常微分方程式と連立方程式，補間法のまとめ

山本昌志*

2007 年 11 月 27 日

1　後期中間試験の傾向と対策

後期中間試験では，常微分方程式の数値解法と連立方程式，補間法について問う．具体的には，次の数値計算法の計算原理とコンピュータープログラムの方法を理解しておくこと．

- 常微分方程式の数値計算法
  - オイラー法
  - 2 次のルンゲ・クッタ法
  - 4 次のルンゲ・クッタ法
  - 高階の常微分方程式
- 連立 1 次方程式 (消去法)
  - ガウス・ジョルダン法
- 連立 1 次方程式 (反復法)
  - ヤコビ法
  - ガウス・ザイデル法
- 補間法
  - ラグランジュ補間
  - スプライン補間

以降，学習すべき内容をまとめておくので，よく理解して試験に臨むこと．試験日時と注意事項は，次の通りである．

試験日	12 月 6 日 (木曜日) 8:45〜9:45(60 分)
場所	教室 (PC ルームではない)
注意事項	教科書やノート，プリント類は持ち込み不可

*秋田工業高等専門学校　電気工学科

2　常微分方程式

数値計算により，近似解を求める微分方程式は，

$$\frac{dy}{dx} = f(x,y) \qquad \text{初期条件} \quad y(a) = b \tag{1}$$

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　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，

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

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，

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

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

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

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 (2)〜式 (4) のいずれかである．

2.2　オイラー法

2.2.1　計算原理

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，

$$y_{i+1} = y(x_i + \Delta x) = y(x_i) + \frac{dy}{dx}\Big|_{x=x_i} \Delta x + \frac{1}{2}\frac{d^2y}{dx^2}\Big|_{x=x_i} \Delta x^2 + \frac{1}{6}\frac{d^3y}{dx^3}\Big|_{x=x_i} \Delta x^3 + \ldots \tag{5}$$

である．この式の右辺第 2 項は，式 (1) 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．

$$y_{i+1} = y_i + f(x_i, y_i)\Delta x + O(\Delta x^2) \tag{6}$$

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，

$$y_{i+1} = y_i + f(x_i, y_i)\Delta x \tag{7}$$

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．

$$\begin{cases} x_0 = a \\ y_0 = b \\ x_{i+1} = x_i + \Delta x \\ y_{i+1} = y_i + f(x_i, y_i)\Delta x \end{cases} \tag{8}$$

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．

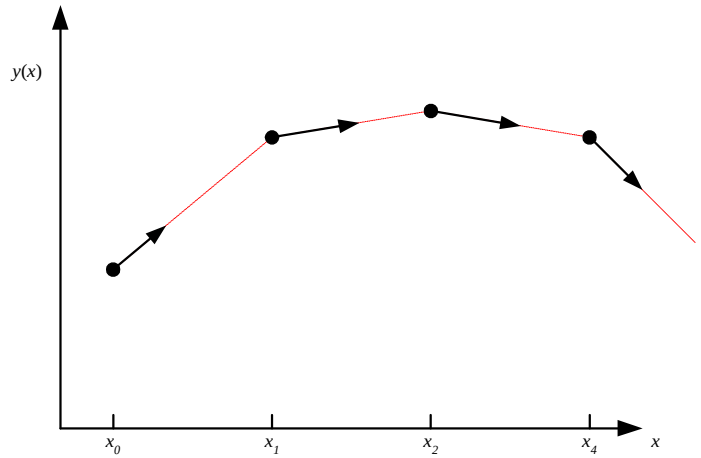

図 1: オイラー法．ある区間での y の変化 Δy は，計算の始めの点の傾きに区間の幅 Δx を乗じて，求めている．

[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

2.2.2 プログラム例

次の微分方程式

$$\frac{\mathrm{d}y}{\mathrm{d}x} = x^2 \sin x + \sqrt{y}\cos y \qquad\qquad y(0) = 0 \tag{9}$$

の近似解をオイラー法で計算するプログラムをリスト 1 に示す．計算結果は，ファイルに格納している．計算領域は $0 \leqq x \leqq 2$，計算のステップ幅は $\mathrm{d}x = 0.001$ となっている．

