有限要素法のサンプル・プログラム 第 2.3 版

数理計算特論バージョン

桂田 祐史

2007 年 8 月, 2011 年 6 月 21 日, 2012 年 5 月 29 日

1 はじめに

この文書で説明するのは、2 次元多角形領域 Ω における Poisson 方程式の境界値問題

$$-\triangle u = f \quad (\text{in } \Omega), \tag{1}$$

$$u = 0 \quad (\text{on } \Gamma_1), \tag{2}$$

$$\frac{\partial u}{\partial n} = 0 \quad (\text{on } \Gamma_2) \tag{3}$$

を有限要素法で解くためのプログラムである。ここで f は与えられた関数、Γ_1, Γ_2 は、境界 $\Gamma := \partial\Omega$ を分割した曲線、n は Γ 上の点における Ω の外向き単位法線ベクトルである。

これは

菊地文雄、有限要素法入門、サイエンス社 (1980)

に掲載されていた Fortran プログラムを元にして作成された。

数理計算特論向けの説明: 入手法

`kikuchi-fem-mac.tar.gz`[1] を入手して (鍵は `fem`, `6701`)、自分のホームディレクトリィなどに保存して、

```
tar xzf kikuchi-fem-mac.tar.gz
```

のように展開すると、`kikuchi-fem-v2` というディレクトリィが出来る。

確認してみる

```
cd kikuchi-fem-mac
ls
```

`naive.c`, `band.c`, `input.dat`, `make-input.c` などが見えるはずである。

Windows 環境での利用を念頭に、ファイルの文字コードは Shift JIS となっている。Linux や Mac では変換が必要になるかもしれない (nkf があれば `make euc`, `make utf8` のようにして変換できる)。

[1] `http://www.math.meiji.ac.jp/~mk/suurikeisantokuron/members/kikuchi-fem-mac.tar.gz`

2　素朴なプログラム

菊地 [1] の 7 章 5 節のプログラムを C 言語に書き換えたものを説明する。

2.1　入力データの作成

次のような入力データを準備しよう (kikuchi-fem-mac にはあるはず)。

input.dat (cat input.dat あるいは type input.dat で確認可能))

```
 9    8    5
0.0       0.0
0.0       0.5
0.0       1.0
0.5       0.0
0.5       0.5
0.5       1.0
1.0       0.0
1.0       0.5
1.0       1.0
 0    3    4    0    4    1
 1    4    5    1    5    2
 3    6    7    3    7    4
 4    7    8    4    8    5
 0    1    2    3    6
```

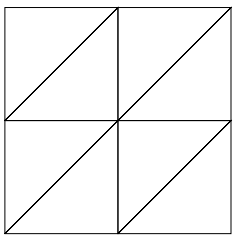

図 1: 各辺を 2 等分してから三角形分割 (input.dat の内容)

1. 1 行目には、節点数 (nnode)、要素数 nelmt、境界の Dirichlet 境界条件を課す部分 Γ_1 に属している節点数 (nbc) を書いてある。

2. その次に (上のデータでは 2 行目以降 10 行目までの 9 行) は、節点の座標 (x_i, y_i) $(i = 0, 1, \cdots, \mathtt{nnode} - 1)$ を書いてある。

3. その次に書いてあるのは (上のデータでは 11 〜 14 行目)、各要素を構成する節点の全体節点番号 (0 から nelmt−1 までの通し番号) を書いてある。ここで、節点は各要素を左回りに回るように順序付けてある (ちょっと、分かり難いけど、本文を読んで下さい)。

4. 最後に Γ_1 に属する節点の全体節点番号を書いてある。

分割数が大きい場合、人力で作成するのは面倒なので、プログラムで自動生成する。

データ作成プログラムの例 make-input.c

```
/*
 * make-input.c
 */

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

int main(void)
{
  int i, j, n;
  int elmt1, elmt11, elmt12, elmt13, elmt2, elmt21, elmt22, elmt23;
  double h;

  /* 分割数 n の入力 */
  scanf("%d", &n);
  h = 1.0 / n;

  /* 総節点数，総要素数，ディリクレ境界条件を課す節点数を表示 */
  printf("%d  %d  %d\n", (n + 1) * (n + 1), 2 * n * n, 2 * n + 1);

  /* 各節点番号に対応する座標を表示  */
  for (i = 0; i <= n; i++)
    for (j = 0; j <= n; j++)
      printf("%f  %f\n", i * h, j * h);

  /* 各要素に属する節点番号を表示 */
  for (j = 0; j < n; j++) {
    for (i = 0; i < n; i++) {
      /*  タイプ I  */
      elmt1 = 2 * (i + n * j);
      elmt11 = i + (n + 1) * j; elmt12 = elmt11 + (n + 1); elmt13 = elmt12 + 1;
      /*  タイプ I  */
      elmt2 = elmt1 + 1;
      elmt21 = elmt11; elmt22 = elmt21 + (n + 2); elmt23 = elmt21 + 1;

      printf("%d  %d  %d ", elmt11, elmt12, elmt13);
      printf("%d  %d  %d\n", elmt21, elmt22, elmt23);
    }
  }

  /* ディリクレ境界条件を課す節点番号を表示  */
  for (j = 0; j <= n; j++) printf("%d ", j);
  for (i = 1; i <= n; i++) printf("%d ", (n + 1) * i);
  printf("\n");

  return 0;
}
```

ccmg コマンドは、桂田研ノートパソコンでのコンパイル用のコマンドです。(glsc -d で代用できるかも。)

input4.dat の作り方

```
mathpc00% ccmg make-input.c
mathpc00% ./make-input  (試し運転)
2

9  8  5
0.000000  0.000000
0.000000  0.500000
0.000000  1.000000
0.500000  0.000000
0.500000  0.500000
0.500000  1.000000
1.000000  0.000000
1.000000  0.500000
1.000000  1.000000
0  3  4 0  4  1
1  4  5 1  5  2
3  6  7 3  7  4
4  7  8 4  8  5
0 1 2 3 6

mathpc00% ./make-input > input4.dat
4
mathpc00% cat input4.dat
(結果省略)
```

