2004年度卒業研究レポート

１，２次元Poisson方程式にたいする有限要素法

明治大学理工学部 数学科
南木　集

2005年3月24日

目 次

第1章　はじめに

卒業研究ではなるべく自分で考えてプログラムを作成し、プログラミングを通して数値解析を勉強することを心がけた。このレポートの内容は教科書として勉強した菊地 [1] に多くの部分をおっている。扱ったテーマは有限要素法を用いて Poisson 方程式を数値的に解くことである。

有限要素法は、有界領域を有限個の有界領域に近似して各領域において解析する数値解析において差分法と二分する手段である。差文法との大きな違いのひとつは、差分法は複雑な領域に対して適用が難しいのに対し、有限要素法は様々な有界領域に対して解析が可能であるところがあげられる。

テーマとして有限要素法の理解とそのプログラミングを扱った。問題としては Poisson 方程式：

$$\begin{aligned} -\triangle u &= f \quad (\text{in } \Omega)) \\ u &= g_1 \quad (\text{on } \Gamma_1) \\ \frac{\partial u}{\partial \boldsymbol{n}} &= g_2 \quad (\text{on } \Gamma_2) \end{aligned}$$

を扱った。ただし、Ω は有界領域、Γ_1, Γ_2 は Ω の境界 Γ の一部とし、互いに交じりあいのなく、両方を合わせると Γ になるものとする。 g_2, g_2 はそれぞれ Γ_1, Γ_2 にて与えられた既知関数である。

第2章　導入

2.1　弱形式

有限要素法は微分方程式を直接扱わないで、微分方程式を積分により変形して扱う。弱形式を使う方法を紹介しよう。

弱形式を導入する例として

$$
\begin{aligned}
-\triangle u &= f \quad (\text{in } \Omega) && (2.1)\\
u &= g_1 \quad (\text{on } \Gamma_1) && (2.2)\\
\frac{\partial u}{\partial \boldsymbol{n}} &= g_2 \quad (\text{on } \Gamma_2) && (2.3)
\end{aligned}
$$

を考える。ただし、Ω は n 次元有界領域、Γ を Ω の境界とし、導関数が存在するくらいの滑らかさを持つものとする。 Γ_1, Γ_2 は Γ の一部で、$\Gamma_1 \cap \Gamma_2 = \emptyset$ かつ $\bar{\Gamma}_1 \cup \bar{\Gamma}_2 = \Gamma$ を満たし、 g_1, g_2 はそれぞれ Γ_1, Γ_2 で定義された関数である。また、 $\triangle$ は Laplace 作用素

$$
\triangle u = \sum_{i=1}^{n} \frac{\partial^2 u}{\partial x_i^2} \tag{2.4}
$$

である。ただし、空間内の点を $x = (x_1, x_2, \ldots, x_n)$ とする。また、Γ での外向き単位法線ベクトルを $\boldsymbol{n} = (n_1, n_2, \ldots, n_n)$ で表し、$\dfrac{\partial u}{\partial \boldsymbol{n}}$ は Γ における u の外向き方向導関数

$$
\frac{\partial u}{\partial \boldsymbol{n}} = \sum_{i=1}^{n} n_i \frac{\partial u}{\partial x_i} \tag{2.5}
$$

である。また、(2.2),(2.3) はそれぞれ **Dirichlet 境界条件**、**Neumann 境界条件**と呼ばれる。

上記の問題の弱形式を求める。次の関数空間 X を定義する。

$$
X \stackrel{\text{def.}}{=} \{v;\ v = 0 \quad (\text{on } \Gamma_1)\}. \tag{2.6}
$$

(2.1) の両辺に任意の $v \in X$ をかけ Ω で積分する。すなわち

$$-\int_\Omega (\triangle u)v \, dx = \int_\Omega fv \, dx. \tag{2.7}$$

ただし、$\int_\Omega dx$ は $n = 1$ ならば Ω での通常の積分、$n = 2$ ならば Ω での面積分、$n = 3$ ならば Ω での体積分を表す。(2.7) の左辺を Green の公式

$$\sum_{i=1}^{n} \int_\Omega \frac{\partial u}{\partial x_i}\frac{\partial v}{\partial x_i} \, dx = \int_\Gamma v\frac{\partial u}{\partial \boldsymbol{n}} \, d\gamma - \int_\Omega (\triangle u)v \, dx \quad (v \in X) \tag{2.8}$$

を用いて計算すると

$$\sum_{i=1}^{n} \int_\Omega \frac{\partial u}{\partial x_i}\frac{\partial v}{\partial x_i} \, dx - \int_\Gamma v\frac{\partial u}{\partial \boldsymbol{n}} \, d\gamma = \int_\Omega fv \, dx. \tag{2.9}$$

ただし、$\int_\Gamma d\gamma$ は $n = 2$ なら Γ での線積分、$n = 3$ ならば Γ での面積分を表す。(2.9) は Neumann 境界条件 (2.3) と $v \in X$ により

$$\sum_{i=1}^{n} \int_\Omega \frac{\partial u}{\partial x_i}\frac{\partial v}{\partial x_i} \, dx = \int_\Omega fv \, dx + \int_{\Gamma_2} g_2 v \, d\gamma \tag{2.10}$$

となる。(2.10) は (2.1) と比べて u の導関数の階数が低く u に要求されている微分可能性は (2.1) よりも弱められており、その代わりに v の導関数が入っている。(2.10) が偏微分方程式の境界値問題 (2.1)〜(2.3) に対する弱形式である。また、このように使用される v を試験関数、重み関数と呼ぶ。

変分法の基本原理を紹介する。

変分法の基本原理

$f \in L^1_{loc}(\Omega)$ が

$$\int_\Omega f(x)\varphi(x) \, dx = 0 \quad (\forall \varphi \in C_0(\Omega))$$

を満たすならば $f(x) = 0$ (a.e. in Ω).

証明はブレジス [2] の第 IV 章に載っている。

記号の補足:$f \in L^1_{loc}(\Omega)$ とは Ω の任意のコンパクト集合 K に対し

$$\int_K |f(x)|\, dx < +\infty$$

が成り立つ事で、このとき f は局所可積分であるという。また、$C_0(\Omega)$ とは台が Ω のコンパクト部分集合であるような連続関数全体である。

この変分法の基本原理を使って弱形式 (2.10) と Dirichlet 境界条件 (2.2) から元の微分方程式 (2.1) と Neumann 境界条件 (2.3) が導かれる事を以下に示す。

v は $\{v;\ v = 0\ (\text{on}\Gamma)\}$ を満たす任意関数とする (この条件は (2.6) より強い条件である)。弱形式 (2.10) に Green の公式 (2.8) を適用すると

$$\int_\Gamma \frac{\partial u}{\partial \boldsymbol{n}} v\, d\gamma - \int_\Omega (\triangle u) v\, dx = \int_\Omega f v\, dx + \int_{\Gamma_2} g_2 v\, d\gamma. \tag{2.11}$$

$v \in \{v;\ v = 0\ (\text{on}\Gamma)\}$ より左辺第 1 項と右辺第 2 項が消えて次式を得る。

$$\int_\Omega (f + \triangle u) v\, dx = 0. \tag{2.12}$$

変分法の基本原理より

$$f + \triangle u = 0 \quad (\text{a.e. in } \Omega). \tag{2.13}$$

ゆえに

$$f = -\triangle u \quad (\text{a.e. in } \Omega). \tag{2.14}$$

すなわち元の微分方程式 (2.1) が示せた。

次に v は $v \in X = \{v;\ v = 0 \in \Gamma_1\}$ を満たす任意の関数と v に対する条件を緩める。Green の公式 (2.8) を考えると

$$\int_{\Gamma_2} \frac{\partial u}{\partial \boldsymbol{n}} v\, d\gamma = \int_\Omega (\triangle u) v\, dx + \sum_{i=1}^{n} \int_\Omega \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_i}\, dx. \tag{2.15}$$

弱形式 (2.10) より

$$\int_{\Gamma_2} \frac{\partial u}{\partial \boldsymbol{n}} v\, d\gamma = \int_\Omega (\triangle u) v\, dx + \int_\Omega f v\, dx + \int_{\Gamma_2} g_2 v\, d\gamma. \tag{2.16}$$

既に示された (2.1) より

$$\int_{\Gamma_2} \frac{\partial u}{\partial \boldsymbol{n}} v\, d\gamma = \int_{\Gamma_2} g_2 v\, d\gamma. \tag{2.17}$$

変分法の基本原理より

$$\frac{\partial u}{\partial \boldsymbol{n}} = g_2 \quad (\text{a.e. in } \Omega). \tag{2.18}$$

すなわち Neumann 境界条件 (2.3) が示せた。

よって (2.1)〜(2.3) を解く事と X の任意関数 v に対して弱形式 (2.10) を解く事は同値である。

今、Dirichlet 境界条件は弱形式と共に前提として用いた。この様に前提として用いられる境界条件を**基本境界条件**といい、一方、Neumann 境界条件は弱形式の条件から自然に導かれた。この様な境界条件を**自然境界条件**という。

ここで次の記号を導入する。

$$\langle u, v \rangle = \sum_{i=1}^{n} \int_{\Omega} \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_i} \, dx, \tag{2.19}$$

$$(u, v) = \int_{\Omega} uv \, dx, \tag{2.20}$$

$$[u, v] = \int_{\Gamma_2} uv \, d\gamma. \tag{2.21}$$

これらの記号を用いると弱形式 (2.10) は

$$\langle u, v \rangle = (f, v) - [g_2, v] \tag{2.22}$$

と書き直せる。以下、弱形式を書くときにはこの様な記号を用いることにする。

2.2 Galerkin 法

微分方程式を近似的に解くにあたって未知関数 u の近似関数 $\hat{u}$ の形を定め、近似方程式を作成する必要がある。有限要素法では近似方程式を構成する方法として Galerkin 法を用いている。

近似関数の選び方としては既知の関数 ψ_i の 1 次結合により作成し、1 次結合の項数 m を増やすにつれてその精度を上げる方法が一般的に行われている。前節では基本境界条件は前提として用いているので、弱形式を基に近似関数 $\hat{u}$ を構成するには基本境界条件を近似的にでも満たすこ

とが望ましい。すなわち

$$\psi_0(x) \ \fallingdotseq \ g_1(x) \quad (x \in \Gamma_1), \tag{2.23}$$

$$\psi_i(x) \ = \ 0 \quad (1 \leq i \leq m,\ x \in \Gamma_1). \tag{2.24}$$

ただし、ψ_i $(1 \leq i \leq m)$ は 1 次独立とする。そして、近似関数 $\hat{u}$ は (2.23),(2.24) の一次結合

$$\hat{u} = \psi_0 + \sum_{i=1}^{m} a_i \psi_i \tag{2.25}$$

とすれば良い。ただし、a_i は未知の結合係数である。このように近似関数を構成すれば基本境界条件を満たす。

なお、ψ_i $(1 \leq i \leq m)$ を基底関数、$\hat{u}$ を試行関数と呼ぶ。後は基底関数をうまく選び、結合係数 a_i を指定すれば未知係数の数 m が大きいほど、$\hat{u}$ は u の良好な近似関数として期待できるだろう。

次に結合係数 a_i をどのように決めるか考える。前節での弱形式の方法を用いよう。すでに近似関数としては (2.25) の形をしているものを用いることに決定したから、弱形式 (2.22) の u の代役は上で考えた (2.25) の $\hat{u}$ でよい。v としては v が $X = \{v; v = 0 \text{ on} \Gamma_1\}$ の任意関数であるから v をそのまま用いてしまうと $\hat{u}$ には m 個の未知数があるのに対し、任意性ゆえ v の候補は無数に存在してしまい、a_i が存在しないので適当でない。そこで v の代わりに (3.13) に似た形のものを次のように採用する。

$$\hat{v} = \sum_{i=1}^{m} b_i \psi_i. \tag{2.26}$$

ただし、ψ_i $(1 \leq i \leq m)$ は (2.24) で考えた先ほどのもので、b_i は任意の係数である。この $\hat{v}$ は今までの v 同様に $\hat{v}$ も $\hat{v} = 0$ $(x \in \Gamma_1)$ を満たし v としての資格を十分に有する。

この時、近似解 $\hat{u}$ を次の条件で決定する

与えられた f, g_1, g_2 に対し (2.25) の形をした u のうち、(2.26) の形の任意の $\hat{v}$ に対し

$$\langle \hat{u}, \hat{v} \rangle = (f, \hat{v}) + [g_2, \hat{v}] \tag{2.27}$$

を満たす $\hat{u}$ を求めよ。

このようにして近似解を求める方法を **Galerkin 法**と呼ぶ。

ところで、(2.26) の形をした $\hat{v}$ で特に

$$\hat{v} = \psi_i \tag{2.28}$$

と取り、Galerkin 法の式 (2.27) に代入すると

$$\langle \hat{u}, \psi_i \rangle = (f, \psi_i) + [g_2, \psi_i]. \tag{2.29}$$

(2.25) より

$$\langle \psi_0 + \sum_{j=1}^{m} a_j \psi_j, \psi_i \rangle = (f, \psi_i) + [g_2, \psi_i]. \tag{2.30}$$

ゆえに

$$\sum_{j=1}^{m} a_j \langle \psi_j, \psi_i \rangle = (f, \psi_i) + [g_2, \psi_i] - \langle \psi_0, \psi_i \rangle. \tag{2.31}$$

(2.31) は $1 \leq i \leq m$ の範囲の全ての i について成立する。全ての i について考え、行列で表すと

$$\begin{pmatrix} \langle \psi_1, \psi_1 \rangle & \cdots & \langle \psi_m, \psi_1 \rangle \\ \vdots & \ddots & \vdots \\ \langle \psi_1, \psi_m \rangle & \cdots & \langle \psi_m, \psi_m \rangle \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix} = \begin{pmatrix} (f, \psi_1) + [g_2, \psi_1] - \langle \psi_0, \psi_1 \rangle \\ \vdots \\ (f, \psi_m) + [g_2, \psi_m] - \langle \psi_0, \psi_m \rangle \end{pmatrix}. \tag{2.32}$$

(2.32) は基底関数 ψ_i を適当に定めれば $\hat{u}$ の 1 次結合の係数 a_i が求まる。

有限要素法において Galerkin 法は領域を有限個の要素に分割したときに、各要素ごとに (2.27) を考えて用いられる。

第3章　１次元有限要素法のモデル

この章では１次元問題に対する有限要素法の基本的な考え方を簡単に述べる。

有限要素法の流れを簡単に説明すると、はじめにある有界領域 Ω で境界条件とともに与えられた微分方程式から弱形式を求める。次に弱形式を求めたら Ω の閉方 $\bar{\Omega}$ を有限個 (m 個) の小領域 $\hat{\bar{\Omega}}_i \ \ (0 \leq i \leq m-1)$ に近似的に分割する。

$$\bar{\Omega} \cong \hat{\bar{\Omega}} \stackrel{\text{def.}}{=} \bigcup_{i=0}^{m-1} \hat{\Omega}_i. \tag{3.1}$$

