レポート課題 No.2

川口廣伊智
学籍番号:051715223

2017/07/06

0　レポートについての注意

0.1　各課題の解答の構成について

まず課題の解釈をし、解答するためのプログラムを記載した。次に得られた結果を記載した。考察すべき内容があった場合は簡単な考察も付けた。すべての課題に考察がついているわけではない。

0.2　プログラムについて

各課題についてその計算を行うためのプログラムを全掲した。長ったらしいが、ソースコードの後に特に工夫した部分がどこかを記した。

1　基本課題 EX3-1

1.1　課題概要

LU 分解を用いて行列の行列式を計算するプログラムを作成した。

ソースコード 1　LU 分解を用いて行列の行列式を計算するプログラム

```
1  #include "matrix_util.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  /* http://www.netlib.org/lapack/explore-html/d3/d6a/dgetrf_8f.html */
6  extern void dgetrf_(int *M, int *N, double *A, int *LDA, int*IPIV, int *INFO);
7
8
9  int main(int argc, char** argv) {
10   char* filename;
11   FILE *fp;
12
13   int i, m, n;
14   double **a;
15   double determinant = 1.0;
16
```

```
17
18  int *ipiv;
19  int info;
20
21  if (argc < 2) {
22    fprintf(stderr, "Usage: %s inputfile\n", argv[0]);
23    exit(1);
24  }
25  filename = argv[1];
26
27  /* read matrix A from a file */
28  fp = fopen(filename, "r");
29  if (fp == NULL) {
30    fprintf(stderr, "Error: file can not open\n");
31    exit(1);
32  }
33  read_dmatrix(fp, &m, &n, &a);
34  if (m != n) {
35    fprintf(stderr, "Error: inconsistent number of equations\n");
36    exit(1);
37  }
38  printf("Matrix A:\n");
39  fprint_dmatrix(stdout, n, n, a);
40
41  /* perform LU decomposition */
42  ipiv = alloc_ivector(n);
43  dgetrf_(&n, &n, &a[0][0], &n, &ipiv[0], &info);
44  if (info != 0) {
45    fprintf(stderr, "Error: LAPACK::dgetrf failed\n");
46    exit(1);
47  }
48  printf("Result of LU decomposition:\n");
49  fprint_dmatrix(stdout, n, n, a);
50  printf("Pivot for LU decomposition:\n");
51  fprint_ivector(stdout, n, ipiv);
52
53  /* calculate the determinant of given matrix */
54  for(i = 0;i < n;i++){
55    determinant *= a[i][i];
56  }
57
58  /* output the value of determinant */
59  printf("determinant is %lf\n", determinant);
60
```

```
  free_dmatrix(a);
  free_ivector(ipiv);
}
```

このプログラムを用いて Vandermonde 行列の行列式を計算した。今回は 5 行 5 列で

$$(x_1, x_2, x_3, x_4, x_5) = (1, 2, 3, 4, 5)$$

の Vandermonde 行列を用いた。5 行 5 列程度の行列なら手打ちで成分を書きだすこともできなくはないが、もっと大きなサイズの Vandermonde 行列を作るとすると手打ちでは苦しい。なので $(x_1, x_2, \cdots, x_n)$ を指定して Vandermonde 行列を生成するプログラムも作成した。

ソースコード 2　Vandermonde 行列を生成するプログラム

```c
/* output vandermonde matrix */

#include<stdio.h>
#include<math.h>

int main(void){

  double x[5] = {1.0, 2.0, 3.0, 4.0, 5.0}; //choose numbers you like
  double a[5][5];  // depends on the size of x[]
  int i, j, n;
  n = 5; // depends on the size of x[]


  printf("%d %d\n", n, n); //output the size of matrix

  for (i = 0;i < n;i++){
    for (j = 0;j < n;j++){
      a[i][j] = pow(x[j], i);
      if(j < n - 1){
        printf("%lf ", a[i][j]);
      }else{
        printf("%lf\n", a[i][j]);
      }
    }
  }

  return 0;
}
```

行列のサイズが大きくなると Vandermonde 行列の行列式も手計算では一苦労なのでこれも $(x_1, x_2, \cdots, x_n)$ を指定して自動で計算するプログラムを作成した。