2.2 分割の様子を図示する — disp-glsc

disp-glsc は、前項のようなデータが、どういう領域をどのように要素分割しているか、表示するためのプログラムである。

2.2.1 オリジナルの GLSC を利用している場合

桂田研ノートパソコンでは、ccmg コマンドを用いると、オリジナルの GLSC を利用するようにコンパイル・リンクされる。

コンパイル

```
mathpc00% ccmg disp-glsc.c
```

のようにコンパイルして、

input4.dat を入力して、分割の様子を図示

```
mathpc00% ./disp-glsc input4.dat
```

あるいは

input4.dat を入力して、分割の様子を図示

```
mathpc00% cat input4.dat | ./disp-glsc
```

として実行する。

後者の使い方は、前者よりただ面倒なだけのようだが、

make-input と disp-glsc の連携プレー

```
mathpc00% ./make-input | ./disp-glsc
4
```

のような応用がある。

プログラムを終了すると[2]、DISP という名前のファイルが出力されているはずである。

DISP を PostScript 形式に変換

```
mathpc00% g_out -i DISP
```

で "DISP.i00" という名前の PostScript ファイルができる。

PostScript データ DISP.i00 を表示

mathpc00% gv DISP.i00 &　あるいは　mathpc00% gsview32 DISP.i00 &

(PostScript ファイルを表示するのに、どういうコマンドを使うかは環境に依存する。)

で画面表示、

PostScript データ DISP.i00 を印刷

```
mathpc00% lpr DISP.i00
```

で印刷出来る。

LaTeX に取り込む場合は、

```
\includegraphics[angle=90,width=10cm]{DISP.i00}
```

のようにして取り込む (90° 回転し、横幅を 10 cm に縮小)。

2.2.2 GLSCWIN を利用している場合

glsc -d コマンドを用いると、オリジナルの GLSC を利用するようにコンパイル・リンクされる、という環境を使っている人が多いようなので、それにそって説明する (6701 号室の環境では glscd コマンドを使うと良い)。

[2]オリジナルの GLSC を利用している場合、グラフィックスのウィンドウをクリックすると、g_sleep() が終了するようになっている。

```
mathpc00% glsc -d disp-glsc
```

のようにコンパイルして、

```
mathpc00% ./disp-glsc < input4.dat
```
(不等号記号 < がミソ)

あるいは

```
mathpc00% cat input4.dat | ./disp-glsc
```

として実行する。

後者の使い方は、前者よりただ面倒なだけのようだが、

```
mathpc00% ./make-input | ./disp-glsc
4
```

のような応用がある。

プログラムを終了すると[3]、DISP000.emf という名前のファイルが出力されているはずである。LaTeX 文書に取り込むためには、例えば

```
mathpc00% convert DISP000.emf DISP000.eps
```

のようにして変換する (ImageMagick がインストールされていると仮定しての話)。

disp-glsc の使用例

input4.dat の内容を disp-glsc に与えてみる。

```
mathpc00% cat input4.dat
25  32  9
0.000000  0.000000
0.000000  0.250000
0.000000  0.500000
0.000000  0.750000
0.000000  1.000000
0.250000  0.000000
0.250000  0.250000
0.250000  0.500000
0.250000  0.750000
0.250000  1.000000
0.500000  0.000000
0.500000  0.250000
0.500000  0.500000
0.500000  0.750000
```

[3]GLSCWIN とリンクしたプログラムは、普通の Windows アプリケーションなので、ウィンドウの右上に付いている×印をクリックすることでウィンドウを閉じる。

```
0.500000  1.000000
0.750000  0.000000
0.750000  0.250000
0.750000  0.500000
0.750000  0.750000
0.750000  1.000000
1.000000  0.000000
1.000000  0.250000
1.000000  0.500000
1.000000  0.750000
1.000000  1.000000
0  5  6 0  6  1
1  6  7 1  7  2
2  7  8 2  8  3
3  8  9 3  9  4
5  10  11 5  11  6
6  11  12 6  12  7
7  12  13 7  13  8
8  13  14 8  14  9
10  15  16 10  16  11
11  16  17 11  17  12
12  17  18 12  18  13
13  18  19 13  19  14
15  20  21 15  21  16
16  21  22 16  22  17
17  22  23 17  23  18
18  23  24 18  24  19
0 1 2 3 4 5 10 15 20
mathpc00% cat input4.dat | ./disp-glsc
basic data
    nnode=  25     nelmt=  32     nbc=   9     nband=   0
x,y-coordinates of nodes (節点の x,y 座標)
    i    x(i)    y(i)   i    x(i)    y(i)
   0   0.0000   0.0000   1   0.0000   0.2500
   2   0.0000   0.5000   3   0.0000   0.7500
   4   0.0000   1.0000   5   0.2500   0.0000
   6   0.2500   0.2500   7   0.2500   0.5000
   8   0.2500   0.7500   9   0.2500   1.0000
  10   0.5000   0.0000  11   0.5000   0.2500
  12   0.5000   0.5000  13   0.5000   0.7500
  14   0.5000   1.0000  15   0.7500   0.0000
  16   0.7500   0.2500  17   0.7500   0.5000
  18   0.7500   0.7500  19   0.7500   1.0000
  20   1.0000   0.0000  21   1.0000   0.2500
  22   1.0000   0.5000  23   1.0000   0.7500
  24   1.0000   1.0000
nodes of elements (各要素を構成する節点)
   0    0    5    6    1    0    6    1
   2    1    6    7    3    1    7    2
   4    2    7    8    5    2    8    3
   6    3    8    9    7    3    9    4
   8    5   10   11    9    5   11    6
  10    6   11   12   11    6   12    7
  12    7   12   13   13    7   13    8
  14    8   13   14   15    8   14    9
  16   10   15   16   17   10   16   11
  18   11   16   17   19   11   17   12
  20   12   17   18   21   12   18   13
```

```
  22    13    18    19    23    13    19    14
  24    15    20    21    25    15    21    16
  26    16    21    22    27    16    22    17
  28    17    22    23    29    17    23    18
  30    18    23    24    31    18    24    19

nodes with zero Dirichlet data (基本境界条件を課す節点)
    0     1     2     3     4     5    10    15    20
mathpc00%
```