リスト 1: オイラー法で微分方程式の近似解を求めるプログラム

```
1  #include <stdio.h>
2  #include <math.h>
3  #define NSTEPS 2000                    // 計算ステップ数
4  #define NDIM 30000                     // 用意する配列の大きさ
5  #define X_MAX 2.0                      // 計算する最大 x
6
7  double f(double x, double y);          // プロトタイプ宣言
8
9  //=============================================================
10 // main 関数
11 //=============================================================
12 int main(void){
13   double x[NDIM], y[NDIM];             // 計算結果を入れる
14   double dx;
15   int i;
16   FILE *out;                           // 計算結果を保存するファイル
17
18
19   dx = X_MAX/NSTEPS;                   // 計算のきざみ幅 dxの計算
20   x[0] = 0;
21   y[0] = 0;
22
23
24   //------- オイラー法の計算 -------
25
26   for(i=0; i<NSTEPS; i++){
27     x[i+1] = x[0]+(i+1)*dx;            //x[i+1]=x[i]+dxよりも精度が良い
28     y[i+1] = y[i]+f(x[i],y[i])*dx;
29   }
30
31
32   //------- 計算結果をファイルに保存 -------
33
34   out = fopen("result.txt","w");
35   for(i=0; i<=NSTEPS; i++){
36     fprintf(out,"%20.15f\t%20.15f\n", x[i],y[i]);
37   }
38
39
40   fclose(out);
41
42   return 0;
43 }
44
45 //=============================================================
46 // dy/dx = f(x,y) の  f(x,y)を計算する関数
47 //=============================================================
48 double f(double x, double y){
49   return x*x*sin(x)+sqrt(y)*cos(y);
```

```
50 }
```

2.3 2 次のルンゲクッタ

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，ホイン法と中点法をしめす．

2.3.1 ホイン法

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

$$y(x_0+h) = y(x_0) + y'(x_0)h + \frac{1}{2}y''(x_0)h^2 + O(h^3) \tag{10}$$

となっていることを意味する．即ち，計算アルゴリズムが，

$$\Delta y = y'(x_0)h + \frac{1}{2}y''(x_0)h^2 + O(h^3) \tag{11}$$

となっている．

式 (11) 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．

$$\Delta y = h\{\alpha y'(x_0) + \beta y'(x_0+h)\} \tag{12}$$

この α, β を上手に選ぶことにより，式 (11) 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

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

となる．これを，式 (11) と比較すると，

$$\begin{cases} \alpha = \dfrac{1}{2} \\ \beta = \dfrac{1}{2} \end{cases} \tag{14}$$

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 (1) 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．

$$\begin{cases} k_1 = hf(x_n, y_n) \\ k_2 = hf(x_n+h, y_n+k_1) \\ y_{n+1} = y_n + \dfrac{1}{2}(k_1+k_2) \end{cases} \tag{15}$$

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．

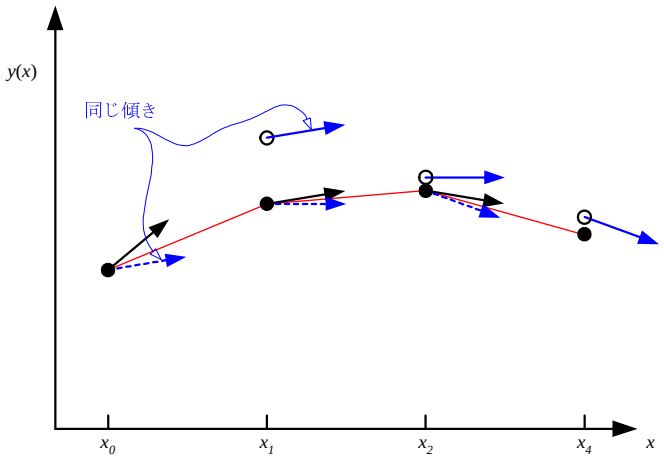

図 2: ホイン法．ある区間での y の変化 Δy は，計算の始めと終わりの点付近の平均傾きに区間の幅 h を乗じて，求めている．

2.3.2 中点法

これも，ホイン法と同じ 2 次の精度である．ホイン法は区間の両端の点の導関数を使ったが，中点法は出発点と中点で漸化式を作る．先ほど同様，2 点を使うので，2 次の精度にすることができる．ホイン法の式 (12) に対応するものは，

$$\Delta y = h\{\alpha y'(x_0) + \beta y'(x_0 + \frac{h}{2})\} \tag{16}$$

である．これを x_0 の回りでテイラー展開すると，

$$\Delta y = (\alpha + \beta)y'(x_0)h + \frac{\beta}{2}y''(x_0)h^2 + O(h^3) \tag{17}$$