ソースコード 3　Vandermonde 行列の行列式の厳密な値を計算するプログラム

```c
/* calculate theoretical value of vandermonde determinant */
```

```
2
3  #include<stdio.h>
4
5  int main(void){
6
7    int i, j, n;
8
9    double determinant;
10   determinant = 1.0;
11
12   double x[] = {1.9, 2.0, 3.1, 4.2, 5.4}; //choose numbers you like
13
14   n = sizeof(x) / 8; //n is the size of x[]
15
16   for(i=0;i<n;i++){
17     for(j=i+1;j<n;j++){
18
19       determinant *= x[j] - x[i]; //calculate the determinant following useful formula
20
21     }
22   }
23
24   printf("%lf\n", determinant); //output the value of the determinant
25
26   return 0;
27 }
```

以上のプログラムを用いて数値計算した行列式と厳密な値とを比較した。

1.2 結果

数値計算では-288 となったが、厳密な行列式の値を計算すると 288 となった。数値計算した値は厳密な値と比べて符号が反転していた。

1.3 考察 (LU 分解のアルゴリズム)

符号が反転した理由は数値計算の際に用いた LU 分解のアルゴリズムを考えることで理解できる。

2 基本課題 EX3-2

2.1 実験概要

LU 分解を用いて Dirichlet 境界条件の下での二次元 Laplace 方程式の解を求めるプログラムを作成した。

ソースコード 4 Laplace 方程式を解くための行列を求めるプログラム

```
1  //方程式の行列とベクトルを計算 laplaceAb
```

```
2
3  #include"matrix_util.h"
4
5  #include<stdio.h>
6  #include<math.h>
7
8  int main(void){
9
10   int i, j, k, l, m, n;
11   double h;
12   double  **a;
13   double *b;
14   n = 30;  // boundaries quantumed by n
15   m = n*n + 2*n; //the number of lattice points -1
16   h = 1.0 / n; //step
17
18
19   /* allocate matrix a and vector b */
20   a = alloc_dmatrix(m+1, m+1);
21   b = alloc_dvector(m+1);
22
23   /* calculate matrix a */
24   for(i=0;i<=m;i++){  //make all matrix elements 0
25     for(j=0;j<=m;j++){
26       a[i][j] = 0;
27     }
28   }
29   //calculate non-zero elements
30   for(i=0;i<=n;i++){
31     a[i][i] = 1;
32   }
33
34   for(k=0;k<=n;k++){
35     for(l=0;l<=n;l++){
36       if((k==(l+1))  && (k!=n)){
37         for(i=1;i<=n-1;i++){
38           a[k*(n+1) + i][l*(n+1) + i] = 1;
39         }
40       }else if((l==(k+1)) && (k!=0)  ){
41         for(i=1;i<=n-1;i++){
42           a[k*(n+1) + i][l*(n+1) + i] = 1;
43         }
44       }else if(k==l && (k!=0) && (k!=n)){
45         a[k*(n+1)][l*(n+1)] = 1;
```

```
        a[k*(n+1) + n][l*(n+1) + n] = 1;
        for(i=1;i<=n-1;i++){
          a[k*(n+1) + i][l*(n+1) + i] = -4;
          a[k*(n+1) + i][l*(n+1) + i+1] = 1;
          a[k*(n+1) + i][l*(n+1) + i-1] = 1;
        }
      }
    }
  }


  for(i=0;i<=n;i++){
    a[n*(n+1) + i][n*(n+1) + i] = 1;
  }









  /* output matrix a */

  printf("%d %d\n", m+1, m+1); //output the size of matrix

  for (i = 0;i <= m;i++){
    for (j = 0;j <= m;j++){
      if(j <= m - 1){
        printf("%lf ", a[i][j]);
      }else{
        printf("%lf\n", a[i][j]);
      }
    }
  }

  /* calculate vector b */
  for(k=0;k<=n;k++){
    for(i=0;i<=n;i++){
      if(i==0){
        b[k*(n+1) + i] = sin(M_PI*h*k);
      }else{
        b[k*(n+1) + i] = 0;
      }
    }
  }

```

```
90
91    /* output vector b */
92    printf("%d\n", m+1, m+1); //output the size of vector
93    for (i=0;i<=m;i++){
94      printf("%lf\n", b[i]);
95    }
96
97
98    free_dmatrix(a);
99    free_dvector(b);
100
101
102   return 0;
103 }
```