分割でできる小領域は１次元では線分、２次元では３角形や長方形、３次元では四面体や六面体、三角柱などを対象とする。分割の際、要素間の重なりや隙間がないようにし、頂点は他の要素の辺上にこないよう注意する。また、有限要素集合体 $\hat{\Omega}$ とその真の境界 Γ との隙間もなるべく小さくする。このとき小領域を (有限) 要素、各要素の頂点を節点と呼ぶ。なお、i 番目の要素を [i] で表し、要素番号、節点番号は０から付けるものとする。

まず、各要素において近似関数 u_i の形[1]を決定する。この u_i を用いて弱形式に対する各要素から全体への寄与分を計算し、組み合わせて全体の近似方程式を作成する。得られた近似方程式 (連立１次方程式) を解けば全体での近似関数 $\hat{u}$ が求まる。

以上が有限要素法の概念である。精度については、一般に有限要素解の精度を上げるには u_i の精度の粗さを向上させるよりも、各要素における近似が良好であれば良いので要素分割を細かくすることで補う。

[1] $x_1, x_2 \ldots$ の比較的低次の多項式がよく使われ, 一般に Galerkin 法の近似関数の形とは異なる

3.1 １次元有限要素法のモデル

有限要素法を説明するにあたってまず簡単な 1 次元の問題で考える。1 次元の場合要素は線分しか考えられないので多次元の場合と比べるととても考えやすい。

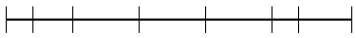

図 3.1: １次元の場合は線分の分割となる

扱う問題は 1 次元の Poisson 方程式とする。

与えられた f に対して次の式を満たす u を求めよ。

$$-\frac{d^2u}{dx^2} = f \quad (x \in (0,1)) \tag{3.2}$$
$$u(0) = \alpha \tag{3.3}$$
$$\frac{du}{dx}(1) = \beta \tag{3.4}$$

ただし、α, β は定数である。

3.2 近似関数の構成

v を $v(0) = 0$ を満たす任意関数として (2.19),(2.20) は

$$\langle u, v\rangle = \int_0^1 \frac{du}{dx}\frac{dv}{dx}\,dx, \tag{3.5}$$
$$(f, v) = \int_0^1 fv\,dx. \tag{3.6}$$

で与えらる。

この問題に対する弱形式は次式で与えられる。

$$\langle u, v\rangle = (f, v) + \beta v(1). \tag{3.7}$$

ただし、v は $v(0) = 0$ を満たす任意関数である。

1 次元の有限要素法ではまず区間 (0,1) を m 個の小区間に分割する。このとき、m 個の要素と $m+1$ 個の節点が存在することになる。節点は $0 = x_0 < x_1 < \cdots < x_m = 1$ という具合に並べられているものとする。

要素 [i] について考える。要素 [i] における u の近似関数 $\hat{u}$ は x の 1 次式であるとする。すなわちパラメータ α_1, α_2 を適当に選んで次式が成立する。

$$\hat{u} = \alpha_1 + \alpha_2 x. \tag{3.8}$$

節点 x_i は要素 [i] と [$i+1$] に属す。このときどちらの要素から見ても $\hat{u}$ の値が u_i と等しくなるようにすれば $\hat{u}$ は要素間で連続になる。すると要素 [i] で次式が成立しなければならない。

$$u_i = \alpha_1 + \alpha_2 x_i, \tag{3.9}$$

$$u_{i+1} = \alpha_1 + \alpha_2 x_{i+1}. \tag{3.10}$$

すなわち

$$\begin{pmatrix} 1 & x_i \\ 1 & x_{i+1} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix} = \begin{pmatrix} u_i \\ u_{i+1} \end{pmatrix}. \tag{3.11}$$

(3.11) は $x_i \neq x_{i+1}$ より解けて

$$\begin{aligned} \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix} &= \begin{pmatrix} 1 & x_i \\ 1 & x_{i+1} \end{pmatrix}^{-1} \begin{pmatrix} u_i \\ u_{i+1} \end{pmatrix} \\ &= \frac{1}{x_{i+1} - x_i} \begin{pmatrix} x_{i+1} & -x_i \\ -1 & 1 \end{pmatrix} \begin{pmatrix} u_i \\ u_{i+1} \end{pmatrix} \\ &= \frac{1}{x_{i+1} - x_i} \begin{pmatrix} x_{i+1}u_i - x_i u_{i+1} \\ -u_i + u_{i+1} \end{pmatrix}. \end{aligned} \tag{3.12}$$

これより

$$\begin{aligned} \hat{u} &= \alpha_1 + \alpha_2 x \\ &= \frac{x_{i+1}u_i - x_i u_{i+1}}{x_{i+1} - x_i} + \frac{-u_i + u_{i+1}}{x_{i+1} - x_i} x \\ &= \frac{x_{i+1} - x}{x_{i+1} - x_i} u_i + \frac{x - x_i}{x_{i+1} - x_i} u_{i+1} \\ &= L_1 u_i + L_2 u_{i+1} \\ &= \sum_{j=1}^{2} L_j u_{i+j-1}.. \end{aligned} \tag{3.13}$$

ただし、

$$L_1 = \frac{x_{i+1} - x}{x_{i+1} - x_i}, \quad L_2 = \frac{x - x_i}{x_{i+1} - x_i} \quad (0 \leq i \leq m-1) \tag{3.14}$$

とする。L_1, L_2 は長さ座標と呼ばれる。

同様に、$\hat{v}$ も要素 [i] で

$$\hat{v} = \sum_{j=1}^{2} L_j v_{i+j-1} \tag{3.15}$$

と定めればよい。

以上により要素 [i] における近似関数 $\hat{u}$ と試験関数 $\hat{v}$ が指定された。

3.3 要素係数行列の計算

要素における近似関数 $\hat{u}$ と試験関数 $\hat{v}$ の節点における値が得られたので、次に Galerkin 法により近似方程式を作成する。通常は各要素から近似方程式への寄与分を計算し、それらを組み合わせて全体の近似方程式を作成する。

各要素から弱形式への寄与分を求めていくのにあたり要素 [i] に対して次の量を定義する。

$$\langle u, v \rangle_i = \int_{x_i}^{x_{i+1}} \frac{du}{dx}(x) \frac{dv}{dx}(x)\, dx, \tag{3.16}$$

$$(f, v)_i = \int_{x_i}^{x_{i+1}} f(x) v(x)\, dx. \tag{3.17}$$

近似関数 $\hat{u}$ と試験関数 $\hat{v}$ を用いると弱形式 (3.7) は次のようになる。

$$\langle \hat{u}, \hat{v} \rangle = (f, \hat{u}) + \beta \hat{v}(1). \tag{3.18}$$

(3.18) を $\langle \hat{u}, \hat{v} \rangle_i$ と $(f, \hat{u})_i$ で書き換えると全ての要素に対して次式で書ける (要素番号は [0] 〜 [$m-1$] とする)。

$$\sum_{i=0}^{m-1} \langle \hat{u}, \hat{v} \rangle_i = \sum_{i=0}^{m-1} (f, \hat{u})_i + \beta \hat{v}(1). \tag{3.19}$$

要素 [i] において $\langle \hat{u}, \hat{v} \rangle_i, (f, \hat{u})_i$ を計算しよう。

$$\begin{aligned}
\langle \hat{u}, \hat{v} \rangle_i &= \langle \sum_{j=1}^{2} L_j u_{i+j-1}, \sum_{k=1}^{2} L_k v_{i+k-1} \rangle_i \\
&= \sum_{j=1}^{2} \sum_{k=1}^{2} u_{i+j-1} \langle L_j, L_k \rangle_i v_{i+k-1} \\
&= \sum_{j=1}^{2} \sum_{k=1}^{2} u_{i+j-1} A_{kj}^{(i)} v_{i+k-1}, \qquad (3.20) \\
(f, \hat{v})_i &= (f, \sum_{k=1}^{2} L_k v_{i+k-1})_i \\
&= \sum_{k=1}^{2} v_{i+k-1} (f, L_k)_i \\
&= \sum_{k=1}^{2} v_{i+k-1} f_k^{(i)}. \qquad (3.21)
\end{aligned}$$

ただし、

$$A_{kj}^{(i)} \stackrel{\text{def.}}{=} \langle L_j, L_k \rangle_i, \qquad (3.22)$$

$$f_k^{(i)} \stackrel{\text{def.}}{=} (f, L_k)_i. \qquad (3.23)$$

とする。また、$\hat{v}(1) = v_m$ であるから

$$\beta \hat{v}(1) = \beta v_m. \qquad (3.24)$$

$0 \leq i \leq m-1$ として次のベクトルと行列を定義する。

$$\boldsymbol{u}_i \stackrel{\text{def.}}{=} \begin{pmatrix} u_i \\ u_{i+1} \end{pmatrix}, \ \boldsymbol{v}_i \stackrel{\text{def.}}{=} \begin{pmatrix} v_i \\ v_{i+1} \end{pmatrix}, \ \boldsymbol{f}_i \stackrel{\text{def.}}{=} \begin{pmatrix} f_1^{(i)} \\ f_2^{(i)} \end{pmatrix}, \ \boldsymbol{g} \stackrel{\text{def.}}{=} \begin{pmatrix} 0 \\ \beta \end{pmatrix}, \qquad (3.25)$$

$$A_i \stackrel{\text{def.}}{=} \begin{pmatrix} A_{11}^{(i)} & A_{12}^{(i)} \\ A_{12}^{(i)} & A_{22}^{(i)} \end{pmatrix}. \qquad (3.26)$$

$\boldsymbol{u}_i, \boldsymbol{v}_i$ は要素節点パラメータ・ベクトル、$\boldsymbol{f}_i$ は要素自由項ベクトル、A_i は要素係数行列と呼ばれる。このとき A_i は $A_{kj}^{(i)} = \langle L_k, L_j \rangle_i = \langle L_j, L_k \rangle_i = A_{jk}^{(i)}$ より対称で

$$\langle \hat{u}, \hat{v} \rangle_i = \boldsymbol{v}_i^T A_i \boldsymbol{u}_i, \qquad (3.27)$$

$$(f, \hat{v})_i = \boldsymbol{v}_i^T \boldsymbol{f}_i, \qquad (3.28)$$

$$\beta \hat{v}(1) = \boldsymbol{v}_{m-1}^T \boldsymbol{g}. \qquad (3.29)$$

ここで T は転置を表す。

A_i, f_i は後で使うことになるので計算しておこう。(3.14) より

$$\frac{dL_1}{dx} = \frac{-1}{x_{i+1} - x_i}, \quad \frac{dL_2}{dx} = \frac{1}{x_{i+1} - x_i}. \tag{3.30}$$

であるから

$$\begin{aligned} A_{jk}^{(i)} &= \langle L_j, L_k \rangle_i \\ &= \int_{x_i}^{x_{i+1}} \frac{dL_j}{dx} \frac{dL_i}{dx}\, dx \\ &= \frac{\mathrm{sgn}(j,k)}{(x_{i+1} - x_i)^2} \int_{x_i}^{x_{i+1}} dx \\ &= \frac{\mathrm{sgn}(j,k)}{x_{i+1} - x_i} \quad (j,k = 1,2). \end{aligned} \tag{3.31}$$

ただし

$$\mathrm{sgn}(j,k) = \begin{cases} 1 & (j = k) \\ -1 & (j \neq k) \end{cases} \tag{3.32}$$

である。これより

$$A_i = \frac{1}{x_{i+1} - x_i} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \tag{3.33}$$

となる。一方

$$f_j^{(i)} = (f, L_j)_i = \int_{x_i}^{x_{i+1}} f(x) L_j(x)\, dx \quad (j = 1,2). \tag{3.34}$$

この右辺の積分は何らかの方法で（近似的にでもいいから）計算しておく。ここでは要素 [i] における f の代わりとして $\bar{f}$（定数関数[2]）を用いて考えると[3]

$$\begin{aligned} f_1^{(i)} = \bar{f} \int_{x_i}^{x_{i+1}} \frac{x_{i+1} - x}{x_{i+1} - x_i} dx &= \frac{\bar{f}(x_{i+1} - x_i)}{2}, \\ f_2^{(i)} = \bar{f} \int_{x_i}^{x_{i+1}} \frac{x - x_i}{x_{i+1} - x_i} dx &= \frac{\bar{f}(x_{i+1} - x_i)}{2}. \end{aligned}$$

より

$$\boldsymbol{f}_i = \frac{\bar{f}(x_{i+1} - x_i)}{2} \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \tag{3.35}$$

[2]例えば要素の両端における f の値の平均とか

[3]他の方法として f を L_1, L_2 の 1 次補間近似して計算する方法など

3.4 直接剛性法 —近似方程式の組み立て—

全小節で与えたベクトル (3.25) と行列 (3.26) を m 次元に拡大する

$$\boldsymbol{u} \stackrel{\text{def.}}{=} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{m-1} \end{pmatrix}, \quad \boldsymbol{v} \stackrel{\text{def.}}{=} \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{m-1} \end{pmatrix}, \tag{3.36}$$

$$\boldsymbol{f}_i^* \stackrel{\text{def.}}{=} \begin{pmatrix} \vdots \\ f_1^{(i)} \\ f_2^{(i)} \\ \vdots \end{pmatrix} \begin{matrix} \\ i \\ i+1 \\ \\ \end{matrix}, \quad \boldsymbol{g}^* \stackrel{\text{def.}}{=} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \beta \end{pmatrix}, \tag{3.37}$$

$$A_i^* = \begin{matrix} \\ \\ \\ \\ \end{matrix} \begin{matrix} \begin{matrix} i & i+1 \end{matrix} \\ \begin{pmatrix} \cdots\cdots\cdots\cdots\cdots \\ \cdots \quad A_{11}^{(i)} \quad A_{12}^{(i)} \quad \cdots \\ \cdots \quad A_{21}^{(i)} \quad A_{22}^{(i)} \quad \cdots \\ \cdots\cdots\cdots\cdots\cdots \end{pmatrix} \end{matrix} \begin{matrix} \\ i \\ i+1 \\ \\ \end{matrix} \tag{3.38}$$

ただし、(3.37)、(3.38) の表示していない成分はすべて 0 である。
これらを用いると

$$\langle \hat{u}, \hat{v} \rangle_i = \boldsymbol{v}^T A_i^* \boldsymbol{u}, \tag{3.39}$$

$$(f, \hat{v})_i = \boldsymbol{v}^T f_k^*, \tag{3.40}$$

$$\beta \hat{v}(1) = \boldsymbol{v}^T \boldsymbol{g}^*. \tag{3.41}$$

また、

$$\langle \hat{u}, \hat{v} \rangle = \sum_{i=0}^{m-1} \langle \hat{u}, \hat{v} \rangle_i, \quad (f, \hat{v}) = \sum_{i=0}^{m-1} (f, \hat{v})_i \tag{3.42}$$