となる．これを，式 (11) と比較すると，

$$\begin{cases} \alpha = 0 \\ \beta = 1 \end{cases} \tag{18}$$

となる必要がある．したがって，中点法の漸化式は，

$$\begin{cases} k_1 = hf(x_n, y_n) \\ k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \\ y_{n+1} = y_n + k_2 \end{cases} \tag{19}$$

となる．この公式のイメージを，図 3 に示しておく．

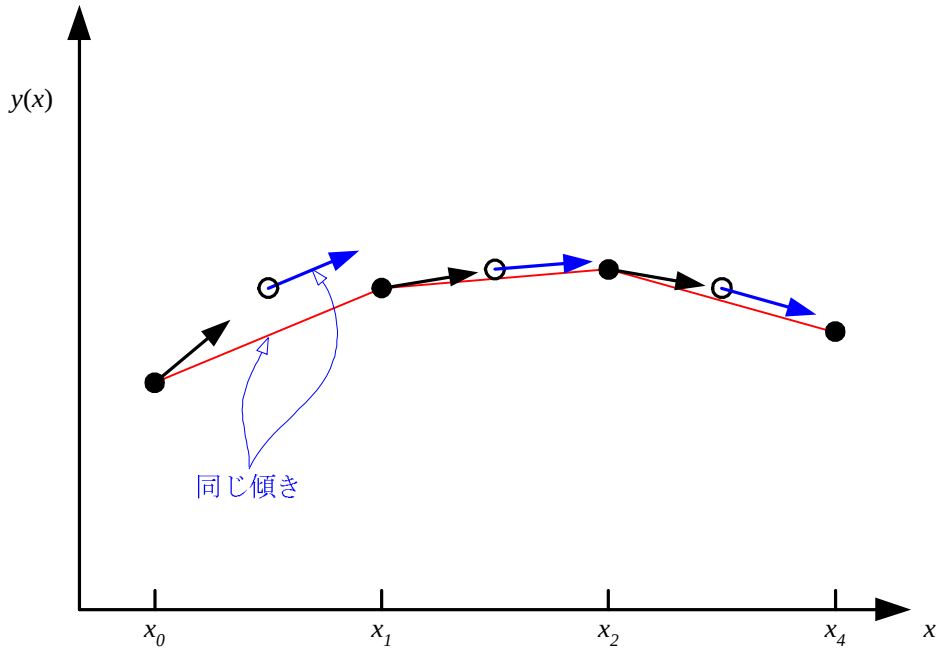

図 3: 中点法．ある区間での y の変化 Δy は，中点付近の傾きに区間の幅 h を乗じて，求めている．

2.4　4 次のルンゲ・クッタ法

前期末試験で出題したオイラー法や 2 次のルンゲ・クッタ法は，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ある．この方法で最良と言われるのが 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 (20) に示す通りである．そして，これは図 4 のように表すことができる．

$$\begin{cases} k_1 = hf(x_n, y_n) \\ k_2 = hf(x_n + \dfrac{h}{2}, y_n + \dfrac{k_1}{2}) \\ k_3 = hf(x_n + \dfrac{h}{2}, y_n + \dfrac{k_2}{2}) \\ k_4 = hf(x_n + h, y_n + k_3) \\ x_{n+1} = x_n + h \\ y_{n+1} = y_n + \dfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{cases} \tag{20}$$

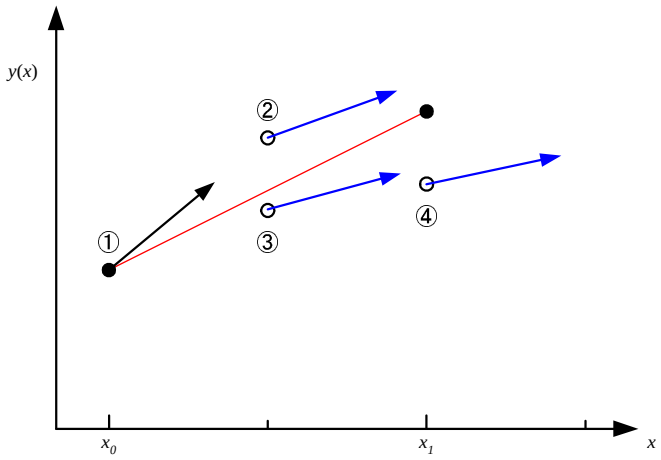

図 4: 4 次のルンゲ・クッタ法．ある区間での y の変化 Δy は，区間内の 4 点の傾きのある種の加重平均に幅 h を乗じて，求めている．

2.5 プログラム (4 次のルンゲ・クッタ法)

