常微分方程式と連立方程式のまとめ (後期中間試験に向けて)

山本昌志*

2004 年 11 月 30 日

後期中間試験は、12 月 7 日 (火) の 6 時間目です。その日の授業は、5E の教室で実施します。

1 基本

1.1 テイラー展開

数値計算は言うに及ばず科学技術全般の考察にテイラー展開 (Taylor expansion) は、重要な役割を果たす。電気の諸問題を考察する場合、いたるところにテイラー展開は顔を出すので、十分理解しなくてはならない。まずは、$x = a$ の回りのテイラー展開は、

$$\begin{aligned} f(x) &= f(a) + f'(a)(x-a) + \frac{1}{2!}f''(a)(x-a)^2 + \frac{1}{3!}f'''(a)(x-a)^3 + \frac{1}{4!}f^{(4)}(a)(x-a)^4 + \cdots \\ &= \sum_{n=0}^{\infty} \frac{1}{n!} f^{(n)}(a)(x-a)^n \end{aligned} \tag{1}$$

と書ける。0 の階乗は $0! = 1$ となることに注意。$f^{(n)}(a)$ は、関数 $f(x)$ を n 回微分したときの $x = a$ の値である。テイラー展開は、任意の関数 $f(x)$ は、無限のべき級数に展開できると言っている。その係数が、$\frac{1}{n!}f^{(n)}(a)$ である。次に、$a = 0$ としてみる。この場合、

$$\begin{aligned} f(x) &= f(0) + f'(0)x + \frac{1}{2!}f''(0)x^2 + \frac{1}{3!}f'''(0)x^3 + \frac{1}{4!}f^{(4)}(0)x^4 + \cdots \\ &= \sum_{n=0}^{\infty} \frac{1}{n!} f^{(n)}(0)x^n \end{aligned} \tag{2}$$

となる。これがマクローリン展開 (Maclaurin expansion) である。次に $x - a = \Delta x$ とする。そして、$a = x_0$ とすると

$$\begin{aligned} f(x_0 + \Delta x) &= f(x_0) + f'(x_0)\Delta x + \frac{1}{2!}f''(x_0)\Delta x^2 + \frac{1}{3!}f'''(x_0)\Delta x^3 + \frac{1}{4!}f^{(4)}(x_0)\Delta x^4 + \cdots \\ &= \sum_{n=0}^{\infty} \frac{1}{n!} f^{(n)}(x_0)\Delta x^n \end{aligned} \tag{3}$$

*国立秋田工業高等専門学校　電気工学科

となる。これは、しばしばお目にかかるパターン。
通常我々がテイラー展開を使う場合、この式 (1)〜式 (3) のいずれかである。

1.2 連立一次方程式の表現

N 個の未知数があり、それが式の数が等し連立方程式を考える。この場合、連立方程式は

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1N}x_N &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2N}x_N &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3N}x_N &= b_3 \\ &\vdots \\ a_{N1}x_1 + a_{NN}x_2 + a_{N3}x_3 + \cdots + a_{NN}x_N &= b_N \end{aligned} \tag{4}$$

と書くのは中学校以来の習慣である。実際の問題を解く場合、これでは見通しが悪いので線形代数の知識を使う。この連立方程式の係数と未知数、非同次項を、

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1N} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2N} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3N} \\ \vdots & & & \ddots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & \cdots & a_{NN} \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_N \end{pmatrix} \tag{5}$$

のような行列やベクトルで表す。すると、連立方程式 (4) は、

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{6}$$

のように表現できる。

2 常微分方程式の数値計算法

数値計算により、近似解を求める微分方程式は、

$$\frac{dy}{dx} = f(x, y) \qquad \text{初期条件} \quad y(a) = b \tag{7}$$

である。これは問題として与えられ、この式に従う x と y の関係を計算する。

2.1 オイラー法

常微分方程式の解を $y = y(x)$ とする。その x_i のまわりのテイラー展開は、

$$y_{i+1} = y(x_i + \Delta x) = y(x_i) + \frac{dy}{dx}\Big|_{x=x_i} \Delta x + \frac{1}{2}\frac{d^2y}{dx^2}\Big|_{x=x_i} \Delta x^2 + \frac{1}{6}\frac{d^3y}{dx^3}\Big|_{x=x_i} \Delta x^3 + \ldots \tag{8}$$