より弱形式は

$$\boldsymbol{v}^T \sum_{i=0}^{m-1} A_i^* \boldsymbol{u} = \boldsymbol{v}^T (\sum_{i=0}^{m-1} \boldsymbol{v}_i^* + \boldsymbol{g}^*). \tag{3.43}$$

また、$A^*, \boldsymbol{f}^*$ を次式で定義する

$$A^* \stackrel{\text{def.}}{=} \sum_{i=0}^{m-1} A_i^*, \quad \boldsymbol{f}^* \stackrel{\text{def.}}{=} \sum_{i=0}^{m-1} \boldsymbol{v}_i^* + \boldsymbol{g}^*. \tag{3.44}$$

(3.44) を用いると (3.43) は

$$\boldsymbol{v}^T A^* \boldsymbol{u} = \boldsymbol{v}^T \boldsymbol{f}^*.$$

すなわち

$$\boldsymbol{v}^T (A^* \boldsymbol{u} - \boldsymbol{f}^*) = 0. \tag{3.45}$$

で与えられる。ここで $\boldsymbol{v}$ は

$$V \stackrel{\text{def.}}{=} \{ \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{m-1} \end{pmatrix} \in \mathbf{R}^m ; v_0 = 0 \}$$

の任意の元であるから、$A^* \boldsymbol{u} - \boldsymbol{f}^*$ は最初の成分以外が 0 になる。すなわち

$$A^{**} \stackrel{\text{def.}}{=} A^* \text{の第 } 0 \text{ 行を除いた } m \times (m+1) \text{行列}, \tag{3.46}$$

$$\boldsymbol{f}^{**} \stackrel{\text{def.}}{=} \boldsymbol{f}^* \text{の第 } 0 \text{ 成分を除いた } m \text{ 次元縦ベクトル}. \tag{3.47}$$

とすれば

$$A^{**} \boldsymbol{u} - \boldsymbol{f}^{**} = 0.$$

すなわち

$$A^{**} \boldsymbol{u} = \boldsymbol{f}^{**}. \tag{3.48}$$

また、Dirichlet 境界条件により $u_0 = \alpha$ は既知なので (3.48) から u_0 を消去できる。A^{**} を列ベクトルで $A^{**} = (\boldsymbol{a_0} \boldsymbol{a_1} \dots \ \boldsymbol{a}_m)$ で表すと

$$\sum_{k=0}^{m} \boldsymbol{a_k} u_k = \boldsymbol{f^{**}}. \tag{3.49}$$

左辺の Dirichlet 境界条件に関する項を移項すれば

$$\sum_{k=1}^{m} \boldsymbol{a_k} u_k = \boldsymbol{f^{**}} - \boldsymbol{a_0} u_0. \tag{3.50}$$

すなわち

$$A \boldsymbol{u}^* = \boldsymbol{f}. \tag{3.51}$$

ただし、

$$A \stackrel{\text{def.}}{=} A^{**} \text{の第 } 0 \text{ 列を除いた } m \text{ 次正方行列}, \tag{3.52}$$

$$\boldsymbol{u}^* \stackrel{\text{def.}}{=} \boldsymbol{u} \text{ の第 } 0 \text{ 成分を除いた } m \text{ 次元縦ベクトル}, \tag{3.53}$$

$$\boldsymbol{f} \stackrel{\text{def.}}{=} \boldsymbol{f}^{**} - \boldsymbol{a_0} \alpha. \tag{3.54}$$

以上のようにして要素ごとに求めた量の和を取って近似方程式を組み立てるという方法を **直接剛性法** と呼ぶ。

3.5　近似方程式の具体例１

この節では具体的な例を与え、近似方程式がどのようになるかを見ていく。

問題は Poisson 方程式の Dirichlet-Neumann 境界条件 (3.2),(3.3),(3.4) を考え、簡単のため $f(x) = \bar{f}$(定数関数) とする。

ここでは領域 $\Omega = (0, 1)$ を 4 等分して考えよう。よって $m = 4$、分割幅 h は $h = \dfrac{1}{4}$ であるから

$$x_i = ih \quad (i = 0, 1, 2, 3, 4). \tag{3.55}$$

各要素は

$$[\, i \,] = (x_i, x_{i+1}) \quad (i = 0, 1, 2, 3). \tag{3.56}$$

で与えられる。すると

$$A_i = \frac{1}{x_{i+1} - x_i} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \tag{3.57}$$

$$\boldsymbol{f}_i = \frac{\bar{f}(x_{i+1} - x_i)}{2} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{\bar{f}h}{2} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \tag{3.58}$$

$$\boldsymbol{g} = \begin{pmatrix} 0 \\ \beta \end{pmatrix}, \tag{3.59}$$

$$
\begin{aligned}
A^* = & \frac{1}{h}\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} + \frac{1}{h}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
& + \frac{1}{h}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} + \frac{1}{h}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \\
= & \frac{1}{h}\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}, \qquad (3.60)
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{f}^* &= \frac{\bar{f}h}{2}\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \frac{\bar{f}h}{2}\begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \frac{\bar{f}h}{2}\begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} + \frac{\bar{f}h}{2}\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} + \frac{\bar{f}h}{2}\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \beta \end{pmatrix} \\
&= \frac{\bar{f}h}{2}\begin{pmatrix} 1 \\ 2 \\ 2 \\ 2 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \beta \end{pmatrix}. \qquad (3.61)
\end{aligned}
$$

よって近似方程式 $A^{**}\boldsymbol{u} = \boldsymbol{f}^{**}$ は

$$
\frac{1}{h}\begin{pmatrix} -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}\begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \frac{\bar{f}h}{2}\begin{pmatrix} 1 \\ 2 \\ 2 \\ 2 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \beta \end{pmatrix}. \qquad (3.62)
$$

$u_0 = \alpha$ を代入して消去すると $A\boldsymbol{u}^* = \boldsymbol{f}$ を得る。

$$\frac{1}{h}\begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \frac{\bar{f}h}{2}\begin{pmatrix} 2 \\ 2 \\ 2 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \beta \end{pmatrix} + \frac{1}{h}\begin{pmatrix} \alpha \\ 0 \\ 0 \\ 0 \end{pmatrix}. \quad (3.63)$$

3.6 １次元問題のプログラム

実際に 1 次元のプログラムを作成し、実験を行った。

例題

１次元の Poisson 方程式

$$\begin{aligned} -\frac{\partial^2 u}{\partial x^2} &= f(x) \quad (x \in (0,1)) \\ u(0) &= \alpha \\ \frac{\partial u}{\partial x}(1) &= \beta \end{aligned}$$

を有限要素法で解け。ただし、α, β は実数である。

以下にソースプログラムの説明をする。

- 関数 input()

  本来有限要素法の入力処理は入力ファイルを用意するのが多いようであるが、１次元のプログラムでは比較的簡単に入力処理ができるため Dirichlet 境界条件の値 α と Neumann 境界条件の値 β を入力するようになっている。このプログラムでは領域 (区間) の分割は等分割にしている。

- 関数 assem()

  直接剛性法を行う。まず、各 $\boldsymbol{f}_i, A_i$ を計算し、$\boldsymbol{f}_i, A_i$ から $\sum_{i=0}^{m-1} \boldsymbol{f}_i, A^*$ を作成する。

  $\bar{f}$ としては要素の両端での値の平均値

  $$\bar{f} = \frac{f(x_i) + f(x_{i+1})}{2} \quad (3.64)$$

  を採用している。

- 関数 bound()

  Dirichlet 境界条件、Neumann 境界条件の処理をする。$A^*, \boldsymbol{f}^*$ を作成。最終的には A^* から α を含む項を移項した

$$\left(\begin{array}{c|c} 1 & \mathbf{0} \\ \hline \mathbf{0} & A^{**} \end{array}\right) \tag{3.65}$$

  を作成する。

- 関数 gauss()

  対称行列に対するガウスの消去法。

- 関数 output()

  節点における有限要素解を出力。

- 関数 func()

  Poisson 方程式 $-u'' = f(x)$ の $f(x)$ に相当。下のプログラムでは $f(x) \equiv 1$（定数関数）としているが、$f(x)$ を他の関数に変更するには DEBUG の外の return 文を変更すればよい。

- 条件付コンパイル部分 DEBUG

  次節の例で有限要素解と厳密解との比較できるよう $f(x) \equiv 1$ の時に有限要素解が厳密解とどの程度差があるか見られるようにした。

- 可視化 $\cdots$ GLSC

  解の様子をグラフィックスライブラリィglsc.h によってグラフにした。g_ や g_ で始まる関数はすべて GLSC の関数である。また、α, β によっては解がグラフ描画範囲 $(0,1) \times (0,1)$ を超えてしまうので注意が必要である。

  main() 中では座標軸と有限要素解の挙動、DEBUG を定義すれば $f(x) \equiv 1$ の場合の厳密解も描く。厳密解は関数 func1() により与えている。なお、有限要素解は赤の実線、厳密解は黒の点線で描いている。

以下にプログラムを載せる。

```
/*--------------------------------------------------------------------------
 *    fem1.c
 *
 *    扱いたい問題は：1 次元の Poisson 方程式
 *
 *                    -(d^2u/dx^2) = f(x)          ( x ∈ (0,1) )
 *                    u(0) = α , (du/dx)(1) = β
 *
 *    を満たす u = u(x) を求めよ。
 *
 *    Gauss の消去法を用いた正方行列版。
 *    分割は区間の等分割である。
 --------------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>

#define G_DOUBLE
#include <glsc.h>

#define nelmt (10)              /* 分割数 , 要素数 */
#define nnode (nelmt + 1)       /* 節点数 , 未知数の数*/

//#define DEBUG

typedef double **matrix;

void input(double *h, double *x, double *alpha, double *beta,
           double a, double b);
void assem(double h, double x[], matrix fi, double f[],
           double Ai[][2], matrix A);
void bound(matrix A, double f[], double alpha, double beta);
void gauss(double f[], matrix A);
void output(double f[], double a, double h);
double func(double x);
#ifdef DEBUG
double func1(double x, double alpha, double beta);
#endif

matrix new_matrix(int m, int n)
{
  int i;
  double *ip;
  matrix a;

  if((a = malloc(sizeof(double *) * m)) == NULL)
    return NULL;
  if((ip = malloc(sizeof(double) * (m * n))) == NULL){
    free(a);
    return NULL;
```

```
  }
  for(i=0; i<m; i++)
    a[i] = ip + (i * n);

  return a;
}

int main()
{
  int i;
  double h;                 /* 分割幅 */
  double *x;                /* 節点 */
  matrix fi;                /* 局所要素節点パラメータ・ベクトル */
  double *f;                /* 大域要素節点パラメータ・ベクトル */
  double Ai[2][2];          /* 局所要素係数行列 */
  matrix A;                 /* 大域要素係数行列 */
  double alpha, beta;       /* 境界値α、β */
  double win_width, win_height, w_margin, h_margin;
  double a, b;              /* 領域 (a,b) */
  double delta;

  if ((x = malloc(sizeof(double) * nnode)) == NULL){
    fprintf(stderr, "x のメモリ確保に失敗\n");
    exit(1);
  }
  if((fi = new_matrix(nelmt, 2)) == NULL){
    fprintf(stderr, "fi のメモリ確保に失敗\n");
    exit(1);
  }
  if((A = new_matrix(nnode, nnode)) == NULL){
    fprintf(stderr, "A のメモリ確保に失敗\n");
    exit(1);
  }
  if((f = malloc(sizeof(double) * nnode)) == NULL){
    fprintf(stderr, "f のメモリ確保に失敗\n");
    exit(1);
  }

  a= 0.0; b= 1.0;
  input(&h, x, &alpha, &beta, a, b);
  assem( h, x, fi, f, Ai, A);
  bound(A, f, alpha, beta);
  gauss( f, A);
  output( f, a, h);
  /* グラフ表示 GLSC */
  win_width = 200.0; win_height = 160.0; w_margin = 40.0; h_margin = 10.0;
  g_init("METAfem1", win_width + 2 * w_margin, win_height + 2 * h_margin);
  g_device(G_BOTH);
  g_def_scale(0,
```

```
              0.0, 1.0, 0.0, 1.0,
              w_margin, h_margin, win_width, win_height);
  g_def_line(0, G_BLACK, 2, G_LINE_SOLID);
  g_def_line(1, G_RED, 2, G_LINE_SOLID);
  g_cls();
  g_sel_scale(0); g_sel_line(0);
  g_move(a, 0.0); g_plot(b, 0.0);  /* 座標軸 */
  g_move(0.0, 0.0); g_plot(0.0, 1.0);
  g_sel_line(1);
  g_move(a, alpha);
  for(i=1; i<nnode; i++){
    delta = a + i * h;
    g_plot(delta, f[i]);
  }
#ifdef DEBUG
  /* f(x)=1.0 の時は厳密解も描く */
  g_def_line(2, G_BLACK, 2, G_LINE_DOTS);
  g_sel_line(2);
  g_move(a, alpha);
  for(i=1; i<1000; i++){
    delta = a + i * (b-a) / 1000.0;
    g_plot(delta, func1(delta, alpha, beta));
  }
#endif
  printf("\n終わりました。X の場合はウィンドウをクリックしてください。\n\n");
  g_sleep(-1.0);
  g_term();

  return 0;
}
/* INPUT 入力処理 */
void input( double *h, double x[], double *alpha, double *beta,
            double a, double b)
{
  int i;

  *h = (b-a)/nelmt;
  x[0] = 0.0;
  for(i=1; i<nnode; i++)
    x[i] = i * (*h);
  printf(" ( α , β ) = ");
  scanf("%lf%lf", alpha, beta);
}

/* ASSEM 直接剛性法を行う関数 全体要素係数行列を作成 */
void assem(double h, double x[], matrix fi, double f[],
           double Ai[][2], matrix A)
{
  int i, j;
```

```
  for(i=0; i<nnode; i++){
    f[i] = 0.0;
    for(j=0; j<nnode; j++)
      A[i][j] = 0.0;
  }
  /* 関数 func の代表値は要素の中点の値とする */
  for(i=0; i<nelmt; i++){
    fi[i][0] = func( (x[i]+x[i+1])/2.0)*0.5*h;
    fi[i][1] = func( (x[i]+x[i+1])/2.0)*0.5*h;
  }
  Ai[0][0] = 1.0/h;   Ai[0][1] = -1.0/h;
  Ai[1][0] = -1.0/h;  Ai[1][1] = 1.0/h;
  for(i=0; i<nelmt; i++){
    f[i] += fi[i][0];
    f[i+1] += fi[i][1];
    A[i][i] += Ai[0][0];
    A[i][i+1] += Ai[0][1];
    A[i+1][i] += Ai[1][0];
    A[i+1][i+1] += Ai[1][1];
  }

}

void bound(matrix A, double f[], double alpha, double beta)
{
  int i;

  A[0][0] = 1.0;
  f[0] = alpha;
  for(i=1; i<nnode; i++){
    f[i] -= A[i][0]*alpha;
    A[i][0] = 0.0;
    A[0][i] = 0.0;
  }
  f[nnode-1] += beta;
}
/* SOLVE ガウスの消去法 対称行列版 */
void gauss(double f[], matrix A)
{
  int i, j, k;
  double q;
  /* 前進消去（対称行列版） */
  for(i=0; i<nnode-1; i++)
    for(j=i+1; j<nnode; j++){
      q = A[i][j] / A[i][i];
      f[j] -= q*f[i];
      for(k=j; k<nnode; k++)
        A[j][k] -= q*A[i][k];
```

```
    }
  /* 後退代入 */
  f[nnode-1] /= A[nnode-1][nnode-1];
  for(i=nnode-2; i>=0; i--){
    for(j=i+1; j<nnode; j++)
      f[i] -= A[i][j]*f[j];
    f[i] /= A[i][i];
  }

}
/* OUTPUT 出力 */
void output(double f[], double a, double h)
{
  int i;

  printf("\n");
  for(i=0; i<nnode; i++){
    printf("u[%8g]=%10g;  ", a + i * h, f[i]);
    if(i%3 == 2)
      printf("\n");
  }

}
/* f (Poisson 方程式 -u''=f) */
double func(double x)
{
#ifdef DEBUG
  return 1.0;
#endif

  return 1.0;
}
/* 厳密解  厳密解は u(x) = -1/2x^2 + (β+1)x + α (f(x)=1.0 の場合)*/
#ifdef DEBUG
double func1(double x, double alpha, double beta){
  return -0.5*x*x + (beta+1.0)*x + alpha;
}
#endif
```

