平成 19 年 6 月 10 日

複雑系科学実験１ 渡辺担当分

渡辺 宙志 *

名古屋大学大学院情報科学研究科複雑系科学専攻多自由度システム情報論講座

概要

目 次

*E-mail: hwatanabe@is.nagoya-u.ac.jp

3　非線形偏微分方程式とカオス

前回、偏微分方程式を数値的に解いた。前回扱った微分方程式は、すべて解の重ね合わせが可能である、という特徴を持っている。たとえば、微分方程式、

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} = -y \tag{1}$$

の解は、$\sin(x)$ や $\cos(x)$ である。このとき、定数倍 $a\sin(x)$ も解であるし、定数倍の和 $a\sin(x) + b\cos(x)$ も解である。このように、ある微分方程式について f_1、f_2 が解であるとき、その定数倍の和 $af_1 + bf_2$ も元の微分方程式の解となるとき、そのような性質を**線形性 (linearity)**、この性質を持つ微分方程式を線形微分方程式と呼ぶ。線形微分方程式は解くことが容易である。もしある物理現象をあらわす方程式が線形であったら、その現象はかなり未来まで容易に予測することができる。しかし、残念ながら、世の中の興味ある現象はほとんど非線形である。今回は、非線形微分方程式に触れる。非線形微分方程式は数値解法も難しいので、単に方程式に触れて、非線形性とはどんなものか、感じをつかむようにしてもらいたい。

3.1　ロジスティック方程式

3.1.1　解析解

生物の個体数について考えよう。毎年決まった時期に繁殖するような生物 (たとえば昆虫) について考える。今年 (t) の個体数が n(t) であったとき、来年の個体数 n(t+1) はどうなっているだろうか。もしも、毎年一匹あたり子孫を k 匹残して、親は死ぬとすると、個体数の増減を表す方程式は

$$n(t+1) = kn(t) \tag{2}$$

となる。連続化して微分方程式であらわすと

$$\frac{\mathrm{d}n(t)}{\mathrm{d}t} = (k-1)n(t) \tag{3}$$

となり、解は

$$n(t) = \exp(k-1)\cdot t \tag{4}$$

である。容易に分かるように、$k < 1$ ならすぐに絶滅、$k > 1$ なら解は発散し、地球上はこの生物であふれかえることになる。

そこで、この式を少し修整することにしよう。生物が少ないときには増えるが、増えすぎると (食料難などで) 逆に減る、という式を考える。すると、もっとも簡単には

$$\frac{\mathrm{d}n(t)}{\mathrm{d}t} = (k-1)\frac{n(t)\,(K - n(t))}{K} \tag{5}$$

という式を考えることができる。K は環境の許容量 (carring capacity) と呼ばれる量で、個体数の最大値を規定する。ここで、$r \equiv k-1$、$x \equiv n/K$ と変数変換すると、

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = rx(1-x) \tag{6}$$

を得る。x は、許容量に対して、現在の個体数がどのくらいの割合を占めるかをあらわす量で、$x = 0$ なら絶滅、$x = 1$ なら最大値となる。さて、式 (6) は非線形である (たとえば x が解として、$2x$ が解でないことを確かめよ)。この方程式の一般解は、初期値を x_0 として、

$$x(t) = \frac{1}{1 + \left(\frac{1}{x_0}\right)\mathrm{e}^{-rt}} \tag{7}$$

となる。

3.1.2 課題１

式 (7) を gnuplot で表示し、初期値 x_0 や、パラメタ r の値を変えてどのように振る舞いが変わるか考察せよ。ただし、初期値 x_0 は $0 < x_0 < 1$ の範囲で、パラメタ r は $-1 < r < 1$ の範囲で変化させよ。

gnuplot で関数の宣言、表示は以下のように行う。

```
% gnuplot                    ← gnuplot を起動

G N U P L O T
Version 3.7 patchlevel 3
(中略)
gnuplot> f(x) = 1/(1+(1/x0)*exp(-r*x))  ←関数を定義
gnuplot> x0 = 0.1                       ←初期値を入れる
gnuplot> r = 0.5                        ←パラメタ設定
gnuplot> plot [0:10] f(x)               ← [0,10] の範囲で f(x) を表示
```

以下、x_0 と r の値を変えて何度もプロットしてみること。

3.1.3 差分化

さて、式 (6) の振る舞いは、いずれ定常状態に近づく、というものであった。それではこの式を差分化したらどうなるか考えよう。もともとこの生物は一年単位で繁殖するのであったので、

$$x(t+1) = rx(t)(1 - x(t)) \tag{8}$$

とあらわすことができるだろう[1]。すなわち、次の年の個体数は、今年の個体数のみで決定される。このように、初期値を決めると後の結果がすべて計算できるような系を**決定論的 (deterministic)** であるという。決定論的であっても、この系が予測可能ということではない。