である。この式の右辺第 2 項は、式 (7) から計算できる。したがって、テイラー展開は、次のように書き表わせる。

$$y_{i+1} = y_i + f(x_i, y_i)\Delta x + O(\Delta x^2) \tag{9}$$

オイラー法での数値計算では、計算の刻み幅 Δx は十分に小さいとして、

$$y_{i+1} = y_i + f(x_i, y_i)\Delta x \tag{10}$$

を計算する。この場合、計算の精度は 1 次と言う[1]。

オイラー法では、次の漸化式に従い数値計算を進める。解である $(x_0, y_0), (x_1, y_1), (x_2, y_2), \cdots$ が同じ手続きで計算できる。実際にプログラムを行うときは、*for* や *while* を用いて繰り返し計算を行い、結果の x_i と y_i は、配列 `x[i]` や `y[i]` に格納するのが常套手段である。

$$\begin{cases} x_0 = a \\ y_0 = b \\ x_{i+1} = x_i + \Delta x \\ y_{i+1} = y_i + f(x_i, y_i)\Delta x \end{cases} \tag{11}$$

この方法の計算のイメージは、図 1 の通りである。明らかに、出発点の導関数のみ利用しているために精度が悪いことが理解できる。式も対称でないため、逆から計算すると元に戻らない。

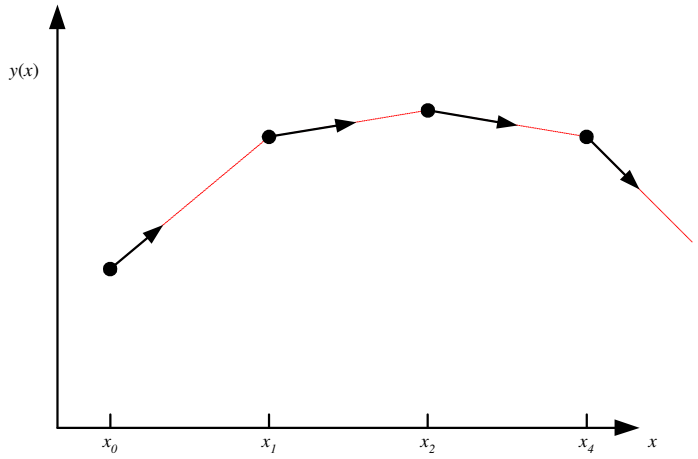

図 1: オイラー法。ある区間での y の変化 Δy は、計算の始めの点の傾きに区間の幅 Δx を乗じて、求めている。

2.2 2 次のルンゲクッタ法 (ホイン法)

2 次のルンゲ・クッタと呼ばれる方法は、いろいろある。ここでは、ホイン法をしめす。

[1]誤差項が $O(\Delta x^{n+1})$ のとき、方法は n 次の精度という慣わしです

先に示したように、オイラー法の精度は 1 次であるが、2 次のルンゲ・クッタ法では 2 次となる。今まで刻み幅を Δx と記述していたが、簡単のため h と表現する。

2 次の精度ということは、テイラー展開より

$$y(x_0+h) = y(x_0) + y'(x_0)h + \frac{1}{2}y''(x_0)h^2 + O(h^3) \tag{12}$$

となっていることを意味する。即ち、計算アルゴリズムが、

$$\Delta y = y'(x_0)h + \frac{1}{2}y''(x_0)h^2 + O(h^3) \tag{13}$$

となっている。

式 (13) から分かるように、y の増分 Δy を計算するためには、1 階微分と 2 階微分の 2 項を満たす式が必要である。そうすると少なくとも、2 点の値が必要となる。2 点として、計算区間の両端の導関数の値を使う。この導関数は問題として与えられているので、計算は簡単である。そうして、区間の増分を α, β をパラメーターとした和で表すことにする。即ち、以下の通りである。

$$\Delta y = h\{\alpha y'(x_0) + \beta y'(x_0+h)\} \tag{14}$$

この α, β を上手に選ぶことにより、式 (13) と同一にできる。この式を x_0 の回りでテイラー展開すると

$$\Delta y = (\alpha+\beta)y'(x_0)h + \beta y''(x_0)h^2 + O(h^3) \tag{15}$$

となる。これを、式 (13) と比較すると、

$$\begin{cases} \alpha = \dfrac{1}{2} \\ \beta = \dfrac{1}{2} \end{cases} \tag{16}$$