ソースコード 5　LU 分解で Laplace 方程式の解を求めるプログラム

```
1  #include "matrix_util.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  /* http://www.netlib.org/lapack/explore-html/d3/d6a/dgetrf_8f.html */
7  extern void dgetrf_(int *M, int *N, double *A, int *LDA, int*IPIV, int *INFO);
8
9  /* http://www.netlib.org/lapack/explore-html/d6/d49/dgetrs_8f.html */
10 extern void dgetrs_(char *TRANS, int *N, int *NRHS, double *A, int *LDA, int *IPIV,
11                      double *B, int *LDB, int *INFO);
12
13 int main(int argc, char** argv) {
14   char* filename;
15   FILE *fp;
16
17   clock_t start, end;
18   int i, k, l, m, n;
19   double h, x, y, z;
20   double **a;
21   double *b;
22
23   int *ipiv;
24   int info;
25   char trans = 'T';
26   int nrhs = 1;
27
28   if (argc < 2) {
```

```
29      fprintf(stderr, "Usage: %s inputfile\n", argv[0]);
30      exit(1);
31    }
32    filename = argv[1];
33
34    /* read matrix A and vector B from a file */
35    fp = fopen(filename, "r");
36    if (fp == NULL) {
37      fprintf(stderr, "Error: file can not open\n");
38      exit(1);
39    }
40    read_dmatrix(fp, &m, &n, &a);
41    if (m != n) {
42      fprintf(stderr, "Error: inconsistent number of equations\n");
43      exit(1);
44    }
45    read_dvector(fp, &n, &b);
46    if (m != n) {
47      fprintf(stderr, "Error: inconsistent number of equations\n");
48      exit(1);
49    }
50
51    start = clock();
52
53    /* perform LU decomposition */
54    ipiv = alloc_ivector(n);
55    dgetrf_(&n, &n, &a[0][0], &n, &ipiv[0], &info);
56    if (info != 0) {
57      fprintf(stderr, "Error: LAPACK::dgetrf failed\n");
58      exit(1);
59    }
60
61    /* solve equations */
62    dgetrs_(&trans, &n, &nrhs, &a[0][0], &n, &ipiv[0], &b[0], &n, &info);
63    if (info != 0) {
64      fprintf(stderr, "Error: LAPACK::dgetrs failed\n");
65      exit(1);
66    }
67
68    end = clock();
69
70    /* output 3d plot data */
71    n = 30; //re-definition n
72    h = 1.0/n; //step
```

```
  for(k=0;k<=n;k++){
    for(i=0;i<=n;i++){
      x = h*i;
      y = h*k;
      z = b[k*(n+1) + i];
      printf("%lf %lf %lf\n", x, y, z);
    }
  }

  free_dmatrix(a);
  free_dvector(b);
  free_ivector(ipiv);
}
```

