応用数理実験

1次元 Poisson 方程式に対する有限要素法

桂田 祐史

2004年1月13日

1 今日やることのあらまし

前回までの講義[1]で、1次元 Poisson 方程式 (常微分方程式と呼ぶべきかもしれないが、確かに Poisson 方程式の 1 次元版) の境界値問題

$$-u'' = f \quad (\text{in } (0,1)) \tag{1}$$

$$u(0) = \alpha, \quad u'(1) = \beta \tag{2}$$

に対する有限要素法を説明した。

そのアルゴリズムを実現したプログラム fem1d.c[2] を公開する。ただし、$f \equiv 1$ で、境界条件は同次、つまり $\alpha = \beta = 0$ の場合のプログラムとなっている。つまり

$$-u'' = 1, \quad u(0) = u'(1) = 0$$

という問題であるから、厳密解は $u(x) = x(2-x)/2$ となる。

このうち f は容易に別のものに替えられるが、境界条件を非同次の問題に対応するようにプログラムを書き換えるのを挑戦課題とする。

2 fem1d.c

2.1 ソースプログラム

```
1  /* fem1d.c -- 1 次元 Poisson 方程式を有限要素法で解く */
2
3  /*
4   * 【境界値問題】
5   * Ω=(a,b) は数直線上の領域（開区間）で、
6   * その境界 Γ は Γ 1, Γ 2 と二つの部分からなる。
7   * Ω 上与えられた関数 f に対して、
8   *
```

[1] 講義ノートは、http://www.math.meiji.ac.jp/~mk/labo/text/members/fem.pdf にある。これを読むための ID は “ouyousuurijikken”, パスワードは “ ” である。

[2] http://www.math.meiji.ac.jp/~mk/lecture/ouyousuurijikken/prog/fem1d.c

```
 9  *              -u''=f  in Ω
10  *
11  *              u=0  on Γ 1,  ∂ u/∂ n=0  on Γ 2
12  *
13  * を満たす u=u(x) を求めよ。ここで n は Γ の外向き単位法線ベクトル。
14  *
15  * f ≡ 1, Γ 1={0}, Γ 2={1} つまり -u''=1, u(0)=u'(1)=0 とする
16  *  u(x)=x(2-x)/2=-x^2/2+x
17  *
18  */
19
20  #include <stdio.h>
21  #include <stdlib.h>
22  #include <math.h>
23
24  typedef double *vector;
25  typedef vector *matrix;
26
27  double  f(double);
28  vector  new_vector(int);
29  matrix  new_matrix(int, int);
30  void    assem(int, int, int, matrix, vector, vector x, int ibc[]);
31  void    ecm(int, vector, double [2][2], double [2]);
32  void    output(int, vector);
33  void    output2(int nnode, vector x, vector u);
34  void    solve(matrix, vector, int);
35
36  /* 方程式の非同次項 */
37  double f(double x)
38  {
39      return 1.0;
40  }
41
42  /*      main program
43   *      the finite element method for the Poisson equation
44   */
45
46  int main()
47  {
48      /*
49       * nnode          節点の総数 (m+1)
50       * nelmt          有限要素の総数 (m)
51       * nbc            基本境界条件を課す節点の総数 (1 or 2)
52       * x[]            節点の座標
53       * ibc[]          基本境界条件を課す節点の番号
54       * am[][]         全体係数行列
55       * fm[]           全体ベクトル
56       *
57       * 注意: 節点番号は 0 から、要素番号は 1 からつける。よって
58       *   節点番号 0,1,...,nnode-1
59       *   要素番号 1,...,nelmt
60       * のような範囲にわたる
61       *
62       */
63
64      int ie, nnode, nelmt, nbc;
65      double h;
66      vector x, fm;
```

```
67      matrix am;
68      int ibc[2];
69
70      /* 総要素数 */
71      nelmt = 10;
72      /* 節点の総数 */
73      nnode = nelmt + 1;
74      /* ディリクレ境界にある節点の個数 */
75      nbc = 1;
76      /* メモリーの確保 */
77      x = new_vector(nnode);
78      fm = new_vector(nnode);
79      am = new_matrix(nnode, nnode);
80      /* 節点の座標（ここでは等分割する）*/
81      h = 1.0 / nelmt;
82      for (ie = 0; ie < nnode; ie++)
83        x[ie] = ie * h;
84      /* ディリクレ境界にある節点の番号 */
85      ibc[0] = 0;
86
87      assem(nnode, nelmt, nbc, am, fm, x, ibc);
88      solve(am, fm, nnode);
89      output(nnode, fm);
90      output2(nnode, x, fm);
91      return 0;
92  }
93
94  /* "直接合成法" --- 連立 1 次方程式を組み上げる */
95  void assem(int nnode, int nelmt, int nbc, matrix am, vector fm,
96             vector x, int ibc[2])
97  {
98    int i, j, ie, ii, jj;
99    /* the direct stiffness method（直接剛性法）*/
100   double ae[2][2], fe[2];
101   /* initial clear */
102   for (i = 0; i < nnode; i++) {
103     fm[i] = 0.0;
104     for (j = 0; j < nnode; j++)
105       am[i][j] = 0.0;
106   }
107   /* assemblage of total matrix and vector（全体行列, 全体ベクトルの組立）*/
108   for (ie = 1; ie <= nelmt; ie++) {
109     /* 要素番号 ie の要素の要素係数行列 ae, 要素自由ベクトル fe を計算する */
110     ecm(ie, x, ae, fe);
111     /* ae の内容を全体行列 am, fe の内容を全体ベクトル fm に算入する
112      * ii: 局所節点番号 i の節点の全体節点番号
113      * jj: 局所節点番号 j の節点の全体節点番号
114      */
115     for (i = 0; i < 2; i++) {
116       ii = ie + i - 1;
117       fm[ii] += fe[i];
118       for (j = 0; j < 2; j++) {
119 jj = ie + j - 1;
120 am[ii][jj] += ae[i][j];
121       }
122     }
123   }
124   /* the homogeneous Dirichlet condition（同次 Dirichlet 境界条件）*/
```