とすれば良いことが分かる。これで、必要な式は求まった。まとめると、式 (7) を数値計算で近似解を求めるには次式を使うことになる。

$$\begin{cases} k_1 = hf(x_n, y_n) \\ k_2 = hf(x_n+h, y_n+k_1) \\ y_{n+1} = y_n + \dfrac{1}{2}(k_1+k_2) \end{cases} \tag{17}$$

この式は、図 2 のようになる。オイラー法の図 1 との比較でも、精度が良いことが分かる。

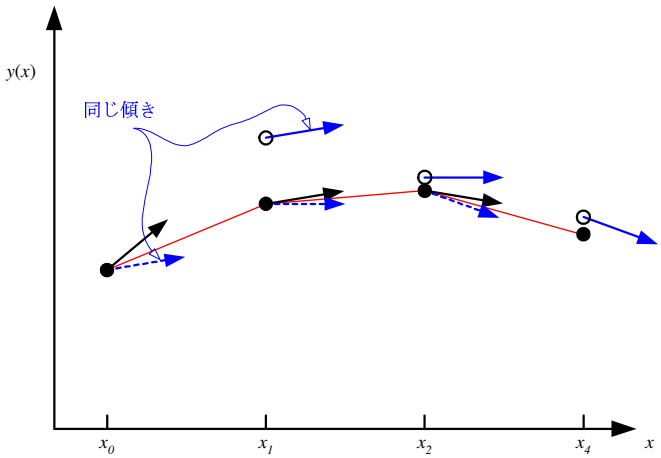

図 2: ホイン法 (教科書の方法)。ある区間での y の変化 Δy は、計算の始めと終わりの点付近の平均傾きに区間の幅 Δx を乗じて、求めている。

2.3 4 次のルンゲ・クッタ法

今まで示したオイラー法や 2 次のルンゲ・クッタ法のように、パラメーターを増やして誤差項の次数を上げていく方法で、最良の方法と言われるのが 4 次のルンゲ・クッタ法である。パラメーターを増やして、5, 6, 7, $\cdots$ と誤差項を小さくすることは可能であるが、同じ計算量であれば 4 次のルンゲ・クッタの刻み幅を小さくするほうが精度が良いと言われている。

ということで、皆さんが常微分方程式を計算する必要が生じたときは、何はともあれ 4 次のルンゲ・クッタで計算すべきである。普通の科学に携わる人にとって、4 次のルンゲ・クッタは常微分方程式の最初で最後の解法である。

4 次のルンゲ・クッタの公式は、式 (18) に示す通りである。そして、これは図 3 のように表すことができる。

$$\begin{cases} k_1 = hf(x_n, y_n) \\ k_2 = hf(x_n + \dfrac{h}{2}, y_n + \dfrac{k_1}{2}) \\ k_3 = hf(x_n + \dfrac{h}{2}, y_n + \dfrac{k_2}{2}) \\ k_4 = hf(x_n + h, y_n + k_3) \\ y_{n+1} = y_n + \dfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{cases} \tag{18}$$

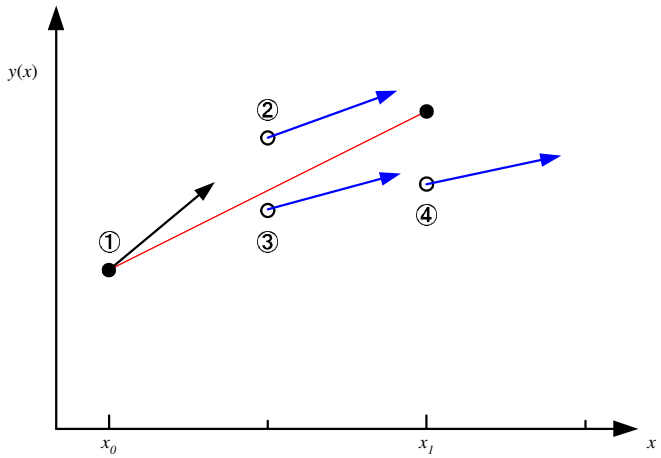

図 3: 4 次のルンゲ・クッタ法。ある区間での y の変化 Δy は、区間内の 4 点の傾きのある種の加重平均に幅 Δx を乗じて、求めている。

2.4 プログラム (4 次のルンゲ・クッタ法)

実際の微分方程式

$$\begin{cases} \dfrac{dy}{dx} = \sin x \cos x - y \cos x \\ y = 0 \quad (\text{初期条件 } x = 0 \text{ の時}) \end{cases} \tag{19}$$

