応用複素関数 第 12 回

〜 ポテンシャル問題 (3) 〜

かつらだ　まさし
桂田 祐史

2023 年 7 月 11 日 (本来 7 月 4 日にするはずの授業)

目次

本日の内容・連絡事項

- FreeFem++ のインストールは、適切に処理すれば出来るようなので、レポート課題 3.2 は出さない (レポート課題 3.1 だけとする)。FreeFem++ のインストールがうまく行かない人は遠慮せず、相談して下さい。
- 本日の目標: レポート課題 3 に必要なことを説明し切ること。
  - Dirichlet の原理 — ポテンシャル問題が、弱定式化できたり、変分問題に変換できる有名な事実の紹介
    この後、より一般の Poisson 方程式の境界値問題に対する弱形式を導出する (少し先の §4.7.2 の一部を説明する)。
  - 有限要素法の概略 (お話)
  - サンプル・プログラム `potential2d-v0.edp` を (もう一度) 見る。
  - FreeFem++ の文法を説明する。

4.5 Dirichlet の原理

Laplace 方程式の Dirichlet 境界値問題

$$\text{(1a)} \qquad \triangle u = 0 \quad (\text{in } \Omega),$$

$$\text{(1b)} \qquad u = g \quad (\text{on } \partial\Omega)$$

の解 u の存在を示すため、Riemann は次のように考えた。

4.5 Dirichlet の原理

Laplace 方程式の Dirichlet 境界値問題

$$\text{(1a)} \qquad \triangle u = 0 \quad (\text{in } \Omega),$$

$$\text{(1b)} \qquad u = g \quad (\text{on } \partial\Omega)$$

の解 u の存在を示すため、Riemann は次のように考えた。

境界条件 (1b) を満たす関数の全体 X と、X 上の汎関数 J を考える。

$$X := \left\{ u \;\middle|\; u\colon \overline{\Omega} \to \mathbb{R},\ (\forall x \in \partial\Omega)\ u(x) = g(x) \right\}.$$

$$J[u] := \iint_{\Omega} \left(u_x^2 + u_y^2 \right) dx\, dy \quad (u \in X).$$

Dirichlet の原理

J の最小値を与える u は $\triangle u = 0$ (in Ω) を満たす。

4.5 Dirichlet の原理

Laplace 方程式の Dirichlet 境界値問題

$$\text{(1a)} \qquad \triangle u = 0 \quad (\text{in } \Omega),$$

$$\text{(1b)} \qquad u = g \quad (\text{on } \partial\Omega)$$

の解 u の存在を示すため、Riemann は次のように考えた。

境界条件 (1b) を満たす関数の全体 X と、X 上の汎関数 J を考える。

$$X := \left\{ u \,\middle|\, u\colon \overline{\Omega} \to \mathbb{R},\ (\forall x \in \partial\Omega)\ u(x) = g(x) \right\}.$$

$$J[u] := \iint_{\Omega} \left(u_x^2 + u_y^2 \right) dx\, dy \quad (u \in X).$$

Dirichlet の原理

J の最小値を与える u は $\triangle u = 0$ (in Ω) を満たす。

したがって J の最小値を与える u は (1a), (1b) の解である。

4.5 Dirichlet の原理

Laplace 方程式の Dirichlet 境界値問題

$$\text{(1a)} \qquad \triangle u = 0 \quad (\text{in } \Omega),$$

$$\text{(1b)} \qquad u = g \quad (\text{on } \partial\Omega)$$

の解 u の存在を示すため、Riemann は次のように考えた。

境界条件 (1b) を満たす関数の全体 X と、X 上の汎関数 J を考える。

$$X := \left\{ u \;\middle|\; u\colon \overline{\Omega} \to \mathbb{R},\ (\forall x \in \partial\Omega)\ u(x) = g(x) \right\}.$$

$$J[u] := \iint_{\Omega} \left(u_x^2 + u_y^2 \right) dx\, dy \quad (u \in X).$$

Dirichlet の原理

J の最小値を与える u は $\triangle u = 0$ (in Ω) を満たす。

したがって J の最小値を与える u は (1a), (1b) の解である。

Riemann (1826–1866) は、Dirichlet (1805–1859) 先生の講義の中で Dirichlet の原理を聴いたそうである。

4.5 Dirichlet の原理

証明

$v\colon \overline{\Omega} \to \mathbb{R}$ は、$v = 0$ (on $\partial\Omega$) を満たす任意の関数とする。任意の $t \in \mathbb{R}$ に対して $u + tv \in X$ である。仮定より

