

応用数値解析特論 第7回

～2次元 Poisson 方程式に対する有限要素法 (2), プログラミング言語
FreeFem++(1)～

かつらだ まさし
桂田 祐史

<https://m-katsurada.sakura.ne.jp/ana2023/>

2023年5月30日

目次

- 1 本日の内容
- 2 2次元 Poisson 方程式に対する有限要素法 (続き)
 - 近似方程式の組み立て — 直接剛性法
 - 連立1次方程式の具体例
 - プログラム
 - 方程式を立てるのに必要なもの
 - サンプルプログラム紹介
- 3 おまけ: C 言語による2次元有限要素法サンプル・プログラムの紹介
 - 進行表
 - 試しに実行
 - 有限要素解を求めるプログラム naive, band の理解
 - プログラム naive の内部構造
 - 参考課題
- 4 FreeFem++の文法
 - はじめに
 - 汎用のプログラミング機能
 - C 言語と良く似ているところ
 - データ型
 - 配列型
 - FreeFem++の real データの入出力の書式指定
 - 有限要素法のための機能

本日の内容

- 2次元 Poisson 方程式に対する有限要素法の説明 2 回目。
C 言語によるサンプル・プログラムの紹介は省略する (資料だけ残す)。
- FreeFem++ の文法の説明を始める。

5.4 近似方程式の組み立て — 直接剛性法

$u_i := \hat{u}(P_i)$, $v_i := \hat{v}(P_i)$ ($i = 1, 2, \dots, m$) として、

$$(1) \quad \mathbf{u} := \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{pmatrix}, \quad \mathbf{v} := \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{pmatrix}$$

とおく。

有限要素 e_k ($k = 1, 2, \dots, N_e$) の節点 N_0, N_1, N_2 に対して、

$$N_0 = P_{i_{k,0}}, \quad N_1 = P_{i_{k,1}}, \quad N_2 = P_{i_{k,2}}$$

となる整数 $i_{k,0}, i_{k,1}, i_{k,2}$ を取る (これらを**全体節点番号**と呼ぶ)。

$\mathbf{f}_k^* \in \mathbb{R}^m$ を $i_{k,0}$ 成分 = $f_0^{(k)}$, $i_{k,1}$ 成分 = $f_1^{(k)}$, $i_{k,2}$ 成分 = $f_2^{(k)}$ で、それ以外の成分はすべて 0 であるようなベクトルとする。例えば $i_0^{(k)} < i_1^{(k)} < i_2^{(k)}$ ならば

$$\mathbf{f}_k^* = \begin{pmatrix} 1 & & & & & & & & & \\ \downarrow & & & & & & & & & \\ 0 & \dots & 0 & f_0^{(k)} & 0 & \dots & 0 & f_1^{(k)} & 0 & \dots & 0 & f_2^{(k)} & 0 & \dots & 0 \end{pmatrix}^\top.$$

このように \mathbf{f}_k^* を定義すると、次が成り立つ。

$$(2) \quad (\mathbf{f}, \hat{v})_{e_k} = \mathbf{v}_k^\top \mathbf{f}_k = \mathbf{v}^\top \mathbf{f}_k^* \quad (k = 1, 2, \dots, N_e).$$

5.4 近似方程式の組み立て — 直接剛性法

同様の考え方で、行列 $A_k^* = (a_{ij}^{(k)})$ を

$$\begin{aligned} a_{i_{k,0}i_{k,0}}^{(k)} &= A_{00}^{(k)}, & a_{i_{k,0}i_{k,1}}^{(k)} &= A_{01}^{(k)}, & a_{i_{k,0}i_{k,2}}^{(k)} &= A_{02}^{(k)}, \\ a_{i_{k,1}i_{k,0}}^{(k)} &= A_{10}^{(k)}, & a_{i_{k,1}i_{k,1}}^{(k)} &= A_{11}^{(k)}, & a_{i_{k,1}i_{k,2}}^{(k)} &= A_{12}^{(k)}, \\ a_{i_{k,2}i_{k,0}}^{(k)} &= A_{20}^{(k)}, & a_{i_{k,2}i_{k,1}}^{(k)} &= A_{21}^{(k)}, & a_{i_{k,2}i_{k,2}}^{(k)} &= A_{22}^{(k)}, \\ && \text{それ以外} &= 0 \end{aligned}$$

で定める。例えば $i_{k,0} < i_{k,1} < i_{k,2}$ ならば

$$A_k^* = \begin{pmatrix} i_{k,0} & i_{k,1} & i_{k,2} & & \\ \downarrow & \downarrow & \downarrow & & \\ A_{00}^{(k)} & A_{01}^{(k)} & A_{02}^{(k)} & \leftarrow i_{k,0} & \\ A_{10}^{(k)} & A_{11}^{(k)} & A_{12}^{(k)} & \leftarrow i_{k,1} & \\ A_{20}^{(k)} & A_{21}^{(k)} & A_{22}^{(k)} & \leftarrow i_{k,2} & \end{pmatrix} \quad (\text{書いてない成分は } 0).$$

これを用いると

$$(3) \quad \langle \hat{u}, \hat{v} \rangle_{e_k} = \mathbf{v}_k^\top A_k \mathbf{u}_k = \mathbf{v}^\top A_k^* \mathbf{u} \quad (k = 1, 2, \dots, N_e).$$

5.4 近似方程式の組み立て — 直接剛性法

ゆえに弱形式 $\sum_{k=1}^{N_e} \langle \hat{u}, \hat{v} \rangle_{e_k} = \sum_{k=1}^{N_e} (f, \hat{v})_{e_k} + [g_2, \hat{v}]$ は (簡単のため $g_2 = 0$ と仮定して)

$$\sum_{k=1}^{N_e} \mathbf{v}^\top \mathbf{A}_k^* \mathbf{u} = \sum_{k=1}^{N_e} \mathbf{v}^\top \mathbf{f}_k^*$$

