カオス・フラクタル 講義ノート #6

担当：井上 純一 (情報科学研究科棟 8-13)

URL：http://chaosweb.complex.eng.hokudai.ac.jp/˜j _inoue/index.html

平成 21 年 6 月 2 日

目 次

まずは前回の復習も兼ねて, レポート課題であったニュートン法の素朴なソースコードを載せておこう.

```
                        /* ニュートン法 */
#include <stdio.h>
#include <math.h>
double func(x,a) /* f(x)=0 を求める. f(x) の定義 */
double x;
double a;
{
  return (pow(a*x,2)-a*(a+1)*x+a+1);
}
double Dfunc(x,a) /* f(x) の 1 階微分の定義. 微分が困難な場合には数値微分値を返すようにする. */
double x;
double a;
{
  return (2*a*a*x-a*(a+1));
}
main()
{
    FILE *fpr;
    int i,imax=100;
    double x,y,a;
     if((fpr = fopen("lower.dat", "wt")) !=NULL){
       for(a=3.0; a<=1.0+sqrt(6.0); a=a+0.01){  /* a の値を振る */
```

```
      for(i = 0, x=0.10; i <= imax; i++){
 x = x - func(x,a)/Dfunc(x,a);
 if(fabs(y-x)<1.0e-5){
            fprintf(fpr,"%lf %lf\n",a,x);
     break;
 }else{
         y = x;
 }
       }}
     }
     fclose(fpr);
}
```

このプログラムでは, a の値に対して解は 2 個しかないことがわかっているので, その 2 つのうち大きい方の解を求めたい場合には x の取りうる上限値に近いところから反復をスタートさせ, 逆に, 小さい方の解を求めたい場合には x の取りうる下限値に近いところから反復をスタートさせればよい. ちなみに, a を少しずつ変化させてその解を求めているが, a を少し変化させても x の値が急激に変化しないようであれば, 一つ手前の a での解 x の値を a に関するループで次の a に対する反復の初期値 x として利用し, この間, 方程式が「断熱的」に振舞うと仮定してプログラムを書いてもよい. 具体的には次のように a のループの部分で x の値を一回だけ初期化すればよい.

```
for(a=3.0,x=0.10; a<=1.0+sqrt(6.0); a=a+0.01){  /* a の値を振る */
   for(i = 0; i <= imax; i++){
```

このプログラムからの出力を解析解とともにプロットしたものを図 24 に載せよう.

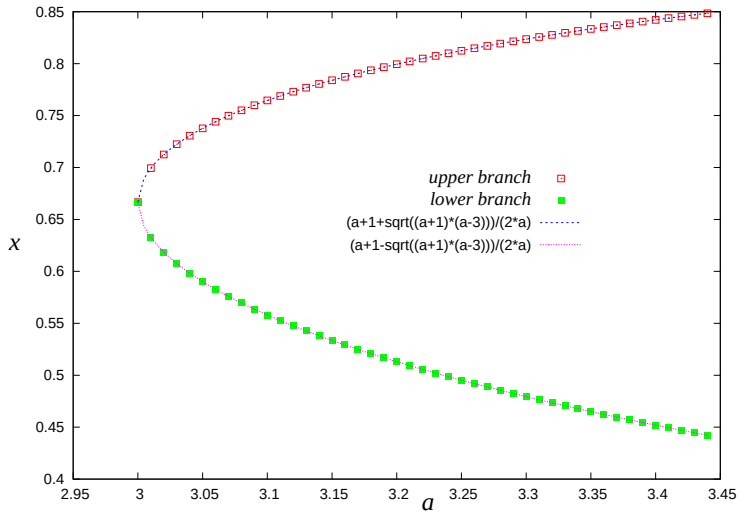

図 24: ニュートン法に基づく数値解と 2 次方程式の解公式による解析解.