$$f(t) := J[u + tv] \quad (t \in \mathbb{R})$$

は $t = 0$ で最小値をとる。

4.5 Dirichlet の原理

証明

$v\colon \overline{\Omega} \to \mathbb{R}$ は、$v = 0$ (on $\partial\Omega$) を満たす任意の関数とする。任意の $t \in \mathbb{R}$ に対して $u + tv \in X$ である。仮定より

$$f(t) := J[u + tv] \quad (t \in \mathbb{R})$$

は $t = 0$ で最小値をとる。ところが

$$f(t) = J[u] + 2t \iint_\Omega (u_x v_x + u_y v_y) dx\, dy + t^2 \iint_\Omega \left(v_x^2 + v_y^2\right) dx\, dy$$

は t の 2 次関数であり、$t = 0$ で最小となるので、1 次の係数は 0 である:

$$\iint_\Omega (u_x v_x + u_y v_y)\, dx\, dy = 0. \tag{2}$$

4.5 Dirichlet の原理

証明

$v\colon \overline{\Omega} \to \mathbb{R}$ は、$v = 0$ (on $\partial\Omega$) を満たす任意の関数とする。任意の $t \in \mathbb{R}$ に対して $u + tv \in X$ である。仮定より

$$f(t) := J[u + tv] \quad (t \in \mathbb{R})$$

は $t = 0$ で最小値をとる。ところが

$$f(t) = J[u] + 2t \iint_\Omega (u_x v_x + u_y v_y) dx\, dy + t^2 \iint_\Omega \left(v_x^2 + v_y^2\right) dx\, dy$$

は t の 2 次関数であり、$t = 0$ で最小となるので、1 次の係数は 0 である:

$$\iint_\Omega (u_x v_x + u_y v_y)\, dx\, dy = 0. \tag{2}$$

Green の公式 ($\iint_\Omega \triangle u v\, dx\, dy = \int_{\partial\Omega} \frac{\partial u}{\partial n} v\, d\sigma - \iint_\Omega \nabla u \cdot \nabla v\, dx\, dy$) より

$$\iint_\Omega \triangle u\, v\, dx\, dy = 0.$$

4.5 Dirichlet の原理

証明

$v\colon \overline{\Omega} \to \mathbb{R}$ は、$v = 0$ (on $\partial\Omega$) を満たす任意の関数とする。任意の $t \in \mathbb{R}$ に対して $u + tv \in X$ である。仮定より

$$f(t) := J[u + tv] \quad (t \in \mathbb{R})$$

は $t = 0$ で最小値をとる。ところが

$$f(t) = J[u] + 2t \iint_\Omega (u_x v_x + u_y v_y) dx\, dy + t^2 \iint_\Omega \left(v_x^2 + v_y^2\right) dx\, dy$$

は t の 2 次関数であり、$t = 0$ で最小となるので、1 次の係数は 0 である:

$$\iint_\Omega (u_x v_x + u_y v_y)\, dx\, dy = 0. \tag{2}$$

Green の公式 ($\iint_\Omega \triangle u v\, dx\, dy = \int_{\partial\Omega} \frac{\partial u}{\partial n} v\, d\sigma - \iint_\Omega \nabla u \cdot \nabla v\, dx\, dy$) より

$$\iint_\Omega \triangle u\, v\, dx\, dy = 0.$$

これが任意の v について成り立つことから (**変分法の基本補題により**)