画面に分割の様子が表示される (図 2 参照)。

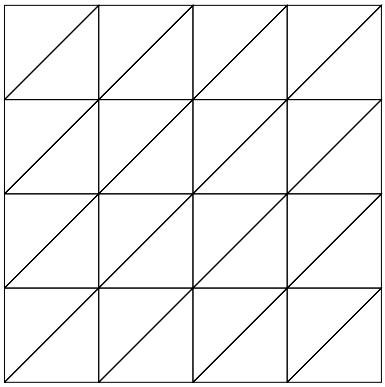

図 2: input4.dat の分割の様子

2.3 ソースプログラム naive.c

菊地 [1] 掲載の FORTRAN プログラムを、C 言語に書き直したプログラム naive.c を紹介する。

```
 1  /* naive.c -- Poisson 方程式を有限要素法で解く */
 2
 3  /*
 4   * 【参考文献】
 5   *   菊地文雄著「有限要素法概説」サイエンス社
 6   *   第 7 章の Fortran プログラムを C に書き直したもの
 7   *
 8   * 【境界値問題】
 9   * Ω は平面内の有界領域で、その境界 Γ は Γ 1, Γ 2 と二つの部分からなる。
10   * Ω 上与えられた関数 f に対して、
11   *
12   *      -Δ u=f  in Ω
13   *
14   *      u=0  on Γ 1,  ∂ u/∂ n=0  on Γ 2
15   *
16   * を満たす u=u(x,y) を求めよ。ここで n は Γ の外向き単位法線ベクトル。
17   */
18
19  #include <stdio.h>
20  #include <stdlib.h>
21  #include <math.h>
22
```

```
23  typedef struct {
24      int node[3];
25  } threenodes;
26
27  typedef int *ivector;
28  typedef double *vector;
29  typedef vector *matrix;
30
31  double  f(double, double);
32  ivector new_ivector(int);
33  vector  new_vector(int);
34  matrix  new_matrix(int, int);
35  void    errorexit(char *);
36  void    input0(FILE *, int *, int *, int *);
37  void    input(FILE *, int, int, int, threenodes *, vector, vector, ivector);
38  void    assem(int, int, int, matrix, vector,
39                threenodes *, vector, vector, ivector);
40  void    solve(matrix, vector, int);
41  void    ecm(int, threenodes *, vector, vector, double[3][3], double[3]);
42  void    output(int, vector);
43
44  /*      main program
45   *      the finite element method for the Poisson equation
46   */
47
48  int main(int argc, char **argv)
49  {
50      /*
51       * nnode          節点の総数
52       * nelmt          有限要素の総数
53       * nbc            基本境界条件 (Dirichlet 境界条件) を課す節点の総数
54       * x[], y[]       節点の座標
55       * ielmt[][3]     各有限要素を構成する節点の番号
56       * ibc[]          基本境界条件を課す節点の番号
57       * am[][]         全体行列
58       * fm[]           全体ベクトル
59       *
60       * 注意: 節点番号や要素番号は 0 からつける。よって
61       *   節点番号 0,1,...,nnode-1
62       *   要素番号 0,1,...,nelmt-1
63       * のような範囲にわたる
64       */
65      int nnode, nelmt, nbc;
66      vector x, y, fm;
67      matrix am;
68      ivector ibc;
69      threenodes *ielmt;
70      FILE *fp;
71
72      if (argc == 2)
73          fp = fopen(argv[1], "r");
74      else
75          fp = stdin;
76
77      /* nnode, nelmt, nbc を入力 */
78      input0(fp, &nnode, &nelmt, &nbc);
79
80      /* 大きな記憶領域を動的に確保 */
```

```
81      if ((ielmt = malloc(sizeof(threenodes) * nelmt)) == NULL)
82          errorexit("要素内節点の番号表 threenodes の確保が出来ませんでした。");
83      if ((ibc = new_ivector(nnode)) == NULL)
84          errorexit("基本境界条件節点の番号表 ibc の確保が出来ませんでした。");
85      if ((x = new_vector(nnode)) == NULL)
86          errorexit("節点の x 座標の表 x の確保が出来ませんでした。");
87      if ((y = new_vector(nnode)) == NULL)
88          errorexit("節点の y 座標の表 y の確保が出来ませんでした。");
89      if ((fm = new_vector(nnode)) == NULL)
90          errorexit("全体自由項ベクトル fm の確保が出来ませんでした。");
91      if ((am = new_matrix(nnode, nnode)) == NULL)
92          errorexit("全体係数行列 am の確保が出来ませんでした。");
93
94      /* 入力 */
95      input(fp, nnode, nelmt, nbc, ielmt, x, y, ibc);
96
97      /* 連立 1 次方程式を組み上げる */
98      assem(nnode, nelmt, nbc, am, fm, ielmt, x, y, ibc);
99      /* 連立 1 次方程式を解く */
100     solve(am, fm, nnode);
101     /* 結果の出力 */
102     output(nnode, fm);
103     return 0;
104 }
105
106 /* 総節点数, 総要素数, Dirichlet 境界に属する節点数を読み込む */
107 void input0(FILE *fp, int *nnode, int *nelmt, int *nbc)
108 {
109     char buf[BUFSIZ];
110     fgets(buf, sizeof(buf), fp);
111     sscanf(buf, "%d %d %d", nnode, nelmt, nbc);
112 }
113
114 /* 節点の座標,
115    要素を構成する節点の節点番号,
116    Dirichlet 境界に属する節点の全体節点番号を読み込む */
117 void input(FILE *fp, int nnode, int nelmt, int nbc,
118            threenodes *ielmt, vector x, vector y, ivector ibc)
119 {
120     int i, j;
121     /* input */
122     for (i = 0; i < nnode; i++)
123         fscanf(fp, "%lf %lf", &(x[i]), &(y[i]));
124     for (i = 0; i < nelmt; i++)
125         for (j = 0; j < 3; j++)
126             fscanf(fp, "%d", &(ielmt[i].node[j]));
127     for (i = 0; i < nbc; i++)
128         fscanf(fp, "%d", &ibc[i]);
129
130     /* output of input data */
131     printf("basic data\n    nnode=%4d     nelmt=%4d     nbc=%4d\n",
132             nnode, nelmt, nbc);
133     printf("x,y-coordinates of nodes (節点の x,y 座標)\n");
134     printf("    i    x(i)    y(i)   i    x(i)    y(i)\n");
135     for (i = 0; i < nnode; i++) {
136         printf("%4d %8.4f %8.4f", i, x[i], y[i]);
137         if (i % 2 == 1) printf("\n");
138     }
```

```
139     printf("\nnodes of elements (各要素を構成する節点)\n"
140            "    i   i1  i2  i3\n    i    i1    i2    i3\n");
141     for (i = 0; i < nelmt; i++) {
142         printf("%4d %4d %4d %4d ",
143                 i, ielmt[i].node[0], ielmt[i].node[1], ielmt[i].node[2]);
144         if (i % 2 == 1)
145             printf("\n");
