応用数値解析特論 第 6 回

～2 次元 Poisson 方程式に対する有限要素法 (2)

かつらだ まさし
桂田 祐史

https://m-katsurada.sakura.ne.jp/ana2022/

2022 年 11 月 7 日

目次

本日の内容など

本日の内容など

- 今後、FreeFem++ を用いたサンプル・プログラムを用いた授業を行なうので、各自自分の Mac で動くようにしておいて下さい。インストールに関する相談に乗ります。

本日の内容など

- 今後、FreeFem++ を用いたサンプル・プログラムを用いた授業を行なうので、各自自分の Mac で動くようにしておいて下さい。インストールに関する相談に乗ります。
- 2 次元の Poisson 方程式に対する有限要素法で、連立 1 次方程式が具体的にどのようになるか説明する。

本日の内容など

- 今後、FreeFem++ を用いたサンプル・プログラムを用いた授業を行なうので、各自自分の Mac で動くようにしておいて下さい。インストールに関する相談に乗ります。
- 2 次元の Poisson 方程式に対する有限要素法で、連立 1 次方程式が具体的にどのようになるか説明する。
- 菊地 [1] 掲載のプログラムを元にしたサンプル・プログラム (C 言語で記述) の紹介

本日の内容など

- 今後、FreeFem++ を用いたサンプル・プログラムを用いた授業を行なうので、各自自分の Mac で動くようにしておいて下さい。インストールに関する相談に乗ります。
- 2 次元の Poisson 方程式に対する有限要素法で、連立 1 次方程式が具体的にどのようになるか説明する。
- 菊地 [1] 掲載のプログラムを元にしたサンプル・プログラム (C 言語で記述) の紹介
- FreeFem++ の文法の説明

5.5 連立 1 次方程式の具体例

簡単な問題に対する有限要素法の連立 1 次方程式を実際に求めてみよう。

5.5 連立１次方程式の具体例

簡単な問題に対する有限要素法の連立１次方程式を実際に求めてみよう。

$$
\begin{aligned}
&\Omega = (0,1)\times(0,1),\\
&\Gamma_1 = \{(x,y) \mid x=0, 0\le y\le 1\} \cup \{(x,y) \mid 0\le x\le 1, y=0\},\\
&\Gamma_2 = \{(x,y) \mid x=1, 0< y\le 1\} \cup \{(x,y) \mid 0< x\le 1, y=1\},\\
&g_1 \equiv 0,\quad g_2 \equiv 0,\quad f \equiv \text{定数関数 } \overline{f}.
\end{aligned}
$$

すなわち

$$
-\triangle u = \overline{f} \quad (\text{in } \Omega),\qquad u = 0 \quad \text{on } \Gamma_1,\qquad \frac{\partial u}{\partial \boldsymbol{n}} = 0 \quad \text{on } \Gamma_2.
$$

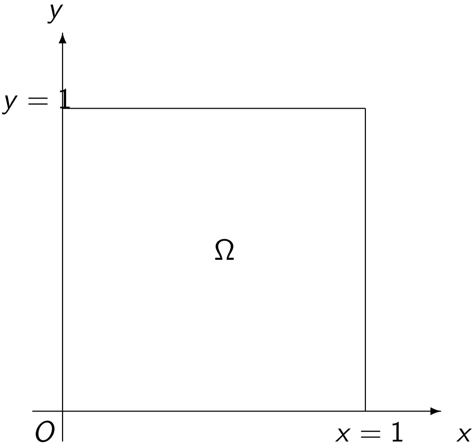


図 1: 領域 Ω

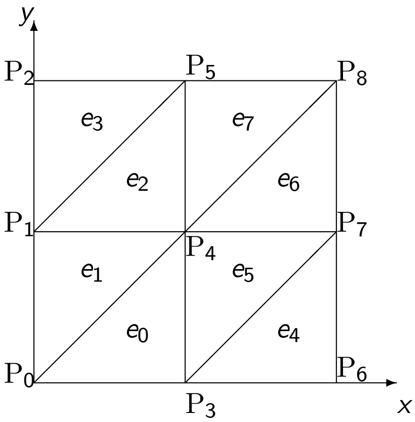


図 2: 要素分割

5.5 連立 1 次方程式の具体例

正方形領域 Ω を図 2 のように 3 角形要素によって要素分割する。

有限要素は次の二つのタイプがある (タイプ I, II と呼ぶことにする)。各々に図 3 のように局所節点番号をつける (左下から反時計回り)。

タイプ I e_0, e_2, e_4, e_6.

タイプ II e_1, e_3, e_5, e_7.

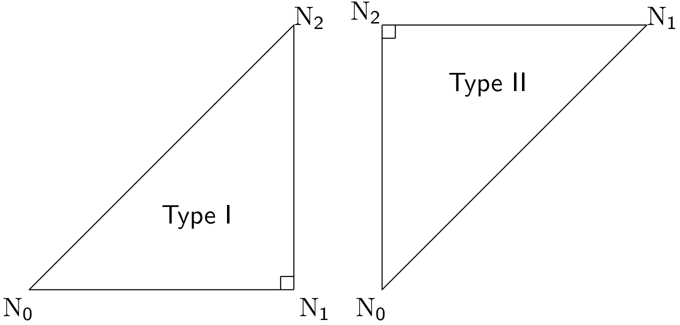


図 3: 二つのタイプの有限要素と局所節点番号

5.5 連立1次方程式の具体例

タイプが同じならば、要素係数行列 A_k, 要素自由項ベクトル $\boldsymbol{f}_k$ が等しいことはすぐ分かる。それぞれ A_{I}, A_{II}, f_{I}, f_{II} で表すことにする。

5.5 連立 1 次方程式の具体例

タイプが同じならば、要素係数行列 A_k, 要素自由項ベクトル $\boldsymbol{f}_k$ が等しいことはすぐ分かる。それぞれ A_{I}, A_{II}, f_{I}, f_{II} で表すことにする。

タイプ I については

$$D = h^2, \quad S = \frac{h^2}{2},$$

$$A_I = \frac{1}{2}\begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}, \quad \boldsymbol{f}_I = \frac{\overline{f}h^2}{6}\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

タイプ II については

$$D = h^2, \quad S = \frac{h^2}{2},$$

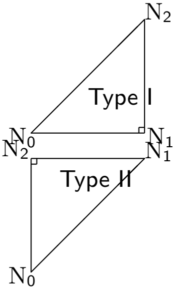
$$A_{II} = \frac{1}{2}\begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & -1 & 2 \end{pmatrix}, \quad \boldsymbol{f}_{II} = \frac{\overline{f}h^2}{6}\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

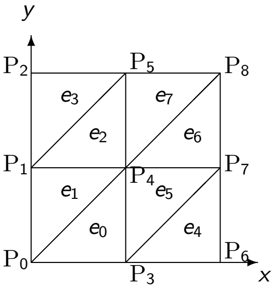
5.5 連立 1 次方程式の具体例

これらから、全体的な近似方程式を作ろう。

そのために局所的な節点番号と、全体的な節点番号の対応づけが必要である。そこで以下のような対応表を用意する (スライド見る場合も手で写すことを推奨 — 要素タイプは不要)。