7　数値計算の準備: 常微分方程式の数値解法

今までこの講義では 1 次元写像の振る舞いを詳しくみてきた. 例えば, ロジスティック写像は次のロジスティック方程式 (第 1 回講義ノート p.2 (9) 式):

$$\frac{dN}{dt} = \lambda N(1-\mu N) \tag{112}$$

を差分化した方程式であり, 本来解くべき方程式の「近似」である. しかし, 現実の自然現象は運動方程式など微分方程式で記述されるものが多く, 従って, これらの現象を正確に調べようとすれば, 差分化された近似式ではなく, 微分方程式そのものを扱わなければならない. いくつかの常微分方程式は解析的に解くことができるが, 非線形項が入り, しかも, 変数が複数絡み合った連立微分方程式は多くの場合に手で解くことはできない. このような手で解くことのできない方程式の多くはカオスなどの複雑で多彩な挙動を示し, 従って, そのような現象の理解を深めることを目的の一つとする本講義では, そのような方程式を解くことを避けて通ることはできない (逆に言えば, 手できれいに解ける方程式は本講義の対象とならない). そこで, 今回はやや本題を外れることになるが, 複雑な挙動を示す常微分方程式を解析するためのほぼ唯一の手段になる数値計算技法についてみていく (一部の人たちにとっては復習になるかもしれないが).

具体的に我々が解こうとする方程式の典型例は次のようなものである.

$$\frac{dx}{dt} = -y - z \tag{113}$$

$$\frac{dy}{dt} = x + ay \tag{114}$$

$$\frac{dz}{dt} = b + z(x-c) \tag{115}$$

この連立微分方程式は**レスラー方程式**と呼ばれる. ここに現れるパラメータ a, b, c を適切に選んだ際, 我々が 1 次元写像でみたようなカオスが生じるかどうか, どのような力学的性質を持つかなどを調べることがここからの目標である. 今回の講義ではそのための準備を行う.

7.1　テーラー展開による漸化式の導出

話を簡単にするために次の 1 変数の微分方程式:

$$\frac{dy}{dt} = f(t,y) \tag{116}$$

を考えよう. ただし, 関数 f は微分可能な関数であるとする. ここで h を十分小さい量とし

$$t_n = t_0 + nh, \quad n = 1, 2, 3, \cdots \tag{117}$$

で「時間」t を離散化すると (t_0 は時間原点), $t_{n+1} = t_0 + (n+1)h$ であるから, $t_{n+1} - t_n = h \ll 1$ となることに注意しょう (あるいは $t_{n+1} = t_n + h$). このとき, 小さな量 h に関するテーラー展開を思い出すと

$$\begin{aligned} y(t_{n+1}) &= y(t_n + h) = y(t_n) + \left.\frac{dy}{dt}\right|_{t_n, y_n} h + \mathcal{O}(h^2) \\ &= y(t_n) + f(t_n, y_n)h + \mathcal{O}(h^2) \end{aligned} \tag{118}$$

となる. 従って, $y(t_n) = y_n, y(t_{n+1}) = y_{n+1}$ と略記することを約束すれば

$$y_{n+1} = y_n + f(t_n, y_n)h + \mathcal{O}(h^2) \tag{119}$$

$$t_n = t_0 + nh \tag{120}$$

が得られる. この方程式 (119) を漸化式として解き, 各 n で (120) 式で与えられる t_n を y_n に対する時刻とみなすことで微分方程式を数値的に解く方法を**オイラー法**と呼ぶ. 上記から明らかなように, 各ステップでの所謂「局所誤差」は h^2 のオーダである ($\mathcal{O}(h^2)$ と表記した部分).

7.2 ルンゲ・クッタ法: 基本的アイデア

上記のテーラー展開で微分係数 $dy/dt = f(t, y)$ を評価する点は (t_n, y_n) であったが, これを (t_n, y_n) と (t_{n+1}, y_{n+1}) の中点である $(t_{n+1/2}, y_{n+1/2})$ に選んだ場合, 上記のテーラー展開のプロセスと結果として得られる漸化式はそれぞれの区間でどうなるかを考えてみよう.