を 4 次のルンゲ・クッタ法で計算するプログラムを示す。計算結果は、配列「x[]」と「y[]」に格納される。実際にプログラムでは、この結果を利用してグラフにしたりするのであるが、ここでは計算のみとする。

```
#include <stdio.h>
#include <math.h>
#define IMAX 100001
double func(double x, double y);

/*=====================================================================*/
/*       main function                                                 */
/*=====================================================================*/
int main(void){
  double x[IMAX], y[IMAX];
  double final_x, h;
  double k1, k2, k3, k4;
  int ncal, i;


  /*--- set initial condition and cal range ---*/

  x[0]=0.0;
  y[0]=0.0;
```

```
  final_x=10.0;
  ncal=10000;

  /* --- size of calculation step --- */

  h=(final_x-x[0])/ncal;


  /* --- 4th Runge Kutta Calculation --- */

  for(i=0; i < ncal; i++){
    k1=h*func(x[i],y[i]);
    k2=h*func(x[i]+h/2.0, y[i]+k1/2.0);
    k3=h*func(x[i]+h/2.0, y[i]+k2/2.0);
    k4=h*func(x[i]+h, y[i]+k3);

    x[i+1]=x[i]+h;
    y[i+1]=y[i]+1.0/6.0*(k1+2.0*k2+2.0*k3+k4);
  }


  return 0;
}


/*=====================================================================*/
/*       define function                                               */
/*=====================================================================*/
double func(double x, double y){
  double dydx;

  dydx=sin(x)*cos(x)-y*cos(x);

  return(dydx);
}
```

2.5 高階の常微分方程式

2.5.1 1 階の連立微分方程式に変換

ここまで示した方法は、1 階の常微分方程式しか取り扱えないので不便である。そこで、高階の常微分方程式を 1 階の連立微分方程式に直す方法を示す。要するに、高階の常微分方程式を連立 1 階常微分方程式に直し、4 次のルンゲ・クッタ法を適用すれば良いのである。例えば、次のような 3 次の常微分方程式があっ

たする。

$$y'''(x) = f(x, y, y', y'') \tag{20}$$

この 3 階常微分方程式を次に示す式を用いて変換する。

$$\begin{cases} y_0(x) = y(x) \\ y_1(x) = y'(x) \\ y_2(x) = y''(x) \end{cases} \tag{21}$$

この式を用いて、式 (20) を書き直すと

$$\begin{cases} y_0'(x) = y_1(x) \\ y_1'(x) = y_2(x) \\ y_2'(x) = f(x, y_0, y_1, y_2) \end{cases} \tag{22}$$

となる。これで、3 階の常微分方程式が 3 元の 1 階の連立常微分方程式に変換できた。2 階であろうが 4 階 $\cdots$ でも同じ方法で連立微分方程式に還元できる。

2.5.2 練習問題

以下の高次常微分方程式を連立 1 階微分方程式に書き換えなさい。

(1) $y'' + 3y' + 5y = 0$
(2) $y'' + 6y' + y = 0$
(3) $5y'' + 2xy' + 3y = 0$
(4) $y''' + y' + xy = 0$
(5) $5y'' + y' + y = \sin(\omega x)$
(6) $xy'' + y' + y = e^x$
(7) $5y''y' + y' + y = 0$
(8) $y''y' + x^2y'y + y = 0$

3 連立一次方程式

3.1 ガウス・ジョルダン法の基本的な考え方

ガウス・ジョルダン (Gauss-Jordan) 法というのは、連立方程式 (6) を次にように変形させて、解く方法である。

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1' \\ b_2' \\ b_3' \\ \vdots \\ b_N' \end{pmatrix} \tag{23}$$

この式から明らかに、求める解 $x_i = b'_i$ となる。これをどうやって求めるか?。コンピューターで実際に計算する前に、人力でガウス・ジョルダン法で計算してみる。例として、

$$\begin{cases} 3x + 2y + z = 10 \\ x + y + z = 6 \\ x + 2y + 2z = 11 \end{cases} \tag{24}$$