実際の微分方程式

$$\begin{cases} \dfrac{dy}{dx} = \sin x \cos x - y \cos x \\ y = 0 \quad (\text{初期条件 } x = 0 \text{ の時}) \end{cases} \tag{21}$$

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);
}
```

2.6 高階の常微分方程式

2.6.1 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．

$$y'''(x) = f(x, y, y', y'') \tag{22}$$

この 3 階常微分方程式を次に示す式を用いて変換する．

$$\begin{cases} y_0(x) = y(x) \\ y_1(x) = y'(x) \\ y_2(x) = y''(x) \end{cases} \tag{23}$$

この式を用いて，式 (22) を書き直すと

$$\begin{cases} y_0'(x) = y_1(x) \\ y_1'(x) = y_2(x) \\ y_2'(x) = f(x, y_0, y_1, y_2) \end{cases} \tag{24}$$

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．

2.6.2 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 (消去法)

3.1 連立方程式の表現方法

連立 1 次方程式 (Linear Equations) は，次のような形をしている．

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1N}x_N &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2N}x_N &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3N}x_N &= b_3 \\ &\vdots \\ a_{M1}x_1 + a_{M2}x_2 + a_{M3}x_3 + \cdots + a_{MN}x_N &= b_M \end{aligned} \tag{25}$$

式 (25) は行列とベクトルで書くと，式がすっきりして考えやすくなる．書き直すと，

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{26}$$

である．それぞれの行列とベクトルは，

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1N} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2N} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3N} \\ \vdots & & & \ddots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & \cdots & a_{NN} \end{bmatrix} \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_N \end{bmatrix} \tag{27}$$

を表す．

通常，連立 1 次方程式 (25) は

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1N} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2N} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3N} \\ \vdots & & & \ddots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & \cdots & a_{NN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_N \end{bmatrix} \tag{28}$$

と書き表せる．このようにすると，見通しがかなり良くなる．

3.2 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 (28) を次にように変形させて，解く方法である．

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \\ \vdots \\ b'_N \end{bmatrix} \tag{29}$$

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，

$$\begin{cases} 3x + 6y + 9z = 6 \\ 2x + 2y + 3z = 1 \\ 2x + 2y + 1z = -1 \end{cases} \tag{30}$$

をガウス・ジョルダン法で解を求める．

解くべき，方程式

$$\begin{bmatrix} 3 & 6 & 9 \\ 2 & 2 & 3 \\ 2 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 9 \\ 1 \\ -1 \end{bmatrix}$$

$\frac{1}{3}\times$1 行

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 2 & 3 \\ 2 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix}$$

2 行 $-2\times$1 行

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -3 \\ 2 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ -1 \end{bmatrix}$$

3 行 $-2\times$1 行

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -3 \\ 0 & -2 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ -5 \end{bmatrix}$$

$-\frac{1}{2}\times$2 行

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & \frac{3}{2} \\ 0 & -2 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ \frac{3}{2} \\ -5 \end{bmatrix}$$

1 行 $-2\times$2 行

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \frac{3}{2} \\ 0 & -2 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ \frac{3}{2} \\ -5 \end{bmatrix}$$

3 行 $+2\times$2 行

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \frac{3}{2} \\ 0 & 0 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ \frac{3}{2} \\ -2 \end{bmatrix}$$

$-\frac{1}{2}\times$3 行

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \frac{3}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ \frac{3}{2} \\ 1 \end{bmatrix}$$

2 行 $-\frac{3}{2}\times$3 行

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

これで，ガウス・ジョルダン法による対角化の作業は完了である．これから，$(x_1, x_2, x_3) = (-1, 0, 1)$ と分かる．

3.3 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 連立一次方程式 (反復法)

実用的なプログラムでは，非常に大きな連立方程式を計算しなくてはならない．数百万元に及ぶことも珍しくない．これを，ガウス・ジョルダン法で計算するの時間的にほとんど不可能である．そこで，これよりは格段に計算の速い方法が用いられる．ここでは，その一つとして反復法を簡単に説明する．

当然ここでも，連立方程式

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{31}$$