要素	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7
要素タイプ	I	II	I	II	I	II	I	II
N_0 の全体節点番号	0	0	1	1	3	3	4	4
N_1 の全体節点番号	3	4	4	5	6	7	7	8
N_2 の全体節点番号	4	1	5	2	7	4	8	5





5.5 連立 1 次方程式の具体例

これから Galerkin 法の弱形式は

$$
\boldsymbol{v}^\top \frac{1}{2}
\begin{pmatrix}
2 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 4 & -1 & 0 & -2 & 0 & 0 & 0 & 0 \\
0 & -1 & 2 & 0 & 0 & -1 & 0 & 0 & 0 \\
-1 & 0 & 0 & 4 & -2 & 0 & -1 & 0 & 0 \\
0 & -2 & 0 & -2 & 8 & -2 & 0 & -2 & 0 \\
0 & 0 & -1 & 0 & -2 & 4 & 0 & 0 & -1 \\
0 & 0 & 0 & -1 & 0 & 0 & 2 & -1 & 0 \\
0 & 0 & 0 & 0 & -2 & 0 & -1 & 4 & -1 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2
\end{pmatrix}
\begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{pmatrix}
= \boldsymbol{v}^\top \frac{\overline{f} h^2}{6}
\begin{pmatrix} 2 \\ 3 \\ 1 \\ 3 \\ 6 \\ 3 \\ 1 \\ 3 \\ 2 \end{pmatrix}
$$

for

$$
\forall \boldsymbol{v} \in \left\{ (v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8)^\top \in \mathbb{R}^9 \;\middle|\; v_0 = v_1 = v_2 = v_3 = v_6 = 0 \right\}.
$$

5.5 連立 1 次方程式の具体例　Dirichlet 境界条件の考慮

$P_i \in \Gamma_1$ となる i について (今の場合 $i = 0, 1, 2, 3, 6$)、第 i 行は削除してよい。

$$\frac{1}{2}\begin{pmatrix} 0 & -2 & 0 & -2 & 8 & -2 & 0 & -2 & 0 \\ 0 & 0 & -1 & 0 & -2 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & -2 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{pmatrix}\begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{pmatrix} = \frac{\overline{f}h^2}{6}\begin{pmatrix} 6 \\ 3 \\ 3 \\ 2 \end{pmatrix}.$$

5.5 連立 1 次方程式の具体例　Dirichlet 境界条件の考慮

$P_i \in \Gamma_1$ となる i について (今の場合 $i = 0, 1, 2, 3, 6$)、第 i 行は削除してよい。

$$\frac{1}{2}\begin{pmatrix} 0 & -2 & 0 & -2 & 8 & -2 & 0 & -2 & 0 \\ 0 & 0 & -1 & 0 & -2 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & -2 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{pmatrix}\begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{pmatrix} = \frac{\bar{f} h^2}{6}\begin{pmatrix} 6 \\ 3 \\ 3 \\ 2 \end{pmatrix}.$$

また $P_i \in \Gamma_1$ となる i について、$u_i = g_1(P_i)$ $(= 0)$. これを代入して移項すると

$$\frac{1}{2}\begin{pmatrix} 8 & -2 & -2 & 0 \\ -2 & 4 & 0 & -1 \\ -2 & 0 & 4 & -1 \\ 0 & -1 & -1 & 2 \end{pmatrix}\begin{pmatrix} u_4 \\ u_5 \\ u_7 \\ u_8 \end{pmatrix} = \frac{\bar{f} h^2}{6}\begin{pmatrix} 6 \\ 3 \\ 3 \\ 2 \end{pmatrix} + \begin{pmatrix} g_1(P_1) + g_1(P_3) \\ g_1(P_2)/2 \\ g_1(P_6)/2 \\ 0 \end{pmatrix}.$$

すなわち

$$\begin{pmatrix} 4 & -1 & -1 & 0 \\ -1 & 2 & 0 & -1/2 \\ -1 & 0 & 2 & -1/2 \\ 0 & -1/2 & -1/2 & 1 \end{pmatrix}\begin{pmatrix} u_4 \\ u_5 \\ u_7 \\ u_8 \end{pmatrix} = \begin{pmatrix} \bar{f} h^2 \\ \bar{f} h^2/2 \\ \bar{f} h^2/2 \\ \bar{f} h^2/3 \end{pmatrix}.$$

5.5 連立1次方程式の具体例　Dirichlet 境界条件の考慮

上のやり方では、係数行列とベクトルが "縮小" される。いくつか留意すべき点:

- 例えば MATLAB では、$(0,1,2,3,6)$, $(4,5,7,8)$ という添字ベクトルを用意すれば、全体係数行列と全体自由項ベクトルを求めるのは (コーディング上は) 容易である。
- 自分で疎行列を扱うコードを書いていたりする場合はそれなりに面倒。
- データの移動にも計算コストがかかる。

5.5 連立 1 次方程式の具体例　Dirichlet 境界条件の考慮

上のやり方では、係数行列とベクトルが “縮小” される。いくつか留意すべき点:

- 例えば MATLAB では、(0, 1, 2, 3, 6), (4, 5, 7, 8) という添字ベクトルを用意すれば、全体係数行列と全体自由項ベクトルを求めるのは (コーディング上は) 容易である。
- 自分で疎行列を扱うコードを書いていたりする場合はそれなりに面倒。
- データの移動にも計算コストがかかる。

Dirichlet 境界条件の処理には、他のやり方 (行列とベクトルを縮小しない方法) もある。

ベクトル、行列の縮小を避ける方法 (1)

$P_i \in \Gamma_1$ となる i (この例では $i = 0, 1, 2, 3, 6$) に対して

- i 番目の方程式 ($\boldsymbol{v}^\top$ のせいで $\mathbf{0}^\top \boldsymbol{u} = 0$) を $u_i = g_1(P_i)$ で置き換える。(結果的に係数行列の第 i 行は $\boldsymbol{e}_i^\top$ で置き換える)

とすることで $\boldsymbol{v}^\top$ が外せる。

5.5 連立 1 次方程式の具体例　Dirichlet 境界条件の考慮

ベクトル、行列の縮小を避ける方法 (1)

$P_i \in \Gamma_1$ となる i (この例では $i = 0, 1, 2, 3, 6$) に対して

- i 番目の方程式 ($\boldsymbol{v}^\top$ のせいで $\boldsymbol{0}^\top \boldsymbol{u} = 0$) を $u_i = g_1(P_i)$ で置き換える。(結果的に係数行列の第 i 行は $\boldsymbol{e}_i^\top$ で置き換える)