をガウス・ジョルダン法で解を求める。

まずは、1 行目の x の係数を 1 に、2 と 3 行目のそれは 0 にします。そのために、1 行目は x の係数の値で割る。2 行目と 3 行目は、1 行目に適当な係数を掛けて引く。次のようにする。

$$\begin{cases} x + \frac{2}{3}y + \frac{1}{3}z = \frac{10}{3} \\ x + y + z = 6 \\ x + 2y + 2z = 11 \end{cases} \Rightarrow \begin{cases} x + \frac{2}{3}y + \frac{1}{3}z = \frac{10}{3} \\ 0 + \frac{1}{3}y + \frac{2}{3}z = \frac{8}{3} \\ 0 + \frac{4}{3}y + \frac{5}{3}z = \frac{23}{3} \end{cases} \tag{25}$$

つぎに、2 行目の y の係数を 1 にして、1 と 3 行目のそれを 0 にする。

$$\begin{cases} x + \frac{2}{3}y + \frac{1}{3}z = \frac{10}{3} \\ 0 + y + 2z = 8 \\ 0 + \frac{4}{3}y + \frac{5}{3}z = \frac{23}{3} \end{cases} \Rightarrow \begin{cases} x + 0 - z = -2 \\ 0 + y + 2z = 8 \\ 0 + 0 - z = -3 \end{cases} \tag{26}$$

同じことを z についても繰り返えす。すると、

$$\begin{cases} x + 0 - z = -2 \\ 0 + y + 2z = 8 \\ 0 + 0 + z = 3 \end{cases} \Rightarrow \begin{cases} x + 0 + 0 = 1 \\ 0 + y + 0 = 2 \\ 0 + 0 + z = 3 \end{cases} \tag{27}$$

となる。従って、連立方程式 (24) の解は、

$$\begin{cases} x = 1 \\ y = 2 \\ z = 3 \end{cases} \tag{28}$$

となる。これがガウス・ジョルダン法である。もっともらしい名前が付けられているが、大したことなはい。

これを係数行列とベクトルで書くと、次のようになる。

3.2 ガウス・ジョルダン法の C 言語の関数

ピボット選択は行わないで、逆行列も求めないのガウス・ジョルダン法で連立方程式を計算するプログラムを示す。このプログラムの動作は、次の通りである。

- 仮引数「`n`」は、解くべき連立方程式の未知数の数である。

- 仮引数の配列「a」と「b」は、係数行列 $\boldsymbol{A}$ と非同次項 $\boldsymbol{b}$ である。値は、呼び出し元からにアドレス渡しで送られる。
  - 係数行列は、配列「a[1][1]」〜「a[n][n]」に格納されている。
  - 非同次項は、配列「b[1]」〜「b[n]」に格納されている。
- 連立方程式の解 $\boldsymbol{x}$ は、配列「b[1]」〜「b[n]」に格納される。
- このプログラムでの処理が終了すると、配列「a[1][1]」〜「a[n][n]」は単位行列になる。。

```
/* ========== ガウスジョルダン法の関数 =================*/
void gauss_jordan(int n, double a[][100], double b[]){

   int ipv, i, j;
   double inv_pivot, temp;

   for(ipv=1 ; ipv <= n ; ipv++){

      /* ---- 対角成分=1(ピボット行の処理) ---- */
      inv_pivot = 1.0/a[ipv][ipv];
      for(j=1 ; j <= n ; j++){
         a[ipv][j] *= inv_pivot;
      }
      b[ipv] *= inv_pivot;

      /* ---- ピボット列=0(ピボット行以外の処理) ---- */
      for(i=1 ; i<=n ; i++){
         if(i != ipv){
            temp = a[i][ipv];
            for(j=1 ; j<=n ; j++){
               a[i][j] -= temp*a[ipv][j];
            }
            b[i] -= temp*b[ipv];
         }
      }

   }
}
```

4 まとめ

この試験範囲で理解すべきことをまとめると、以下のようになる。

- 基本
  - テーラー展開ができること。
  - 連立方程式が行列で書き表せること。

- 常微分方程式
  - オイラー法の内容が理解できること。
  - 2 次のルンゲクッタ法 (ホイン法) の漸化式が導け、その内容が理解できること。
  - 4 次のルンゲクッタ法の漸化式くらいは、暗記すること。イメージが湧けば、そんなに難しくはない。
  - 4 次のルンゲクッタ法のプログラムの内容が理解できること。
  - 高階の微分方程式が連立の 1 階の微分方程式に直せること。
- 連立 1 次方程式
  - ガウス・ジョルダン法で連立方程式が、計算できること。
  - ガウス・ジョルダン法の C 言語の関数が、理解できること。