すなわち

$$\mathbf{v}^\top \left(\sum_{k=1}^{N_e} \mathbf{A}_k^* \mathbf{u} - \sum_{k=1}^{N_e} \mathbf{f}_k^* \right) = 0$$

と同値になる。

ゆえに

$$\mathbf{A}^* := \sum_{k=1}^{N_e} \mathbf{A}_k^*, \quad \mathbf{f}^* := \sum_{k=1}^{N_e} \mathbf{f}_k^*$$

とおけば

$$(4) \quad \mathbf{v}^\top (\mathbf{A}^* \mathbf{u} - \mathbf{f}^*) = 0.$$

5.4 近似方程式の組み立て — 直接剛性法

ここで \mathbf{v} は

$$Y := \left\{ \begin{pmatrix} v_1 \\ \vdots \\ v_m \end{pmatrix} \in \mathbb{R}^m; v_i = 0 \quad (P_i \in \Gamma_1 \text{ なる } i) \right\}$$

の任意の元であるから、(4) は次と同値である。

$$\mathbf{A}^* \mathbf{u} - \mathbf{f} \in Y^\perp = \left\{ \begin{pmatrix} w_1 \\ \vdots \\ w_m \end{pmatrix} \in \mathbb{R}^m; w_i = 0 \quad (P_i \notin \Gamma_1 \text{ なる } i) \right\}$$

すなわち

$$(5) \quad \mathbf{A}^{**} \mathbf{u} = \mathbf{f}^{**}.$$

ここで

$\mathbf{A}^{**} := \mathbf{A}^*$ の第 i 行 ($P_i \in \Gamma_1$ なる i) を除いた行列,

$\mathbf{f}^{**} := \mathbf{f}^*$ の第 i 成分 ($P_i \in \Gamma_1$ なる i) を除いた縦ベクトル.

5.4 近似方程式の組み立て — 直接剛性法

$$I := \{i \in \mathbb{N} \mid 1 \leq i \leq m\}, \quad I_1 := \{i \in I \mid P_i \in \Gamma_1\}$$

とおく (I_1 は Γ_1 上にある節点の節点番号全体の集合)。

条件

$$u_i = g_1(P_i) \quad (i \in I_1)$$

があるから、これを代入して u_i ($i \in I_1$) を消去できる。以下それを実行する。

A^{**} を列ベクトルで

$$A^{**} = (\mathbf{a}_1 \ \mathbf{a}_2 \ \cdots \ \mathbf{a}_m)$$

のように表示すると、(5) は

$$(6) \quad \sum_{i=1}^m \mathbf{a}_i u_i = \mathbf{f}^{**}.$$

左辺の $\sum_{i=1}^m$ を $\sum_{i \in I \setminus I_1} + \sum_{i \in I_1}$ と分解して、移項すると

$$\sum_{i \in I \setminus I_1} \mathbf{a}_i u_i = \mathbf{f}^{**} - \sum_{i \in I_1} \mathbf{a}_i u_i.$$

5.4 近似方程式の組み立て — 直接剛性法

ゆえに

$$A\mathbf{u}^* = \mathbf{f},$$

ただし

$\mathbf{u}^* := \mathbf{u}$ の第 i 成分 ($i \in I_1$) を除いた縦ベクトル,

$A := A^{**}$ の第 i 列 ($i \in I_1$) を除いた正方行列,

$$\mathbf{f} := \mathbf{f}^{**} - \sum_{i \in I_1} \mathbf{a}_i u_i.$$

実際にプログラムを作成するとき、 A や \mathbf{f} が容易に求まる (後述する)。

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。

$$\Omega = (0, 1) \times (0, 1),$$

$$\Gamma_1 = \{(x, y) \mid x = 0, 0 \leq y \leq 1\} \cup \{(x, y) \mid 0 \leq x \leq 1, y = 0\},$$

$$\Gamma_2 = \{(x, y) \mid x = 1, 0 < y \leq 1\} \cup \{(x, y) \mid 0 < x \leq 1, y = 1\},$$

$$g_1 \equiv 0, \quad g_2 \equiv 0, \quad f \equiv \text{定数関数 } \bar{f}.$$

すなわち

$$-\Delta u = \bar{f} \quad (\text{in } \Omega), \quad u = 0 \quad \text{on } \Gamma_1, \quad \frac{\partial u}{\partial \mathbf{n}} = 0 \quad \text{on } \Gamma_2.$$

5.5 連立1次方程式の具体例

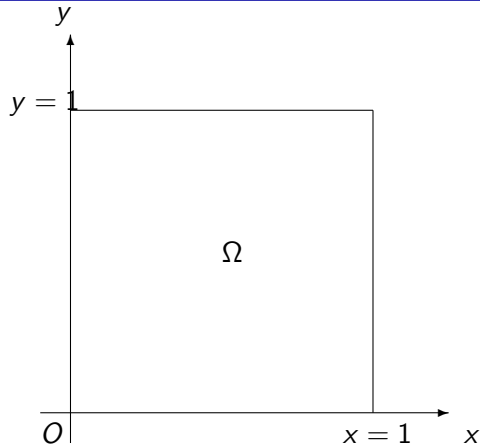


図 1: 領域 Ω

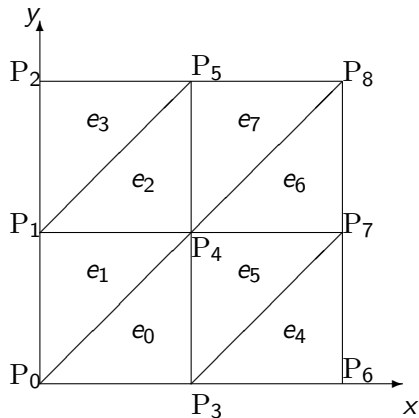


図 2: 要素分割

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3三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 .