を満たす $\boldsymbol{x}$ を数値計算により，求めることになる．ここで，真の解 $\boldsymbol{x}$ とする．

ここで，ある計算により n 回目で求められたものを $\boldsymbol{x}^{(}n)$ とする．そして，計算回数を増やして，

$$\lim_{n\to\infty} \boldsymbol{x}^{(n)} = \boldsymbol{x} \tag{32}$$

になったとする．この様に計算回数を増やして，真の解に近づける方法を反復法という．

この様な方法は，行列 $\boldsymbol{A}$ を $\boldsymbol{S}-\boldsymbol{T}$ と分解するだけで，容易に作ることができる．たとえば，

$$\boldsymbol{S}\boldsymbol{x}^{(k+1)}=\boldsymbol{T}\boldsymbol{x}^{(k)}+\boldsymbol{b} \tag{33}$$

とすればよい．ここで，$\boldsymbol{x}^{(k)}$ が $\boldsymbol{\alpha}$ に収束するとする．すると，式 (33) と式 (31) を比べれば，$\boldsymbol{\alpha}$ と $\boldsymbol{x}$ は等しいことがわかる．すなわち，式 (33) で元の方程式 (31) を表した場合，$\boldsymbol{x}^{(k)}$ が収束すれば，必ず真の解 $\boldsymbol{x}$ に収束するのである．別の解に収束することはなく，真の解に収束するか，発散するかのいずれかである．

4.1 ヤコビ法

係数行列 $\boldsymbol{A}$ の対角行列を反復計算の行列 $\boldsymbol{S}$ としたものがヤコビ (Jacobi) 法である．係数行列を

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & 0 & 0 & \dots & 0 \\ 0 & a_{22} & 0 & \dots & 0 \\ 0 & 0 & a_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & a_{nn} \end{bmatrix} + \begin{bmatrix} 0 & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & 0 & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & 0 & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & 0 \end{bmatrix} \tag{34}$$

と分解し，右辺第 1 項が行列 $\boldsymbol{S}$ で第 2 項が $-\boldsymbol{T}$ となる．$\boldsymbol{x}^{(k+1)}$ の解の計算に必要な $\boldsymbol{S}$ の逆行列は，それが対角行列なので，

$$\boldsymbol{S}^{-1}=\begin{bmatrix} a_{11}^{-1} & 0 & 0 & \dots & 0 \\ 0 & a_{22}^{-1} & 0 & \dots & 0 \\ 0 & 0 & a_{33}^{-1} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & a_{nn}^{-1} \end{bmatrix} \tag{35}$$

と簡単に求めることができる．$k+1$ 番目の近似解は $\boldsymbol{x}^{(k+1)}=\boldsymbol{S}^{-1}\left(\boldsymbol{b}+\boldsymbol{T}\boldsymbol{x}^{(k)}\right)$ となるなので，容易に求めることができる．実際，k 番目の解を

$$x_1^{(k)},\, x_2^{(k)},\, x_3^{(k)},\, \cdots,\, x_n^{(k)}$$

とすると，k+1 番目の解は

$$\begin{aligned} x_1^{(k+1)} &= a_{11}^{-1}\left\{b_1-\left(a_{12}x_2^{(k)}+a_{13}x_3^{(k)}+a_{14}x_4^{(k)}+\cdots+a_{1n}x_n^{(k)}\right)\right\} \\ x_2^{(k+1)} &= a_{22}^{-1}\left\{b_2-\left(a_{21}x_1^{(k)}+a_{23}x_3^{(k)}+a_{24}x_4^{(k)}+\cdots+a_{2n}x_n^{(k)}\right)\right\} \\ x_3^{(k+1)} &= a_{33}^{-1}\left\{b_3-\left(a_{31}x_1^{(k)}+a_{32}x_2^{(k)}+a_{34}x_4^{(k)}+\cdots+a_{3n}x_n^{(k)}\right)\right\} \\ &\vdots \\ x_n^{(k+1)} &= a_{nn}^{-1}\left\{b_n-\left(a_{n1}x_1^{(k)}+a_{n2}x_2^{(k)}+a_{n3}x_3^{(k)}+\cdots+a_{nn-1}x_{n-1}^{(k)}\right)\right\} \end{aligned} \tag{36}$$

と計算できる．これを繰り返して連立方程式の解を求める方法が，ヤコビ法である．

4.2 ガウス・ザイデル法

ヤコビ法の特徴では，$\boldsymbol{x}^{(k+1)}$ の近似値は，すべてその前の値 $\boldsymbol{x}^{(k)}$ を使うことにある．大きな行列を扱う場合，全ての $\boldsymbol{x}^{(k+1)}$ と $\boldsymbol{x}^{(k)}$ を記憶する必要があり，大きなメモリーが必要となり問題が生じる．今では，個人で大きなメモリーを使い計算することは許されるが，ちょっと前まではできるだけメモリーを節約したプログラムを書かなくてはならなかった．

そこで，$\boldsymbol{x}^{(k+1)}$ の各成分の計算が終わると，それを直ちに使うことが考えば，メモリーは半分で済む．即ち，$x_i^{(k+1)}$ を計算するときに，

$$x_i^{(k+1)} = a_{ii}^{-1}\Big\{b_i - (a_{i1}x_1^{(k+1)} + a_{i2}x_2^{(k+1)} + a_{i3}x_3^{(k+1)} + \cdots + a_{ii-1}x_{i-1}^{(k+1)} + a_{ii+1}x_{i+1}^{(k)} + a_{ii+2}x_{i+2}^{(k)} + a_{ii+3}x_{i+3}^{(k)} + \cdots + a_{in}x_n^{(k)})\Big\} \tag{37}$$

とするのである．実際の計算では，$k+1$ 番目の解は

$$\begin{aligned}
x_1^{(k+1)} &= a_{11}^{-1}\left\{b_1 - \left(a_{12}x_2^{(k)} + a_{13}x_3^{(k)} + a_{14}x_4^{(k)} + \cdots + a_{1n}x_n^{(k)}\right)\right\} \\
x_2^{(k+1)} &= a_{22}^{-1}\left\{b_2 - \left(a_{21}x_1^{(k+1)} + a_{23}x_3^{(k)} + a_{24}x_4^{(k)} + \cdots + a_{2n}x_n^{(k)}\right)\right\} \\
x_3^{(k+1)} &= a_{33}^{-1}\left\{b_3 - \left(a_{31}x_1^{(k+1)} + a_{32}x_2^{(k+1)} + a_{34}x_4^{(k)} + \cdots + a_{3n}x_n^{(k)}\right)\right\} \\
&\vdots \\
x_n^{(k+1)} &= a_{nn}^{-1}\left\{b_n - \left(a_{n1}x_1^{(k+1)} + a_{n2}x_2^{(k+1)} + a_{n3}x_3^{(k+1)} + \cdots + a_{nn-1}x_{n-1}^{(k+1)}\right)\right\}
\end{aligned} \tag{38}$$

と計算できる．これを繰り返して連立方程式の解を求める方法が，ガウス・ザイデル法である．

4.3 プログラム例 (ガウス・ザイデル法)

ガウス・ザイデル法のような反復法は大きな連立方程式の計算に敵している．しかし，ここではその計算原理を分かり易くするため，次の連立方程式を計算する．

$$\begin{bmatrix} 3 & 2 & 1 \\ 1 & 4 & 1 \\ 2 & 2 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 12 \\ 21 \end{bmatrix} \tag{39}$$

式 (38) の漸化式に従うプログラム例を以下に示す．