146     }
147     if (nbc > 0) {
148         printf("\nnodes with zero Dirichlet data (基本境界条件を課す節点)\n");
149         for (i = 0; i < nbc; i++)
150             printf("%5d", ibc[i]);
151         if (i % 8 == 7) printf("\n");
152     }
153     printf("\n");
154 }
155
156 /* "直接剛性法" --- 連立 1 次方程式を組み上げる */
157 void assem(int nnode, int nelmt, int nbc,
158            matrix am, vector fm, threenodes *ielmt,
159            vector x, vector y, ivector ibc)
160 {
161     int i, j, ie, ii, jj;
162     /* the direct stiffness method */
163     double ae[3][3], fe[3];
164     /* initial clear */
165     for (i = 0; i < nnode; i++) {
166         fm[i] = 0.0;
167         for (j = 0; j < nnode; j++)
168             am[i][j] = 0.0;
169     }
170     /* assemblage of total matrix and vector; */
171     for (ie = 0; ie < nelmt; ie++) {
172         ecm(ie, ielmt, x, y, ae, fe);
173         for (i = 0; i < 3; i++) {
174             ii = ielmt[ie].node[i];
175             fm[ii] += fe[i];
176             for (j = 0; j < 3; j++) {
177                 jj = ielmt[ie].node[j];
178                 am[ii][jj] += ae[i][j];
179             }
180         }
181     }
182     /* the homogeneous Dirichlet condition */
183     for (i = 0; i < nbc; i++) {
184         ii = ibc[i];
185         fm[ii] = 0.0;
186         for (j = 0; j < nnode; j++)
187             am[ii][j] = am[j][ii] = 0.0;
188         am[ii][ii] = 1.0;
189     }
190 #ifdef DEBUG
191     for (i = 0;  i < nnode; i++) {
192         for (j = 0;  j < nnode; j++)
193             printf("%5g ", am[i][j]);
194         printf("  %5g\n", fm[i]);
195     }
196 #endif
```

```
197 }
198
199 /* 要素係数行列，要素自由ベクトルを計算する */
200 void ecm(int ie, threenodes *ielmt, vector x, vector y,
201          double ae[3][3], double fe[3])
202 {
203     int i, j, k;
204     double d, s;
205     /* element coefficient matrix and free vector */
206     double xe[3], ye[3], b[3], c[3];
207     /* 節点の座標を求める */
208     for (i = 0; i < 3; i++) {
209         j = ielmt[ie].node[i];
210         xe[i] = x[j];
211         ye[i] = y[j];
212     }
213     /* 要素の面積を求める */
214     d = xe[0] * (ye[1] - ye[2]) + xe[1] * (ye[2] - ye[0])
215       + xe[2] * (ye[0] - ye[1]);
216     s = fabs(d) / 2.0;
217     /* calculation of element coefficient matrix */
218     for (i = 0; i < 3; i++) {
219         /* 次の節点 j, 前の節点 k */
220         j = (i == 2) ? 0 : i + 1;
221         k = (i == 0) ? 2 : i - 1;
222         b[i] = (ye[j] - ye[k]) / d;
223         c[i] = (xe[k] - xe[j]) / d;
224     }
225     for (i = 0; i < 3; i++)
226         for (j = 0; j < 3; j++)
227             ae[i][j] = s * (b[j] * b[i] + c[j] * c[i]);
228     /* calculation of element free vector */
229     for (i = 0; i < 3; i++)
230         fe[i] = s * f(xe[i], ye[i]) / 3.0;
231 }
232
233 /* Gauss の消去法により連立 1 次方程式を解く */
234 void solve(matrix a, vector f, int m)
235 {
236     int m1, i, j, k;
237     double aa;
238     /* forward elimination; */
239     m1 = m - 1;
240     for (i = 0; i < m1; i++) {
241         for (j = i + 1; j < m; j++) {
242             aa = a[j][i] / a[i][i];
243             f[j] -= aa * f[i];
244             for (k = i + 1; k < m; k++)
245                 a[j][k] -= aa * a[i][k];
246         }
247     }
248     /* backward substitution */
249     f[m-1] /= a[m-1][m-1];
250     for (i = m - 2; i >= 0; i--) {
251          for (j = i + 1; j < m; j++)
252             f[i] -= a[i][j] * f[j];
253          f[i] /= a[i][i];
254     }
```

```
255 }
256
257 void output(int nnode, vector fm)
258 {
259     int i;
260     /* output of approximate nodal values of u */
261     printf("nodal values of u (節点での u の値)\n");
262     for (i = 0; i < 3; i++)
263         printf("   i      u    ");
264     for (i = 0; i < nnode; i++) {
265         if (i % 3 == 0) printf("\n");
266         printf("%4d %11.3e", i, fm[i]);
267     }
268     printf("\n");
269 }
270
271 double f(double x, double y)
272 {
273     return 1.0;
274 }
275
276 vector new_vector(int n)
277 {
278     return malloc(sizeof(double) * n);
279 }
280
281 ivector new_ivector(int n)
282 {
283     return malloc(sizeof(int) * n);
284 }
285
286 void del_vector(vector a)
287 {
288     free(a);
289 }
290
291 void del_ivector(ivector a)
292 {
293     free(a);
294 }
295
296 matrix new_matrix(int m, int n)
297 {
298     int i;
299     matrix a;
300     if ((a = malloc(m * sizeof(vector))) == NULL) {
301         return NULL;
302     }
303     for (i = 0; i < m; i++) {
304         if ((a[i] = new_vector(n)) == NULL) {
305             while (--i >= 0) free(a[i]);
306             free(a);
307             return NULL;
308         }
309     }
310     return a;
311 }
312
```

```
313  void errorexit(char *msg)
314  {
315      fprintf(stderr, msg);
316      exit(1);
317  }
```