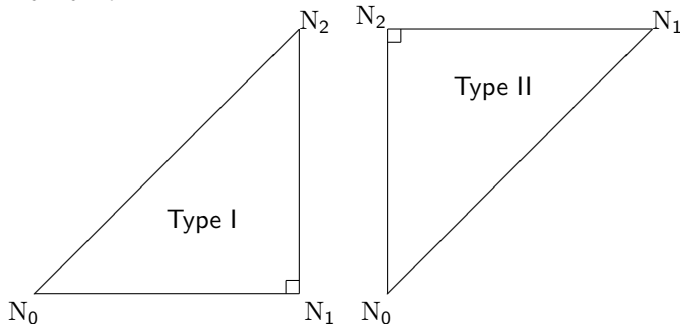


図 3: 二つのタイプの有限要素と局所節点番号

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 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

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

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

タイプ II については

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

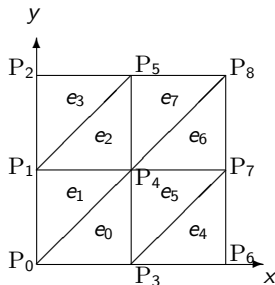
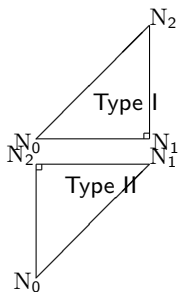
$$A_{II} = \frac{1}{2} \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & -1 & 2 \end{pmatrix}, \quad f_{II} = \frac{\bar{f}h^2}{6} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

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

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

for

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

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。

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

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。

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。

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 (\mathbf{v}^\top のせいで $\mathbf{0}^\top \mathbf{u} = 0$) 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 \mathbf{e}_i^\top で置き換える)

とすることで \mathbf{v}^\top が外せる。

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1/2 & 0 & -1 & 2 & 0 & 0 & -1/2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1/2 & 2 & -1/2 \\ 0 & 0 & 0 & 0 & 0 & -1/2 & 0 & -1/2 & 1 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{pmatrix} = \begin{pmatrix} g_1(P_0) \\ g_1(P_1) \\ g_1(P_2) \\ g_1(P_3) \\ \bar{f}h^2 \\ \bar{f}h^2/2 \\ g_1(P_6) \\ \bar{f}h^2/2 \\ \bar{f}h^2/3 \end{pmatrix}$$

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

$$\begin{pmatrix}
 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\
 0 & 0 & 0 & 0 & -1 & 2 & 0 & 0 & -1/2 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & -1 & 0 & 0 & 2 & -1/2 & 0 \\
 0 & 0 & 0 & 0 & 0 & -1/2 & 0 & -1/2 & 1 & 0
 \end{pmatrix}
 \begin{pmatrix}
 u_0 \\
 u_1 \\
 u_2 \\
 u_3 \\
 u_4 \\
 u_5 \\
 u_6 \\
 u_7 \\
 u_8
 \end{pmatrix}
 =
 \begin{pmatrix}
 g_1(P_0) \\
 g_1(P_1) \\
 g_1(P_2) \\
 g_1(P_3) \\
 \bar{f}h^2 \\
 \bar{f}h^2/2 \\
 g_1(P_6) \\
 \bar{f}h^2/2 \\
 \bar{f}h^2/3
 \end{pmatrix}
 +
 \begin{pmatrix}
 0 \\
 0 \\
 0 \\
 0 \\
 g_1(P_1) + g_1(P_3) \\
 g_1(P_2)/2 \\
 0 \\
 g_1(P_6)/2 \\
 0
 \end{pmatrix}.$$

5.5 連立1次方程式の具体例 Dirichlet境界条件の考慮

(説明のため、Galerkin法の弱形式を再度掲示)

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

for

$$\forall \mathbf{v} \in \left\{ (v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8)^T \in \mathbb{R}^9 \mid v_0 = v_1 = v_2 = v_3 = v_6 = 0 \right\}.$$

5.5 連立1次方程式の具体例 Dirichlet境界条件の考慮

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 $\text{tgv} \times g_1(P_i)$ 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。

- 解は実質的にはほぼ変わらない (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 $\text{tgv} (= 10^{30})$ として

$$\begin{pmatrix} T & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & T & -1 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & -1 & T & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & T & -2 & 0 & -1 & 0 & 0 \\ 0 & -2 & 0 & -2 & 8 & -2 & 0 & -2 & 0 \\ 0 & 0 & -1 & 0 & -2 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & T & -1 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{pmatrix} = \begin{pmatrix} Tg_1(P_0) \\ Tg_1(P_1) \\ Tg_1(P_2) \\ Tg_1(P_3) \\ f_4 \\ f_5 \\ Tg_1(P_6) \\ f_7 \\ f_8 \end{pmatrix}$$

(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?))

5.6 プログラム 5.6.1 方程式を立てるのに必要なもの

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 ($f \equiv \bar{f}$ (定数), $g_1 \equiv 0$, $g_2 \equiv 0$) では

- 節点の座標 ($i = 1, \dots, m$ 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。

- Ⓐ Ω に属する節点 P_i での f の値 $f(P_i)$
- Ⓑ Γ_1 上にある節点での g_1 の値
- Ⓒ Γ_2 上にある節点の全体節点番号
- Ⓓ 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。

($\Omega, \Gamma_1, \Gamma_2, \{e_k\}, \{P_i\}, f, g_1, g_2$ 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できる。)