```
#include <stdio.h>
#include <math.h>
#define N (3)                  // 連立方程式の大きさ
#define EPS (1e-15)            // 計算誤差の許容値

int main(void){
  double a[N+1][N+1], x[N+1], b[N+1];
  double dx, absx, sum, new;
  int i,j;
```

```
  a[1][1]=3.0;  a[1][2]=2.0;  a[1][3]=1.0;  // 係数行列
  a[2][1]=1.0;  a[2][2]=4.0;  a[2][3]=1.0;
  a[3][1]=2.0;  a[3][2]=2.0;  a[3][3]=5.0;

  b[1]=10.0;                          // 同次項
  b[2]=12.0;
  b[3]=21.0;

  x[1]=0.0;                           // 近似解の初期値
  x[2]=0.0;
  x[3]=0.0;

  do{                                 // 反復計算のループ
    dx=0.0;
    absx=0.0;

    for(i=1;i<=N;i++){
      sum=0;
      for(j=1;j<=N;j++){
if(i != j){
  sum+=a[i][j]*x[j];
}
      }

      new=1.0/a[i][i]*(b[i]-sum);   // 反復計算後の近似解
      dx+=fabs(new-x[i]);           // 近似解の変化量を加算
      absx+=fabs(new);              // 近似解の総和計算
      x[i]=new;                     // 新しい近似解を代入
    }