2.4 naive.c 解読

主な関数には以下のようなものがある。

`main()`

`input()` 入力データ読み込み

`assem()` 全体係数行列 A, 全体自由項ベクトル f の計算 (直接剛性法)

`ecm()` 要素係数行列 A_e, 要素自由項ベクトル f_e の計算

`solve()`

`output()` 節点パラメーター (節点での解の値) を出力

`f()` Poisson 方程式 $-\triangle u = f$ の右辺の既知関数 f

`nnode`	節点の総数
`nelmt`	有限要素の総数
`nbc`	基本境界条件 (Dirichlet 境界条件) を課す節点の総数
`x[], y[]`	節点の座標
`ielmt[][3]`	各有限要素を構成する節点の番号
`ibc[]`	基本境界条件を課す節点の番号
`am[][]`	全体行列
`fm[]`	全体ベクトル

2.4.1 ecm()

`ecm()` は `assem()` の中に 1 箇所呼び出しがある。

```
void ecm(int ie, threenodes *ielmt, vector x, vector y,
         double ae[3][3], double fe[3])
```

要素番号 `ie` の三角形要素 $e = e_{\mathrm{ie}}$ における要素係数行列 A_e, 要素自由項ベクトル f_e を計算するための関数である。

`x[], y[]` は節点の座標を記憶している ($\mathrm{P_j} = $ (`x[j],y[j]`) となっている)。

```
    /* 節点の座標を求める */
    for (i = 0; i < 3; i++) {
        j = ielmt[ie].node[i];
        xe[i] = x[j];
        ye[i] = y[j];
    }
```

要素 $e = e_{\mathrm{ie}}$ に属する節点の座標を求めている。局所節点番号 i の節点 N_i の全体節点番号が `j` で、その座標が (`xe[`i`],ye[`i`]`) となる。

```
/* 要素の面積を求める */
d = xe[0] * (ye[1] - ye[2]) + xe[1] * (ye[2] - ye[0])
  + xe[2] * (ye[0] - ye[1]);
s = fabs(d) / 2.0;
```

要素 $e = e_{\mathrm{ie}}$ の面積 s を求めている。次式を思い出すと良い。

$$|e| = \frac{1}{2}\det\begin{pmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix} = \frac{1}{2}\left[(x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)\right].$$

```
/* calculation of element coefficient matrix */
for (i = 0; i < 3; i++) {
    /* 次の節点 j, 前の節点 k */
    j = (i == 2) ? 0 : i + 1;
    k = (i == 0) ? 2 : i - 1;
    b[i] = (ye[j] - ye[k]) / d;
    c[i] = (xe[k] - xe[j]) / d;
}
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        ae[i][j] = s * (b[j] * b[i] + c[j] * c[i]);
```

面積座標 $L_i(x, y) = a_i + b_i x + c_i y$ の係数について、

$$a_i := \frac{x_j y_k - x_k y_j}{2|e|}, \quad b_i := \frac{y_j - y_k}{2|e|}, \quad c_i := \frac{x_k - x_j}{2|e|} \quad ((i, j, k) \text{ は } (0, 1, 2) \text{ の偶置換})$$

が成り立つこと、「(i, j, k) は $(0, 1, 2)$ の偶置換」とは要するに

$$(i, j, k) = (0, 1, 2), (1, 2, 0), (2, 0, 1)$$

ということ、A_e の (i, j) 成分は $\langle L_j, L_i \rangle_e$ で、

$$\langle L_j, L_i \rangle_e = \iint_e \nabla L_j \cdot \nabla L_i \, dxdy = \iint_e \begin{pmatrix} b_j \\ c_j \end{pmatrix} \cdot \begin{pmatrix} b_i \\ c_i \end{pmatrix} dxdy = (b_j b_i + c_j c_i)|e|$$