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。

6 おまけ: 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()` の解説 (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
```

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 (GLSC3D に含まれるコンパイル用スクリプト) 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。

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

$$(7) \quad -\Delta u(x, y) = f(x, y) := \mathbf{1} \quad ((x, y) \in \Omega),$$

$$(8) \quad u(x, y) = g_1(x, y) := \mathbf{0} \quad ((x, y) \in \Gamma_1),$$

$$(9) \quad \frac{\partial u}{\partial \mathbf{n}}(x, y) = g_2(x, y) := \mathbf{0} \quad ((x, y) \in \Gamma_2)$$

を有限要素法で解くプログラムである。

Q ここで $\Omega, \Gamma_1, \Gamma_2$ は何か？

A 実は $\Omega, \Gamma_1, \Gamma_2$ についてはデータとして入力する。

naive, band とともに、**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 $\mathbf{1}, \mathbf{0}, \mathbf{0}$ 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
```

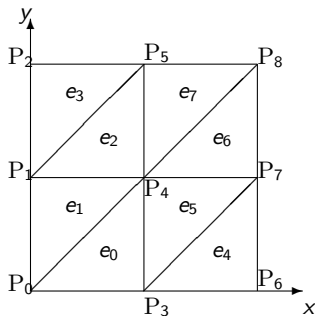


図 4: 要素分割 (各辺を 2 等分してから要素分割)

- 1 行目には、節点数 (nnode)、要素数 (nelmt)、 Γ_1 に属している節点数 (nbc)
- 2~10 行は、節点の座標 (x_i, y_i) ($i = 0, 1, \dots, \text{nnode} - 1$)
- 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。
- 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。

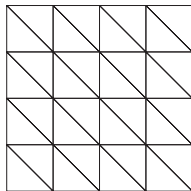
—— 三角形と (結果として) Ω を描く ——

```
./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
```

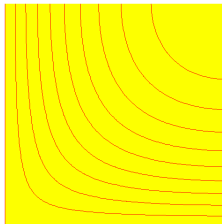
```
./disp-glsc3d input64.dat
```

```
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 $-\Delta u = f$ 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 $f^* := \sum_{k=0}^{N_e-1} f_k^*$ 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 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 , 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知っているが、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

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

であるが、これを非同次境界条件

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

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

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ソフトウェアであり、言語処理系である (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 ([3] の付録に書いておいた)。

参考: FreeFem++ は C++ で記述されている。

マニュアル Hecht[4] は事例集の性格が強く、言語仕様が整理され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 [3] に基づく。

マニュアル以外の情報源として、日本応用数理学会のチュートリアル (鈴木 [5], [6])、テキスト大塚・高石 [7] (本学学生は Maruzen eBook で読める) などがある。

7.1 はじめに 基本的な Poisson 方程式のプログラム

`curl -O https://m-katsurada.sakura.ne.jp/program/fem/poisson.edp` で入手できる。

```
// 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  $-\Delta u=f$  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。

- // 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

- 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`)
例えば `complex a=1+2i;` 入出力は 2次元ベクトル風の (1,2)
- 論理を表すための `bool` がある (C 言語の `bool` に相当). `true`, `false` という値があるが、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。

配列 a の第 i 要素は、 $a[i]$ としてアクセスできるが、 $a(i)$ としてアクセスするのが普通？

```
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次元配列は少し違う。2次元配列 a の (i,j) 要素にアクセスするには $a(i,j)$ 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;
```


7.2.4 FreeFem++の real データの入出力の書式指定

- 何も指定しないと C 言語の %g 相当の出力になる。
- `cout.precision(n);` とすると、以下小数点以下の桁数は n になる。

```
cout.precision(15);  
cout << "pi=" << pi << endl;
```
- 幅を指定するには `<< setw(桁数)` とする (これは毎回必要)。

```
cout << "pi=" << setw(20) << pi << endl;
```
- `cout.fixed;` とすると、以下固定小数点数形式 (C 言語の %f 相当) になる。

```
cout.fixed;  
cout << "NA=" << NA << endl;
```
- `cout.scientific;` とすると、以下指数形式 (C 言語の %e 相当) になる。

```
cout.scientific;  
cout << "pi=" << pi << endl;
```
- `cout.default;` とすると、以下デフォルト (C 言語の %g) に戻る。

7.2.4 FreeFem++の real データの入出力の書式指定 例

```
// testfloat.edp
real NA = 6.022e+23;
// デフォルト %g に相当
cout << "pi=" << pi << ", NA=" << NA << ", pi*NA=" << pi * NA << endl << endl;
// 幅を 20 に指定 %20g に相当
cout << "pi=" << setw(20) << pi << ", NA=" << setw(20) << NA
    << ", pi*NA=" << setw(20) << pi * NA << endl << endl;
// 小数点以下の桁数を 15 に指定 %20.15g に相当?
cout.precision(15);
cout << "pi=" << setw(20) << pi << ", NA=" << setw(20) << NA
    << ", pi*NA=" << setw(20) << pi * NA << endl << endl;
// 固定小数点数形式 %.15f に相当
cout.fixed;
cout << "pi=" << pi << ", NA=" << NA << ", pi*NA=" << pi * NA << endl << endl;
// %20.15f に相当
cout << "pi=" << setw(20) << pi << ", NA=" << setw(20) << NA
    << ", pi*NA=" << setw(20) << pi * NA << endl << endl;
// 指数形式 %20.15e に相当
cout.scientific;
cout << "pi=" << setw(20) << pi << ", NA=" << setw(20) << NA
    << ", pi*Na=" << setw(20) << pi * NA << endl << endl;
// %g 形式に戻す %.15g に相当
cout.default;
cout << "pi=" << pi << ", NA=" << NA << ", pi*Na=" << pi * NA << endl;
```

7.3 有限要素法のための機能

7.3.1 有限要素法のプログラムの構成

有限要素法のプログラムと言っても色々あるが、基本的と考えられる2次元 Poisson 方程式の境界値問題のプログラムを例にして説明する。

- ① 領域の定義と領域の三角形分割
- ② 有限要素空間の定義
- ③ 弱形式を次のいずれかで定義して解く。
 - a solve()
弱形式を与えると同時にそれを解く (弱解を求める)。
 - b problem()
弱形式を与えて問題を解く関数を定義する。発展問題で便利。
 - c varf, matrix
弱形式を与えて連立1次方程式を作る。