まず, (117) 式から $t_{n+1/2} = t_0 + (n + 1/2)h$ であるから

$$t_{n+1} \quad = \quad t_{n+1/2} + h/2, \quad t_n = t_{n+1/2} - h/2 \tag{121}$$

であることに注意して

$$\begin{aligned} y_{n+1} = y(t_{n+1}) &= y(t_{n+1/2} + h/2) \\ &= y_{n+1/2} + f(t_{n+1/2}, y_{n+1/2}) \left(\frac{h}{2}\right) + \frac{1}{2} f^{'}(t_{n+1/2}, y_{n+1/2}) \left(\frac{h}{2}\right)^2 + \mathcal{O}(h^3) \end{aligned} \tag{122}$$

および

$$\begin{aligned} y_n = y(t_n) &= y(t_{n+1/2} - h/2) \\ &= y_{n+1/2} - f(t_{n+1/2}, y_{n+1/2}) \left(\frac{h}{2}\right) + \frac{1}{2} f^{'}(t_{n+1/2}, y_{n+1/2}) \left(\frac{h}{2}\right)^2 + \mathcal{O}(h^3) \end{aligned} \tag{123}$$

が得られる. ここで, $f^{'} \equiv df/dt$ と略記したことに注意されたい. 従って, これらの方程式 (122)(123) を辺々引くと

$$y_{n+1} - y_n \quad = \quad f(t_{n+1/2}, y_{n+1/2})h + \mathcal{O}(h^3) \tag{124}$$

となり

$$y_{n+1/2} \quad = \quad y(t_{n+1/2}) = y(t_n + h/2) = y(t_n, y_n) + \frac{h}{2} f(t_n, y_n) \tag{125}$$

であるから, 問題の微分方程式 (116) を

$$y_{n+1} \quad = \quad y_n + k_2 + \mathcal{O}(h^3) \tag{126}$$

$$k_1 \quad = \quad h f(t_n, y_n) \tag{127}$$

$$k_2 \quad = \quad h f(t_n + h/2, y_n + k_1/2) \tag{128}$$

$$t_n \quad = \quad t_0 + nh \tag{129}$$

の漸化式で書き直すことで, ステップごとの局所誤差をオイラー法での $\mathcal{O}(h^2)$ から $\mathcal{O}(h^3)$ へまで低減させることができることがわかる[1]. これら (126)(127)(128)(129) 式を 2 次の**ルンゲ・クッタ法**と呼ぶ.

このような方法でうまく中継点を選び, 上記の手続きを繰り返すことにより, 局所誤差を系統的に減らしていくことができることは容易に推測できるであろう. 導出が煩雑なので詳細は省略するが, 局所誤差が

[1] h は十分に小さな量であったことを思い出す. 例えば $h = 0.01$ などとして, $\mathcal{O}(h^2)$ と $\mathcal{O}(h^3)$ を比較してみるとよい.

$\mathcal{O}(h^5)$ まで低減できるものとして, 次の 4 次のルンゲ・クッタ法が知られており, 多くの場面で用いられている (本講義でも数値計算では専らこの 4 次のルンゲ・クッタ法を用いる).

$$y_{n+1} = y_n + \frac{(k_1 + 2k_2 + 2k_3 + k_4)}{6} + \mathcal{O}(h^5) \tag{130}$$
$$k_1 = hf(t_n, y_n) \tag{131}$$
$$k_2 = hf(t_n + h/2, y_n + k_1/2) \tag{132}$$
$$k_3 = hf(t_n + h/2, y_n + k_2/2) \tag{133}$$
$$k_4 = hf(t_n + h, y_n + k_3) \tag{134}$$
$$t_n = t_0 + nh \tag{135}$$

次で具体的なコーディングについてみていこう.

7.2.1 適用例 1: ロジスティック方程式

以上が一般論であるが, 感じをつかむためにいくつかの具体的な方程式への適用例を見ておこう.
まずは最も簡単な例の一つとして, 上記の 4 次のルンゲ・クッタ法 (130)-(135) でロジスティック方程式:

$$\frac{dN}{dt} = \lambda N(1 - \mu N) \tag{136}$$

を数値的に解いてみる. その結果を図 25 に載せよう. この図より, 解析解と極めて良好な一致を見せていることがわかる[2] . また, そのプログラムを載せる.