であることを思い出すと、「そのまんま」であることが分かる。

```
/* calculation of element free vector */
for (i = 0; i < 3; i++)
    fe[i] = s * f(xe[i], ye[i]) / 3.0;
```

これは要素自由項ベクトル $\boldsymbol{f}_e$ の成分 $f_i^{(e)} := (f, L_i)_e$ を計算している部分であるが、f を

$$f \fallingdotseq \sum_{j=0}^{2} f(\mathrm{N}_j) L_j$$

と 1 次関数近似をするのならば、

$$f_0^{(e)} \fallingdotseq \frac{|e|}{12}(2f(\mathrm{N}_0) + f(\mathrm{N}_1) + f(\mathrm{N}_2)),$$
$$f_1^{(e)} \fallingdotseq \frac{|e|}{12}(f(\mathrm{N}_0) + 2f(\mathrm{N}_1) + f(\mathrm{N}_2)),$$
$$f_2^{(e)} \fallingdotseq \frac{|e|}{12}(f(\mathrm{N}_0) + f(\mathrm{N}_1) + 2f(\mathrm{N}_2)).$$

となるのであったから、修正すべきかも知れない。

2.4.2 assem()

この関数は有限要素法の連立 1 次方程式を組み立てるもので、1 つの問題につき 1 度だけ呼び出されます。

```
void assem(int nnode, int nelmt, int nbc,
           matrix am, vector fm, threenodes *ielmt,
           vector x, vector y, ivector ibc)
```

引数がたくさんありますが、出力は am, fm だけで、後はすべて入力です。

最初に変数の 0 クリアをします (以下見るように代入演算子 = でなく、+= を使うため、最初に変数の値を 0 にしておくことが必要です)。

その後、$\boldsymbol{A}^* = \sum_{k=0}^{N_e-1} A_k^*$, $\boldsymbol{f}^* = \sum_{k=0}^{N_e-1} \boldsymbol{f}_k^*$ を計算するため、次のようなコードがあります。

```
    /* assemblage of total matrix and vector; */
    for (ie = 0; ie < nelmt; ie++) {
        ecm(ie, ielmt, x, y, ae, fe);
        for (i = 0; i < 3; i++) {
            ii = ielmt[ie].node[i];
            fm[ii] += fe[i];
            for (j = 0; j < 3; j++) {
                jj = ielmt[ie].node[j];
                am[ii][jj] += ae[i][j];
            }
        }
    }
```

要素番号 ie に関する for 文で、すべての要素についての処理をするわけです。要素番号 ie の要素の要素係数行列 $A_{\mathtt{ie}}$, 要素自由項ベクトル $\boldsymbol{f}_{\mathtt{ie}}$ を求め、変数 am, fm に加えていきます。この段階で、am は $\boldsymbol{A}^*$, fm は $\boldsymbol{f}^*$ になっています。

先日の講義での説明では、この後、

A^{**} (A^* から、 $P_i \in \Gamma_1$ となるすべての i について、第 i 行を削除))
$\boldsymbol{f}^{**}$ ($\boldsymbol{f}^*$ から、 $P_i \in \Gamma_1$ となるすべての i について、第 i 成分を削除)

を経て、

A (A^* から、 $P_i \in \Gamma_1$ となるすべての i について、第 i 行、第 i 列を削除したもの)
$\boldsymbol{f}$ ($\boldsymbol{f}^{*^*}$ から、$\displaystyle\sum_{P_i \in \Gamma_1} \boldsymbol{a}_i g_1(P_i)$ を引いたもの)

を求めた。

この後プログラムでは、$\mathrm{P}_i \in \Gamma_1$ となる i について、A^* の第i行, 第i列を $\boldsymbol{e}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ に、$\boldsymbol{f}^*$ の第 i 成分を 0 に置き換えている。これは第 i 行目に $u_i = 0$ という方程式を設定することになる (同次 Dirichlet 境界条件 $u = 0$ on Γ_1 に対応している)。

```
    /* the homogeneous Dirichlet condition */
    for (i = 0; i < nbc; i++) {
        ii = ibc[i];
        fm[ii] = 0.0;
        for (j = 0; j < nnode; j++)
            am[ii][j] = am[j][ii] = 0.0;
        am[ii][ii] = 1.0;
    }
}
```

3 係数行列が帯行列であることを利用したプログラム band

菊地 [1] 7 章 7 節には、係数行列が帯行列であることを利用して、効率的に計算するプログラムが載っている。

3.1 ソース・プログラム band.c

菊地 [1] 掲載の FORTRAN プログラムを、C 言語に書き直したプログラム `band.c` を用意したが、紙の節約のため、ここでは説明を省略する。`kikuchi-fem-mac` には入っている。

3.2 band.c 解読

係数行列の半バンド幅 m, すなわち

$$|i - j| > m \quad \Longrightarrow \quad a_{ij} = 0$$

を満たす m を記憶する変数 `nband` が増えている。係数行列は対称性も持っているので、一つの行につき記憶すべき成分は m 個ということになる。`am[nnode][nnode]` が `am[nnode][nband]` になっている。

`ecm()` は `naive.c` と全く違いがない。`assem()` と `solve()` が変更されている。

3.2.1 assem()

引数並びには nband が増えている。

```
void assem(int nnode, int nelmt, int nbc, int nband, matrix am, vector fm,
           threenodes *ielmt, vector x, vector y, ivector ibc)
```

