

A.3 printf() の代わりに

C で計算結果等を画面 (正しくは標準出力) に表示するには、printf() を使うのが普通であるが、C++ では、std::cout に << 演算子で送る。

C では

```
#include <stdio.h>
...
  printf("こんにちは、世界\n");
```

C++では

```
#include <iostream>
...
  std::cout << "こんにちは、世界" << std::endl;
```

std:: を書くのが面倒ならば、最初に using namespace std; としておけばよい。

C++では

```
#include <iostream>
using namespace std;
...
  cout << "こんにちは、世界" << endl;
```

以下の話は using namespace std; と宣言してあると仮定して説明する。

C で整数型の式は %d, 浮動小数点数の式は %g などの書式を使ったが、C++ では

```
  int i;
  double x;
...
  printf("i=%d, 2*x=%f\n", i, 2 * x);
```

```
  int i;
  double x;

  cout << "i=" << i << ", 2*x=" << 2 * x << endl;
```

書式の指定をしないで済むのは便利なようだが、本当に書式を指定したい場合は、少々面倒なことになる。

C の場合

```
  int i;
...
  printf("i=%4d\n", i);
```

C++の場合

```
  int i;
...
  cout << "i=" << setw(4) << i << endl;
```

C の場合

```
  double pi;
...
  printf("pi=%20.15g\n", pi);
```

C++の場合

```
#include <iomanip> // setprecision() に必要

  double i;
...
  cout << "pi=" << setw(20) << setprecision(15) << pi << endl;
```

- 通常の小数表記をするのに、C では %f という書式指定をするが、C++ では << std::fixed とする。
- 指数形式で表記するのに、C では %e という書式指定をするが、C++ では << std::scientific とする。
- C における %g に相当するのは、C++ では << std::defaultfloat とする。

```
/*
 * test1.cpp
 */

#include <iostream>  // #include <stdio.h> に相当
#include <cmath>     // #include <math.h> に相当
#include <iomanip>   // setprecision() に必要
using namespace std; // これはいつもやっておくと便利かも

int main(void)       // C と同じ
{
  double pi;
  pi = 4 * atan(1);
  cout << "12345678901234567890123456789O" << endl;
  cout << setw(25) << setprecision(15) << pi << endl;//printf("%25.15g\n", pi);

  cout << setw(25) << setprecision(15) << scientific << pi << endl;//printf("%25.15e\n", pi);

  cout << setw(25) << setprecision(15) << fixed << pi << endl;//printf("%25.15f\n", pi);
  return 0;
}
```

実行結果

```
$ c++ test1.cpp ; ./a.out
12345678901234567890123456789O
          3.14159265358979
    3.141592653589793e+00
        3.141592653589793
$
```

http://nalab.mind.meiji.ac.jp/~mk/labo/studying-C/Programing-in-C/node227.html

A.4 scanf() の代わりに

C ではキーボード (正しくは標準入力) からの入力を変数にセットするのに、例えば scanf() を使ったりするが、C++ では std::cin >> 変数名 とする。

```
// test2.cpp

#include <iostream>  // #include <stdio.h> に相当
using namespace std; // これはいつもやっておくと便利かも

int main(void)       // C と同じ
{
  int a, b;
  double x, y;

  cout << "input a,b: ";
  cin >> a >> b;     // C だったら scanf("%d%d", &a, &b);
  cout << "a=" << a << ", b=" << b << endl;
  cin >> x >> y;     // C だったら scanf("%lf%lf", &x, &y);
  cout << "x=" << x << ", y=" << y << endl;

  return 0;
}
```