とすることで $\boldsymbol{v}^\top$ が外せる。

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1/2 & 0 & -1 & 2 & 0 & 0 & -1/2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1/2 & 2 & -1/2 \\ 0 & 0 & 0 & 0 & 0 & -1/2 & 0 & -1/2 & 1 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{pmatrix} = \begin{pmatrix} g_1(\mathrm{P}_0) \\ g_1(\mathrm{P}_1) \\ g_1(\mathrm{P}_2) \\ g_1(\mathrm{P}_3) \\ \overline{f} h^2 \\ \overline{f} h^2/2 \\ g_1(\mathrm{P}_6) \\ \overline{f} h^2/2 \\ \overline{f} h^2/3 \end{pmatrix}$$

5.5 連立 1 次方程式の具体例　Dirichlet 境界条件の考慮

ベクトル、行列の縮小を避ける方法 (1)

$P_i \in \Gamma_1$ となる i (この例では $i = 0, 1, 2, 3, 6$) に対して

- i 番目の方程式 ($\boldsymbol{v}^\top$ のせいで $\boldsymbol{0}^\top \boldsymbol{u} = 0$) を $u_i = g_1(P_i)$ で置き換える。
(結果的に係数行列の第 i 行は $\boldsymbol{e}_i^\top$ で置き換える)

とすることで $\boldsymbol{v}^\top$ が外せる。

$$
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\
0 & 0 & -1/2 & 0 & -1 & 2 & 0 & 0 & -1/2 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & -1/2 & 2 & -1/2 \\
0 & 0 & 0 & 0 & 0 & -1/2 & 0 & -1/2 & 1
\end{pmatrix}
\begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{pmatrix}
=
\begin{pmatrix} g_1(\mathrm{P}_0) \\ g_1(\mathrm{P}_1) \\ g_1(\mathrm{P}_2) \\ g_1(\mathrm{P}_3) \\ \overline{f} h^2 \\ \overline{f} h^2/2 \\ g_1(\mathrm{P}_6) \\ \overline{f} h^2/2 \\ \overline{f} h^2/3 \end{pmatrix}
$$

これは正しい方程式であるが、係数行列が対称でなくなっている (数値計算で不利)。

ベクトル、行列の縮小を避ける方法 (1)

$P_i \in \Gamma_1$ となる i (この例では $i = 0, 1, 2, 3, 6$) に対して

- i 番目の方程式 ($\boldsymbol{v}^\top$ のせいで $\boldsymbol{0}^\top \boldsymbol{u} = 0$) を $u_i = g_1(P_i)$ で置き換える。
  (結果的に係数行列の第 i 行は $\boldsymbol{e}_i^\top$ で置き換える)

とすることで $\boldsymbol{v}^\top$ が外せる。

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1/2 & 0 & -1 & 2 & 0 & 0 & -1/2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1/2 & 2 & -1/2 \\ 0 & 0 & 0 & 0 & 0 & -1/2 & 0 & -1/2 & 1 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{pmatrix} = \begin{pmatrix} g_1(\mathrm{P}_0) \\ g_1(\mathrm{P}_1) \\ g_1(\mathrm{P}_2) \\ g_1(\mathrm{P}_3) \\ \overline{f} h^2 \\ \overline{f} h^2/2 \\ g_1(\mathrm{P}_6) \\ \overline{f} h^2/2 \\ \overline{f} h^2/3 \end{pmatrix}$$

これは正しい方程式であるが、係数行列が対称でなくなっている (数値計算で不利)。

そこで、係数行列の i 列に 0 でない非対角要素があれば、それと $g_1(P_i)$ との積を右辺に移項する。　(その結果は次のスライド)

5.5 連立1次方程式の具体例　Dirichlet 境界条件の考慮

$$
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 4 & -1 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 2 & 0 & 0 & -1/2 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 2 & -1/2 \\
0 & 0 & 0 & 0 & 0 & -1/2 & 0 & -1/2 & 1
\end{pmatrix}
\begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{pmatrix}
=
\begin{pmatrix} g_1(\mathrm{P}_0) \\ g_1(\mathrm{P}_1) \\ g_1(\mathrm{P}_2) \\ g_1(\mathrm{P}_3) \\ \overline{f} h^2 \\ \overline{f} h^2/2 \\ g_1(\mathrm{P}_6) \\ \overline{f} h^2/2 \\ \overline{f} h^2/3 \end{pmatrix}
+
\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ g_1(P_1) + g_1(P_3) \\ g_1(P_2)/2 \\ 0 \\ g_1(P_6)/2 \\ 0 \end{pmatrix}.
$$

(説明のため、Galerkin 法の弱形式を再度掲示)

$$
\boldsymbol{v}^\top \frac{1}{2}\begin{pmatrix}
2 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0\\
-1 & 4 & -1 & 0 & -2 & 0 & 0 & 0 & 0\\
0 & -1 & 2 & 0 & 0 & -1 & 0 & 0 & 0\\
-1 & 0 & 0 & 4 & -2 & 0 & -1 & 0 & 0\\
0 & -2 & 0 & -2 & 8 & -2 & 0 & -2 & 0\\
0 & 0 & -1 & 0 & -2 & 4 & 0 & 0 & -1\\
0 & 0 & 0 & -1 & 0 & 0 & 2 & -1 & 0\\
0 & 0 & 0 & 0 & -2 & 0 & -1 & 4 & -1\\
0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2
\end{pmatrix}\begin{pmatrix} u_0\\ u_1\\ u_2\\ u_3\\ u_4\\ u_5\\ u_6\\ u_7\\ u_8\end{pmatrix} = \boldsymbol{v}^\top \begin{pmatrix} f_0\\ f_1\\ f_2\\ f_3\\ f_4\\ f_5\\ f_6\\ f_7\\ f_8\end{pmatrix}
$$

for

$$
\forall \boldsymbol{v} \in \left\{ (v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8)^\top \in \mathbb{R}^9 \;\middle|\; v_0 = v_1 = v_2 = v_3 = v_6 = 0 \right\}.
$$

ベクトル、行列の縮小を避ける方法 (2) (FreeFem++で採用？)

$P_i \in \Gamma_1$ となる i に対して、行列の (i,i) 成分を “terrible great value” tgv $(=10^{30})$ で置き換え、右辺のベクトルの第 i 成分を $\mathtt{tgv} \times g_1(P_i)$ で置き換える。方程式が近似方程式に置き換わってしまうが、以下の利点がある。

- 解は実質的にほぼ変わらない (演算精度を 10 進 16 桁弱と想定してる)。
- 行列、ベクトルの縮小 (サイズ変更) は不要。
- 係数行列の対称性は保たれる。
- コーディングの負担 (手間) が少ない。