3.7 数値実験

解いた問題は

$$-\frac{d^2u}{dx^2} = 1 \quad (x \in (0,1)\ ) \tag{3.66}$$
$$u(0) = \alpha \tag{3.67}$$
$$\frac{du}{dx}(1) = \beta \tag{3.68}$$

である。この問題の厳密解は

$$u(x) = -\frac{1}{2}x^2 + (\beta+1)x + \alpha \tag{3.69}$$

で与えられる。

（$\alpha = 0, \beta = 0$, 分割数 $m = 10$ の場合の実行結果）

```
u[         0]=         0;  u[       0.1]=     0.095;  u[       0.2]=      0.18;
u[       0.3]=     0.255;  u[       0.4]=      0.32;  u[       0.5]=     0.375;
u[       0.6]=      0.42;  u[       0.7]=     0.455;  u[       0.8]=      0.48;
u[       0.9]=     0.495;  u[         1]=       0.5;
```

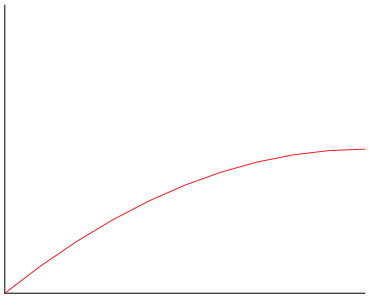

図 3.2: $\alpha = 0, \beta = 0$ の有限要素解の挙動

$\alpha = 0, \beta = 0$ の場合の厳密解は

$$\begin{aligned} u(x) &= -\frac{1}{2}x^2 + x \\ &= -\frac{1}{2}(x-1)^2 + \frac{1}{2} \end{aligned} \tag{3.70}$$

で与えられる。

分割数 $m = 10$ とした上の実行結果と厳密解を比較すると、各節点における有限要素解と厳密解との差は 0 で見事に有限要素解と厳密解が一致している。分割数を変えて比較してみてもやはり各節点における有限要素解と厳密解の誤差は発生しない。そのためこの問題の場合、分割数を増やせば増やすほど各節点における有限要素解の挙動と厳密解の挙動と次第に近い挙動が得られる。

1 次元問題の場合では、領域 Ω は区間であり分割しても過不足なく分割できる。つまり $\Omega = \bigcup_{i=0}^{m-1} \hat{\Omega}_i$ が成り立っている。この事も各節点における有限要素解と厳密解が一致していることに役立っていることだろう。

第4章　2次元有限要素法のモデル

次に2次元問題を考える。1次元問題は要素分割例が線分のみであるのに対して、2次元問題では3角形、長方形などの要素分割が考えられる。ここでは3角形分割により有限要素法の2次元問題を考える。

4.1　近似関数の構成

2次元問題の例として次のPoisson方程式を考える。

与えられた f, g_1, g_2 に対して次式を満たす u を求めよ。

$$-\Delta u = f \quad (\ (x,y) \in \Omega) \tag{4.1}$$

$$u(x,y) = g_1 \quad (\ (x,y) \in \Gamma_1) \tag{4.2}$$

$$\frac{du}{d\boldsymbol{n}} = g_2 \quad (\ (x,y) \in \Gamma_2) \tag{4.3}$$

ただし、Ω は2次元有界領域、Γ を Ω の境界、Γ_1, Γ_2 は Γ の一部であり、$\Gamma_1 \cap \Gamma_2 = \emptyset$ かつ $\Gamma_1 \cup \Gamma_2 = \Gamma$ を満たすものとする。

領域 Ω を m 個の3角形要素に近似的に分割する

$$\Omega \cong \hat{\Omega} \stackrel{\text{def.}}{=} \bigcup_{i=0}^{m-1} e_i. \tag{4.4}$$

ただし、e_i $(i = 0, 1, \ldots m-1)$ は3角形要素である。

領域の種類が線分くらいであろう1次元領域とは異なり、2次元領域では領域の境界が曲がっている場合がある。そういった領域の場合には $\Omega = \hat{\Omega}$ となることは期待できないことを注意する。

有限要素の頂点を節点と呼び、$\hat{\Omega}$ の全体の節点に 0 番から通し番号をつけておく。また、各要素が全体のどの節点に対応するのかわかるようにするために各要素の局所的な節点に対しても番号をつける。ここでは 3 角形分割を用いるため各要素の局所的な節点番号を $1^*, 2^*, 3^*$ とする。この各要素の局所的な節点番号と全体的な節点番号との対応は表として記憶しておく。

1 次元の時同様に要素 [e] における近似関数 $\hat{u}$ を x, y の 1 次関数とする。すなわち

$$\hat{u} = \alpha_1 + \alpha_2 x + \alpha_3 y. \tag{4.5}$$

ただし、$\alpha_1, \alpha_2, \alpha_3$ はパラメータである。

要素間の連続性を保障するため要素 [e] の節点における近似関数 $\hat{u}$ の値 u_i を節点パラメータとして用いる。このとき次式が成立する

$$\text{節点 } i \text{ で} \qquad \hat{u} = u_i = \alpha_1 + \alpha_2 x_i + \alpha_3 y_i \quad (1 \le i \le 3).$$

ただし、節点の座標を (x_i, y_i) $(1 \le i \le 3)$ と置いた。

行列で書くと

$$\begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}.$$

すなわち

$$\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{pmatrix}^{-1} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}. \tag{4.6}$$

右辺の逆行列は

$$\begin{aligned} D &\overset{\text{def.}}{=} \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} \\ &= \begin{vmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_1 & y_3 - y_1 \end{vmatrix} = 2|e| \neq 0. \end{aligned} \tag{4.7}$$

ただし、$|e|$ は要素 [e] の面積である。

よって逆行列を持ち、

$$\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \frac{1}{D} \begin{pmatrix} x_2 y_3 - x_3 y_2 & x_3 y_1 - x_1 y_3 & x_1 y_2 - x_2 y_1 \\ y_2 - y_3 & y_3 - y_1 & y_1 - y_2 \\ x_3 - x_2 & x_1 - x_3 & x_2 - x_1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}. \tag{4.8}$$

(i,j,k) を $(1,2,3)$ の偶置換として次式を定義する。

$$a_i \overset{\text{def.}}{=} \frac{x_j y_k - x_k y_j}{D}, \quad b_i \overset{\text{def.}}{=} \frac{y_j - y_k}{D}, \quad c_i \overset{\text{def.}}{=} \frac{x_k - x_j}{D}. \tag{4.9}$$

また、次式を定義する。

$$L_i(x,y) \overset{\text{def.}}{=} a_i + b_i x + c_i y \quad (1 \le i \le 3). \tag{4.10}$$

この (L_1, L_2, L_3) は (x,y) の面積座標と呼ばれる。

以上の記号を用いると $\hat{u}$ は次式で書ける

$$\hat{u} = \sum_{i=1}^{3} u_i (a_i + b_i x + c_i y) = \sum_{i=1}^{3} u_i L_i(x,y) \quad (\ (x,y) \in [\ e\ ]\ ). \tag{4.11}$$

以上により各要素内では x, y の 1 次式で、領域 Ω では連続な区分多項式の近似関数 $\hat{u}$ が得られた。

また、Dirichlet 境界条件の近似については対応する境界上の節点での節点パラメータ u_i を関数 g_1 (on Γ_1) の値に等しくすればよい。

$\hat{v}$ も同様に

$$\hat{v} = \sum_{i=1}^{3} v_i L_i(x,y) \tag{4.12}$$

として $\hat{v} = 0$ (on Γ_1) を満たすように構成する。すなわち

$$v_i(x,y) = 0 \quad (\ (x,y) \in \Gamma_1,\ 1 \le i \le 3\ ). \tag{4.13}$$

4.2 要素係数行列の計算

$$\langle u, v \rangle_e \overset{\text{def.}}{=} \iint_e \left(\frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y} \right) dxdy. \tag{4.14}$$

$$(f, v)_e \overset{\text{def.}}{=} \iint_e f v \, dxdy. \tag{4.15}$$

という記号を用いれば要素数 m、要素を e_k ($0 \le k \le m-1$) として弱形式 $\langle \hat{u}, \hat{v} \rangle = (f, \hat{v}) + [g_2, \hat{v}]$ は次式になる

$$\sum_{k=0}^{m-1} \langle \hat{u}, \hat{v} \rangle_{e_k} = \sum_{k=0}^{m-1} (f, \hat{v})_{e_k} + [g_2, \hat{v}]\,. \tag{4.16}$$

$\langle \hat{u}, \hat{v} \rangle_{e_k}, (f, \hat{v})_{e_k}$ を計算する

$$\begin{aligned} \langle \hat{u}, \hat{v} \rangle_{e_k} &= \langle \sum_{j=1}^{3} u_j L_j, \sum_{i=1}^{3} v_i L_i \rangle_{e_k} = \sum_{i=1}^{3} \sum_{j=1}^{3} v_i \langle L_j, L_i \rangle_{e_k} u_j \\ &= \sum_{i=1}^{3} \sum_{j=1}^{3} v_i A_{ij}^{(e_k)} u_j, \end{aligned} \tag{4.17}$$

$$(f, \hat{v})_{e_k} = (f, \sum_{i=1}^{3} v_i L_i)_{e_k} = \sum_{i=1}^{3} v_i (f, L_i)_{e_k} = \sum_{i=1}^{3} v_i f_i^{(e_k)}. \tag{4.18}$$

ただし、

$$A_{ij}^{(e_k)} = \langle L_j, L_i \rangle_{e_k}, \quad f_i^{(e_k)} = (f, L_i)_{e_k} \quad (\ 1 \leq i, j \leq 3\ ). \tag{4.19}$$

次のベクトルと行列を定義する

$$\boldsymbol{u}_{e_k} \stackrel{\text{def.}}{=} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}, \quad \boldsymbol{v}_{e_k} \stackrel{\text{def.}}{=} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}, \quad \boldsymbol{f}_{e_k} \stackrel{\text{def.}}{=} \begin{pmatrix} f_1^{(e_k)} \\ f_2^{(e_k)} \\ f_3^{(e_k)} \end{pmatrix}, \tag{4.20}$$

$$A_{e_k} \stackrel{\text{def.}}{=} \begin{pmatrix} A_{11}^{(e_k)} & A_{12}^{(e_k)} & A_{13}^{(e_k)} \\ A_{21}^{(e_k)} & A_{22}^{(e_k)} & A_{23}^{(e_k)} \\ A_{31}^{(e_k)} & A_{32}^{(e_k)} & A_{33}^{(e_k)} \end{pmatrix}. \tag{4.21}$$

すると A_{e_k} は対称行列で

$$\langle \hat{u}, \hat{v} \rangle_{e_k} = \boldsymbol{v}_{e_k}^T A_{e_k} \boldsymbol{u}_{e_k}, \quad (f, \hat{v})_{e_k} = \boldsymbol{v}_{e_k} \boldsymbol{f}_{e_k}. \tag{4.22}$$

となる。

$A_{e_k}, \boldsymbol{f}_{e_k}$ を具体的に求める。

$$\begin{aligned} A_{ij}^{(e_k)} &= \langle L_j, L_i \rangle_{e_k} \\ &= \iint_{e_k} \left(\frac{\partial L_j}{\partial x} \frac{\partial L_i}{\partial x} + \frac{\partial L_j}{\partial y} \frac{\partial L_i}{\partial y} \right) dxdy \\ &= \iint_{e_k} (b_j b_i + c_j c_i)\, dxdy \\ &= (b_j b_i + c_j c_i) |e_k|. \end{aligned} \tag{4.23}$$

$\boldsymbol{f}_{e_k}$ の第 i 成分を計算すると

$$f_i^{(e_k)} = (f, L_i)_{e_k} = \iint_{e_k} f(x,y) L_i(x,y)\, dxdy \quad (\ 1 \le i \le 3\ ). \tag{4.24}$$

$f(x,y)$ が要素 e_k において $f(x,y) \equiv \bar{f}$（定数関数）[1]として考えると

$$f_i^{(e_k)} = \bar{f} \iint_{e_k} L_i(x,y)\, dxdy. \tag{4.25}$$

ここで右辺の積分を計算するには次の面積座標の積分公式が便利である。

面積座標の積分公式

L_1, L_2, L_3 を 3 角形 e に関する面積座標とすると、次式が成立する。

$$\begin{aligned} I(l,m,n) &= \iint_e L_1^l(x,y) L_2^m(x,y) L_3^n(x,y)\, dxdy & (4.26)\\ &= 2S \frac{l!m!n!}{(l+m+n+2)!} & (4.27) \end{aligned}$$

ただし、l, m, n は非負整数、S は e の面積である。

証明は菊地 [1] に載っている。

この公式を使えば、

$$f_i^{(e_k)} = \frac{\bar{f}S}{3} \quad (1 \le i \le 3). \tag{4.28}$$

よって、

$$\boldsymbol{f}_{e_k} = \frac{\bar{f}S}{3} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \tag{4.29}$$