  }while(dx/absx > EPS);            // 計算終了条件

  for(i=1;i<=N;i++){
    printf("x[%d]=%25.20f\n",i,x[i]);
  }

  return 0;
}
```

5 補間法

実験やシミュレーションを行うと，離散的にデータが得られるのは普通である．例えば，半導体の電圧・電圧特性の測定では，(V_0, I_0), (V_1, I_1), (V_2, I_2), (V_3, I_3), $\cdots$, (V_N, I_N) のようなデータが得られる．通常，このデータはグラフ化して解析を進める．このデータの場合，2 次元のグラフ上に測定点が並ぶことは，今まで学習してきたとおりである．

実験等を通して得られる結果は離散的であるが，実際の現象は連続的なことが多い．この離散的な値を用いて，測定点の間の値—例えば電流と電圧の関係—を求めるのが補間法の役割である．ここで学習したラグランジュ補間もスプライン補間も，全てのグラフ上の測定点を通る曲線の方程式を求めている．

2 次元のグラフ上の点は，数学では座標 $(x,\, y)$ の点として与えられる．以降の説明では，電圧・電流などのように特定の問題にとらわれないよう，一般化した座標 $(x,\, y)$ で話を進める．

5.1 ラグランジュ補間

平面座標上に $N+1$ 個の点がある場合，その全ての点を通る曲線は N 次関数で表すことができる[2]．2 個の場合には 1 次関数，すなわち直線で，その 2 点を通る関数を決めることができる．3 点の場合は，2 次関数になる．

この性質を利用すると，$N+1$ 個の点がある場合，N 次関数で補間できることが分かる．ラグランジュ補間とは，まさにこの補間を行っていうる．数学の授業で，ある 3 点 $(x_0, y_0),\, (x_1, y_1),\, (x_2, y_2)$ を通る 2 次関数 $y = ax^2 + bx + c$ の a, b, c を求めたことがあるだろうが，それと同じである．そこでは，それぞれの x と y の値を代入して，連立方程式をつくり a, b, c を求めたはずである．

コンピューターを用いて，$N+1$ 個の点を通る N 次方程式を表す $N+1$ 個の係数を連立方程式を解くことにより求めることは可能である．しかし，最終目的の N 次関数の値を求めると言う意味では不経済である．補間という目的からすると，関数を形成する係数なんか，全く興味の対象外なのである．そこで，係数が分からなくても，N 次関数を示すものとして，ラグランジュ補間が使われる．

2 次元座標上に $N+1$ 個の点，$(x_0, y_0)\,, (x_1, y_1),\, (x_2, y_2),\, \cdots,\, (x_N, y_N)$ のラグランジュ補間は，

$$
\begin{aligned}
y = {} & \frac{(x-x_1)(x-x_2)(x-x_3)\cdots(x-x_N)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)\cdots(x_0-x_N)} y_0 + \frac{(x-x_0)(x-x_2)(x-x_3)\cdots(x-x_N)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)\cdots(x_1-x_N)} y_1 \\
& + \frac{(x-x_0)(x-x_1)(x-x_3)\cdots(x-x_N)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)\cdots(x_2-x_N)} y_2 + \frac{(x-x_0)(x-x_1)(x-x_2)\cdots(x-x_N)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)\cdots(x_3-x_N)} y_3 \\
& \cdots + \frac{(x-x_0)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_N)}{(x_k-x_0)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_N)} y_k + \cdots \\
& + \frac{(x-x_0)(x-x_1)(x-x_2)\cdots(x-x_{N-1})}{(x_N-x_0)(x_N-x_1)(x_N-x_2)\cdots(x_N-x_{N-1})} y_N
\end{aligned}
\tag{40}
$$

となる．

この式 (40) を見ると，

- 各項の分母は定数で，分子は N 次関数である．このことから，全ての項は N 次関数になっていることが理解できる．したがって，この式は N 次関数 (N 次多項式) である．
- x に $x_0, x_1, x_2, \cdots, y_N$ を代入すると，y の値は $y_0, y_1, y_1, \cdots, x_N$ になることが分かる．これは，データ点 $(x_0, y_0)\,, (x_1, y_1),\, (x_2, y_2),\, \cdots,\, (x_N, y_N)$ の全てを通過していることを示している．

となっている．

[2] x が同じで y が異なる複数の点が存在するような特別な場合を除く．

式 (40) をもうちょっと格好良く書けば，