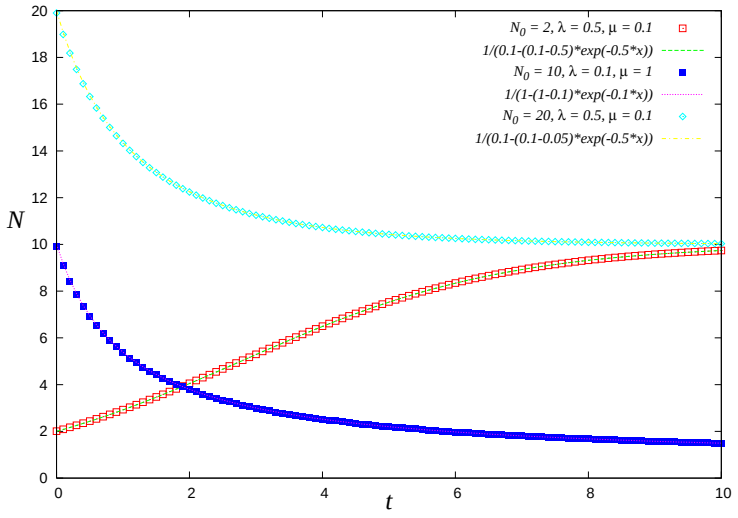

図 25: ロジスティック方程式の 4 次のルンゲ・クッタ法に基づく数値解と解析解.

```
/* ルンゲ・クッタ法のロジスティック方程式への適用例 */
#include <stdio.h>
#include <math.h>
#define h 0.01
```

[2] 大域誤差 (真の解 $y(t)$ と y_n の差) についての評価は後の演習などで触れたいと思う.

```
#define mu 0.1
#define lambda 0.5
double func(y)
double y;
{return (lambda*y*(1.0-mu*y));}  /* 関数 f の定義 */
main()
{
    FILE *fpr;
    int i,imax=1000;
    double y,k1,k2,k3,k4,k;
if((fpr = fopen("rk4.dat", "wt")) !=NULL){
       for(i = 0,y=20.0; i <= imax; i++){
          k1 = h*func(y);
  k2 = h*func(y+0.5*k1);
  k3 = h*func(y+0.5*k2);
  k4 = h*func(y+k3);
  k = (k1+2.0*k2+2.0*k3+k4)/6.0;
  y = y + k;
  fprintf(fpr,"%lf %lf\n",i*h,y);
   }
}
     fclose(fpr);
}
```

7.2.2 適用例 2: レスラー方程式

変数が 3 つに増えたレスラー方程式 (113)(114)(115) も上記 1 変数の場合のプログラムを拡張することで容易に解くことができる. 参考までにプログラミングのソースコードを載せておく.

```
                /* ルンゲ・クッタ法のレスラー方程式への適用例 */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define h 0.01
#define a 0.5
#define b 0.5
#define c 5.7
double func1(x,y,z) /* 関数その 1 */
double x;
double y;
double z;
{return (-(y+z));}
double func2(x,y,z) /* 関数その 2 */
double x;
double y;
```

```
double z;
{return (x+a*y);}
double func3(x,y,z) /* 関数その 3 */
double x;
double y;
double z;
{return (b+z*(x-c));}

main()
{
    FILE *fpr;
    int i,imax=50000;
    double x,k1x,k2x,k3x,k4x,kx,y,k1y,k2y,k3y,k4y,ky,z,k1z,k2z,k3z,k4z,kz;
if((fpr = fopen("rossler.dat", "wt")) !=NULL){
       for(i = 0,x=-0.01,y=-0.01,z=0.01; i <= imax; i++){
          k1x = h*func1(x,y,z);
          k1y = h*func2(x,y,z);
          k1z = h*func3(x,y,z);
             k2x = h*func1(x+0.5*k1x,y+0.5*k1y,z+0.5*k1z);
             k2y = h*func2(x+0.5*k1x,y+0.5*k1y,z+0.5*k1z);
             k2z = h*func2(x+0.5*k1x,y+0.5*k1y,z+0.5*k1z);
          k3x = h*func1(x+0.5*k2x,y+0.5*k2y,z+0.5*k2z);
          k3y = h*func2(x+0.5*k2x,y+0.5*k2y,z+0.5*k2z);
          k3z = h*func3(x+0.5*k2x,y+0.5*k2y,z+0.5*k2z);
             k4x = h*func1(x+k3x,y+k3y,z+k3z);
             k4y = h*func2(x+k3x,y+k3y,z+k3z);
             k4z = h*func3(x+k3x,y+k3y,z+k3z);
          kx = (k1x+2.0*k2x+2.0*k3x+k4x)/6.0;
          ky = (k1y+2.0*k2y+2.0*k3y+k4y)/6.0;
          kz = (k1z+2.0*k2z+2.0*k3z+k4z)/6.0;
             x = x + kx;
             y = y + ky;
             z = z + kz;
fprintf(fpr,"%lf %lf %lf %lf\n",i*h,x,y,z);
   }
}
     fclose(fpr);
}
```