次に Neumann 境界条件の処理を考えよう。

弱形式 (4.16) の右辺第 2 項が Neumann 境界条件の対象となる。

要素 e の辺には Γ_2 を含む辺があるものとし、Γ_2 のうち e 属す部分を $\Gamma_2^{(e)}$ とし、e の 3 頂点をそれぞれ $1^*, 2^*, 3^*$、Γ_2 は辺 2^*3^* 上にあるとする。L_{23} を辺 2^*3^* の長さとし、辺 2^*3^* 上に長さに沿って座標 s を考える。s は頂点 2^* で $s = 0$、頂点 3^* で $s = L_{23}$ とすると、$\hat{v}$ は

$$\begin{aligned} \hat{v} &= \frac{L_{23} - s}{L_{23}} v_2 + \frac{s}{L_{23}} v_3 \\ &= (1 - \frac{s}{L_{23}}) v_2 + \frac{s}{L_{23}} v_3. \end{aligned} \tag{4.30}$$

[1]要素内での平均的な値を取ればよい。例えば、各節点におけるの f の値の平均値など。

よって、

$$
\begin{aligned}
[g_2, \hat{v}] &= \int_{\Gamma_2^{(e)}} g_2 \hat{v}\, d\gamma = \int_{\Gamma_2^{(e)}} g_2 \{(1 - \frac{s}{L_{23}})v_2 + \frac{s}{L_{23}} v_3\}\, d\gamma \\
&= \begin{pmatrix} v_1 & v_2 & v_3 \end{pmatrix} \begin{pmatrix} 0 \\ \displaystyle\int_{\Gamma_2^{(e)}} (1 - \frac{s}{L_{23}}) g_2\, ds \\ \displaystyle\int_{\Gamma_2^{(e)}} \frac{s}{L_{23}} g_2\, ds \end{pmatrix}. \qquad (4.31)
\end{aligned}
$$

特に、$\Gamma_2^{(e)}$ において g_2 の代表値をとり $g_2 \equiv \bar{g}_2$ とすれば

$$
\int_{\Gamma_2^{(e)}} \frac{s}{L_{23}} g_2\, ds = \frac{\bar{g}_2}{L_{23}} \int_{\Gamma_2^{(e)}} s\, ds = \frac{\bar{g}_2}{L_{23}} \int_0^{L_{23}} s\, ds = \frac{\bar{g}_2 L_{23}}{2}.
$$

よって

$$
[g_2, \hat{v}] = \frac{\bar{g}_2 L_{23}}{2} \boldsymbol{v}_e^T \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}. \qquad (4.32)
$$

同様に、Γ_2 が辺 1^*2^*, 辺 3^*1^* 上であればそれぞれ

$$
[g_2, \hat{v}] = \frac{\bar{g}_2 L_{12}}{2} \boldsymbol{v}_e^T \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \text{(辺 } 1^*2^* \text{ が } \Gamma_1 \text{ 上の場合)}, \qquad (4.33)
$$

$$
[g_2, \hat{v}] = \frac{\bar{g}_2 L_{31}}{2} \boldsymbol{v}_e^T \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \text{(辺 } 3^*1^* \text{ が } \Gamma_1 \text{ 上の場合)}. \qquad (4.34)
$$

となる。また、要素によっては要素の 2 つの辺が Γ_2 を含む要素もある。例えば要素 [e] の辺 1^*2^* と 2^*3^* が Γ_2 の一部である場合は

$$
[g_2, \hat{v}] = \frac{\bar{g}_2}{2} \boldsymbol{v}_e^T \begin{pmatrix} L_{12} \\ L_{12} + L_{23} \\ L_{23} \end{pmatrix} \qquad (4.35)
$$

となる。

4.3 直接剛性法 —近似方程式の組み立て—

Ω に節点番号を与え節点数を m 個とする。前ページで与えたベクトル、行列を m 次元に拡大する

$$\boldsymbol{u} \overset{\text{def.}}{=} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{m-1} \end{pmatrix}, \quad \boldsymbol{v} \overset{\text{def.}}{=} \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{m-1} \end{pmatrix}.$$

要素 e_k $(1 \leq k \leq m-1)$ を 1 つ取って考える。e_k の局所的な節点番号 $1^*, 2^*, 3^*$ に対して Ω の全体節点番号がそれぞれ i, j, k が対応しているとする。

次のベクトル $\boldsymbol{f}^*_{e_k}, \boldsymbol{g}^*_{ij}$、行列 $A^*_{e_k}$ を定義する。

$$\boldsymbol{f}^*_{e_k} \overset{\text{def.}}{=} (\ldots \overset{i}{f_1} \ldots \overset{j}{f_2} \ldots \overset{k}{f_3} \ldots)^T \tag{4.36}$$

f_1は第 i 成分、f_2は第 j 成分、f_3は第 k 成分であり、示していない成分はすべて 0 である。

$$A^*_{e_k} \overset{\text{def.}}{=} \begin{pmatrix} & \overset{i}{} & & \overset{j}{} & & \overset{k}{} & \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & A^{(e_k)}_{11} & \cdots & A^{(e_k)}_{12} & \cdots & A^{(e_k)}_{13} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & A^{(e_k)}_{21} & \cdots & A^{(e_k)}_{22} & \cdots & A^{(e_k)}_{23} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & A^{(e_k)}_{31} & \cdots & A^{(e_k)}_{31} & \cdots & A^{(e_k)}_{33} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} \begin{matrix} \\ i \\ \\ j \\ \\ k \\ \end{matrix} \tag{4.37}$$

示していない成分はすべて 0 であり、行列の上側に書いた i, j, k は $A^*_{e_k}$ の成分のくる列番号、右側に書いた i, j, k は行番号である。

$$\boldsymbol{g}^*_{ij} \overset{\text{def.}}{=} \frac{g_2 L_{ij}}{2} (\ldots \overset{I}{1} \ldots \overset{J}{1} \ldots)^T \quad (1 \leq i, j \leq 3,\ i \neq j). \tag{4.38}$$

ただし、示していない成分はすべて 0 であり、i, j は成分番号で、それぞれ局所節点番号 i^*, j^* $(1 \leq i, j \leq 3,\ i \neq j)$ が対応する全体節点番号である。

これらの記号を用いると弱形式は

$$\sum_{k=0}^{m-1} \boldsymbol{v}^T A^*_{e_k} \boldsymbol{u} = \sum_{k=0}^{m-1} \boldsymbol{v}^T \boldsymbol{f}^*_k + \sum_{\text{辺 } ij \subset \Gamma_2 \text{ なる } ij} \boldsymbol{v}^T g^*_{ij}. \tag{4.39}$$

すなわち

$$\boldsymbol{v}^T \sum_{k=0}^{m-1} A^*_{e_k} \boldsymbol{u} = \boldsymbol{v}^T (\sum_{k=0}^{m-1} \boldsymbol{f}^*_k + \sum_{\text{辺 } ij \subset \Gamma_2 \text{ なる } ij} g^*_{ij}), \tag{4.40}$$

$$A^* \stackrel{\text{def.}}{=} \sum_{i=0}^{m-1} A_{e_k}, \quad \boldsymbol{f}^* \stackrel{\text{def.}}{=} (\sum_{k=0}^{m-1} \boldsymbol{f}^*_k + \sum_{\text{辺 } ij \subset \Gamma_2 \text{ なる } ij} g^*_{ij}) \tag{4.41}$$

と置けば弱形式は

$$\boldsymbol{v}^T A^* \boldsymbol{u} = \boldsymbol{v}^T \boldsymbol{f}^* \tag{4.42}$$

となる。

v が $v = 0$ (on Γ_1) を満たす任意関数であることを考えると上式において節点番号が Γ_1 上にきている行を除いた

$$A^{**} \boldsymbol{u} = \boldsymbol{f}^{**} \tag{4.43}$$

が成立する。ただし、

$A^{**} \stackrel{\text{def.}}{=}$ A^*から Γ_1 のくる節点番号の行を除いた行列,

$\boldsymbol{f}^{**} \stackrel{\text{def.}}{=}$ $\boldsymbol{f}^*$ から Γ_1 のくる節点番号の成分を除いた縦ベクトル

とする。

また Γ_1 では $\hat{u} = g_1$(既知) であるから、これを代入して消去できる。A^{**} を列ベクトルで $A^{**} = (\boldsymbol{a}_0 \boldsymbol{a}_1 \ldots \ \boldsymbol{a}_{m-1})$ と表示すると (4.43) は

$$\sum_{k=0}^{m-1} \boldsymbol{a}_k u_k = \boldsymbol{f}^{**}. \tag{4.44}$$

左辺を Γ_1 に属す部分と属さない部分に分解して移行すると

$$\sum_{N_k \notin \Gamma_1 \text{なる } k} \boldsymbol{a}_k u_k = \boldsymbol{f}^{**} - \sum_{N_k \in \Gamma_1 \text{なる } k} \boldsymbol{a}_k u_k. \tag{4.45}$$

ただし、N_k $(0 \leq k \leq m-1)$ は節点番号である。

これより

$$A\boldsymbol{u}^* = \boldsymbol{f}. \tag{4.46}$$

ここで

$$\begin{aligned} A &= A^{**}\text{から第 } k \text{ 列 } (N_k \in \Gamma_1\text{なる } k) \text{ を除いた正方行列,} \\ \boldsymbol{u}^* &= \boldsymbol{u} \text{ から第 } k \text{ 成分 } (N_k \in \Gamma_1\text{なる } k) \text{ を除いた縦ベクトル,} \\ \boldsymbol{f} &= \boldsymbol{f}^{**} - \sum_{N_k \in \Gamma_1\text{なる } k} \boldsymbol{a}_k u_k. \end{aligned}$$

(4.46) が最終的な方程式である。

4.4 近似方程式の具体例２

この節で扱う問題は (4.1),(4.2),(4.3) で $\Omega = (0,1) \times (0,1), \Gamma_1$ は $x = 0$ または $y = 0$ で与えられる 2 辺、Γ_2 は $x = 1$ または $y = 1$ で与えられる 2 辺であり、$f = \bar{f}$(定数関数), $g_1 = 0,\ g_2 = 0$ の場合を扱う。つまり

$$-\Delta u = \bar{f} \quad (\ (x,y) \in (0,1) \times (0,1)\ ) \tag{4.47}$$

$$u(x,y) = 0 \quad (\ (x,y) \in \Gamma_1) \tag{4.48}$$

$$\frac{du}{d\boldsymbol{n}} = 0 \quad (\ (x,y) \in \Gamma_2) \tag{4.49}$$

を考える。以下に Ω の分割方法を描いた。

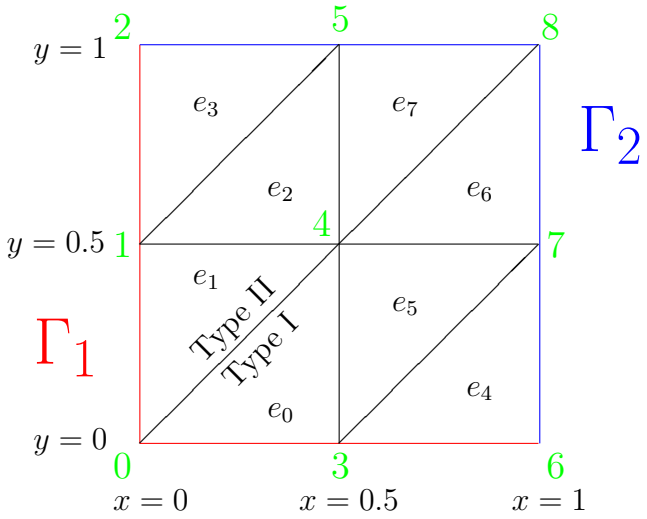

図 4.1: 分割の様子

図 4.1 中の緑色で書いた数字は全体節点番号であり、e_i $(0 \leq i \leq 7)$ は要素である。

図 4.1 にも書いてあるがこの分割には要素の 2 つのタイプ、Type I と Type II がある。Type I は右下が直角な直角二等辺 3 角形、Type II は左上が直角な直角二等辺 3 角形とすると Type I は要素 e_0, e_2, e_4, e_6、Type II は要素 e_1, e_3, e_5, e_7 が該当する。

タイプが同じであれば $A_{e_k}, \boldsymbol{f}_{e_k}$ は等しいことは要素の面積が等しく、b_i, c_i $(1 \leq i \leq 3)$ もそれぞれ等しいことからわかる。そこで Type I の要素にはそれぞれ $A_{\mathrm{I}}, \boldsymbol{f}_{\mathrm{I}}$、Type II の要素には $A_{\mathrm{II}}, \boldsymbol{f}_{\mathrm{II}}$ と表すことにする。$A_{\mathrm{I}}, \boldsymbol{f}_{\mathrm{I}}, A_{\mathrm{II}}, \boldsymbol{f}_{\mathrm{II}}$ を計算すると

$$A_{\mathrm{I}} = \frac{1}{2}\begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & -1 \end{pmatrix}, \quad \boldsymbol{f}_{\mathrm{I}} = \frac{\bar{f}}{24}\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \tag{4.50}$$

$$A_{\mathrm{II}} = \frac{1}{2}\begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & -1 & 2 \end{pmatrix}, \quad \boldsymbol{f}_{\mathrm{II}} = \frac{\bar{f}}{24}\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \tag{4.51}$$

これらから全体的な近似方程式を作る。そのためには局所的な節点番号 $1^*, 2^*, 3^*$ と全体的な節点番号の対応づけが必要となる。下の表を準備する。

要素	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7
タイプ	I	II	I	II	I	II	I	II
1^*	0	0	1	1	3	3	4	4
2^*	3	4	4	5	6	7	7	8
3^*	4	1	5	2	7	4	8	5

表 4.1: 要素節点対応表

(4.37),(4.38) をすべての要素について考えると弱形式は

$$\boldsymbol{v}^T \frac{1}{2} \begin{pmatrix} 2 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -2 & 0 & -1 & 0 & 0 \\ 0 & -2 & 0 & -2 & 8 & -2 & 0 & -2 & 0 \\ 0 & 0 & -1 & 0 & -2 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{pmatrix} = \boldsymbol{v}^T \frac{\bar{f}}{24} \begin{pmatrix} 2 \\ 3 \\ 1 \\ 3 \\ 6 \\ 3 \\ 1 \\ 3 \\ 2 \end{pmatrix}. \tag{4.52}$$

ただし

$$\forall \boldsymbol{v} \in \{(v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8)^T \in \boldsymbol{R}^9; v_0 = v_1 = v_2 = v_3 = v_6 = 0\}.$$

Dirichlet 境界条件の処理をすると

$$\begin{pmatrix} 8 & -2 & 0 & -2 & 0 \\ -2 & 4 & 0 & 0 & -1 \\ -2 & 0 & -1 & 4 & -1 \\ 0 & -1 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_4 \\ u_5 \\ u_7 \\ u_8 \end{pmatrix} = \frac{\bar{f}}{12} \begin{pmatrix} 6 \\ 3 \\ 3 \\ 2 \end{pmatrix}. \tag{4.53}$$