以下、この式の振る舞いを調べることにする。もし、個体数がある数に落ち着いたら、$x(t+1) = x(t)$ である。したがって、

$$x = rx(1 - x) \tag{9}$$

より、定常状態の解があるとすれば[2]

$$x = \begin{cases} 0 \\ \dfrac{r-1}{r} \end{cases} \tag{10}$$

となる。

3.1.4 課題 2

3.4.1 節のプログラムを実行し、パラメタ r によって、x の値がどのように収束していくか調べよ。また、初期値を $0 < x < 1$ の範囲で変えて実行せよ。

プログラムを logic1.cc という名前で保存した後、r の値を少なくとも 0.5、1.5、2.5 の三つの値について結果を保存し、gnuplot で表示せよ。結果を定常解、$x = (r-1)/r$ と比べよ。

[1] この式が厳密には式 (6) の離散化ではない事に注意せよ。
[2] 定常状態があるかどうかは非自明である。コラム参照のこと。

3.1.5 課題 3

課題 2 では一度に一つの r の値についてしか計算できなかった。そこで、r の値をいろいろ変えて、収束後の x の値を表示するプログラムを組むことにする。具体的には 3.4.2 節のプログラムを入力し、logic2.cc などの名前で保存してコンパイルした後、

```
% ./a.out > R1-3.dat
```

として結果を保存し、

```
% gnuplot      ← gnuplot 起動
% plot "R1-3.dat", (x-1)/x
```

として定常解と一致するか調べよ。

3.1.6 課題 4

課題 3 のプログラムで、`rs = 1`、`re = 3` であった。これを `rs = 3`、`re = 4` として同様に結果を表示し、定常解と一致するか調べよ。

3.1.7 課題 5

課題 2 のプログラムで、$r = 3.6$ とする。$x = 0.1$ の場合の結果を `1.dat` に、$x = 0.10001$ の場合の結果を `2.dat` に保存し、gnuplot で両方を表示して、どのようになるか考察せよ。なお、このような性質は**初期値鋭敏性**と呼ばれ、カオスの重要な性質である。

3.2 ローレンツアトラクタ

カオスにはいくつか特徴的な性質がある。そのうち、軌道不安定性 (初期値鋭敏性) を前回調べた。今回はカオスのもうひとつの特徴である構造安定性、いわゆる**アトラクタ (attractor)** を見ることにしよう。

有名なアトラクタとして、ローレンツアトラクタ (Lorentz's Attractor) がある。これは、次の三変数の連立微分方程式の解として現れる。

$$\left\{ \begin{array}{lcl} \dot{x} &=& -s(x-y) \\ \dot{y} &=& -y - xz + rx \\ \dot{z} &=& xy - bz \end{array} \right. \tag{14}$$

余談コラム 3

問題を解くには、いろんな条件 (たとえば境界条件) が必要となる。しかし、もっとも基本的な条件として**解が存在する**という条件を仮定していることを忘れてはならない。世の中には解が存在しない方程式も存在し、それを解の存在を仮定して解くと誤った答えが出る。たとえば、$\sum_{n=0}^{\infty}(-1)^n$ という数列を考えよう。

$$\begin{aligned} I &= \sum_{n=0}^{\infty}(-1)^n = 1-1+1-1+1-1+1\cdots && (11) \\ &= (1-1)+(1-1)+(1-1)+(1-1)+\cdots && (12) \\ &= 1-(1-1)-(1-1)-(1-1)+\cdots && (13) \end{aligned}$$

括弧内はいずれもゼロであるから、式 (12) より $I = 0$、式 (13) より $I = 1$ である。したがって、$0 = 1$ が結論された。このように、実際には存在しない解を仮定すると、矛盾した結論が導かれる。特に非線形方程式では解の存在は自明ではないので注意したい。

ただし、$\dot{x}$ は、温度微分 $\mathrm{d}x/\mathrm{d}t$ をあらわす。通常、$s = 10, r = 28, b = 8/3$ とする。この方程式は、もともと重力下における二次元流体方程式を簡略化したものであり、対流現象を記述する方程式系である。x は対流の強さ、y は系に生じる温度差、z は温度プロファイルの線形からのずれをあらわす。