5.5 連立 1 次方程式の具体例　Dirichlet 境界条件の考慮

T を tgv $(= 10^{30})$ として

$$\boldsymbol{v}^\top \frac{1}{2} \begin{pmatrix} 2 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -2 & 0 & -1 & 0 & 0 \\ 0 & -2 & 0 & -2 & 8 & -2 & 0 & -2 & 0 \\ 0 & 0 & -1 & 0 & -2 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{pmatrix} = \boldsymbol{v}^\top \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \end{pmatrix}$$

for

$$\forall \boldsymbol{v} \in \left\{ (v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8)^\top \in \mathbb{R}^9 \;\middle|\; v_0 = v_1 = v_2 = v_3 = v_6 = 0 \right\}.$$

5.5 連立 1 次方程式の具体例　Dirichlet 境界条件の考慮

T を tgv $(= 10^{30})$ として

$$
\begin{pmatrix}
T & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & T & -1 & 0 & -2 & 0 & 0 & 0 & 0 \\
0 & -1 & T & 0 & 0 & -1 & 0 & 0 & 0 \\
-1 & 0 & 0 & T & -2 & 0 & -1 & 0 & 0 \\
0 & -2 & 0 & -2 & 8 & -2 & 0 & -2 & 0 \\
0 & 0 & -1 & 0 & -2 & 4 & 0 & 0 & -1 \\
0 & 0 & 0 & -1 & 0 & 0 & T & -1 & 0 \\
0 & 0 & 0 & 0 & -2 & 0 & -1 & 4 & -1 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2
\end{pmatrix}
\begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{pmatrix}
=
\begin{pmatrix} T g_1(P_0) \\ T g_1(P_1) \\ T g_1(P_2) \\ T g_1(P_3) \\ f_4 \\ f_5 \\ T g_1(P_6) \\ f_7 \\ f_8 \end{pmatrix}
$$

(実は、この方法が数値計算的にも妥当なものであるか、私自身は納得できていないところがある (行列の条件数が大きくなるが大丈夫？)。)

5.5 連立 1 次方程式の具体例　Dirichlet 境界条件の考慮

有限要素解を計算する (連立 1 次方程式を作る) ため、何が必要かまとめる。

上の例 ($f \equiv \overline{f}$ (定数), $g_1 \equiv 0$, $g_2 \equiv 0$) では

- 節点の座標 ($i = 1, \cdots, m$ に対して P_i の座標 (x_i, y_i))
- Γ_1 上にある節点の全体節点番号
- 各要素 e_k を構成する節点の全体節点番号 $i_{k,0}$, $i_{k,1}$, $i_{k,2}$

が必要になった。

有限要素解を計算する (連立 1 次方程式を作る) ため、何が必要かまとめる。

上の例 ($f \equiv \overline{f}$ (定数), $g_1 \equiv 0$, $g_2 \equiv 0$) では

- 節点の座標 ($i = 1, \cdots, m$ に対して P_i の座標 (x_i, y_i))
- Γ_1 上にある節点の全体節点番号
- 各要素 e_k を構成する節点の全体節点番号 $i_{k,0}$, $i_{k,1}$, $i_{k,2}$

が必要になった。

一般の問題では、次のものも必要になる。

(a) Ω に属する節点 P_i での f の値 $f(P_i)$

(b) Γ_1 上にある節点での g_1 の値

(c) Γ_2 上にある節点の全体節点番号

(d) Γ_2 上にある節点での g_2 の値

有限要素解を計算する (連立 1 次方程式を作る) ため、何が必要かまとめる。

上の例 ($f \equiv \overline{f}$ (定数), $g_1 \equiv 0$, $g_2 \equiv 0$) では

- 節点の座標 ($i = 1, \cdots, m$ に対して P_i の座標 (x_i, y_i))
- Γ_1 上にある節点の全体節点番号
- 各要素 e_k を構成する節点の全体節点番号 $i_{k,0}$, $i_{k,1}$, $i_{k,2}$

が必要になった。

一般の問題では、次のものも必要になる。

(a) Ω に属する節点 P_i での f の値 $f(P_i)$

(b) Γ_1 上にある節点での g_1 の値

(c) Γ_2 上にある節点の全体節点番号

(d) Γ_2 上にある節点での g_2 の値

以上の情報があれば、Poisson 方程式の境界値問題を解くための一般的な方程式が作成できる。

(Ω, Γ_1, Γ_2, $\{e_k\}$, $\{P_i\}$, f, g_1, g_2 などの情報は、プログラムの中に埋め込まずに入力データとして与えることが出来る。)

5.6.2 サンプルプログラム紹介

菊地 [1] にはサンプル・プログラム (FORTRAN, C 言語) も用意されている。

[1] の初版の FORTRAN プログラムを、移植した C 言語プログラムを紹介する。長いので別資料として紹介する。

1. 百聞は一見しかず。まず実行例を見てもらう。

1. 百聞は一見しかず。まず実行例を見てもらう。
2. プログラムが何をするか、入力と出力を理解する。
   有限要素解を求めるプログラム (`naive`, `band`) では、領域や三角形分割の情報を入力データとする。そのため一般性が高くなっている。

1. 百聞は一見しかず。まず実行例を見てもらう。
2. プログラムが何をするか、入力と出力を理解する。
   有限要素解を求めるプログラム (`naive`, `band`) では、領域や三角形分割の情報を入力データとする。そのため一般性が高くなっている。
3. naive と band の比較をする。数学的にはやること同じ。効率の違いは？

6 C言語による2次元要素法サンプル・プログラムの紹介
6.1 進行表

1. 百聞は一見しかず。まず実行例を見てもらう。
2. プログラムが何をするか、入力と出力を理解する。
   有限要素解を求めるプログラム (`naive`, `band`) では、領域や三角形分割の情報を入力データとする。そのため一般性が高くなっている。
3. naive と band の比較をする。数学的にはやること同じ。効率の違いは？
4. プログラムの心臓部分 `assem()` と `ecm()` の解読 (説明したことの確認)。

6.2 試しに実行

参考　授業 WWW サイトの「有限要素法のサンプル C プログラム」

入手、展開、ファイル名確認

```
curl -O https://m-katsurada.sakura.ne.jp/program/fem/fem-mac-20221031.tar.gz
tar xzf fem-mac-20221031.tar.gz
cd fem-mac-20221031
ls
```

6.2 試しに実行

参考　授業 WWW サイトの「有限要素法のサンプル C プログラム」

入手、展開、ファイル名確認

```
curl -O https://m-katsurada.sakura.ne.jp/program/fem/fem-mac-20221031.tar.gz
tar xzf fem-mac-20221031.tar.gz
cd fem-mac-20221031
ls
```

とりあえず動作チェック (実行には、cc, ccg (あるいは cglsc), make 等が必要)

コンパイル＆テスト

`make`	プログラムのコンパイル
`make test1`	naive の動作確認 (辺を 2,4,8 分割したときの有限要素解の数値データ)
`make test2`	band の動作確認 (辺を 2,4,8 分割したときの有限要素解の数値データ)
`make test3`	band の動作確認 (辺を 2,4,8,16,32 分割したときの有限要素解の等高線表示) 等高線を描いたウィンドウをクリックすると次を表示