この式が最終的に解くべき方程式 $A\boldsymbol{u}^* = \boldsymbol{f}$ である。

4.5 数値実験

4.5.1 例題１

例題 1

Poisson 方程式の Dirichlet 境界条件を解く。

$$\Delta u = 1 \quad ((x, y) \in \Omega) \tag{4.54}$$

$$u(x, y) = 0 \quad ((x, y) \in \Gamma) \tag{4.55}$$

ただし、Ω は $x = 0, x = 1, y = 0, y = 1$ で囲まれた領域とし、Γ は Ω の境界である。

この問題に対するプログラムを以下に載せる。

```
/*--------------------------------------------------------------------------
 * fem2.c 2次元の Poisson 方程式を有限要素法で解くプログラム
 *  例 7.5
 * 扱った問題: 2次元 Poisson 方程式
 *             領域:Ω=(0,1) × (0,1), Ωの境界:Γ
 *             f:領域Ω内で与えられた関数
 *             g:Γ上で与えられた関数
 *             に対し
 *               -△u = f  (in Ω = (0,1) × (0,1))
 *                  u = g  (on Γ)
 *             を満たす u(x,y) を求めよ.
 *
 * 領域の分割は x 方向に X 等分、y 方向に Y 等分してできる小長方形を対角線で
 * 二分割してできる規則的な三角形要素による分割.
 * 等高線描画のためのファイル contourdata_fem2.dat を作成
 --------------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define X (30)              /* x 方向分割数 */
#define Y (30)              /* y 方向分割数 */
#define nnode (X+1)*(Y+1) /* 全節点数*/
#define nelmt 2*X*Y       /* 全要素数 */
#define nbc   2*(X+Y)     /* Γ上の節点の総数 */

typedef double **matrix;

void input(double *a, double *b, double *c, double *d,
           double *x, double *y,double *hx, double *hy,
           int first[], int second[], int third[], int *ibc);
void assem(int i, int first[], int second[], int third[],
           matrix A, double f[], double Ae[][3], double fe[]);
void ecm(int i, int first[], int second[], int third[],
         double hx, double hy, double x[], double y[],
         double Ae[][3], double fe[]);
void output(double f[]);
void node_correspondence(int *first, int *second, int *third);
void bound(double x[], double y[], int ibc[], matrix A, double f[nnode]);
void gauss(matrix A, double f[]);
double func(double x, double y);
double g(double x, double y);
void make_contour_data(int first[], int second[], int third[], int nband,
                       double x[], double y[], int ibc[], double f[]);
void make_gnudata(double x[], double y[], double f[]);

matrix new_matrix(int m, int n)
{
  int i;
```

```
  double *ip;
  matrix a;

  if((a = malloc(sizeof(double *) * m)) == NULL)
    return NULL;
  if((ip = malloc(sizeof(double) * (m * n))) == NULL){
    free(a);
    return NULL;
  }
  for(i=0; i<m; i++)
    a[i] = ip + (i * n);

  return a;
}

int main()
{
  int i, j, k;
  double a, b, c, d;          /* 領域Ω=(a,b) × (c,d) */
  double hx, hy;              /* hx:x 方向分割幅, hy:y 方向分割幅 */
  double *x, *y;              /* x, y 座標 */
  double Ae[3][3];            /* 要素係数行列 */
  double fe[3];               /* 要素自由項ベクトル */
  matrix A;                   /* 全体係数行列 */
  double *f;                  /* 全体自由項ベクトル */
  int *ibc;
  int *first, *second, *third;

  if ((x = malloc(sizeof(double) * nnode)) == NULL){
    fprintf(stderr, "x のメモリ確保に失敗\n");
    exit(1);
  }
  if((y = malloc(sizeof(double) * nnode)) == NULL){
    fprintf(stderr, "y のメモリ確保に失敗\n");
    exit(1);
  }
  if((A = new_matrix(nnode, nnode)) == NULL){
    fprintf(stderr, "A のメモリ確保に失敗\n");
    exit(1);
  }
  if((f = malloc(sizeof(double) * nnode)) == NULL){
    fprintf(stderr, "f のメモリ確保に失敗\n");
    exit(1);
  }
  if((ibc = malloc(sizeof(int) * nbc)) == NULL){
    fprintf(stderr, "ibc のメモリ確保に失敗\n");
    exit(1);
  }
  if((first = malloc(sizeof(int) * nelmt)) == NULL){
```

```
    fprintf(stderr, "first のメモリ確保に失敗\n");
    exit(1);
  }
  if((second = malloc(sizeof(int) * nelmt)) == NULL){
    fprintf(stderr, "second のメモリ確保に失敗\n");
    exit(1);
  }
  if((third = malloc(sizeof(int) * nelmt)) == NULL){
    fprintf(stderr, "third のメモリ確保に失敗\n");
    exit(1);
  }
  node_correspondence(first, second, third);
  input( &a, &b, &c, &d, x, y, &hx, &hy, first, second, third, ibc);
  for(j=0; j<nnode; j++){
      f[j] = 0.0;
      for(k=0; k<nnode; k++)
        A[j][k] = 0.0;
    }
  for(i=0; i<nelmt; i++){
    ecm( i, first, second, third, hx, hy, x, y, Ae, fe);
    assem( i, first, second, third, A, f, Ae, fe);
  }
  bound(x, y, ibc, A, f);
  gauss(A, f);
  make_contour_data(first, second, third, 0, x, y, ibc, f);
  make_gnudata(x, y, f);
  output( f);

  return 0;
}

void input(double *a, double *b, double *c, double *d,
           double *x, double *y, double *hx, double *hy,
           int *first, int *second, int *third, int *ibc)
{
  int i;
  int count;

  *a = 0.0; *b = 1.0; *c = 0.0; *d = 1.0;
  *hx = (*b-*a)/X; *hy = (*d-*c)/Y;
  for(i=0; i<nnode; i++){
    x[i] = 0.0;
    y[i] = 0.0;
  }
  for(i=0; i<nnode-1; i++){
    x[i+1] = x[i];
    y[i+1] = y[i] + *hy;
    if( (i+1) % (Y+1) == 0){
      x[i+1] = x[i] + *hx;
```

```
      y[i+1] = 0.0;
    }
  }
  count = 0;
  for(i=0; i<nnode; i++){
    if (fabs(x[i]) < 1e-6 || fabs(x[i] - 1) < 1e-6 ||
        fabs(y[i]) < 1e-6 || fabs(y[i] - 1) < 1e-6) {
      ibc[count] = i;
      count++;
    }
  }
  printf("nnode:%d  nelmt:%d  nbc:%d\n", nnode, nelmt, nbc);
  printf("\nhx = %5g  hy = %5g\n", *hx, *hy);
  printf("\n[  i]       x[i]          y[i]\n");
  for(i=0; i<nnode; i++)
    printf("[%3d] %10g    %10g\n", i, x[i], y[i]);
  printf("\n[  i]  first  second   third\n");
  for(i=0; i<nelmt; i++)
    printf("[%3d] %6d  %6d  %6d\n",i,  first[i], second[i], third[i]);
  printf("\n");

}

void assem(int i, int first[], int second[], int third[],
           matrix A, double *f, double Ae[][3], double fe[])
{
  A[first[i]][first[i]]   += Ae[0][0];
  A[first[i]][second[i]]  += Ae[0][1];
  A[first[i]][third[i]]   += Ae[0][2];
  A[second[i]][first[i]]  += Ae[1][0];
  A[second[i]][second[i]] += Ae[1][1];
  A[second[i]][third[i]]  += Ae[1][2];
  A[third[i]][first[i]]   += Ae[2][0];
  A[third[i]][second[i]]  += Ae[2][1];
  A[third[i]][third[i]]   += Ae[2][2];

  f[first[i]] += fe[0];
  f[second[i]] += fe[1];
  f[third[i]] += fe[2];

}
/* 要素係数行列と要素自由項ベクトルの計算 */
void ecm(int i, int first[], int second[], int third[],
         double hx, double hy, double x[], double y[],
         double Ae[][3], double fe[])
{
  int j, k;
  double bb[3], cc[3], D, S;
```

```
  bb[0] = (y[second[i]] - y[third[i]] )  / (hx*hy);
  bb[1] = (y[third[i]]  - y[first[i]] )  / (hx*hy);
  bb[2] = (y[first[i]]  - y[second[i]] ) / (hx*hy);
  cc[0] = (x[third[i]]  - x[second[i]] ) / (hx*hy);
  cc[1] = (x[first[i]]  - x[third[i]] )  / (hx*hy);
  cc[2] = (x[second[i]] - x[first[i]] )  / (hx*hy);
  D = x[first[i]]*(y[second[i]]-y[third[i]])
    + x[second[i]]*(y[third[i]]-y[first[i]])
    + x[third[i]]*(y[first[i]]-y[second[i]]);
  S = fabs(D)/2.0;
  for(j=0; j<3; j++)
    for(k=0; k<3; k++)
      Ae[j][k] = hx*hy*(bb[j]*bb[k] + cc[j]*cc[k])/2.0;
  fe[0] = func( x[first[i]],y[first[i]])*S/3.0;
  fe[1] = func( x[second[i]],y[second[i]])*S/3.0;
  fe[2] = func( x[third[i]],y[third[i]])*S/3.0;
}

void output(double f[])
{
  int i;

  printf("\n 有限要素解 \n");
  for(i=0; i<nnode; i++){
    printf("u[%3d]=%10g;  ", i, f[i]);
    if(i%3 == 2)
      printf("\n");
  }

  printf("\n");

}
/* 要素節点対応表 */
void node_correspondence(int *first, int *second, int *third)
{
  int j;

  first[0] = 0; second[0] = Y+1; third[0] = Y+2;
  first[1] = 0; second[1] = Y+2; third[1] = 1;

  for(j=2; j<nelmt; j++){
    first[j]= 0; second[j] = 0; third[j] = 0;
  }
  for(j=0; j<nelmt-2; j+=2){
    if((j+2)%(2*Y)){
      first[j+2] = first[j] + 1;
      second[j+2] = second[j] + 1;
      third[j+2] = third[j] + 1;
    }
```

```
    else {
      first[j+2] = first[j] + 2;
      second[j+2] = second[j] + 2;
      third[j+2] = third[j] + 2;
    }
  }
  for(j=1; j<nelmt-2; j+=2){
    if((j+1)%(2*Y)){
      first[j+2] = first[j] + 1;
      second[j+2] = second[j] + 1;
      third[j+2] = third[j] + 1;
    }
    else {
      first[j+2] = first[j] + 2;
      second[j+2] = second[j] + 2;
      third[j+2] = third[j] + 2;
    }
  }

}
/* 境界条件を考慮する */
void bound(double x[], double y[], int ibc[], matrix A, double f[])
{
  int i, j;

  for(i=0; i<nbc; i++){
    printf("ibc[%3d]= %4d ", i, ibc[i]);
    if((i%4)==3)
      printf("\n");
  }
  for(i=0; i<nbc; i++){
    for(j=0; j<nnode; j++){
      A[ibc[i]][j] = 0.0;
      A[j][ibc[i]] = 0.0;
      f[j] -= g( x[ibc[i]],y[ibc[i]])*A[j][ibc[i]]; /* 移項 */
    }
    A[ibc[i]][ibc[i]] = 1.0;
    f[ibc[i]] = g(x[ibc[i]],y[ibc[i]]);
  }

}
/* 対称な行列に対するガウスの消去法 */
void gauss(matrix A, double f[])
{
  int i, j, k;
  double q;
  /* 前進消去（対称行列版） */
  for(i=0; i<nnode-1; i++)
    for(j=i+1; j<nnode; j++){
```

```
      q = A[i][j] / A[i][i];
      for(k=j; k<nnode; k++)
        A[j][k] -= q*A[i][k];
      f[j] -= q*f[i];
    }
  /* 後退代入 */
  f[nnode-1] /= A[nnode-1][nnode-1];
  for(i=nnode-2; i>=0; i--){
    for(j=i+1; j<nnode; j++)
      f[i] -= A[i][j]*f[j];
    f[i] /= A[i][i];
  }

}

double func(double x, double y)  /* Ω内の関数 -△u(x,y) = func(x,y) */
{
  return 1.0;
}

double g(double x, double y)     /* Γ上の関数 u(x,y) = g(x,y) */
{
  return 0.0;
}

void make_contour_data(int first[], int second[], int third[], int nband,
                           double x[], double y[],int ibc[], double f[])
{
  int i;
  FILE *fp;

  fp = fopen("contourdata_fem2.dat", "w");
  fprintf(fp, "%d %d %d %d\n", nnode, nelmt, nbc, nband);
  for(i=0; i<nnode; i++){
    fprintf(fp, "%f %f  ", x[i], y[i]);
    if(i%4==3)
      fprintf(fp, "\n");
  }
  fprintf(fp, "\n");
  for(i=0; i<nelmt; i++){
    fprintf(fp, "%d %d %d ", first[i], second[i], third[i]);
    if((i+1)%5==0)
      fprintf(fp, "\n");
  }
  fprintf(fp, "\n");
  for(i=0; i<nbc; i++){
    fprintf(fp, "%d ", ibc[i]);
    if((i+1)%10 == 0)
      fprintf(fp, "\n");
```

```
  }
  fprintf(fp, "\n");
  for(i=0; i<nnode; i++){
    fprintf(fp, "%f ", f[i]);
    if((i+1)%5 == 0)
      fprintf(fp, "\n");
  }

  fclose(fp);

}

void make_gnudata(double x[], double y[], double f[])
{
  int i;
  FILE *fp;

  fp = fopen("gnu_fem2.dat","w");
  for(i=0; i<nnode; i++)
    fprintf(fp, "%f %f %f\n", x[i], y[i], f[i]);
  fclose(fp);

}
```

１次元問題のプログラムとは各関数の役割がいろいろ違っているので関数の変更点等を説明をする。

- 関数 node_corrspondence()

  全体節点番号と局所節点番号の対応処理。局所節点番号 $1^*, 2^*, 3^*$ をそれぞれ first[],second[],third[] として記憶する。

- 関数 input()[2]

  １次元問題のプログラムでは Dirichlet 境界条件と Neumann 境界条件の値を得ていたが、今度は Dirichlet 境界条件が適応される節点の全体節点番号を ibc[] として記憶する。

- 関数 assem(),ecm()

  １次元問題のプログラムでは assem() だけで処理していたが、今度は ecm() で $A_e, \boldsymbol{f}_e$ を作成し、assem() で $A^*, \boldsymbol{f}^*$ を作成する。

- 関数 g()

  Dirichlet 境界条件 $u(x) = g(x)$ の $g(x)$ に相当。

[2] この関数は四角形領域にしか適応できないため汎用性が低い。

- 関数 make_contour_data()