このプログラムを用いていくつかのメッシュ数で解の形と解の計算にかかった時間を記載した。

また解の計算にかかった時間がメッシュ数を増やしていくとどのように変わるかをグラフで分かりやすく示した。

2.2 実験結果

境界条件はどれも課題で与えられた通り

$$u(0,y) = \sin \pi y, u(x,0) = u(x,1) = u(1,y) = 0$$

とした。

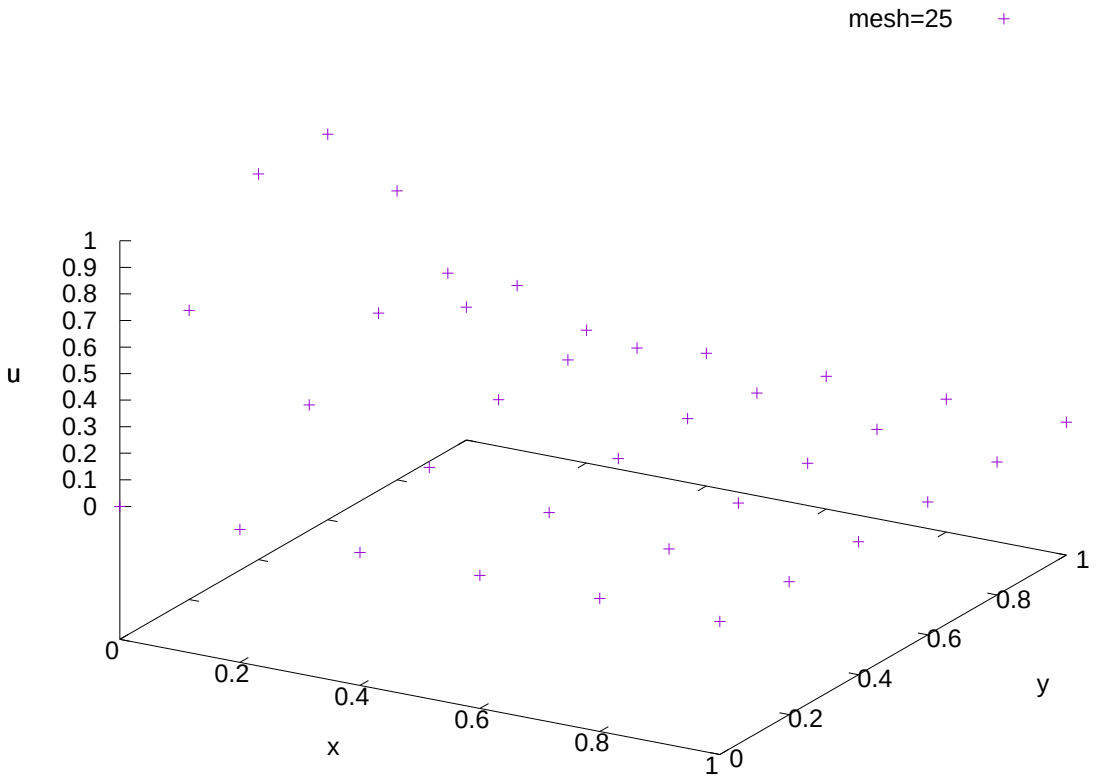

図 1　mesh 数が 25 の解。概形は分かりにくい。

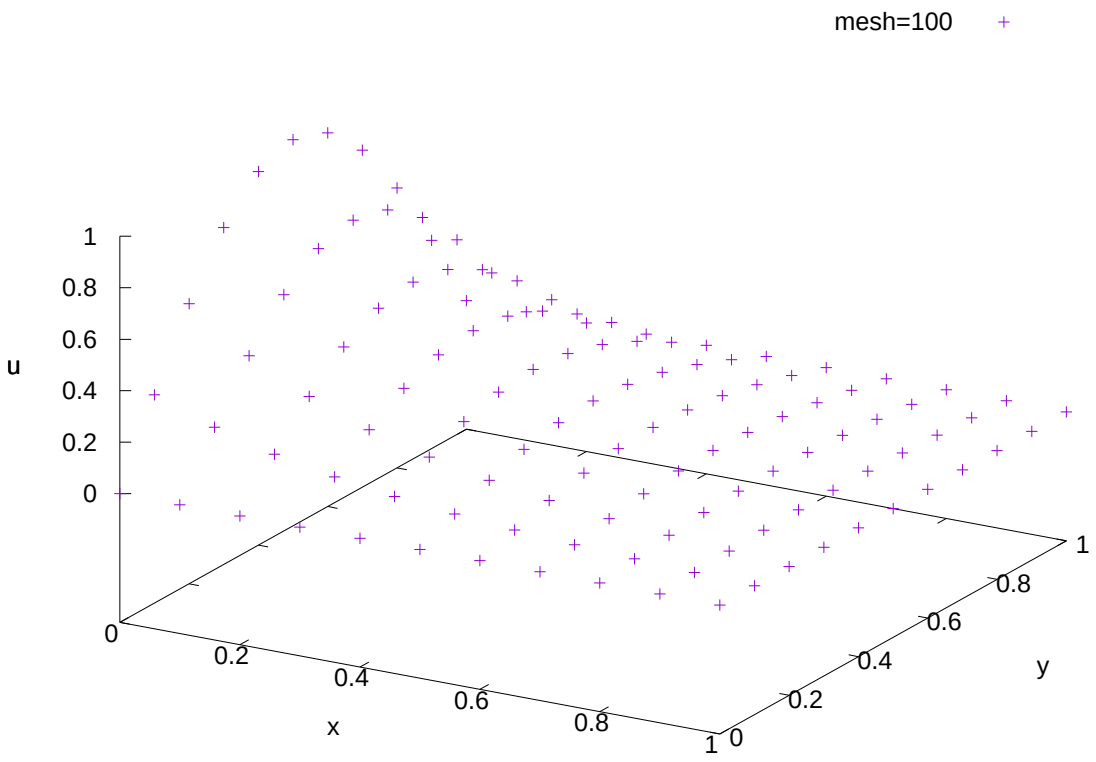

図 2　mesh 数が 100 の解。

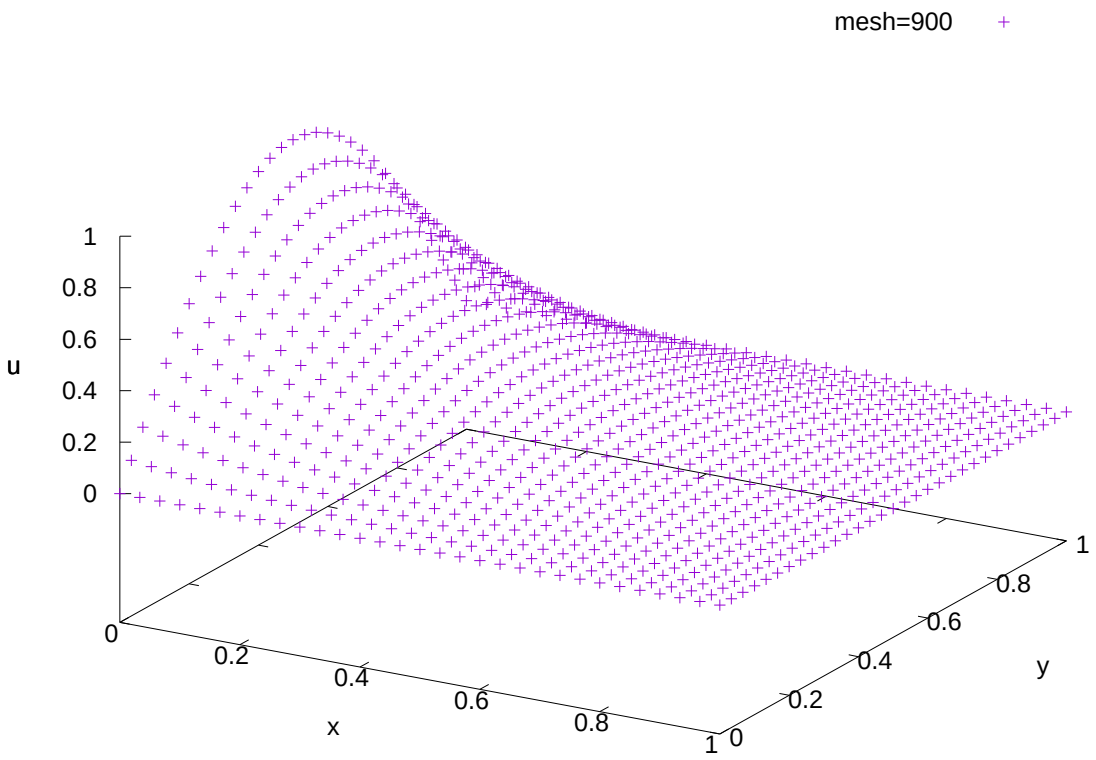

図 3　mesh 数が 900 の解。

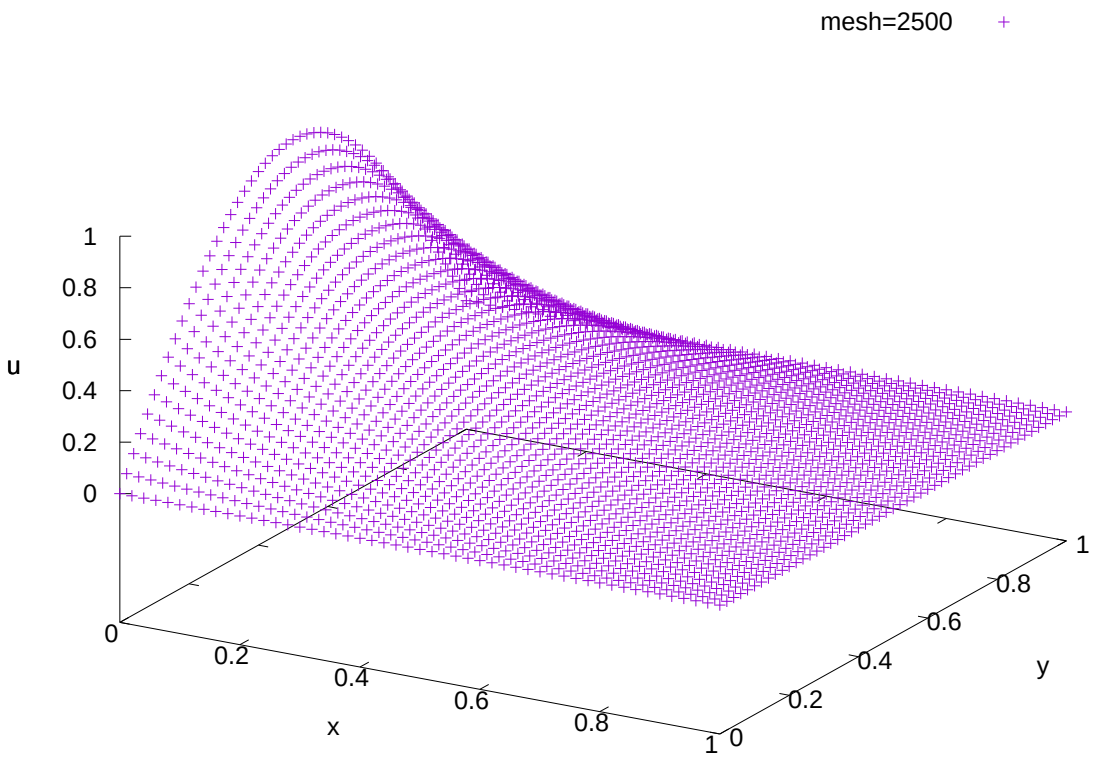

図 4　mesh 数が 2500 の解。

2.3　考察

3　基本課題 EX3-3

3.1　実験概要

3.2　実験結果

3.3　考察

4　応用課題 EX3-1

4.1　実験概要

pointer.c のソースコードを見て出力される結果を予想し、実際にコンパイルして得た出力と比較した。まず、ベクトルの方について予想される出力を考え、結果と比較する。行列の方についても同様にする。

4.2　実験結果

pointer.c のソースコードはこのようであった。

ソースコード 6　pointer.c

```c
#include "matrix_util.h"
#include <stdio.h>

int main() {
  int n, i, j;