3.2.1 課題 1

以下のソースコードを参照し、連立微分方程式 (14) を数値的に解くプログラムを完成せよ。

```
#include <stdio.h>
#include <stdlib.h>

const int LOOP = 1000;
//ここで、必要であれば他の定数を宣言せよ

int
main(void){
  double x,y,z;
  //初期値を代入
  x = 1;
  y = 1;
  z = 1;
  for(int i=0;i<LOOP;i++){
    //ここを埋めよ
    printf("%f␣%f␣%f\n",x,y,z);
  }
}
```

3.2.2 課題 2

プログラムが完成したら、実行結果を gnuplot で観察せよ。ただし、三次元の結果をプロットするためには、コマンド”splot”を用いる。

```
% g++ lorentz.cc              ←コンパイル
% ./a.out > result.dat        ←実行結果を保存
% gnuplot                     ← gnuplot 起動
gnuplot> splot "result.dat    ←結果を三次元表示
```

また、`printf("%f %f %f\n",x,y,z);` の部分を `printf("%f\n",x);` と変えて、x の値のみを出力するように変更せよ。出力結果を `x.dat` に保存してプロットし、その振る舞いについて考察せよ。

3.2.3 課題 3

(x, y, z) の初期値を適当に変えて、どうなるか観察せよ。たとえば、$(60, 1, 1)$ と $(-60, 1, 1)$ ではどうなるか？

3.3 マンデルブロ集合

3.3.1 写像

ロジスティック方程式では、ある変数 x_n にたいして、次の変数 x_{n+1} を対応させるような関数 f を考えた。すなわち、

$$x_{n+1} = f(x_n) \tag{15}$$

であり、前回の場合では、関数 f として $f(x) = rx(1-x)$ を考えた。変数 x の取りうる値は $0 < x < 1$ の範囲の実数であるので、$x \in \mathbf{R}$ である[3]。このように、ある値と別の値を結びつけるような操作を、一般に**写像 (map)** という。今回の写像は実数から実数への写像であるので、$f : \mathbf{R} \to \mathbf{R}$ と表記する。

さて、ロジスティック方程式では一次元の写像を考えたが、今回は二次元の写像を扱うことにしよう。二次元の世界を表現するために、実数ではなく複素数 $z = x + iy$ を変数とする。複素数から複素数へ写す写像 f は $f : \mathbf{C} \to \mathbf{C}$ と表現される。ある複素数 $c = a + bi$ について、$z_0 = 0$ を初期値として f として以下の操作を考える。

$$z_{n+1} = f(z_n) \tag{16}$$
$$= z_n^2 + c \tag{17}$$

この計算を繰り返した時、z_n が無限大に発散しない c の集合を**マンデルブロ集合 (Mandelbrot Set)** と呼ぶ[4]。この集合はガウス平面で複雑な形をとるが、それを数値計算で表示することを考えよう。

無限大まで計算するわけにはいかないので、絶対値がある値以上になったら発散したとみなし、そこで計算を打ち切ることにする。したがって計算の手順は以下のとおり。

1. $c = a + ib$
2. $x \leftarrow 0$,$y \leftarrow 0$
3. $\begin{cases} x \leftarrow x^2 - y^2 + a \\ y \leftarrow 2xy + b \end{cases}$
4. $x^2 + y^2 > 4$ か？もしそうならここで停止。そうでなければ 3 へ。

上記の手順を、様々な c の値について計算し、発散しなければその c の値に対応する複素平面上の点を表示する。マンデルブロ集合の描画はこれでよいが、よりきれいな図を得るために $|z_n| > 2$ となるまでに反復した回数を使って濃淡を作成する。

3.3.2 課題 1

3.4.3 節に掲載したソースコードを打ち込み、コンパイル、実行せよ。ファイルは”mandelbrot.java”という名前で保存し、

```
% javac mandelbrot.java     ←コンパイル
% java  mandelbrot          ←実行
```

として実行する。

余談コラム 4

ローレンツ方程式のところで、x の時間微分を $\dot{x}$ であらわした。このような表記方法はニュートン (Newton) によって導入されたものである。現在使われている微分の表記法 $\mathrm{d}x/\mathrm{d}t$ は、ライプニッツにより導入された。こちらの方が、より直感的に物理量の次元がわかるなどの利点があり、また偏微分なども定義しやすい。ニュートンとライプニッツは、どちらも微積分法を確立した科学者であるが、どちらが微積分法を発明したかでかなり確執があったらしい。こういった話は「数学を作った人々(E. T. Bell 著、田中 勇、銀林 浩訳、ハヤカワ文庫)」などに書いてあるので興味があったら読んでみると良いだろう。数学という、一種無味乾燥とも思える学問の構築の裏に様々な人間ドラマが存在することを知れば、数学を勉強する面白さが増えるに違いない。