上記のプログラムを用いて, まずは, 3 つの変数 x, y, z の時間変化をプロットすると図 26 のようになる. この図 26 より, 1 次元写像に見たような複雑な挙動を x, y, z のそれぞれが示すことがわかる. この図 26 を見方を変えて, xyz の 3 次元空間内にその軌道をプロットすると興味深い. これは上記プログラムでファイルに書き出されたデータの第 2 軸を x 軸, 第 3 軸を y 軸, 第 4 軸を z 軸に選んだプロットに相当する.

ちなみに, 「計算機プログラミング I・同演習」や実験などで gnuplot を学んだものは gnuplot を起動したのちに

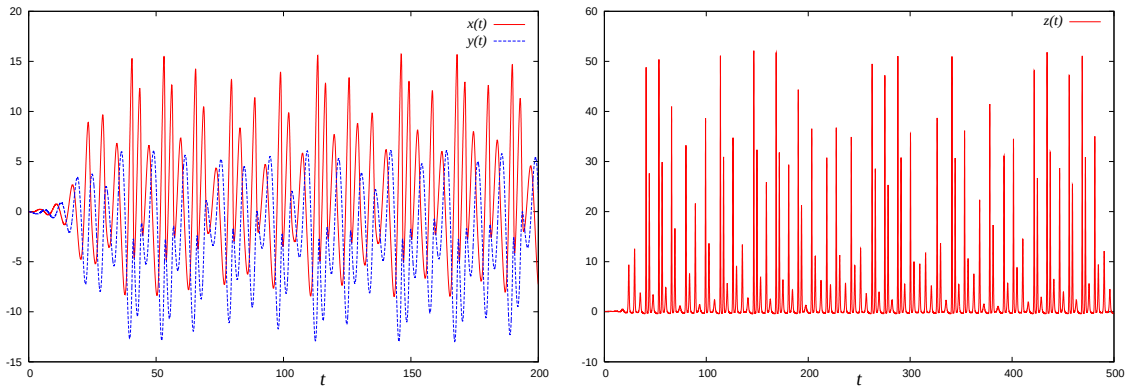

図 26: レスラー方程式の 4 次のルンゲ・クッタ法に基づく数値解. $x(t), y(t)$ および $z(t)$ の振る舞い.

```
gnuplot> set parametric
gnuplot> splot 'lossler.dat' u 2:3:4 w line
```

とすればよい (先頭の gnuplot> は gnuplot のカーソルであることに注意). このようにして 3 次元プロットした結果を図 27 に載せる[3] . 軌道の詳細を調べるため, 図 27 の 3 次元プロットを x-y 平面, x-z 平面に射影して再プロットしたものが図 28 である.

以上で本講義の [カオス編] 後半で必要な数値計算上の道具立ては全て揃ったことになる. これらの図 27,28 の意味するところは次回以降に詳しくみていくことにしよう.

レポート課題 6

次の x に関する 2 階の常微分方程式:

$$\frac{d^2x}{dt^2} - \epsilon(1-x^2)\frac{dx}{dt} + x = 0 \tag{137}$$

は速度を $dx/dt = v$ で定義すれば

$$\frac{dv}{dt} = \epsilon(1-x^2)v - x \tag{138}$$

$$\frac{dx}{dt} = v \tag{139}$$

と x, v に関する 1 階の連立微分方程式に書き直すことができ[4] , これは今回学んだルンゲ・クッタ法を用いて解くことができる. そのプログラムを書き (ソースコードの提出), いくつかの ϵ の値, 初期条件に対する x, v の振る舞いをプロットせよ (gnuplot などの作図ツールを知らないものはソースコードのみでよい).