7.3.2 領域の定義と領域の三角形分割 (2次元の場合)

問題を考える領域を定義し、三角形分割をすることが必要である。

- 2次元 (有界) 領域の多くは、その境界曲線を定義することで定まる。
- 境界曲線は `border` という型の変数として定義される。
- 三角形分割は `mesh` という型の変数として定義される。
- `buildmesh()` という関数は、各 `border` を何等分するか指定することで、`border` の囲む領域を三角形分割して、`mesh` 型のデータを作る。
- `mesh` 型のデータは、`readmesh()`, `writemesh()` という関数を用いて入出力できる (フォーマットはテキスト・ファイル)。
- `mesh` 型のデータは、`plot()` により可視化できる。
- 矩形領域 (辺が座標軸に平行な長方形) は、`square()` という命令で `mesh` 型データが作れる (参考「[FreeFem++ノート](#)」)。

円周全体を C とする

```
border C(t=0,2*pi) { x=cos(t); y=sin(t); }
```

円周の上半分、下半分を別々に Gamma1 , Gamma2 と定義する

```
int C=1;
...
border Gamma1(t=0,pi) { x=cos(t); y=sin(t); label=C; }
border Gamma2(t=pi,2*pi) { x=cos(t); y=sin(t); label=C; }
```

正方形領域 $(0,1) \times (0,1)$ の4つの辺 $C1, C2, C3, C4$ を定義

```
border C1(t=0,1) { x=t; y=0; label=1; }
border C2(t=0,1) { x=1; y=t; label=2; }
border C3(t=0,1) { x=1-t; y=1; label=3; }
border C4(t=0,1) { x=0; y=1-t; label=4; }
```

(label の値指定は必須ではない。指定する場合は 0 以外の値を選ぶ。)

それぞれ表示してみる

```
// 例 1
```

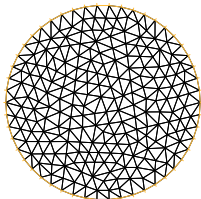
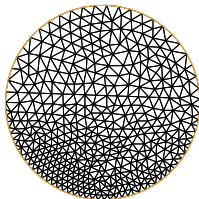
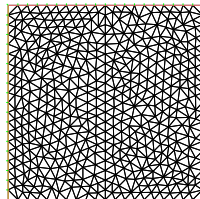
```
border C(t=0,2*pi) { x=cos(t); y=sin(t); }  
mesh Th1=buildmesh(C(50));  
plot(Th1,wait=true,ps="Th1.eps");
```

```
// 例 2
```

```
int C0=1;  
border Gamma1(t=0,pi) { x=cos(t); y=sin(t); label=C0; }  
border Gamma2(t=pi,2*pi) { x=cos(t); y=sin(t); label=C0; }  
mesh Th2=buildmesh(Gamma1(25)+Gamma2(50));  
plot(Th2,wait=true,ps="Th2.eps");
```

```
// 例 3
```

```
border C1(t=0,1) { x=t; y=0; label=1; }  
border C2(t=0,1) { x=1; y=t; label=2; }  
border C3(t=0,1) { x=1-t; y=1; label=3; }  
border C4(t=0,1) { x=0; y=1-t; label=4; }  
mesh Th3=buildmesh(C1(20)+C2(20)+C3(20)+C4(20));  
plot(Th3,wait=true,ps="Th3.eps");
```

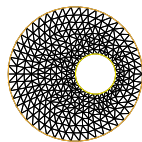
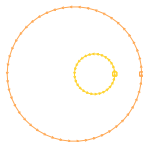
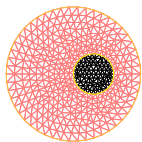
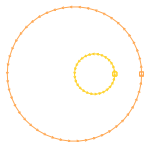
図 5: $C(50)$ 図 6: $\text{Gamma}1(25)+\text{Gamma}2(50)$ 図 7: $C1(20)+C2(20)+\dots$

7.3.2 領域の定義と領域の三角形分割 メッシュ (mesh)

有限個の Jordan 閉曲線で囲まれた多重連結領域を三角形分割することもできる。

sampleMesh.edp

```
border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}  
border b(t=0,2*pi){ x=0.3+0.3*cos(t); y=0.3*sin(t);label=2;}  
plot(a(50)+b(+30),wait=true,ps="border.eps");  
mesh ThWithoutHole = buildmesh(a(50)+b(+30));  
plot(ThWithoutHole,wait=1,ps="Thwithouthole.eps");  
plot(a(50)+b(-30),wait=true,ps="borderminus.eps");  
mesh ThWithHole = buildmesh(a(50)+b(-30));  
plot(ThWithHole,wait=1,ps="Thwithhole.eps");
```



7.3.2 領域の定義と領域の三角形分割 メッシュ (mesh)

普通は mesh データの細かいことは見る必要がないかもしれないが…

- Th をメッシュとするとき、 Th.n t は三角形の数 (the number of triangles)、 Th.n v は節点の数 (the number of vertices)、 Th.area は領域の面積 (area) である。
- $\text{Th}(i)$ は i 番目の節点 ($i = 0, 1, \dots, \text{Th.nv} - 1$) で、その座標は $\text{Th}(i).x$ と $\text{Th}(i).y$ である。 $\text{Th}(i).label$ はその節点のラベル (領域内部にあれば 0, それ以外は境界のどこか) を表す。
- $\text{Th}[i]$ は i 番目の三角形 ($i = 0, 1, \dots, \text{Th.nt} - 1$)、 $\text{Th}[i][j]$ は i 番目の三角形の j 番目の節点 ($j = 0, 1, 2$) の全体節点番号、その節点の座標は $\text{Th}[i][j].x$ と $\text{Th}[i][j].y$ である。三角形の面積は $\text{Th}[i].area$ である。
- 点 (x, y) を含む三角形の番号は $\text{Th}(x, y).nuTriangle$ で得られる。