  double *v;
  double **m;
  n = 10;

  /* test for vector */
  v = alloc_dvector(n);
  for (i = 0; i < n; ++i) v[i] = i;
  fprint_dvector(stdout, n, v);

  printf("v     = %lu\n", (long)v);
  printf("&v[0] = %lu\n", (long)&v[0]);

  printf("(v+2) = %lu\n", (long)(v+2));
  printf("&v[2] = %lu\n", (long)&v[2]);

  printf("*v       = %10.5f\n", *v);
  printf("v[0]     = %10.5f\n", v[0]);

  printf("*(v+2)   = %10.5f\n", *(v+2));
  printf("v[2]     = %10.5f\n", v[2]);

  printf("(v+2)[3] = %10.5f\n", (v+2)[3]);
  printf("*(v+2+3) = %10.5f\n", *(v+2+3));

  free_dvector(v);

  /* test for matrix */
  m = alloc_dmatrix(n, n);
  for (i = 0; i < n; ++i)
    for (j = 0; j < n; ++j)
      m[i][j] = 100 * i + j;
  fprint_dmatrix(stdout, n, n, m);

  printf("m        = %lu\n", (long)m);
  printf("&m[0]    = %lu\n", (long)&m[0]);

  printf("m[0]     = %lu\n", (long)m[0]);
  printf("&m[0][0] = %lu\n", (long)&m[0][0]);
```

```
44
45    printf("m[2]     = %lu\n", (long)m[2]);
46    printf("&m[2][0] = %lu\n", (long)&m[2][0]);
47
48    printf("m+2      = %lu\n", (long)(m+2));
49    printf("&m[2]    = %lu\n", (long)&m[2]);
50
51    printf("(*(m+2))[3] = %10.5f\n", (*(m+2))[3]);
52    printf("*(*(m+2)+3) = %10.5f\n", *(*(m+2)+3));
53    printf("m[2][3]     = %10.5f\n", m[2][3]);
54
55    printf("*(m+2)[3]   = %10.5f\n", *(m+2)[3]);
56    printf("*((m+2)[3]) = %10.5f\n", *((m+2)[3]));
57    printf("*(m[5])     = %10.5f\n", *(m[5]));
58    printf("m[5][0]     = %10.5f\n", m[5][0]);
59
60    free_dmatrix(m);
61  }
```