6.2 試しに実行

参考　授業 WWW サイトの「有限要素法のサンプル C プログラム」

入手、展開、ファイル名確認

```
curl -O https://m-katsurada.sakura.ne.jp/program/fem/fem-mac-20221031.tar.gz
tar xzf fem-mac-20221031.tar.gz
cd fem-mac-20221031
ls
```

とりあえず動作チェック (実行には、cc, ccg (あるいは cglsc), make 等が必要)

コンパイル＆テスト

`make`	プログラムのコンパイル
`make test1`	naive の動作確認 (辺を 2,4,8 分割したときの有限要素解の数値データ)
`make test2`	band の動作確認 (辺を 2,4,8 分割したときの有限要素解の数値データ)
`make test3`	band の動作確認 (辺を 2,4,8,16,32 分割したときの有限要素解の等高線表示) 等高線を描いたウィンドウをクリックすると次を表示

途中で引っかかった場合、相談して下さい。

もしかすると、今の院生の Mac には ccg がインストールされていないかも。その場合は `make test3` は実行できない。

ソースプログラム `naive.c`, `band.c` は、それぞれ 321 行、397 行である。

6.3 有限要素解を求めるプログラム naive, band の理解

2 次元多角形領域 Ω における Poisson 方程式の同次 Dirichlet, Neumann 境界値問題

$$
\begin{aligned}
&(1) \qquad & -\triangle u(x,y) = f(x,y) := 1 \quad ((x,y)\in\Omega),\\
&(2) \qquad & u(x,y) = g_1(x,y) := 0 \quad ((x,y)\in\Gamma_1),\\
&(3) \qquad & \frac{\partial u}{\partial \boldsymbol{n}}(x,y) = g_2(x,y) := 0 \quad ((x,y)\in\Gamma_2)
\end{aligned}
$$

を有限要素法で解くプログラムである。

6.3 有限要素解を求めるプログラム naive, band の理解

2次元多角形領域 Ω における Poisson 方程式の同次 Dirichlet, Neumann 境界値問題

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

$$u(x,y) = g_1(x,y) := 0 \quad ((x,y) \in \Gamma_1), \tag{2}$$

$$\frac{\partial u}{\partial \boldsymbol{n}}(x,y) = g_2(x,y) := 0 \quad ((x,y) \in \Gamma_2) \tag{3}$$

を有限要素法で解くプログラムである。

Q　ここで Ω, Γ_1, Γ_2 は何か？

2 次元多角形領域 Ω における Poisson 方程式の同次 Dirichlet, Neumann 境界値問題

$$(1) \qquad -\triangle u(x,y) = f(x,y) := 1 \quad ((x,y)\in\Omega),$$

$$(2) \qquad u(x,y) = g_1(x,y) := 0 \quad ((x,y)\in\Gamma_1),$$

$$(3) \qquad \frac{\partial u}{\partial \boldsymbol{n}}(x,y) = g_2(x,y) := 0 \quad ((x,y)\in\Gamma_2)$$

を有限要素法で解くプログラムである。

Q ここで Ω, Γ_1, Γ_2 は何か？

A 実は Ω, Γ_1, Γ_2 についてはデータとして入力する。

6.3 有限要素解を求めるプログラム naive, band の理解

2 次元多角形領域 Ω における Poisson 方程式の同次 Dirichlet, Neumann 境界値問題

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

$$u(x,y) = g_1(x,y) := 0 \quad ((x,y)\in\Gamma_1), \tag{2}$$

$$\frac{\partial u}{\partial \boldsymbol{n}}(x,y) = g_2(x,y) := 0 \quad ((x,y)\in\Gamma_2) \tag{3}$$

を有限要素法で解くプログラムである。

Q ここで Ω, Γ_1, Γ_2 は何か？

A 実は Ω, Γ_1, Γ_2 についてはデータとして入力する。

naive, band ともに、任意の領域＆境界についての計算ができる。

6.3 有限要素解を求めるプログラム naive, band の理解

2 次元多角形領域 Ω における Poisson 方程式の同次 Dirichlet, Neumann 境界値問題

$$
\begin{aligned}
&(1) && -\triangle u(x,y) = f(x,y) := 1 \quad ((x,y)\in\Omega),\\
&(2) && u(x,y) = g_1(x,y) := 0 \quad ((x,y)\in\Gamma_1),\\
&(3) && \frac{\partial u}{\partial \boldsymbol{n}}(x,y) = g_2(x,y) := 0 \quad ((x,y)\in\Gamma_2)
\end{aligned}
$$

を有限要素法で解くプログラムである。

Q ここで Ω, Γ_1, Γ_2 は何か?

A 実は Ω, Γ_1, Γ_2 についてはデータとして入力する。

naive, band ともに、任意の領域&境界についての計算ができる。

(f, g_1, g_2 については、簡単のため、特殊な値 1, 0, 0 が仮定されている。これを一般化するのは適度の演習問題である。)

6.3 有限要素解を求めるプログラム naive, band の理解

入力データの例 input.dat

```
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
```

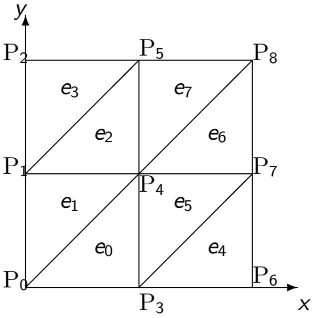


図 4: 要素分割 (各辺を 2 等分してから要素分割)

1. 1 行目には、節点数 (nnode)、要素数 (nelmt)、Γ_1 に属している節点数 (nbc)
2. 2〜10 行は、節点の座標 (x_i, y_i) $(i = 0, 1, \cdots, \mathtt{nnode} - 1)$
3. 11〜14 行は、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) 節点は各要素を左回りに回るように順序付けてある。
4. 最後に Γ_1 に属する節点の全体節点番号 (nbc 個の番号)

6.3 有限要素解を求めるプログラム naive, band の理解

この形式のデータがあれば、図が描ける (幾何的状況が分かる) ことを理解しよう。

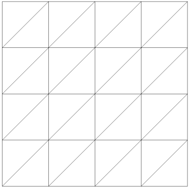
三角形と (結果として)Ω を描く

```
./disp-glsc3d input.dat
./disp-glsc3d input4.dat
cat input4.dat | ./disp-glsc3d
./make-input | ./disp-glsc3d
```

(最後のコマンドに対して、辺を何等分するか、数値 (例えば 64 とか) を入力しよう。)

コマンド 1 | コマンド 2 でコマンド 1 の出力をコマンド 2 に入力できる (パイプ機能)。

`disp-glsc3d` は上の形式のデータを図示するプログラム、`make-input` は正方形領域に対して上の形式のデータを作成するプログラムである。



6.3 有限要素解を求めるプログラム naive, band の理解

naive, band は上の形式の入力データから、有限要素解を計算するプログラム。

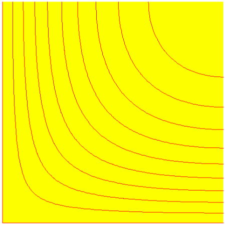
両者は同じ計算を行う。連立 1 次方程式の係数行列が**帯行列** (band matrix) であることを利用して、計算の効率化の工夫をしたのが band で、それをしないのが naive である。