7.3.2 領域の定義と領域の三角形分割 メッシュ (mesh)

次のような場合に `readmesh()`, `writemesh()` は有効である。

- ① FreeFem++ を用いて三角形分割を行い、得られたメッシュ・データを外部のプログラムで利用する (有限要素解の計算は自作プログラムで行う等)。
- ② 自作のプログラムで三角形分割を行い、そのメッシュ・データを FreeFem++ で利用する。

`readmesh()`, `writemesh()` で入出力される **メッシュ・データのフォーマット** については、「FreeFem++ノート §6.2 mesh ファイルの構造」を見よ。

- Mesh 型の変数 `Th` の内容は、

```
writemesh(Th, "bunkatsu.msh");
```

のようにしてファイルに出力できる。
- そのフォーマットにのっとって作られたファイルがあれば、

```
Mesh Th=readmesh("bunkatsu.msh");
```

のようにして読み込める。

7.3.3 有限要素空間

既に定義しておいた mesh 型データと、要素の種類を表す名前 (P1, P2, ...) を用いて、有限要素空間 (この講義では \tilde{X} のように表したが、 V_h などの記号で表すことが多い) を定義する。

fespace 型の変数は関数空間を表すことになる。

例えば Th という mesh 型の変数があるとき、

```
fespace Vh(Th,P1);
```

とすると有限要素空間 Vh が定義される。

これは文法的には型名で

```
Vh u,v;
```

として変数 u, v が定義できる。これらが個々の関数を表す。

(数学語では $u, v \in V_h$ という調子)

(注 これまでの授業で、三角形要素分割して、区分的 1 次多項式 (P1 要素) しか紹介しなかったが、Poisson 方程式の境界値問題以外では、他の要素 (P1b, P2, P2Morley,...) が必要になることがある。)

7.3.3 有限要素空間

- u の節点での値を集めた配列は $u[]$ で表す。
 $u[].n$ ($u.n$ でも同じ) は $Th.nv$ と同じである。
 i 番目の節点での値 (授業中の式で $u^i = \hat{u}(P_i)$) は $u[](i)$
- u は補間多項式でもあり、 (x, y) での値は $u(x, y)$ で得られる。

7.3.3 弱形式を定義して解く

いよいよ弱形式を定義する方法の説明である。大きく分けて3通りある。

- Ⓐ `solve` — 弱形式を与えると同時にそれを解く (弱解を求める)。
- Ⓑ `problem` — 弱形式を与えて問題を解く関数を定義する。
- Ⓒ `varf, matrix` — 弱形式を与えて連立1次方程式を作る。

これまで説明して来た次の Poisson 方程式の境界値問題を元に説明する。

$$(10a) \quad -\Delta u(x, y) = f(x, y) \quad ((x, y) \in \Omega)$$

$$(10b) \quad u(x, y) = g_1(x, y) \quad ((x, y) \in \Gamma_1)$$

$$(10c) \quad \frac{\partial u}{\partial \mathbf{n}}(x, y) = g_2(x, y) \quad ((x, y) \in \Gamma_2).$$

弱解 u とは、 X_{g_1} に属し、次の弱形式を満たすものである。

$$(11) \quad \langle u, v \rangle = (f, v) + [g_2, v] \quad (v \in X).$$

ただし

$$(12) \quad X_{g_1} := \{w \mid w = g_1 \text{ on } \Gamma_1\}, \quad X := \{v \mid v = 0 \text{ on } \Gamma_1\}.$$

7.3.3 弱形式を定義して解く (a) solve を利用

Poisson 方程式の境界値問題を解くサンプル・プログラム `poisson.edp` では、(a) を用いた。

— solve で弱形式を定義して解く —

```
solve Poisson(u,v)=
  int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))-int2d(Th)(f*v)-int1d(Th,2,3)(g2*v)
  +on(1,4,u=g1);
```

次はどの方法でも共通である。

- `dx()`, `dy()` はそれぞれ x , y での微分を表す。
高階の微分は `dxx()`, `dxy()`, `dyy()` のようにする。
- `int2d(Th)` は、`Th` の領域全体の積分 (重積分) を表す。
また `int1d(Th,2,3)` は境界のうち、ラベルが 2,3 である部分 (正方形の右と上) の積分 (境界積分、今の場合は線積分) を表す。
- `+on(1,4,u=g1)` は境界のうち、ラベルが 1,4 である部分 (正方形の下と左) で、 $u = g_1$ という Dirichlet 境界条件を課すことを表す (`+on(1,u=g1)+on(4,u=g1)` と分けて書くことも可能)。
ベクトル値関数の場合は、`+on(1,u1=g1,u2=g2)` のように複数の方程式を書くこともできる。

以下、この問題の場合に、(b), (c) がどうなるか示す。

7.3.3 弱形式を定義して解く (b) problem を利用

(b) problem を利用する方法では、次のようになる。

problem で弱形式を定義して解く

```
problem Poisson(u,v)=  
  int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))-int2d(Th)(f*v)-int1d(Th,2,3)(g2*v)  
  +on(1,4,u=g1);  
  
Poisson;
```

この問題の場合は、`solve` と比べての利点は特に感じられないかもしれないが、時間発展の問題では、同じ形の弱形式を何度も解く必要が生じるので、有効である (効率が上がる可能性がある — 後述)。

7.3.3 弱形式を定義して解く (c) varf, matrix を利用

varf, matrix を利用

```
real Tgv=1.0e+30; // tgv と小文字でも可
varf a(u,v)=
  int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))
  +on(1,4,u=g1);
matrix A=a(Vh,Vh,tgv=Tgv,solver=CG);
varf l(UNUSED,v)=
  int2d(Th)(f*v)+int1d(Th,2,3)(g2*v)
  +on(1,4,UNUSED=0);
Vh F;
F[]=l(0,Vh,tgv=Tgv);
u[]=A^-1*F[];
```