$$L(x) = \sum_{k=0}^{N} L_k(x) y_k \tag{41}$$

ただし，

$$\begin{aligned} L_k(x) &= \frac{(x-x_0)(x-x_1)(x-x_2)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_N)}{(x_k-x_0)(x_k-x_1)(x_k-x_2)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_N)} \\ &= \frac{x-x_0}{x_k-x_0} \times \frac{x-x_1}{x_k-x_1} \times \frac{x-x_2}{x_k-x_2} \times \cdots \times \frac{x-x_{k-1}}{x_k-x_{k-1}} \times \frac{x-x_{k+1}}{x_k-x_{k+1}} \times \cdots \frac{x-x_N}{x_k-x_N} \\ &= \prod_{j=0}^{N(j\neq k)} \frac{x-x_j}{x_k-x_j} \end{aligned} \tag{42}$$

となる．

ラグランジュ補間の考え方は単純で，その計算も簡単である．しかし，補間の点数が増えてくると，ラグランジュの補間には問題が生じる．ラグランジュの補間では，補間の点数が増えてくると大きな振動が発生して，もはや補間とは言えなくなる．ラグランジュの補間には常にこの問題が付きまうので，データ点数が多い場合は使えなくなる．

5.2 スプライン補間

5.2.1 区分多項式

ラグランジュの補間は，データ点数が増えてくると関数が振動し問題が発生する．補間する関数の次数が増加するからである．そこでこの問題を解決するために，補間する領域をデータ間隔 $[x_i, x_{i+1}]$ で区切り，その近傍の値を使い低次の多項式で近似することを考える．区分的な関数を使うことになるが，上手に近似をしないと境界でその導関数が不連続になる．導関数が連続になるように，上手に近似する方法がスプライン補間 (spline interpolation) である．

補間をするデータは，先と同じように $(x_0, y_0), (x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)$ とする．そして，区間 $[x_j, x_{j+1}]$ で補間をする関数を $S_j(x)$ とする．この様子を図 5 に示す．

5.2.2 係数が満たす式

3 次のスプライン補間を考えるので，

$$S_j(x) = a_j(x-x_j)^3 + b_j(x-x_j)^2 + c_j(x-x_j) + d_j \qquad (j = 0, 1, 2, 3, \cdots, N-1) \tag{43}$$

となる．スプライン補間を行う場合，この a_j, b_j, c_j, d_j を決めることが，主な作業である．

これらの $4N$ 個の未知数を決めるためには，$4N$ 個の方程式が必要である．そのために，3 次のスプライン補間に以下の条件を課すことにする．

- 全てのデータ点を通る．各々の $S_j(x)$ に対して両端での値が決まるため，$2N$ 個の方程式ができる．

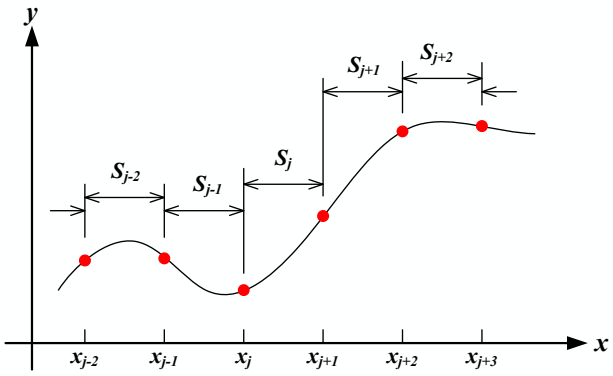

図 5: スプライン補間の区分

- 各々の区分補間式は，境界点の 1 次導関数は連続とする．これにより，$N-1$ 個の方程式ができる．
- 各々の区分補間式は，境界点の 2 次導関数は連続とする．これにより，$N-1$ 個の方程式ができる．

以上の条件を課すと $4N-2$ 個の方程式が決まる．未知数は $4N$ 個なので，2 個方程式が不足している．この不足を補うために，いろいろな条件が考えられるが，通常は両端 x_0 と x_N での 2 次導関数の値を 0 とする．すなわち，$S_0''(x_0) = S_{N-1}''(x_N) = 0$ である．これを自然スプライン (natural spline) と言う．自然スプライン以外には，両端の 1 次導関数の値を指定するものもある．

これで全ての条件が決まった．あとは，この条件に満たす連立方程式を求めるだけである．

6 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．

- 常微分方程式
  - テイラー展開を使って，2 次のルンゲクッタ法の漸化式を導けるようになること．
  - 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．
  - 高階の微分方程式を連立の 1 階の微分方程式に直せること．
- 連立 1 次方程式 (消去法)
  - 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・ジョルダン法で逆行列が計算できること．
  - ガウス・ジョルダン法の C 言語の関数が理解できること．

- 連立 1 次方程式 (反復法)
  - ガウス・ザイデル法の漸化式くらい導けること.
  - ガウス・ザイデル法で C 言語のプログラムがかけること.
- 補間法
  - ラグランジュ補間の式とその意味が説明できること.
  - スプライン補間の基本的な考え方が説明できること. 3 次のスプライン補間の係数の決め方が説明できること.