一辺 64 分割で解き比べ (CPU 時間計測), 解の等高線表示

```
echo 64 | ./make-band-input > input64.dat
./disp-glsc3d input64.dat
time ./naive input64.dat
time ./band input64.dat
```

あるマシンで 17.5 秒 vs 0.02 秒　(naive は実際的ではない, ちなみに節点数 4225)

```
./contour-glsc3d band.out
```



6.4 プログラム naive の内部構造

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

6.4 プログラム naive の内部構造

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

主な変数名

`nnode`	節点の総数
`nelmt`	有限要素の総数
`nbc`	Γ_1 (Dirichlet 境界条件を課す) 上の節点の総数
`x[nnode], y[nnode]`	節点の座標
`ielmt[nelmt].node[3]`	各有限要素を構成する節点の番号
`ibc[nbc]`	基本境界条件を課す節点の番号
`am[][]`	全体係数行列
`fm[]`	全体自由項ベクトル

6.4 プログラム naive の内部構造　assem()

assem() は連立 1 次方程式を組み立てる関数。

$A^* := \sum_{k=0}^{N_e-1} A_k^*$ と $\boldsymbol{f}^* := \sum_{k=0}^{N_e-1} \boldsymbol{f}_k^*$ を次のように計算する。

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

ielmt[k].node[i] は、要素 e_k の、局所節点番号が i の節点 N_i の全体節点番号

6.4 プログラム naive の内部構造　ecm()

ecm() は要素係数行列 A_k、要素自由項ベクトル $\boldsymbol{f}_k$ を求める関数。

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

節点の座標さえ求まれば、A_k, $\boldsymbol{f}_k$ の成分は公式に従って計算するだけである。(それを確かめたければ、naive.c あるいは band.c を見よ。)

6.5 参考課題

以前、FreeFem++ が使えなかった頃は、授業で次のような課題を出していた。

私は「百見は一験にしかず」と考えていて、次のような実験をすることは有益と思っているが、この科目では要求しない。

(a) このプログラムで解ける問題は、境界条件が同次境界条件

$$u = 0 \quad \text{on } \Gamma_1, \qquad \frac{\partial u}{\partial \boldsymbol{n}} = 0 \quad \text{on } \Gamma_2$$

であるが、これを非同次境界条件

$$u = g_1 \quad \text{on } \Gamma_1, \qquad \frac{\partial u}{\partial \boldsymbol{n}} = g_2 \quad \text{on } \Gamma_2$$

に変える。

(b) 自分で選んだ領域を三角形分割して、このプログラムに入力できるデータを生成するプログラムを書く。

7 FreeFem++の文法
7.1 はじめに

すでに FreeFem++ のインストール手順と簡単な解説を行ってある。

すでに FreeFem++ のインストール手順と簡単な解説を行ってある。

FreeFem++ は有限要素法によって微分方程式の数値シミュレーションを行うためのプログラミング言語、そして言語処理系の名前でもある (Hecht [2])。

インタープリターである (その点は MATLAB や Python と似ている)。

7 FreeFem++の文法
7.1 はじめに

すでに FreeFem++ のインストール手順と簡単な解説を行ってある。

FreeFem++ は有限要素法によって微分方程式の数値シミュレーションを行うためのプログラミング言語、そして言語処理系の名前でもある (Hecht [2])。
インタープリターである (その点は MATLAB や Python と似ている)。

今回は、プログラミング言語としての FreeFem++を説明する (マニュアル Hect[3] を見ても良く分からない — 少なくとも私は)。

7 FreeFem++の文法

7.1 はじめに

すでに FreeFem++ のインストール手順と簡単な解説を行ってある。

FreeFem++ は有限要素法によって微分方程式の数値シミュレーションを行うためのプログラミング言語、そして言語処理系の名前でもある (Hecht [2])。
インタープリターである (その点は MATLAB や Python と似ている)。

今回は、プログラミング言語としての FreeFem++を説明する (マニュアル Hect[3] を見ても良く分からない — 少なくとも私は)。

FreeFem++のことを「有限要素法専用ツール」と考える人もいる。確かに有限要素法に便利な命令が組み込まれているが、それ以外の目的のプログラミングに必要な機能も十分に備わっている (実際、有限体積法や差分法のプログラムも記述可能である)。効率を度外視すれば、C のようなプログラミング言語で出来ることは FreeFem++でも出来る、と考えよう。

7 FreeFem++の文法

7.1 はじめに

すでに FreeFem++ のインストール手順と簡単な解説を行ってある。

FreeFem++ は有限要素法によって微分方程式の数値シミュレーションを行うためのプログラミング言語、そして言語処理系の名前でもある (Hecht [2])。
インタープリターである (その点は MATLAB や Python と似ている)。

今回は、プログラミング言語としての FreeFem++を説明する (マニュアル Hect[3] を見ても良く分からない — 少なくとも私は)。

FreeFem++のことを「有限要素法専用ツール」と考える人もいる。確かに有限要素法に便利な命令が組み込まれているが、それ以外の目的のプログラミングに必要な機能も十分に備わっている (実際、有限体積法や差分法のプログラムも記述可能である)。効率を度外視すれば、C のようなプログラミング言語で出来ることは FreeFem++でも出来る、と考えよう。

文法は、C++に似ている (ゆえに C にも似ている)。C しか知らない人は、C++のストリームを使った入出力 (cout, cin の利用) を調べておくこと。
参考: FreeFem++ は C++ で記述されている。

7.1 はじめに

すでに FreeFem++ のインストール手順と簡単な解説を行ってある。

FreeFem++ は有限要素法によって微分方程式の数値シミュレーションを行うためのプログラミング言語、そして言語処理系の名前でもある (Hecht [2])。
インタープリターである (その点は MATLAB や Python と似ている)。

今回は、プログラミング言語としての FreeFem++を説明する (マニュアル Hect[3] を見ても良く分からない — 少なくとも私は)。

FreeFem++のことを「有限要素法専用ツール」と考える人もいる。確かに有限要素法に便利な命令が組み込まれているが、それ以外の目的のプログラミングに必要な機能も十分に備わっている (実際、有限体積法や差分法のプログラムも記述可能である)。効率を度外視すれば、C のようなプログラミング言語で出来ることは FreeFem++でも出来る、と考えよう。

文法は、C++に似ている (ゆえに C にも似ている)。C しか知らない人は、C++のストリームを使った入出力 (cout, cin の利用) を調べておくこと。
参考: FreeFem++ は C++ で記述されている。