6 行目で v がポインタ変数として定義されている。

15 行目のプリント関数は v を出力している。v は v[0] のアドレスを表すので出力されるのは v[0] のアドレス (具体的にはわからない) であるはずだ。

16 行目は&v[0] を出力している。&v[0] は v[0] が格納されているアドレスを表すので出力されるのは 15 行目と同じ v[0] のアドレスであるはずだ。ただしアドレスは具体的にはわからないし、プログラムの実行環境によって異なる。

18 行目は v+2 を出力している。v+2 は v[2] のアドレスを表すので出力されるのは v[2] のアドレスである。*v は double 型で定義されていたので 8byte である。なので v[0] のアドレスに 16 を足したものが出力される。

19 行目は&v[2] を出力している。&v[2] は v[2] が格納されているアドレスを表すので出力されるのは 18 行目と同じ v[2] のアドレスである。

21 行目は*v を出力している。*v は v のアドレスに格納されている数値を表すので出力されるのは v[0] の値である。出力の表示桁数の指定が %10.5f になっているので出力は全体の桁数が最大で 10、小数点以下の桁数が最大 5 である。なので出力は 0.00000 である。(数値の出力は以下もこれと同じ理由で小数点以下 5 桁まで表示される。)

22 行目は v[0] を出力している。これは取りも直さず v[0] の値を表すので出力されるのは 21 行目と同じ 0.00000 である。

24 行目は*(v+2) を出力している。*(v+2) はアドレス v+2 に格納されている数値を表すので出力されるのじゃ v[2]、すなわち 2.00000 である。

25 行目は v[2] を出力している。これは取りも直さず v[2] の値なので出力は 2.00000 である。

27 行目は (v+2)[3] を出力している。これはアドレス v+2 から 3 つ (24 バイト) 進んだ先のアドレスに格納されている値を表すので、出力されるのは v[5]、すなわち 5.00000 である。

28 行目は*(v+2+3) を出力している。これはアドレス v+5 に格納されている値を表すので出力は v[5] す、すなわち 5.00000 である。

次に行列のテストについても同様に出力を予想する。

7 行目で m がポインタ変数 (ポインタのポインタ変数) として定義されている。

39 行目は m を出力している。これは m[0] のアドレスを表すので出力されるのは m[0] のアドレス。

40 行目は&m[0] を出力している。&m[0] は m[0] のアドレスを表す。出力されるのは m[0] のアドレス。

42 行目は m[0] を出力している。これは m[0][i] の配列の先頭アドレス、すなわち m[0][0] のアドレスを表す。なので出力されるのは m[0][0] のアドレス。

43 行目は&m[0][0] を出力している。これは m[0][0] のアドレスを表すので出力されるのは m[0][0] のアドレス。

45 行目は m[2] を出力している。m[2] は m[2][i] の配列の先頭アドレスを表す。なので出力されるのは m[2][0] のアドレスで配列は double 型で定義されていて (8byte)、配列のサイズが 10 なのでこのアドレスは m[0][0] のアドレスに 160 を足したものになると考えられる。

46 行目は&m[2][0] を出力している。これは m[2][0] のアドレスを表すので出力されるのは m[2][0] のアドレス。

48 行目は m+2 を出力している。これは m[2] のアドレスをあらわすので出力されるのは m[2] のアドレスである。またこれは配列の定義から m[0] のアドレスに 16 を足したものになると考えられる。

49 行目は&m[2] を出力している。これは m[2] のアドレスを表すので出力されるのは m[2] のアドレス。

51 行目は (*(m+2))[3] を出力している。ちょっと複雑なので丁寧に考える。まず、m+2 は m[2] のアドレスを表すのであった。そして*をポインタ変数に作用させるとそのアドレスに格納されている値を返すのであった。なのでと考えることができて、出力されるのは m[2][3] つまり 203.00000 である。