あらすじは、連立1次方程式 $A\mathbf{u} = \mathbf{f}$ の A, \mathbf{f} を別々に計算して、 $A^{-1}\mathbf{f}$ を計算することで \mathbf{u} を得る、ということである (詳細は、実は現時点で把握していないので省略する)。

tgv (terrible great value) は以前説明した ($tgv = 10^{30}$)。

solver= は連立1次方程式の解法を指定する。

<i>CG</i>	CG 法 (共役勾配法) (反復法, 正値対称行列 (<i>spd</i>) 用)
<i>GMRES</i>	GMRES 法 (反復法, 一般の正則行列用)
<i>UMFPACK</i>	UMFPACK を利用 (直接法, 一般の正則行列用)
<i>sparsesolver</i>	ダイナミック・リンクで 外部のソルバー を呼ぶ

参考プログラム — 有限要素解の誤差を見る (1)

有限要素解の収束、誤差の減衰は本科目の最後に説明する予定であるが、見ておこう。
真の解 u , 有限要素解 \hat{u} について

$$\|u - \hat{u}\|_{L^2} = \left(\iint_{\Omega} |u(x, y) - \hat{u}(x, y)|^2 dx dy \right)^{1/2}$$

を L^2 誤差、

$$\|u - \hat{u}\|_{H^1} = \left(\|u - \hat{u}\|_{L^2}^2 + \|u_x - \hat{u}_x\|_{L^2}^2 + \|u_y - \hat{u}_y\|_{L^2}^2 \right)^{1/2}$$

を H^1 誤差と呼ぶ。

領域の分割を細かくしたとき、これらの誤差がどのように減衰するか、厳密解が分かる問題で調べてみよう。

```
curl -O https://m-katsurada.sakura.ne.jp/program/fem/poisson-mixedBC-mk.edp
FreeFem++ poisson-mixedBC-mk.edp
FreeFem++ poisson-mixedBC-mk.edp 2
FreeFem++ poisson-mixedBC-mk.edp 4
...
FreeFem++ poisson-mixedBC-mk.edp 64
```

正方形の辺を $20 \cdot 2^m$ ($m = 0, 1, \dots, 6$) 分割したときの誤差を近似計算する。

参考プログラム — 有限要素解の誤差を見る (2)

このプログラムで解いている問題は、先に厳密解 (真の解) u を選んで、それから f, g, h を計算して作ったのであろう。

$$\Omega = (0, 1) \times (0, 1),$$

$$\Gamma_1 = \{(1, y) \mid 0 \leq y \leq 1\} \cup \{(x, 1) \mid 0 \leq x \leq 1\} \cup \{(0, y) \mid 0 \leq y \leq 1\},$$

$$\Gamma_2 = \{(x, 0) \mid 0 \leq x \leq 1\}.$$

$$u(x, y) = \sin(\pi x) \sin \frac{\pi y}{2}$$

より

$$f(x, y) = -\Delta u(x, y) = \frac{5\pi^2}{4} \sin(\pi x) \sin \frac{\pi y}{2},$$

$$g(x, y) = u(x, y) = \sin(\pi x) \sin \frac{\pi y}{2},$$

$$h(x, y) = \frac{\partial u}{\partial n}(x, 0) = -\frac{\partial u}{\partial y}(x, 0) = -\frac{\pi}{2} \sin(\pi x) \cos \frac{\pi y}{2} \Big|_{y=0} = -\frac{\pi}{2} \sin(\pi x).$$

参考プログラム — 有限要素解の誤差を見る (3)

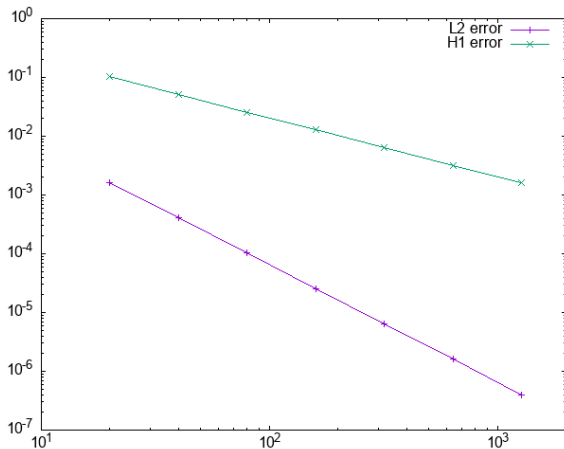


図 8: 1 辺を $n = 20, 40, 80, \dots, 1280$ 分割したときの誤差 (横軸 n)

($h := 1/n$ として) L^2 誤差は $O(h^2)$, H^1 誤差は $O(h)$ となっている (実は理論からの予想と一致)。

参考プログラム — 図8 の作成方法

データは数が少ないので、一つ一つの分割数について、`poisson-mixedBC.edp` を使って誤差を得た。それを次のように記録したファイルを作った。(工夫すれば自動化できるであろう。)

— `error.txt` —

```
20 0.00162987 0.102169
40 0.000408387 0.0511309
80 0.000102155 0.0255713
160 2.55422e-05 0.0127864
320 6.38579e-06 0.00639328
640 1.59646e-06 0.00319665
1280 3.99119e-07 0.00159833
```

— `error.gp` … `gnuplot` でグラフを描き、イメージ・データも作る —

```
set logscale
set format y "10^{%L}"
set format x "10^{%L}"
plot [10:2000] "error.txt" using 1:2 with lp title "L2 error", \
    "error.txt" using 1:3 with lp title "H1 error"
set term png
set output "error.png"
replot
```