マニュアル Hect[3] は事例集の性格が強く、言語仕様は整理した形では載っていない。以下の説明は、個人的なノートである桂田 [4] に基づく。

7.1 はじめに　基本的な Poisson 方程式のプログラム

```
// poisson.edp
// 境界の定義 (単位円), いわゆる正の向き
border Gamma(t=0,2*pi) { x=cos(t); y=sin(t); }
// 三角形要素分割を生成 (境界を 50 に分割)
mesh Th = buildmesh(Gamma(50));
plot(Th,wait=true); // plot(Th,wait=true,ps="Th.eps");
// 有限要素空間は P1 (区分的 1 次多項式) 要素
real [int] levels =-0.012:0.001:0.012;
fespace Vh(Th,P1);
Vh u,v;
// Poisson 方程式 -△ u=f の右辺
func f = x*y;
// 問題を解く
solve Poisson(u,v)
 = int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))-int2d(Th)(f*v)
   +on(Gamma,u=0);
// 可視化 (等高線)
plot(u,wait=true);
//plot(u,viso=levels,fill=true,wait=true);
// 可視化 (3 次元) --- マウスで使って動かせる
plot(u,dim=3,viso=levels,fill=true,wait=true);
```

→ 独特の命令ばかりで、汎用のプログラミング言語の機能があることは分かりにくい。

7.2 汎用のプログラミング機能

7.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

7.2 汎用のプログラミング機能

7.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

- // から行末までは注釈、/* と */ で挟まれた部分は注釈 (共に C 言語と同じ)

改めて数えるととても多い。

- // から行末までは注釈、/* と */ で挟まれた部分は注釈 (共に C 言語と同じ)
- 文の最後は ;

7.2 汎用のプログラミング機能

7.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

- // から行末までは注釈、/* と */ で挟まれた部分は注釈 (共に C 言語と同じ)
- 文の最後は ;
- 四則演算 (+, -, *, /) や、代入 (=) などの演算子

改めて数えるととても多い。

- // から行末までは注釈、/* と */ で挟まれた部分は注釈 (共に C 言語と同じ)
- 文の最後は ;
- 四則演算 (+, −, ∗, /) や、代入 (=) などの演算子
- 0 は偽、0 以外の整数は真とみなす。一方、比較演算・論理演算などの結果は 0 (false) または 1 (true).

改めて数えるととても多い。

- // から行末までは注釈、/* と */ で挟まれた部分は注釈 (共に C 言語と同じ)
- 文の最後は ;
- 四則演算 (+, −, ∗, /) や、代入 (=) などの演算子
- 0 は偽、0 以外の整数は真とみなす。一方、比較演算・論理演算などの結果は 0 (false) または 1 (true).
- 変数宣言の文法も C 言語と同様。型名の後に , で区切った名前のリストを書く。

改めて数えるととても多い。

- // から行末までは注釈、/* と */ で挟まれた部分は注釈 (共に C 言語と同じ)
- 文の最後は ;
- 四則演算 (+, −, *, /) や、代入 (=) などの演算子
- 0 は偽、0 以外の整数は真とみなす。一方、比較演算・論理演算などの結果は 0 (false) または 1 (true).
- 変数宣言の文法も C 言語と同様。型名の後に , で区切った名前のリストを書く。
- 関数呼び出しの文法も C 言語と同様。

7.2 汎用のプログラミング機能

7.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

- // から行末までは注釈、/* と */ で挟まれた部分は注釈 (共に C 言語と同じ)
- 文の最後は ;
- 四則演算 (+, -, *, /) や、代入 (=) などの演算子
- 0 は偽、0 以外の整数は真とみなす。一方、比較演算・論理演算などの結果は 0 (false) または 1 (true).
- 変数宣言の文法も C 言語と同様。型名の後に , で区切った名前のリストを書く。
- 関数呼び出しの文法も C 言語と同様。
- ブロックは { と } で複数 (0 個以上) の文を囲んで作る。

改めて数えるととても多い。

- `//` から行末までは注釈、`/*` と `*/` で挟まれた部分は注釈 (共に C 言語と同じ)
- 文の最後は `;`
- 四則演算 (+, −, ∗, /) や、代入 (=) などの演算子
- 0 は偽、0 以外の整数は真とみなす。一方、比較演算・論理演算などの結果は 0 (false) または 1 (true).
- 変数宣言の文法も C 言語と同様。型名の後に，で区切った名前のリストを書く。
- 関数呼び出しの文法も C 言語と同様。
- ブロックは { と } で複数 (0 個以上) の文を囲んで作る。
- 比較演算子 (==, !=, <, <=, >, >=)、論理演算子 (&&, ||, !)、`if`, `if else` などの制御構造。
  ただし `switch` はない。

改めて数えるととても多い。

- `//` から行末までは注釈、`/*` と `*/` で挟まれた部分は注釈 (共に C 言語と同じ)
- 文の最後は `;`
- 四則演算 (+, -, *, /) や、代入 (=) などの演算子
- 0 は偽、0 以外の整数は真とみなす。一方、比較演算・論理演算などの結果は 0 (false) または 1 (true).
- 変数宣言の文法も C 言語と同様。型名の後に, で区切った名前のリストを書く。
- 関数呼び出しの文法も C 言語と同様。
- ブロックは { と } で複数 (0 個以上) の文を囲んで作る。
- 比較演算子 (`==`, `!=`, <, <=, >, >=)、論理演算子 (`&&`, `||`, `!`)、`if`, `if else` などの制御構造。
  ただし `switch` はない。
- `for`, `while` などの繰り返し制御。`break` (ループを抜ける), `continue` (次の繰り返し) など。
  ただし `do while` はない。

7.2 汎用のプログラミング機能

7.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

- // から行末までは注釈、/* と */ で挟まれた部分は注釈 (共に C 言語と同じ)
- 文の最後は ;
- 四則演算 (+, -, *, /) や、代入 (=) などの演算子
- 0 は偽、0 以外の整数は真とみなす。一方、比較演算・論理演算などの結果は 0 (false) または 1 (true).
- 変数宣言の文法も C 言語と同様。型名の後に , で区切った名前のリストを書く。
- 関数呼び出しの文法も C 言語と同様。
- ブロックは { と } で複数 (0 個以上) の文を囲んで作る。
- 比較演算子 (==, !=, <, <=, >, >=)、論理演算子 (&&, ||, !)、`if`, `if else` などの制御構造。
  ただし `switch` はない。
- `for`, `while` などの繰り返し制御。`break` (ループを抜ける), `continue` (次の繰り返し) など。
  ただし `do while` はない。
- 数学関数の名前