行列 $A = (a_{ij})$ の対角線より上 (対角線を含む) を記憶する、a_{ij} $(j \geq i)$ を am[i][j-i] に記憶する、というのが要点である。

```
    /* assemblage of total matrix and vector; */
    for (ie = 0; ie < nelmt; ie++) {
        ecm(ie, ielmt, x, y, ae, fe);
        for (i = 0; i < 3; i++) {
            ii = ielmt[ie].node[i];
            fm[ii] += fe[i];
            for (j = 0; j < 3; j++) {
                jj = ielmt[ie].node[j];
                if ((ij = jj - ii) >= 0) am[ii][ij] += ae[i][j];
            }
        }
    }
```

```
    /* the homogeneous Dirichlet condition */
    for (i = 0; i < nbc; i++) {
        ii = ibc[i];
        fm[ii] = 0.0;
        for (j = 0; j < nband; j++) {
            if ((ij = ii - j) >= 0) am[ij][j] = 0.0;
            /* a_{ii,jj} = 0 */
            am[ii][j] = 0.0;
        }
        am[ii][0] = 1.0;
    }
}
```

3.3 入力データの自動作成

プログラム band を利用するには、係数行列の半バンド幅を入力する必要がある。
行列 $A = (a_{ij})$ が半バンド幅 m であるとは、

$$|i - j| > m \quad \Longrightarrow \quad a_{ij} = 0$$

を満たすことをいう[4]。対偶を取ると

$$a_{ij} \neq 0 \quad \Longrightarrow \quad |i-j| \leq m$$

となる。

make-band-input.c は、半バンド幅も出力するように直したバージョンである。

```
mathpc00% ccmg make-band-input.c
mathpc00% ./make-band-input
2
9  8  5  4       ← この 4 が半バンド幅（の見積り）
0.000000  0.000000
0.000000  0.500000
0.000000  1.000000
0.500000  0.000000
0.500000  0.500000
0.500000  1.000000
1.000000  0.000000
1.000000  0.500000
1.000000  1.000000
0  3  4 0  4  1
1  4  5 1  5  2
3  6  7 3  7  4
4  7  8 4  8  5
0 1 2 3 6
mathpc00% ccmg band.c
mathpc00% ./make-band-input | ./band
4
mathpc00% ccmg contour.c
mathpc00% ./contour < band.out
```

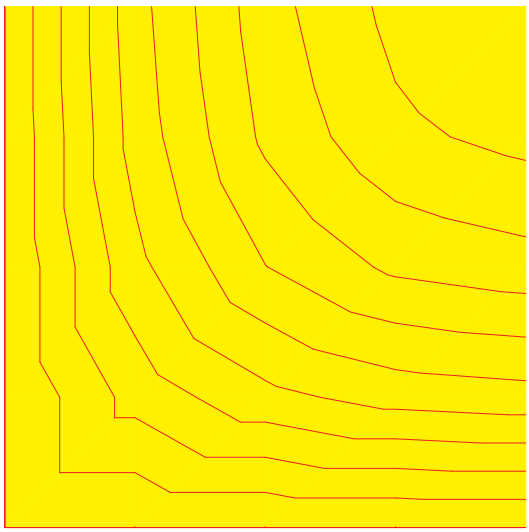

図 3: 各辺 4 等分の場合

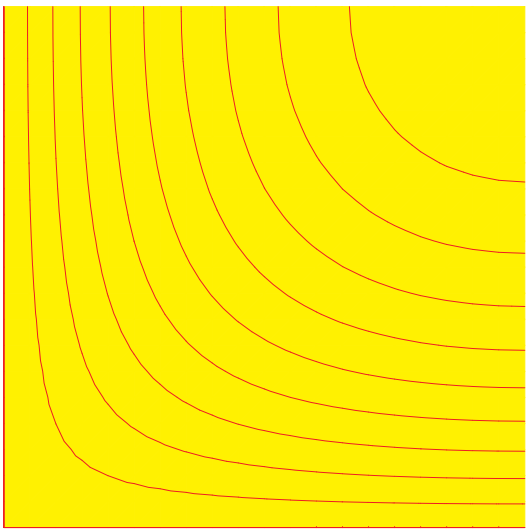

図 4: 各辺 20 等分の場合

[4] この定義によると、A が半バンド幅 3 であれば、A は半バンド幅 4 でもある。一意に決めるためには、$\max_{a_{ij}\neq 0} |i-j|$ を取るべきかもしれない。しかし、この最大値を厳密に求めるのは面倒な場合もありそうで、上のような緩く定義した方が便利かも知れない。

4 課題

次のうち (1) は必修、(2) と (3) はいずれか一方だけで良い。

(1) 正方形領域でない領域を一つ以上選び、naive または band 用のインプット・データを出力するプログラムを作成せよ。分割をどんどん細かく出来るようにして、有限要素解が収束する様子を示す図をつけること。

(2) 境界条件 $u = 0$ on Γ_1 を、非同次境界条件 $u = g_1$ on Γ_1 に変更した問題を解くためのプログラムを作成せよ (帯行列について慣れていなければ、naive の方を修正すると良い)。$\sum_{P_i \in \Gamma_1} \boldsymbol{a}_i g_1(P_i)$ を移項するということ。

(3) 境界条件 $\dfrac{\partial u}{\partial n} = 0$ on Γ_2 を、非同次境界条件 $\dfrac{\partial u}{\partial n} = g_2$ on Γ_2 に変更した問題を解くためのプログラムを作成せよ (帯行列について慣れていなければ、naive の方を修正すると良い)。線積分 $[g_2, \widehat{v}] = \displaystyle\int_{\Gamma_2} g_2 \widehat{v}\, d\sigma$ を計算するということ。