1, 4-5 行目が必須。2,3 行目は凡例の体裁を整える。最後の 3 行はイメージ・データ作成のため。ターミナルで `gnuplot error.gp` とすれば、グラフが描かれ、`error.png` が出来る。

参考プログラム — 図8 の作成方法 (続き)

ちなみに (このやり方を勧めているわけではないが)、次のようなシェル・スクリプトを実行して `error.txt` を作成した (データ作成の自動化)。

```
make-data.sh
```

```
#!/bin/sh
rm -f error2.txt
for i in 1 2 4 8 16 32 64
do
    echo ${i}
    FreeFem++ poisson-mixedBC.edp ${i} | grep '^n=' | grep H1 >> error2.txt
done
sed 's/n=//;s/L2-error=//;s/H1-error=/' error2.txt>error.txt
```

```
chmod +x make-data.sh
./make-data.sh
```

データ作成を自動化すると、作成手順の記録が残り、結果の再現も容易になる。記録を残し、再現性を確保することは重要であり、工夫すべきである。

関数定義の話。
gnuplot の紹介

A. (おまけ) C++のストリーム入出力

すでに述べたように、FreeFem++ の入出力は、C++の**ストリーム入出力**の機能に良く似ている (似ているけれど同じではない。同じにすれば良いのに。)

ここでは C++ のストリーム入出力機能の大まかな説明を行う。

A.1 標準入力 cin, 標準出力 cout, 標準エラー出力 cerr

C++のソース・プログラムで次のようにしてあることを仮定する。

```
#include <iostream> // Cの <stdio.h> に相当するような定番
#include <iomanip> // setprecision() 等に必要
using namespace std; // こうしないと std::cin, std::cout, std::cerr とする必要
```

通常は、標準入力は端末のキーボードからの入力、標準出力は端末 (ターミナル) の画面への文字出力、標準エラー出力も端末の画面への文字出力、に結びつけられている (入出力のリダイレクトで、指定したファイルに結びつけることもできる)。

とりあえず、C言語のプログラムの `printf()` を使う代わりに `cout << 式, scanf()` を使う代わりに `cin >> 変数名` を使う、と覚える。

```
double a, b;
cout << "Hello, world" << endl; // endl は改行 \n である。
cout << "Please input two numbers: ";
cin >> a >> b;
cout << "a+b=" << a + b << ", a-b=" << a - b << ", a*b=" << a * b
    << ", a/b=" << a / b << endl;
```


A.2 数値の書式指定

C 言語の `printf()` での書式指定 `"%4d"`, `"%7.2f"`, `"%20.15e"`, `"%25.15g"` は、C++ で使うのはあきらめることを勧める。

- 幅の指定は `<< setw(桁数)` で行う。これは次のフィールドにしか影響しない (必要ならば毎回指定する)。
- 浮動小数点数の小数点以下の桁数の指定は `<< setprecision(桁数)` で行う。
- 浮動小数点数で固定小数点形式での出力の指定は、`<< fixed` で行う。
(C 言語の `%f` に相当)
- 浮動小数点数で指数形式での出力の指定は、`<< scientific` で行う。
(C 言語の `%e` に相当)
- 浮動小数点数でデフォルト形式での出力の指定は、`<< defaultfloat` で行う。
(C 言語の `%g` に相当)

A.2 数値の書式指定

```
// testfloat.cpp --- ナンセンスな計算 (円周率とアボガドロ数の積)
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main(void)
{
    double pi, NA;
    pi = 4.0 * atan(1.0);
    NA = 6.022e+23;
    cout << setprecision(15); //double は 10 進 16 桁弱なので。cout.precision(15); も可
    cout << fixed;
    cout << "π=" << setw(20) << pi << ", NA=" << setw(20) << NA
         << ", π Na=" << setw(20) << pi * NA << endl; // %20.15f 相当
    cout << scientific;
    cout << "π=" << setw(24) << pi << ", NA=" << setw(24) << NA
         << ", π Na=" << setw(24) << pi * NA << endl; // %24.15e 相当
    cout << defaultfloat;
    cout << "π=" << setw(20) << pi << ", NA=" << setw(20) << NA
         << ", π Na=" << setw(20) << pi * NA << endl; // %20.15g 相当
}
```

(実行してみると分かるが、意外と難しい…)

参考文献 I

- [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), <https://www.um.es/freefem/ff++/uploads/Main/NewDevelopmentsInFreeFem.pdf>.
- [3] 桂田祐史：FreeFEM++ ノート,
<https://m-katsurada.sakura.ne.jp/lab/text/freefem-note.pdf>
(2012～).
- [4] 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。
(??).

- [5] Suzuki, A.: Finite element programming by FreeFem++ —intermediate course, 日本応用数学会「産業における応用数理」研究部会のソフトウェアセミナー「FreeFem++ による有限要素プログラミング — 中級編 —」(2016/2/11-12)の配布資料で、<https://www.ljll.math.upmc.fr/~suzukia/FreeFempp-tutorial-JSIAM2016/> から入手できる (2016).
- [6] Suzuki, A.: Finite element programming by FreeFem++ —advanced course, 日本応用数学会「産業における応用数理」研究部会のソフトウェアセミナー「FreeFem++ による有限要素プログラミング — 中級編 —」(2016/6/4-5)の配布資料で、<https://www.ljll.math.upmc.fr/~suzukia/FreeFempp-tutorial-JSIAM2016b/> から入手できる (2016).
- [7] 大塚厚二, 高石武史: 有限要素法で学ぶ現象と数理 — FreeFem++ 数理思考プログラミング —, 共立出版 (2014),
<https://sites.google.com/a/comfos.org/comfos/ffempp> というサポート WWW サイトがある. Maruzen eBook に入っているので,
<https://elib.maruzen.co.jp/elib/html/BookDetail/Id/3000018545> でアクセス出来る.