7.2 汎用のプログラミング機能

7.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

- // から行末までは注釈、/* と */ で挟まれた部分は注釈 (共に C 言語と同じ)
- 文の最後は ;
- 四則演算 (+, -, *, /) や、代入 (=) などの演算子
- 0 は偽、0 以外の整数は真とみなす。一方、比較演算・論理演算などの結果は 0 (false) または 1 (true).
- 変数宣言の文法も C 言語と同様。型名の後に , で区切った名前のリストを書く。
- 関数呼び出しの文法も C 言語と同様。
- ブロックは { と } で複数 (0 個以上) の文を囲んで作る。
- 比較演算子 (==, !=, <, <=, >, >=)、論理演算子 (&&, ||, !)、`if`, `if else` などの制御構造。
  ただし `switch` はない。
- `for`, `while` などの繰り返し制御。`break` (ループを抜ける), `continue` (次の繰り返し) など。
  ただし `do while` はない。
- 数学関数の名前

他にもあるだろう…

7.2.2 データ型

7.2.2 データ型

- 整数を表すための int がある (C 言語の `int` に相当)

7.2.2 データ型

- 整数を表すための int がある (C 言語の `int` に相当)
- 実数を表すための real がある (C 言語の `double` に相当)

7.2.2 データ型

- 整数を表すための int がある (C 言語の `int` に相当)
- 実数を表すための real がある (C 言語の `double` に相当)
- 複素数を表すための complex がある (C 言語の `complex` に相当, 実部・虚部が `double`)

7.2.2 データ型

- 整数を表すための int がある (C 言語の `int` に相当)
- 実数を表すための real がある (C 言語の `double` に相当)
- 複素数を表すための complex がある (C 言語の `complex` に相当, 実部・虚部が `double`)
- 論理を表すための bool がある (C 言語の `bool` に相当). `true`, `false` と言う値があるが、それぞれ 1, 0 の別名と考えて良い。
  例えば `plot(u,wait=true);` は `plot(u,wait=1);` と同じ。

7.2.2 データ型

- 整数を表すための int がある (C 言語の `int` に相当)
- 実数を表すための real がある (C 言語の `double` に相当)
- 複素数を表すための complex がある (C 言語の `complex` に相当, 実部・虚部が `double`)
- 論理を表すための bool がある (C 言語の `bool` に相当). `true`, `false` と言う値があるが、それぞれ 1, 0 の別名と考えて良い。
  例えば `plot(u,wait=true);` は `plot(u,wait=1);` と同じ。
- 文字列を表すための string がある (C++言語の `string` に相当, 日本語不可？)。

7.2.2 データ型

- 整数を表すための int がある (C 言語の `int` に相当)
- 実数を表すための real がある (C 言語の `double` に相当)
- 複素数を表すための complex がある (C 言語の `complex` に相当, 実部・虚部が `double`)
- 論理を表すための bool がある (C 言語の `bool` に相当). `true`, `false` と言う値があるが、それぞれ 1, 0 の別名と考えて良い。
  例えば `plot(u,wait=true);` は `plot(u,wait=1);` と同じ。
- 文字列を表すための string がある (C++言語の `string` に相当, 日本語不可？)。
  - 2 つの string `s1`, `s2` を、(+ 演算子を用いて) `s1+s2` で連結できる。

7.2.2 データ型

- 整数を表すための int がある (C 言語の `int` に相当)
- 実数を表すための real がある (C 言語の `double` に相当)
- 複素数を表すための complex がある (C 言語の `complex` に相当, 実部・虚部が `double`)
- 論理を表すための bool がある (C 言語の `bool` に相当). `true`, `false` と言う値があるが、それぞれ 1, 0 の別名と考えて良い。
  例えば `plot(u,wait=true);` は `plot(u,wait=1);` と同じ。
- 文字列を表すための string がある (C++言語の `string` に相当, 日本語不可？)。
  - 2 つの string `s1`, `s2` を、(+ 演算子を用いて) `s1+s2` で連結できる。
  - string+数値 とすると、数値を文字列に変換してから連結する。

```
real a=1.23, b=4.56;
string s;
s= "a=" + a + ", b=" + b + ".";
cout << s << endl;
```

7.2.2 データ型

- 整数を表すための int がある (C 言語の `int` に相当)
- 実数を表すための real がある (C 言語の `double` に相当)
- 複素数を表すための complex がある (C 言語の `complex` に相当, 実部・虚部が `double`)
- 論理を表すための bool がある (C 言語の `bool` に相当). `true`, `false` と言う値があるが、それぞれ 1, 0 の別名と考えて良い。
  例えば `plot(u,wait=true);` は `plot(u,wait=1);` と同じ。
- 文字列を表すための string がある (C++言語の `string` に相当, 日本語不可？)。
  - 2 つの string `s1`, `s2` を、(+ 演算子を用いて) `s1+s2` で連結できる。
  - string+数値 とすると、数値を文字列に変換してから連結する。

```
real a=1.23, b=4.56;
string s;
s= "a=" + a + ", b=" + b + ".";
cout << s << endl;
```

  - string を int に変換する `atoi()`, string を real に変換する `atof()` がある (C 言語の真似)。

7.2.3 配列型　1 次元

1 次元配列は、C 言語に (少し) 似ている。

```
real[int] a1(3); // C で double a1[3]; とするのに似ている
for (int i=0;i<3;i++)
  a1[i]=i; // a1(i)=i; としても良い。

real[int] a2 = [0,1,2]; // C で double a[]={0,1,2}; とするのに似てる
real[int] a3 = 0:2; // これは少し MATLAB 風

cout << "a1=" << a1 << endl;
cout << "a2=" << a2 << endl;
cout << "a3=" << a3 << endl;
```

追記: `a1` の要素数は `a1.n` で得られる。

7.2.3 配列型　2次元

2 次元配列はかなり違う。要素にアクセスするには 名前 (i,j) とする。

```
real[int,int] kuku(9,9);
int i,j;
for (i=0; i<kuku.n; i++) {
  for (j=0; j<kuku.m; j++) {
      kuku(i,j)=(i+1)*(j+1);
      cout << setw(3) << kuku(i,j);
  }
  cout << endl;
}
cout << kuku << endl;

real[int,int] kuku2=[[1,2,3,4,5,6,7,8,9],
                    [2,4,6,8,10,12,14,16,18],
                    [3,6,9,12,15,18,21,24,27],
                    [4,8,12,16,20,24,28,32,36],
                    [5,10,15,20,25,30,35,40,45],
                    [6,12,18,24,30,36,42,48,54],
                    [7,14,21,28,35,42,49,56,63],
                    [8,16,24,32,40,48,56,64,72],
                    [9,18,27,36,45,54,63,72,81]];
cout << kuku2 << endl;
```

参考文献

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

[2] Hecht, F.: New development in FreeFem++, *J. Numer. Math.*, Vol. 20, No. 3-4, pp. 251–265 (2012).

[3] Hecht, F.: Freefem++, `https://doc.freefem.org/pdf/FreeFEM-documentation.pdf`, 以前は `http://www3.freefem.org/ff++/ftp/freefem++doc.pdf` にあった。

[4] 桂田祐史：FreeFEM++ ノート, `https://m-katsurada.sakura.ne.jp/labo/text/freefem-note.pdf` (2012〜).