  桂田先生作の等高線描画プログラム contour.c[3](GLSC を用いている) に対するデータファイル contourdata_fem2.dat を作成する。

- 関数 make_gnudata()

  gnuplot を用いてグラフを書くためのデータファイル gnu_fem2.dat を作成する

(X=2,Y=2 の場合の実行結果)

```
nnode:9  nelmt:8  nbc:8

hx =   0.5  hy =   0.5

[  i]       x[i]       y[i]
[  0]          0          0
[  1]          0        0.5
[  2]          0          1
[  3]        0.5          0
[  4]        0.5        0.5
[  5]        0.5          1
[  6]          1          0
[  7]          1        0.5
[  8]          1          1

[  i]  first  second  third
[  0]      0       3      4
[  1]      0       4      1
[  2]      1       4      5
[  3]      1       5      2
[  4]      3       6      7
[  5]      3       7      4
[  6]      4       7      8
[  7]      4       8      5

ibc[  0]=    0 ibc[  1]=    1 ibc[  2]=    2 ibc[  3]=    3
ibc[  4]=    5 ibc[  5]=    6 ibc[  6]=    7 ibc[  7]=    8

 有限要素解
u[  0]=         0;  u[  1]=         0;  u[  2]=         0;
u[  3]=         0;  u[  4]=     0.625;  u[  5]=         0;
u[  6]=         0;  u[  7]=         0;  u[  8]=         0;
```

分割を細かくした場合の等高線は次のようになる。

[3]10 本の等高線を引くプログラム。

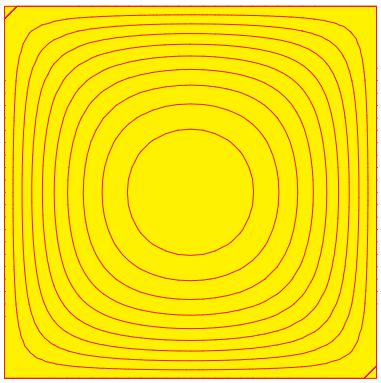

図 4.2: 中心 (0.5,0.5) が一番高い等高線

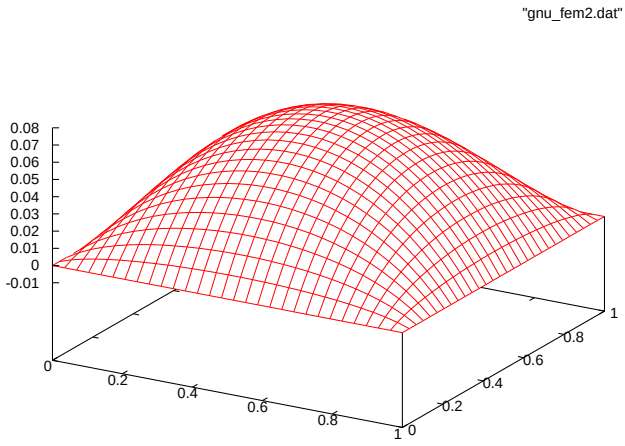

図 4.3: 全体の様子

座標 (0.5,0.5) を頂点とするなだらかで山型が得られた。
この問題より少しだけ一般的な問題

$$-\Delta u = f \quad (\text{in } \Omega) \tag{4.56}$$

$$u = 0 \quad (\text{on } \Gamma) \tag{4.57}$$

に対する厳密解は m, n を正の整数として

$$u(x,y) = \sum_{m,n=1}^{\infty} a_{mn} \sin(m\pi x) \sin(n\pi y), \tag{4.58}$$

$$a_{mn} = \frac{4}{(m^2+n^2)\pi^2} \int_0^1 \int_0^1 f(x,y) \sin(m\pi x) \sin(n\pi y)\ dxdy. \tag{4.59}$$

で与えられる (杉島 [4] 参照)。この例題では $f(x,y) \equiv 1$ であるから

$$a_{mn} = \frac{4}{(m^2+n^2)\pi^2} \int_0^1 \int_0^1 \sin(m\pi x) \sin(n\pi y)\ dxdy. \tag{4.60}$$

$(x,y) = (0.5, 0.5)$ の場合の誤差を調べる。厳密な値は $u(0.5, 0.5) = 0.07367$ である。表 4.2 に有限要素解の誤差の推移を書いておいた。

X=Y	$\hat{u}(0.5, 0.5)$	誤差 (%)	X=Y	$\hat{u}(0.5, 0.5)$	誤差 (%)
2	0.0625	15	4	0.0703125	4.5
8	0.0727826	1.2	16	0.0734458	0.30
32	0.0736147	0.075	厳密解	0.07367	

表 4.2: 誤差の推移

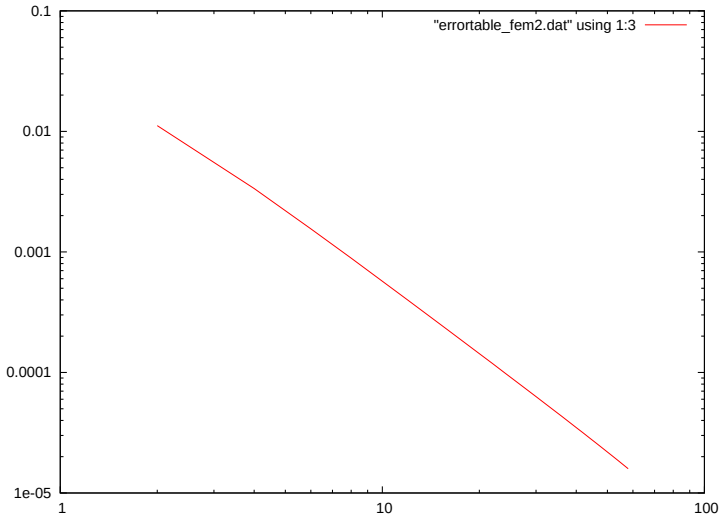

図 4.4: 収束の様子

１次元問題では有限要素解と厳密解の節点における誤差がまったくなかったのに対し、２次元問題ではそうはいかない。

4.5.2 例題２

例題２

Poisson 方程式の Dirichlet-Neumann 境界条件問題を解く。

$$\Delta u = 1 \quad ((x,y) \in \Omega) \tag{4.61}$$

$$u(x,y) = 0 \quad ((x,y) \in \Gamma_1) \tag{4.62}$$

$$\frac{\partial u}{\partial \boldsymbol{n}}(x,y) = 0 \quad ((x,y) \in \Gamma_2) \tag{4.63}$$

ただし、Ω は $x=0, x=1, y=0, y=1$ で囲まれた領域とし、Γ_1 は Ω の境界のうち $x=0, y=0$ の部分、Γ_2 は Ω の境界のうち $x=1, y=1$ の部分である。

この問題は前節において考えた Poisson 方程式の Dicichlet-Neumann 境界条件問題について分割をもっと細かくして数値的に解くというものである。

この問題に対するプログラムを以下に載せる。

```
/*--------------------------------------------------------------------------
 * fem3.c ２次元の Poisson 方程式を有限要素法で解くプログラム
 * 境界は Dirichlet 条件--Γ_1;  Neumann 条件--Γ_2;
 * 扱った問題: ２次元 Poisson 方程式
 *             領域:Ω, Ωの境界:Γ=Γ_1∪Γ_2(Γ_2∩Γ_2=φ)
 *             f:領域Ω内で与えられた関数
 *             g_1:Γ_2 上で与えられた関数
 *             g_2:Γ_2 上で与えられた関数
 *             に対し
 *                  -Δ u = f  (in Ω)
 *                     u = g_1  (on Γ_1)
 *             (∂ u)/(∂ n) = g_2  (on Γ_2)
 *             を満たす u(x,y) を求めよ.
 *
 * 入力ファイル input1.dat を受け取り、等高線を描くために contourdata_fem3.dat
 * を生成
 --------------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
//#define SWITCH1
#define SWITCH2   /* SWITCH1 と SWITCH2 はどちらか片方定義する */

typedef int **imatrix;
typedef double **matrix;

void input0(FILE *fp, int *nnode, int *nelmt, int *Dnbc, int *Nnbc);
void input(FILE *fp, int nnode, int nelmt, int Dnbc, int Nnbc, imatrix ielmt,
           int *Dibc, int *Nibc, double *x, double *y);
void assem(int nnode, int nelmt, imatrix ielmt, int Dnbc, int Dibc[],
           int *nband, double x[], double y[],
           matrix A, double f[], matrix Ae, double fe[]);
void ecm(int ie, imatrix ielmt, double x[], double y[],
         double *fe, matrix Ae);
void band(int m, int nband, double f[], matrix A);
void output(int nnode, double f[]);
double func(double x, double y);
double g_1(double x, double y);
double g_2(double x, double y);
void make_contour_data(FILE *fp, int nnode, int nelmt, int Dnbc, int nband,
                  double x[], double y[], imatrix ielmt, int Dibc[],
                  double f[]);
void make_gnudata(FILE *fp, int nnode, double x[], double y[], double f[]);

imatrix new_imatrix(int m, int n)
{
  int i;
  int *ip;
  imatrix a;

  if ((a = malloc(sizeof(int *) * m)) == NULL)
    return NULL;
  if ((ip = malloc(sizeof(int) * (m * n))) == NULL) {
    free(a);
    return NULL;
  }
  for (i = 0; i < m; i++)
    a[i] = ip + (i * n);

  return a;
}

matrix new_matrix(int m, int n)
{
  int i;
  double *ip;
  matrix a;
```

```
  if((a = malloc(sizeof(double *) * m)) == NULL)
    return NULL;
  if((ip = malloc(sizeof(double) * (m * n))) == NULL){
    free(a);
    return NULL;
  }
  for(i=0; i<m; i++)
    a[i] = ip + (i * n);

  return a;
}

int main()
{
#ifdef SWITCH
  char filename[30];
#endif
  int nnode, nelmt, Dnbc, Nnbc, nband;
  int *Dibc, *Nibc;
  imatrix ielmt;
  matrix A, Ae;
  double *f, *fe;
  double *x, *y;
  FILE *fp;

#ifdef SWITCH
  printf("ファイル名を入力してください : ");
  gets(filename);
  if( (fp = fopen(filename,"r")) == NULL){
    printf("%s を開くことができません", filename);
    printf("\n");
    exit(1);
  }
#endif
#ifdef SWITCH2
  fp = fopen("input1.dat", "r");
#endif
  input0(fp, &nnode, &nelmt, &Dnbc, &Nnbc);

  if ((x = malloc(sizeof(double)*nnode)) == NULL){
    fprintf(stderr, "x のメモリ確保に失敗\n");
    exit(1);
  }
  if(( y = malloc(sizeof(double) * nnode)) == NULL){
    fprintf(stderr, "y のメモリ確保に失敗\n");
    exit(1);
  }
  if((ielmt = new_imatrix(nelmt, 3)) == NULL){
    fprintf(stderr, "ielmt のメモリ確保に失敗\n");
```

```
    exit(1);
  }
  if((Dibc = malloc(sizeof(int) * Dnbc)) == NULL){
    fprintf(stderr, "Dibc のメモリ確保に失敗\n");
    exit(1);
  }
  if((Nibc = malloc(sizeof(int) * Nnbc)) == NULL){
    fprintf(stderr, "Nibc のメモリ確保に失敗\n");
    exit(1);
  }
  if((A = new_matrix(nnode, nnode)) == NULL){
    fprintf(fp, "A のメモリ確保に失敗\n");
    exit(1);
  }
  if((Ae = new_matrix(3, 3)) == NULL){
    fprintf(fp, "Ae のメモリ確保に失敗\n");
    exit(1);
  }
  if((f = malloc(sizeof(double) * nnode)) == NULL){
    fprintf(stderr, "f のメモリ確保に失敗\n");
    exit(1);
  }
  if((fe = malloc(sizeof(double) * 3)) == NULL){
    fprintf(stderr, "fe のメモリ確保に失敗\n");
    exit(1);
  }

  input(fp, nnode, nelmt, Dnbc, Nnbc, ielmt, Dibc, Nibc, x, y);
  assem(nnode, nelmt, ielmt, Dnbc, Dibc, &nband, x,  y, A, f, Ae, fe);
  band(nnode, nband, f, A);
  make_contour_data(fp, nnode, nelmt, Dnbc, 0, x, y, ielmt, Dibc, f);
  make_gnudata(fp, nnode, x, y, f);
  output(nnode, f);

  return 0;
}

void input0(FILE *fp, int *nnode, int *nelmt, int *Dnbc, int *Nnbc)
{
  fscanf(fp, "%d %d %d %d", nnode, nelmt, Dnbc, Nnbc);
}

void input(FILE *fp, int nnode, int nelmt, int Dnbc, int Nnbc, imatrix ielmt,
           int *Dibc, int *Nibc, double *x, double *y)
{
  int i, j;

  for(i=0; i<nnode; i++)
    fscanf(fp, "%lf %lf",&(x[i]), &(y[i]));
```

```
  for(i=0; i<nelmt; i++)
    for(j=0; j<3; j++)
      fscanf(fp, "%d", &(ielmt[i][j]));
  for(i=0; i<Dnbc; i++)
    fscanf(fp, "%d", &(Dibc[i]));
  for(i=0; i<Nnbc; i++)
    fscanf(fp, "%d", &(Nibc[i]));
  /* 入力データの出力 */
  printf("\nnnode:%d  nelmt:%d  Dnbc:%d  Nnbc:%d\n\n",
         nnode, nelmt, Dnbc, Nnbc);
  printf("座標 (x,y)\n");
  for(i=0; i<nnode; i++){
    printf("x[%3d]:%f , y[%3d]:%f   ", i, x[i], i, y[i]);
    if(i%2 == 1)
      printf("\n");
  }
  printf("\n\nielmt\n");
  for(i=0; i<nelmt; i++) {
    printf("[%3d]  ", i);
    for(j=0; j<3; j++)
      printf("%4d   ", ielmt[i][j]);
    if( i % 2 == 1)
      printf("\n");
  }
  printf("\nDirichlet 境界条件節点番号 Dnbc=%d\n", Dnbc);
  for(i=0; i<Dnbc; i++)
    printf("%d ", Dibc[i]);
  printf("\nNeumann 境界条件節点番号 Nnbc=%d\n", Nnbc);
  for(i=0; i<Nnbc; i++)
    printf("%d ", Nibc[i]);
  printf("\n\n");
#ifdef SWITCH2
  fclose(fp);
#endif

}

void assem(int nnode, int nelmt, imatrix ielmt, int Dnbc, int Dibc[],
           int *nband, double x[], double y[],
           matrix A, double f[], matrix Ae, double fe[])
{
  int i, j, k, ii, jj, kk, count;

  for(i=0; i<nnode; i++){
    f[i] = 0.0;
    for(j=0; j<nnode; j++)
      A[i][j] = 0.0;
  }
  for(i=0; i<nelmt; i++){
```

```
    ecm(i, ielmt, x, y, fe, Ae);
    for(j=0; j<3; j++){
      jj = ielmt[i][j];
      f[jj] += fe[j];
      for(k=0; k<3; k++){
        kk = ielmt[i][k];
        A[jj][kk] += Ae[j][k];
      }
    }
  }

  count = nnode;
  for(i=nnode-1; i>=0; i--){
    if( A[0][i] != 0)
      break;
    count--;
  }
  *nband = count;
  printf("nband = %d\n", *nband);
  /* Dirichlet 境界条件の考慮 Gauss の消去法半バンド版*/
  for(i=0; i<Dnbc; i++){
    ii = Dibc[i];
    f[ii] = g_1( x[ii], y[ii]);
    for(j=0; j<nnode; j++){
      A[ii][j] = A[j][ii] = 0.0;
      f[j] -= g_1( x[ii], y[ii]) * A[j][ii];
    }
    A[ii][ii] = 1.0;
  }
  for(i=1; i<nnode; i++){
    count = 0;
    for(j=i; count < (*nband)+1; j++){
      A[i][j-i] = A[i][j];
      count++;
      }
  }

}
 /* ecm 要素番号を受け取って要素の Ae,fe を作成 */
void ecm(int ie, imatrix ielmt, double x[], double y[],
         double *fe, matrix Ae)
{
  int i, j, k;
  double xe[3], ye[3], D, b[3], c[3], S;

  for(i=0; i<3; i++){
    j = ielmt[ie][i];
    xe[i] = x[j];
    ye[i] = y[j];
```

```
  }
  D = xe[0]*(ye[1]-ye[2]) + xe[1]*(ye[2]-ye[0]) + xe[2]*(ye[0]-ye[1]);
  S = fabs(D) / 2.0;
  for(i=0; i<3; i++){
    j = i + 1;
    k = i + 2;
    if( j >= 3) /* (i, j, k):(0,1,2) の偶置換 */
      j -= 3;
    if( k >= 3)
      k -= 3;
    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]);
  for(i=0; i<3; i++)                /* func の要素内の代表値 */
    fe[i] = S * func(xe[i], ye[i]) / 3.0;

}
/* Gauss の消去法半バンド版  */
void band(int m, int nband, double f[], matrix A)
{
  int i, j, k, mm;
  double q;
  /* m : number of unknowns , nband : half_band_width */
  /* 前進消去 */
  for(i=0; i<m-1; i++){
    mm = i + nband; /* mm = min(i+nband, nnode) */
    if(mm > m)
      mm = m;
    for(j=i+1; j<mm; j++){
      q = A[i][j-i] / A[i][0];
      f[j] -= q*f[i];
      for(k=j; k<mm; k++)
        A[j][k-j] -= q*A[i][k-i];
    }
  }
  /* 後退代入 */
  f[m-1] /= A[m-1][0];
  for(i=m-2; i>=0; i--){
    mm = i + nband;
    if(mm > m)
      mm = m;
    for(j=i+1; j<mm; j++)
      f[i] -= A[i][j-i]*f[j];
    f[i] /= A[i][0];
  }
```

```
}

void output(int nnode, double f[])
{
  int i;
  /* 有限要素解の出力 */
  printf("\n 有限要素解 \n");
  for(i=0; i<nnode; i++){
    printf("u[%3d] = %10f   ", i, f[i]);
    if(i%3 == 2)
      printf("\n");
  }

  printf("\n");

}

double func(double x, double y) /* Poisson 方程式 -Δ u = func */
{
  return 1.0;
}

double g_1(double x, double y) /* Dirichlet 境界条件 */
{
  return 0.0;
}

double g_2(double x, double y) /* Neumann 境界条件 */
{
  return 0.0;
}

void make_contour_data(FILE *fp, int nnode, int nelmt, int Dnbc, int nband,
                   double x[], double y[], imatrix ielmt, int Dibc[],
                   double f[])
{
  int i, j;

  fp = fopen("contourdata_fem3.dat", "w");
  fprintf(fp, "%d %d %d %d ", nnode, nelmt, Dnbc, nband);
  for(i=0; i<nnode; i++){
    fprintf(fp, "%f %f ", x[i], y[i]);
    if(i%3 == 2)
      fprintf(fp, "\n");
  }
  for(i=0; i<nelmt; i++){
    for(j=0; j<3; j++)
      fprintf(fp, "%d ", ielmt[i][j]);
    if(i%5 == 4)
```

```
      fprintf(fp, "\n");
  }
  for(i=0; i<Dnbc; i++){
    fprintf(fp, "%d ", Dibc[i]);
    if( i%5 == 4)
      fprintf(fp, "\n");
  }
  fprintf(fp, "\n");
  for(i=0; i<nnode; i++){
    fprintf(fp, "%f ", f[i]);
    if(i%5 == 4)
      fprintf(fp, "\n");
  }

  fclose(fp);

}

void make_gnudata(FILE *fp, int nnode, double x[], double y[], double f[])
{
  int i;

  fp = fopen("gnu_fem3.dat","w");
  for(i=0; i<nnode; i++)
    fprintf(fp, "%f %f %f\n", x[i], y[i], f[i]);
  fclose(fp);
}
```