4.1 (2) について

簡単のため、`naive.c` の修正から始めると良い。

$P_i \in \Gamma_1$ となる i に対して、$g_1(P_i)$ を入力データとして用意する。例えば `input.data` を次のように書き換えることになる ($g_1 \equiv 0$ の場合)。

```
....
0 1 2 3 6
0.0 0.0 0.0 0.0 0.0    ← この行を書き加える。
```

プログラムの側は、直接剛性法の部分のコードを修正する必要がある。`assem()` に、移項の処理を書き足す。

4.2 (3) について

これも簡単のため、`naive.c` の修正から始めると良い。

Γ_2 をどうするか。ちょっと考えると、「有限要素法への入門」図 4 のケースでは、$\boxed{6, 7, 8, 5, 2}$ とすれば良いようだが、Γ_2 の連結成分が 2 以上である場合を扱えるようにするには、$\boxed{4}$, $\boxed{6, 7}$, $\boxed{7, 8}$, $\boxed{8, 5}$, $\boxed{5, 2}$ のように、辺の数 N_n と、各辺を構成する節点の番号の組 I_j, J_j $(j = 0, \ldots, N_n - 1)$ を並べたリストを入力するのが良いかもしれない。

$$[g_2, \widehat{v}] = \int_{\Gamma_2} g_2 \widehat{v}\, ds = \sum_{j=0}^{N_n - 1} \int_{P_{I_j} P_{J_j}} g_2 \widehat{v}\, ds.$$

線分 PQ において、$g_2(P)$, $g_2(Q)$, $\widehat{v}(P)$, $\widehat{v}(Q)$ が分かっているとする。線分 PQ は、$\boldsymbol{\varphi}(t) = (1-t)\overrightarrow{OP} + t\overrightarrow{OQ}$ $(t \in [0,1])$ と表される。g_2 を 1 次補間すれば、

$$
\begin{aligned}
&g_2(\boldsymbol{\varphi}(t)) = (1-t)g_2(P) + tg_2(Q),\\
&\widehat{v}(\boldsymbol{\varphi}(t)) = (1-t)\widehat{v}(P) + t\widehat{v}(Q),\\
&ds = \overline{PQ}\, dt.
\end{aligned}
$$

であるから

$$
\begin{aligned}
\int_{PQ} g_2\widehat{v}\, ds &= \overline{PQ}\int_0^1 [(1-t)g_2(P) + tg_2(Q)]\,[(1-t)\widehat{v}(P) + t\widehat{v}(Q)]\, dt\\
&= (\widehat{v}(P)\ \widehat{v}(Q))\overline{PQ}\begin{pmatrix}\frac{1}{3} & \frac{1}{6}\\ \frac{1}{6} & \frac{1}{3}\end{pmatrix}\begin{pmatrix}g_2(P)\\ g_2(Q)\end{pmatrix} = (\widehat{v}(P)\ \widehat{v}(Q))\frac{\overline{PQ}}{6}\begin{pmatrix}2g_2(P)+g_2(Q)\\ g_2(P)+2g_2(Q)\end{pmatrix}.
\end{aligned}
$$

Γ_2 に含まれる線分 P_IP_J について、節点の番号 I と J と、そこにおける g_2 の値 $g_2(P_I)$ と $g_2(P_J)$ を入力データとして与える必要が生じる。例えば次のような感じ。

```
....
0 1 2 3 6
0.0 0.0 0.0 0.0 0.0
4                          ← この行以下を書き加える。
6 7 0.0 0.0
7 8 0.0 0.0
8 5 0.0 0.0
5 2 0.0 0.0
```

プログラムの側は、直接剛性法の部分のコードを修正する必要がある。`assem()` に、線積分の処理を書き足す。

A kikuchi-fem-mac.tar.gz の README

`kikuchi-fem-v2.tar.gz (2010/6/8 バージョン)`

`1.` はじめに文字コードについて

貧弱な (?) Windows 環境での利用を鑑み、Shift JIS にしておきました。

最近の nkf があれば、SunOS, CentOS などでは make euc で euc に出来ます。
あるいは qkc があれば qkc -eu *.c としても OK です。

Windows では make sjis で Shift JIS に出来ますが、まあ nkf がないことが多いでしょう (だから最初から ShiftJIS にしておきました)。

2．デモをする

(a) SunOS, CentOS などでは、make demo で良いはず。

(b) Cygwin では、Makefile を書き変える。

```
CFLAGS = -W -Wall -O -finput-charset=cp932 -fexec-charset=cp932
```

を選択してから、また適当な CCGLSC を選択してから、make demo

(c) 二宮研パソコン（多分次のようにして動くはず）

(i) make.bat を実行する（glsc -d を使ってコンパイルする)。
(ii) make-demo.bat を実行する。

例: Cygwin で glscd コマンドを利用する場合の Makefile

```
#CFLAGS = -W -Wall -O
CFLAGS = -W -Wall -O -finput-charset=cp932 -fexec-charset=cp932

#CCGLSC = ccmg
#CCGLSC = glsc -d
CCGLSC = glscd
```

参考文献

[1] 菊地 文雄, 有限要素法概説, サイエンス社 (初版 1980, 新訂版 1999).

[2] 桂田 祐史, 有限要素法への入門, http://www.math.meiji.ac.jp/~mk/lecture/suurikeisantokuron/members/fem.pdf, 1996 年〜2011 年.
2011 年度は 6 月 14 日に配付済み。

[3] 桂田 祐史, 連立 1 次方程式 1, http://www.math.meiji.ac.jp/~mk/labo/text/linear-eq-1.pdf, 2002 年.

[4] 桂田 祐史, 数理計算特論, http://www.math.meiji.ac.jp/~mk/suurikeisantokuron/
今年度の鍵は fem, 6701