[3] 良く使う無限集合の記号としては実数全体の集合 $\mathbf{R}$ をはじめ、自然数全体 $\mathbf{N}$、整数全体 $\mathbf{Z}$、有理数全体 $\mathbf{Q}$、複素数全体 $\mathbf{C}$ などがある。

[4] ここでは初期値を固定して、パラメータ c についての集合を定義したが、逆にパラメータ c を固定して、初期値についての集合を考えることもできる。この集合を**ジュリア集合 (Julia Set)** と呼ぶ。

3.3.3 課題 2

打ち込んだプログラムは、(X_s, Y_s) と (X_e, Y_e) を対角線とする長方形の領域を表示するプログラムである。この範囲は、プログラムの冒頭で

```
static final double XS = -2.5;
static final double YS = -2;
static final double XE = 1.5;
static final double YE = 2;
```

として宣言してある。この範囲をいろいろ変えてみよ。たとえば、$(-0.62, -0.7)$ から $(-0.42, -0.5)$ の範囲では何が見えるか。いろんな場所をためし、一番自分の気に入った場所をメールで報告すること。

3.3.4 応用課題

余力のある人は以下のことを試せ。このソースコードは、初期値を $z = 0 + 0i$ に固定し、定数 $c = a + bi$ を変化させて、その収束を調べた。これを、定数 $c = a + bi$ を固定し、初期値をいろいろ変えてプロットすると**ジュリア集合 (Julia Set)** と呼ばれる集合となる。ソースコードを修整し、ジュリア集合を表示するプログラムにせよ。範囲は、まずは

```
static final double XS = -1;
static final double YS = -1;
static final double XE = 1;
static final double YE = 1;
```

を、定数は $c = -0.9 + 0i$、すなわち $a = -0.9, b = 0$ のあたりから試すと良いだろう。

3.4 ソースコード

3.4.1 ロジスティック方程式１

```
#include <stdio.h>

const int LOOP = 1000;

int
main(void){
  double x = 0.1;
  double r = 0.5;

  for(int i=0;i<LOOP;i++){
    x = r*x*(1-x);
    printf("%f\n",x);
  }

}
```

3.4.2 ロジスティック方程式２

```
#include <stdio.h>

const int LOOP = 1000; //繰り返しの数
const int LOOP2 = 900; //どこから表示するか
const double rs = 1.0; //パラメタの範囲
const double re = 3.0; //(rs< r < re)
const int N = 1000; //分割数

int
main(void){
  for(int j=0;j<N;j++){
    double r = (re-rs)*(double)j/(double)N + rs;
    double x = 0.1;
    for(int i=0;i<LOOP;i++){
      x = r*x*(1-x);
      if(i > LOOP2){ //ある程度収束したと思われるところだけ表示
        printf("%f␣%f␣\n",r,x);
        }
    }
  }
}
```

3.4.3 マンデルブロ集合 (Java)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
//-----------------------------------------------------------------
public class mandelbrot extends JFrame{

// 描画に必要な情報
  static final int N = 400; //ウィンドウの大きさ
  static final int MAX = 256; //繰り返しの数

// 描画範囲

  static final double XS = -2.5;
  static final double YS = -2;
  static final double XE = 1.5;
  static final double YE = 2;

  static final int WIDTH = N;
  static final int HEIGHT = N;
//-----------------------------------------------------------------
  Image offImage; //Background Surface
  Graphics offg; //Graphics of BG Surface
//-----------------------------------------------------------------
  public mandelbrot(String s){
    super(s);
  }
//-----------------------------------------------------------------
// 計算ルーチン
//-----------------------------------------------------------------
  int calculate(double a, double b){
    double x = 0;
    double y = 0;
    for (int i=0;i<MAX;i++){
      double nx = x*x - y*y + a;
      double ny = 2*x*y + b;
      if(nx*nx+ny*ny > 4)return i;
      x = nx;
      y = ny;
    }
    return MAX-1;
  }
//-----------------------------------------------------------------
// 描画ルーチン
//-----------------------------------------------------------------
  void drawall(){
    for(int i=0;i<WIDTH;i++){
      for(int j=0;j<HEIGHT;j++){
        double x = (XE-XS)/(double)N*(double)i + XS;
        double y = (YE-YS)/(double)N*(double)j + YS;
        int c = calculate(x,y);
        if(c==MAX-1)c=255;
        drawpoint(i,j, new Color(c,c,c));
      }
      paint(getGraphics());
    }
  }
//-----------------------------------------------------------------
  public void paint(Graphics g){
    if(offImage==null){
      offImage = createImage(WIDTH,HEIGHT);
      offg = offImage.getGraphics();
      offg.setColor(Color.black);
      offg.fillRect(0,0,WIDTH,HEIGHT);
```