[3] ここで出てきた gnuplot の使用法などは演習の時間に取り上げたいと思う.
[4] 多くの運動方程式はこの手の書き換えができることに注意しょう.

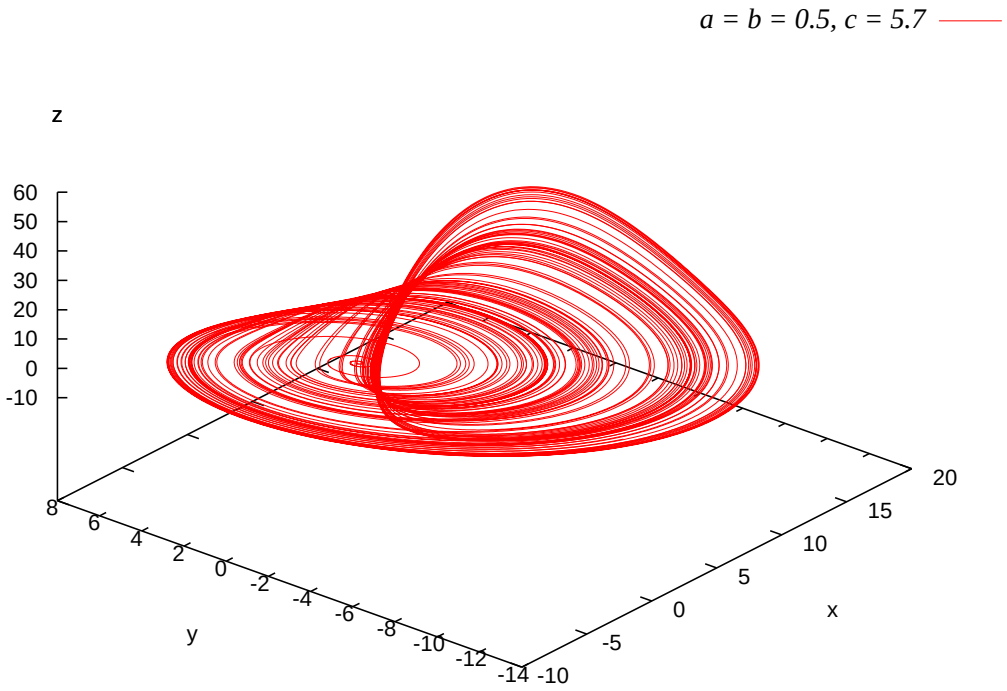

図 27: レスラー方程式の数値解から得られる軌道の様子. パラメータは $a = b = 0.5, c = 5.7$ に選んである.

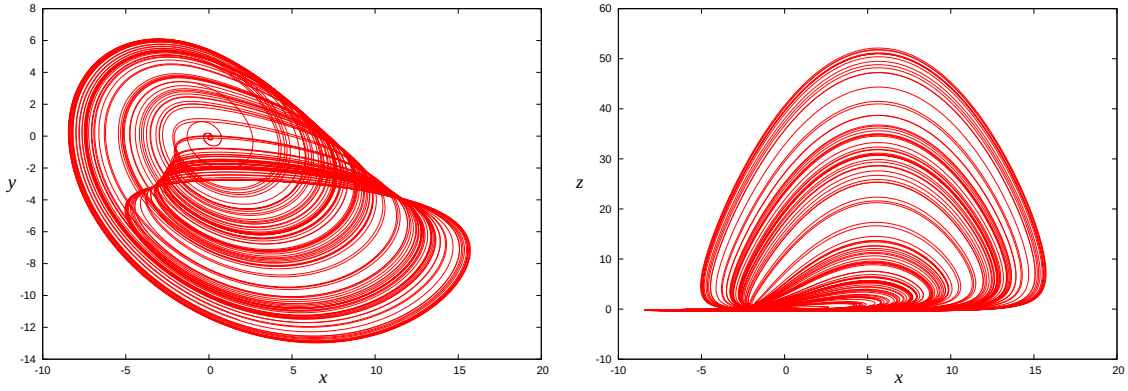

図 28: 図 27 の 3 次元プロットを x-y 平面, x-z 平面に射影してプロットしたもの