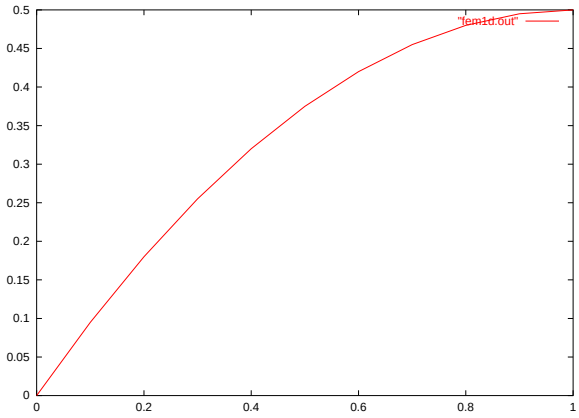
```
125   /* 番号 ibc[0]..ibc[nbc-1] の節点での値は既知であるから、
126    * その番号 (=:ii) の行と列は対角成分以外 := 0、対角成分 := 1 とする。
127    */
128   for (i = 0; i < nbc; i++) {
129     ii = ibc[i];
130     fm[ii] = 0.0;
131     for (j = 0; j < nnode; j++)
132       am[ii][j] = am[j][ii] = 0.0;
133     am[ii][ii] = 1.0;
134   }
135 }
136
137 /* 要素係数行列, 要素自由ベクトルを計算する */
138 void ecm(int ie, vector x, double ae[2][2], double fe[2])
139 {
140   double f0 = f(x[ie-1]), f1 = f(x[ie]);
141   /* ie 番目の要素 [x_{ie-1},x_{ie}] の長さ */
142   double d = x[ie] - x[ie-1];
143
144   /* 要素係数行列 A_e=(<Li,Lj>)
145    *
146    *   これは講義でもやったように
147    *
148    *            1       [ 1  -1]
149    *  A_e=------------- |      |
150    *      x[ie]-x[ie-1] [-1   1]
151    */
152
153   ae[0][0] =   1 / d; ae[0][1] = - 1 / d;
154   ae[1][0] = - 1 / d; ae[1][1] =   1 / d;
155
156   /* 要素自由項ベクトル
157    *
158    * f_e=((f,L0),(f,L1))^T
159    *
160    * 準備として
161    *  (L0,L0)=(L1,L1)=(x[ie]-x[ie-1]) ∫     x^2 dx=(x[ie]-x[ie-1])/3
162    *                                  [0,1]
163    *  (L0,L1)=(L1,L0)=(x[ie]-x[ie-1]) ∫     x(1-x) dx=(x[ie]-x[ie-1])/6
164    *                                  [0,1]
165    * f を 1 次補間する
166    *  f   f0 L0+f1 L1
167    *
168    * 1 次補間に基づいた近似
169    *  (f,L0)   f0(L0,L0)+f1(L1,L0)=(x[ie]-x[ie-1])(2f0+f1)/6
170    *  (f,L1)   f0(L0,L1)+f1(L1,L1)=(x[ie]-x[ie-1])(f0+2f1)/6
171    *
172    */
173
174   fe[0] = d * (2 * f0 + f1) / 6;
175   fe[1] = d * (f0 + 2 * f1) / 6;
176 }
177
178 /* Gauss の消去法により連立一次方程式を解く */
179 void solve(matrix a, vector f, int m)
180 {
181   int m1, i, j, k;
182   double aa;
```

```
183   /* forward elimination; */
184   m1 = m - 1;
185   for (i = 0; i < m1; i++) {
186     for (j = i + 1; j < m; j++) {
187       aa = a[j][i] / a[i][i];
188       f[j] -= aa * f[i];
189       for (k = i + 1; k < m; k++)
190         a[j][k] -= aa * a[i][k];
191     }
192   }
193   /* backward substitution */
194   f[m-1] /= a[m-1][m-1];
195   for (i = m - 2; i >= 0; i--) {
196     for (j = i + 1; j < m; j++)
197       f[i] -= a[i][j] * f[j];
198     f[i] /= a[i][i];
199   }
200 }
201
202 void output(int nnode, vector fm)
203 {
204     int i;
205     /* output of approximate nodal values of u */
206     printf("nodal values of u (節点での u の値)\n");
207     for (i = 0; i < 3; i++)
208         printf("   i      u    ");
209     for (i = 0; i < nnode; i++) {
210         if (i % 3 == 0) printf("\n");
211         printf("%4d %11.3e", i, fm[i]);
212     }
213     printf("\n");
214 }
215
216 void output2(int nnode, vector x, vector u)
217 {
218   int i;
219   FILE *fp;
220   fp = fopen("fem1d.out", "w");
221   for (i = 0; i < nnode; i++)
222     fprintf(fp, "%f %f\n", x[i], u[i]);
223   fclose(fp);
224 }
225
226 vector new_vector(int n)
227 {
228     return (double *)malloc(sizeof(double) * n);
229 }
230
231 void del_vector(vector a)
232 {
233     free(a);
234 }
235
236 matrix new_matrix(int m, int n)
237 {
238   int i;
239   matrix a;
240   if ((a = (matrix)malloc(m * sizeof(vector))) == NULL) {
```

```
241       return NULL;
242     }
243     for (i = 0; i < m; i++) {
244       if ((a[i] = new_vector(n)) == NULL) {
245         while (--i >= 0) free(a[i]);
246         free(a);
247         return NULL;
248       }
249     }
250     return a;
251   }
```