52 行目は*(*(m+2)+3) を出力している。これも丁寧に考える。まず*(m+2) は m[2] に格納されている値つまり m[2][0] のアドレスを表している。ここで v がポインタ変数であるとすると v+2 は v[2] のアドレスを表す。なので m[2]+3 は m[2][3] のアドレスを表す。同じく v がポインタ変数だとすると*v は v に格納されている値を表すので*(m[2]+3) は m[2][3] を表す。よって出力されるのは m[2][3] の値、すなわち 203.00000 である。

53 行目は m[2][3] を出力している。これは取りも直さず m[2][3] の値なので出力されるのは 203.00000 である。

55 行目は*(m+2)[3] を出力している。間接演算子*より添え字演算子 [3] の方が優先順位が高い。なのでまず (m+2)[3] について考える。m[3] は m+3 を表すのだから (m+2)[3] は m+5、つまり m[5] を表す。m[5] は m[5][i] の先頭アドレス、つまり m[5][0] のアドレスを表すので出力されるのは 500.00000 である。

56 行目は*((m+2)[3]) を出力している。(m+2)[3] は m[5] のことである。m[5] は m[5][0] のアドレスを表すので出力されるのは m[5][0]、つまり 500.00000 である。

57 行目は*(m[5]) を出力している。これはアドレス m[5] に格納されている値を表す。m[5] は m[5][0] のアドレスなので出力されるのは 500.00000 である。

58 行目は m[5][0] を出力している。これは取りも直さず m[5][0] の値を表すので出力されるのは 500.00000 である。

以上が予想される出力である。

実際に pointer.c を走らせて得られた出力をまとめた。

表 1　最大次数と残差

変数	出力結果
v	25769804768
&v[0]	25769804768
(v+2)	25769804784
&v[2]	25769804784
*v	0.00000
v[0]	0.00000
*(v+2)	2.00000
v[2]	2.00000
(v+2)[3]	5.00000
*(v+2+3)	5.00000
m	25769804768
&m[0]	25769804768
m[0]	25770100688
&m[0][0]	25770100688
m[2]	25770100848
&m[2][0]	25770100848
m+2	25769804784
&m[2]	25769804784
(*(m+2))[3]	203.00000
((m+2)+3)	203.00000
m[2][3]	203.00000
*(m+2)[3]	500.00000
*((m+2)[3])	500.00000
*(m[5])	500.00000
m[5][0]	500.00000

結果は予想と一致していた。

5　応用課題 EX3-2

6　基本課題 EX3-3

7　基本課題 EX4-1

7.1　実験概要

7.2　実験結果

7.3　考察

8　基本課題 EX4-2

8.1　実験概要

ファイル measurement.dat に収められている実験データを最小二乗法により任意の次数の多項式でフィッティングできるプログラムを作成した。このプログラムを用いて多項式の最大次数を変えた時に最も良いと考えられる最大次数を推定した。最後にその最良の最大次数の多項式を用いて実験データをフィッティングした。

8.2　実験結果

まず多項式の最良の最大次数を推定した。推定するにあたっては、各最大次数の多項式について実験データとの残差を計算し最も残差の小さい最大次数を最良であると判断した。

以下が各最大次数に対して計算した残差の表である。

表 2　最大次数と残差

次数	多項式	残差
1	$2.209534 + -0.708812\ x^1$	482.6702
2	$0.555836 + 0.318843\ x^1 + -0.102766\ x^2$	270.2837
3	$0.417774 + 0.500040\ x^1 + -0.148841\ x^2 + 0.003072\ x^3$	259.5323
4	$0.527741 + 0.240453\ x^1 + -0.027370\ x^2 + -0.016031\ x^3 + 0.000955\ x^4$	266.3886
5	$0.514604 + 0.291929\ x^1 + -0.065848\ x^2 + -0.005524\ x^3 + -0.000236\ x^4 + 0.000048\ x^5$	265.7403
6	$0.530055 + 0.195282\ x^1 + 0.040494\ x^2 + -0.049690\ x^3 + 0.008180\ x^4 + -0.000697\ x^5 + 0.000025\ x^6$	266.2998

8.3　考察

参考文献

[1] 増原英彦 + 東京大学情報教育連絡会『情報科学入門 Ruby を使って学ぶ』(東京大学出版会, 2010)

[2] http://www.cp.cmc.osaka-u.ac.jp/~kikuchi/texts/conservation.pdf