```
64        return;
65      }
66      g.drawImage(offImage,0,0,this);
67    }
68  //---------------------------------------------------------------
69    void drawpoint(int x, int y, java.awt.Color c){
70      int x1 = x;
71      int y1 = y;
72      int x2 = x + 1;
73      int y2 = y + 1;
74      offg.setColor(c);
75      offg.fillRect(x1,y1,x2,y2);
76    }
77  //---------------------------------------------------------------
78  // 描画準備ができるまで待つ
79    void waitImage(){
80      while(null==offImage){
81        try{
82          Thread.sleep(10);
83        }catch(InterruptedException e){
84          System.out.println(e);
85        }
86      }
87    }
88  //---------------------------------------------------------------
89    public static void main(String arg[]){
90      mandelbrot f = new mandelbrot("Mandelbrot␣Set");
91      f.setSize(WIDTH,HEIGHT);
92      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
93      f.show();
94      f.waitImage();
95      f.drawall();
96    }
97  }//End of class mandelbrot
```

3.4.4 マンデルブロ集合 (Java) の解説

前節のソースコードが何をやっているか、少し解説しておこう。一般に、プログラムで何か絵を描くのは難しい。特に、ウィンドウがひらいて、その中に描画する、というプログラムは、たとえばウィンドウが隠れてから再表示されたら書き直さなければならないし、右上の「×」印がクリックされたらウィンドウを閉じなければならないなど、面倒をみなければならないことが多い。それらを最低限実現したのが、前節のソースコードである。

まず、ウィンドウプログラムを書くためには、`public void paint(Graphics g)` という関数を実装しなくてはならない。この関数は、ウィンドウの再描画の必要が生じたときに OS によって呼ばれる関数であり、ここに描画ルーチンを書く。しかし、ここで毎回描画していると、一度画面をクリアして、そこから描画をしようとするため、画面がちらつくことになる。これを防ぐために、一度内部で描くべき内容を別のバッファに描画しておき、そのバッファをコピーすることでちらつきを防ぐ。このような表示法はダブルバッファリングと呼ばれ、ウィンドウプログラミングにおける描画の基本技術である。前節のコードでは、`Image offImage` に描画しておき、必要なときにその内容を表にコピーすることでダブルバッファリングを実現しているが、Java にはダブルバッファの面倒を見るクラスもあるので、それを利用しても良い。

なお、今回は簡単のためにグレースケール、すなわち白黒でのみ描画したが、もちろんカラーで描画することもできる。カラーで描画するためには、描画ルーチン `void drawall()` の中で、

```
drawpoint(i,j, new Color(c,c,c));
```

の場所を変更すればよい。`Color(int, int, int)` の中は、それぞれ三原色 (RGB) を 256 段階で指定することができる。現在はすべて同じ重みにしてあるのでグレースケールとなっているが、たとえば赤のみにすることも、もしくは繰り返しの数によって色が変わる、などと変更することもできるので、いろいろ試してみると良い。

3.5 参考資料

授業の参考資料として、以下をあげておく。これらは直接課題の解答となるものではないが、より詳しく学びたいときに役に立つことだろう。

数学的な内容としては、Wolfram (Mathematica の作者) のサイト、Math World (http://mathworld.wolfram.com/) がすばらしい。数学に関して、基礎的な内容はほとんど載っている。もちろん英語だが、ブックマークの価値があるサイトである。たとえば、ローレンツアトラクタに関しては http://mathworld.wolfram.com/LorenzAttractor.html に記述がある。

カオスの入門書としては、ジェイムズ・グリックの「カオス－新しい科学をつくる (新潮文庫)」と、ミッチェル・ワードロップ「複雑系 (新潮社)」が良い。特に前者は、カオスという新しい科学を切り開いていった研究者たちの活躍を生き生きと描いており、単純に読み物として面白い。

3.6 出席及び課題の提出について

授業終了時に、学生番号、名前、課題の回答、感想などをまとめ、hwatanabe@is.nagoya-u.ac.jp までメールにて提出すること。その際、メールの題名 (subject) は、「実験レポート (日付) 学籍番号」とせよ。たとえば、2007 年 5 月 28 日、学籍番号が 050500000 なら、「実験レポート (070528) 050500000」とせよ。感想や改善の指摘等を歓迎する。