2.2 実験

```
oyabun% gcc -o fem1d fem1d.c
oyabun% ./fem1d
nodal values of u (節点での u の値)
   i       u        i       u        i       u
   0    0.000e+00    1    9.500e-02    2    1.800e-01
   3    2.550e-01    4    3.200e-01    5    3.750e-01
   6    4.200e-01    7    4.550e-01    8    4.800e-01
   9    4.950e-01   10    5.000e-01
oyabun% cat fem1d.out
0.000000 0.000000
0.100000 0.095000
0.200000 0.180000
0.300000 0.255000
0.400000 0.320000
0.500000 0.375000
0.600000 0.420000
0.700000 0.455000
0.800000 0.480000
0.900000 0.495000
1.000000 0.500000
oyabun% gnuplot
gnuplot> plot "fem1d.out" with lines
```



2.3 解説

- main() を読むと分かるように、最初に

- nnode 総節点数
- nelmt 総要素数
- nbc ディリクレ境界にある接点の個数 (1 または 2)
- x[] 節点の座標
- ibc ディリクレ境界にある接点の節点番号

を決めている。

- 連立 1 次方程式を構成するのは、関数 assem() で行っている。作業内容は 3 つに分かれる。

  1. am, fm を 0 クリアする。
  2. すべての有限要素について、要素係数行列 ae, 要素自由ベクトル fe を計算して、それぞれ全体行列 am、全体自由ベクトル fm に算入する。
  3. ディリクレ境界上にある節点に対応する部分を修正する。

- ecm() で必要となる事項の復習。$e_i = [x_{i-1}, x_i]$ とすると、

$$A_{e_i} = \frac{1}{x_i - x_{i-1}} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad f_{e_i} = \begin{pmatrix} (f, L_0)_{e_i} \\ (f, L_1)_{e_i} \end{pmatrix}$$

であったが、f は

$$f(x) \fallingdotseq f(x_{i-1})L_0(x) + f(x_i)L_1(x) \quad (x \in e_i)$$

と 1 次近似することにすれば、

$$(f, L_j)_{e_i} \fallingdotseq (f(x_{i-1})L_0 + f(x_i)L_1, L_j)_{e_i} = f(x_{i-1})(L_0, L_j)_{e_i} + f(x_i)(L_1, L_j)_{e_i}.$$

ここで

$$\begin{aligned} (L_0, L_0)_{e_i} &= (L_1, L_1)_{e_i} = (x_i - x_{i-1}) \int_0^1 x^2 \, dx = \frac{x_i - x_{i-1}}{3}, \\ (L_0, L_1)_{e_i} &= (L_1, L_0)_{e_i} = (x_i - x_{i-1}) \int_0^1 x(1-x) \, dx = \frac{x_i - x_{i-1}}{6} \end{aligned}$$

であるから、

$$f_{e_i} \fallingdotseq \frac{x_i - x_{i-1}}{6} \begin{pmatrix} 2f(x_{i-1}) + f(x_i) \\ f(x_{i-1}) + 2f(x_i) \end{pmatrix}.$$

3 やってみよう

1. 両側ディリクレ条件 $u(0) = u(1) = 0$ の問題を解いてみよう。
2. 非同次ディリクレ条件 $u(0) = \alpha$ の問題を解いてみよう。
3. 非同次 Neumann 条件 $u'(1) = \beta$ の問題を解いてみよう。
4. $-(pu')' = f$ という一般の楕円型方程式の問題をといてみよう。