例題１のプログラムと違う点の説明をする。

- 関数 input0()

  input1.dat から節点数、要素数、Dirichlet 境界条件が適応される節点番号の個数、Neumann 境界条件が適応される節点番号の個数を読み込む。

- 関数 input()

  input1.dat から各節点の座標、局所節点番号の対応する全体節点番号、Dirichlet 境界条件の適用される節点番号、Neumann 境界条件の適用される節点番号を読み込む。

- 関数 assem()

  $\boldsymbol{f}, A$ を作成に加え対称なバンド行列に対する Gauss の消去法半バンド版のための半バンド幅を見積もり、A を左寄せする (菊地 [1] 参照)。

- 関数 band()[4]

  対称なバンド行列に対する Gauss の消去法。

- 関数 make_contour_data()

  等高線描画のためにデータファイル contourdata_fem3.dat を作成。

- 関数 make_gnudata()

  gnuplot を用いてグラフを書くためのデータファイル gnu_fem3.dat を作成する

例題１のプログラムと比べると入力ファイルを使うことによりだいぶ汎用性のあるプログラムになった。

このプログラム中に出てきた input1.dat について説明する。

input1.dat は以下のプログラム input1.c によって作成された。

```
/*****************************************************************************
 * input1.c
 *
 * 長方領域を直角二等辺三角形で自動分割するプログラム
 *
 * 領域Ω=(a,b) × (c,d)
 * Dirichlet 境界条件: x=a || y=c
 * Neumann 境界条件  : x=b || y=d
 *****************************************************************************/

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

int main()
{
  int i, j;
  int nx, ny;
  int first, second, third;
  double a, b, c, d;
  double hx, hy;
  FILE *fp;

  a = 0.0; b = 1.0; c = 0.0; d = 1.0;
  /* nx:x 方向の分割数, ny:y 方向の分割数 */
  printf("x 方向分割数 と y 方向分割数を入力してください: ");
  scanf("%d%d", &nx, &ny);
  fp = fopen("input1.dat", "w");
```

[4]プログラムを書いていた当初、半バンド幅が正しく見積もれていなかったことに気づかなかったため、大変苦労した。

```
  hx = ( fabs(a)+fabs(b)) / nx;
  hy = ( fabs(c)+fabs(d)) / ny;
  /* 総節点数,総要素数,Dirichlet 境界条件の節点数,Neumann 境界条件の節点数 */
  fprintf(fp, "%d %d %d %d\n", (nx+1)*(ny+1), 2*nx*ny, nx+ny+1, nx+ny+1);
  /* 節点番号の座標 */
  for(i=0; i<nx+1; i++)
    for(j=0; j<ny+1; j++)
      fprintf(fp, "%f %f\n", i*hx, j*hy);
  /* 局所節点番号 */
  for(i=0; i<nx; i++)
    for(j=0; j<ny; j++){
      first  = j + (ny+1)*i;
      second = first + ny + 1;
      third  = second + 1;
      fprintf(fp, "%d %d %d %d %d %d\n",
             first, second, third, first, third, first+1);
    }
  /* Dirichlet 境界条件の節点番号 */
  for(i=0; i<(nx+1)*(ny+1); i++)
    if( i<=ny || i%(ny+1) == 0)
      fprintf(fp, "%d ", i);
  fprintf(fp,"\n");
  /* Neumann 境界条件の節点番号 */
  for(i=0; i<(nx+1)*(ny+1); i++)
    if( (i-ny) % (ny+1) == 0 || i >= nx*(ny+1))
      fprintf(fp, "%d ", i);
  fprintf(fp,"\n");
  fclose(fp);

  return 0;
}
```

局所節点番号 $1^*, 2^*, 3^*$（プログラム中では first,second,third）について補足しておく。

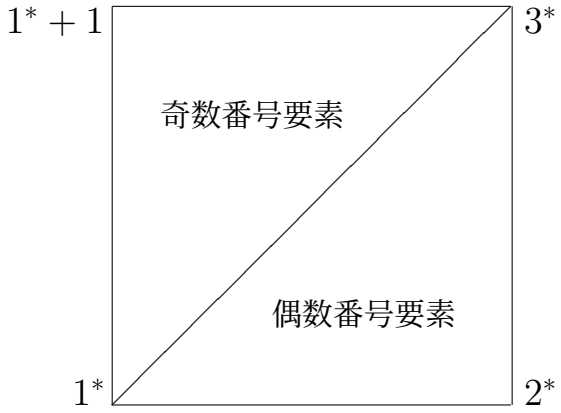

図 4.5: 局所節点番号

この分割の場合、図 4.5 からわかるよう奇数番号要素の局所節点番号に対する全体節点番号は偶数番号要素の局所節点番号を使って構成できる。つまり、奇数番号要素の全体節点番号は偶数番号要素の局所節点番号 $1^*, 2^*, 3^*$ を使って奇数番号要素の局所節点番号 $1^*, 2^*, 3^*$ の全体節点番号が $1^*, 3^*, 1^* + 1$ として書ける。

input1.dat は input1.c の実行において x 方向の分割数=2,y 方向の分割数=2 とした場合以下のデータファイルになる。

9 8 5 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

2 5 6 7 8

分割を細かくした場合の等高線は次のようになる。

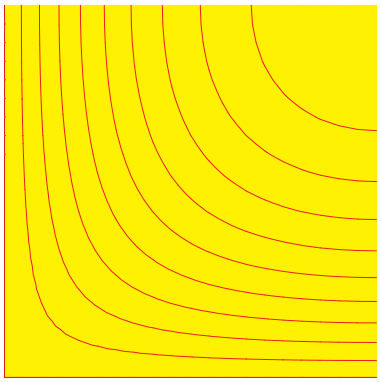

図 4.6: 左、下方向が一番低く、右上が一番高い等高線

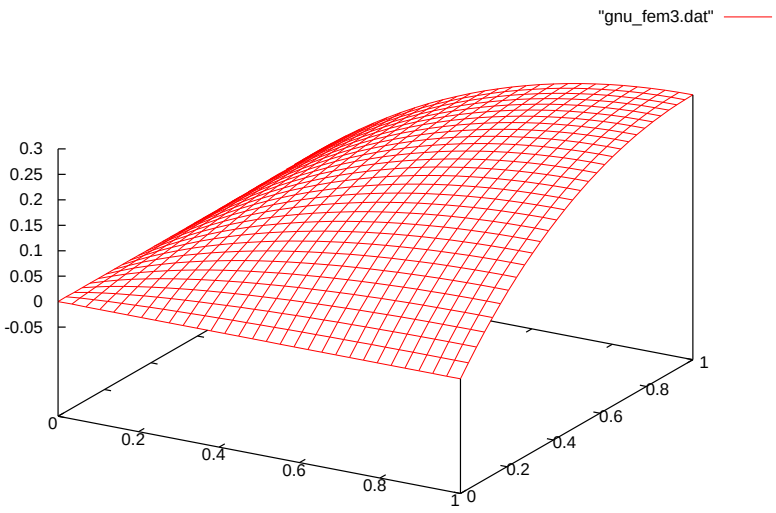

図 4.7: 全体の様子

Γ_1 から離れている所ほど高くなっている。

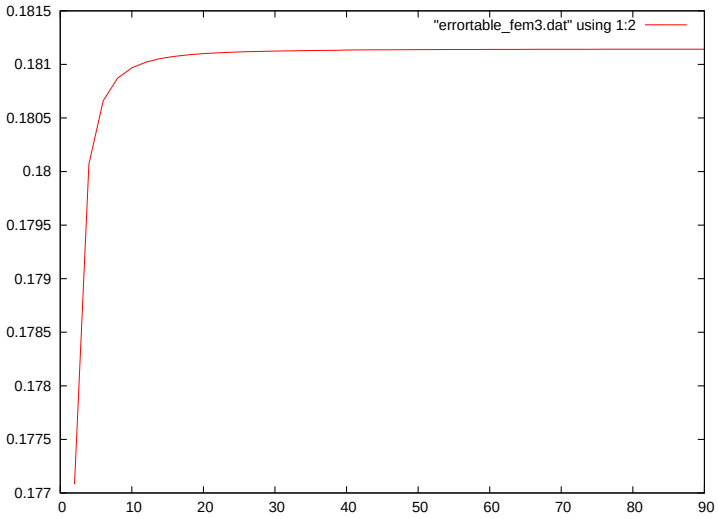

図 4.8: (0.5,0.5) における有限要素解の収束の様子

第5章　最後に

「はじめに」で書いた『卒業研究ではなるべく自分で考えてプログラムを作成し、プログラミングを通して数値解析を勉強することを心がけた。』ということを目標として掲げたが、実際にはその為に他人のプログラムをあまり参考にしないでプログラムを作り、結果、はじめに作ったプログラムは汎用性の低いプログラムになってしまった。有限要素法のプログラムは有限要素法自体が本来ブラックボックスとしても使えるようなプログラムであるべきである（当時は知らなかった）。他人のプログラムを読むことも勉強であると思い知らされた。有限要素法らしいプログラムは最後の fem3.c でようやく実現できたが、まだ完成していない。非同時 Neumann 境界条件問題に対しては対応できていない。アイディアはあるのだけれどもタイムアップなのでこの問題については大学院へ行く機会を得られたので今後の課題とした。

また、数値解析の勉強自体簡単にいかないものだと痛感した。自分の勘違い、指摘されないとわからなかったような間違いは山ほどあった。プログラムについてはまだまだ、自分一人では解決できないと痛感した。そういった事を適宜指導してもらった桂田先生には大変お世話になりました。

関連図書

[1] 菊地文雄, 有限要素法概説 [新訂版], サイエンス社.

[2] ブレジス, 関数解析, 産業図書.

[3] 陳小君・山本哲郎 共著, 英語で学ぶ数値解析, コロナ社.

[4] 2000 年度桂田研卒業生卒業論文 杉島武, Poisson 方程式に対する差分法.