$$\triangle u = 0 \quad (\text{in } \Omega). \quad \square$$

4.5 Dirichlet の原理

反省

Riemann は、汎関数 $J[u]$ を最小にする $u \in X$ の存在は明らかだと考えた。

J は下に有界 $(J[u] \geq 0)$ であるから、J は下限を持つ。それは最小値のはず…

4.5 Dirichlet の原理

反省

Riemann は、汎関数 $J[u]$ を最小にする $u \in X$ の存在は明らかだと考えた。

J は下に有界 $(J[u] \geq 0)$ であるから、J は下限を持つ。それは最小値のはず…

それに Weierstrass が疑義を呈した (「下限は本当に最小値？」とツッコミを入れた)。これに Riemann は存命中に答えられなかった。

4.5 Dirichlet の原理

反省

Riemann は、汎関数 $J[u]$ を最小にする $u \in X$ の存在は明らかだと考えた。

J は下に有界 $(J[u] \geq 0)$ であるから、J は下限を持つ。それは最小値のはず…

それに Weierstrass が疑義を呈した (「下限は本当に最小値？」とツッコミを入れた)。これに Riemann は存命中に答えられなかった。

現代的な解説をすると、関数空間は無限次元空間なので、有界閉集合上の連続関数であっても、最小値を持たないことがありえる。

4.5 Dirichlet の原理

反省

Riemann は、汎関数 $J[u]$ を最小にする $u \in X$ の存在は明らかだと考えた。

J は下に有界 $(J[u] \geq 0)$ であるから、J は下限を持つ。それは最小値のはず…

それに Weierstrass が疑義を呈した (「下限は本当に最小値？」とツッコミを入れた)。これに Riemann は存命中に答えられなかった。

現代的な解説をすると、関数空間は無限次元空間なので、有界閉集合上の連続関数であっても、最小値を持たないことがありえる。

ポテンシャル問題は重要なため、解の存在について、多くの人が努力して Dirichlet 原理を用いない証明がいくつか発見されたが、Riemann の発表から約 50 年後 (1900 年頃)、D. Hilbert が Dirichlet 原理に基づく証明を発表し、肯定的に解決した。

4.5 Dirichlet の原理

反省

Riemann は、汎関数 $J[u]$ を最小にする $u \in X$ の存在は明らかだと考えた。

J は下に有界 $(J[u] \geq 0)$ であるから、J は下限を持つ。それは最小値のはず…

それに Weierstrass が疑義を呈した (「下限は本当に最小値？」とツッコミを入れた)。これに Riemann は存命中に答えられなかった。

現代的な解説をすると、関数空間は無限次元空間なので、有界閉集合上の連続関数であっても、最小値を持たないことがありえる。

ポテンシャル問題は重要なため、解の存在について、多くの人が努力して Dirichlet 原理を用いない証明がいくつか発見されたが、Riemann の発表から約 50 年後 (1900 年頃)、D. Hilbert が Dirichlet 原理に基づく証明を発表し、肯定的に解決した。

今では解の存在証明は、このルートをたどるのがスタンダードになっている。…でも応用複素関数としては、ここから数値計算法に舵を切る (存在証明については、関数解析か偏微分方程式論で学んでください)。

4.6 ポテンシャル問題の数値解法 (1) 有限要素法

ポテンシャル問題を数値的に解くことを考えよう。この「応用複素関数」では、**有限要素法**と**基本解の方法**を簡単に紹介する。

4.6 ポテンシャル問題の数値解法 (1) 有限要素法

ポテンシャル問題を数値的に解くことを考えよう。この「応用複素関数」では、**有限要素法**と**基本解の方法**を簡単に紹介する。

差分法で解くこともできるが、長方形領域でない問題を解くには工夫が必要になり、あまり便利でない。

4.6 ポテンシャル問題の数値解法 (1) 有限要素法

ポテンシャル問題を数値的に解くことを考えよう。この「応用複素関数」では、**有限要素法**と**基本解の方法**を簡単に紹介する。

差分法で解くこともできるが、長方形領域でない問題を解くには工夫が必要になり、あまり便利でない。

有限要素法の主たるアイディアは次の 2 つ:

1. **弱形式**を用いる。
2. 領域を三角形、四面体などの**有限要素**に分割し、近似解や試験関数に**区分的多項式**を採用する。

4.6 ポテンシャル問題の数値解法 (1) 有限要素法

ポテンシャル問題を数値的に解くことを考えよう。この「応用複素関数」では、**有限要素法**と**基本解の方法**を簡単に紹介する。

差分法で解くこともできるが、長方形領域でない問題を解くには工夫が必要になり、あまり便利でない。

有限要素法の主たるアイディアは次の 2 つ:

1. **弱形式**を用いる。
2. 領域を三角形、四面体などの**有限要素**に分割し、近似解や試験関数に**区分的多項式**を採用する。

この講義では有限要素法の詳細は解説できないが、幸い **FreeFem++** というソフトを用いると、弱形式さえ分かれば、有限要素についてはソフトに任せにして、数値計算ができる。

4.6 ポテンシャル問題の数値解法 (1) 有限要素法

ポテンシャル問題を数値的に解くことを考えよう。この「応用複素関数」では、**有限要素法**と**基本解の方法**を簡単に紹介する。

差分法で解くこともできるが、長方形領域でない問題を解くには工夫が必要になり、あまり便利でない。

有限要素法の主たるアイディアは次の 2 つ:

1. **弱形式**を用いる。
2. 領域を三角形、四面体などの**有限要素**に分割し、近似解や試験関数に**区分的多項式**を採用する。

この講義では有限要素法の詳細は解説できないが、幸い **FreeFem++** というソフトを用いると、弱形式さえ分かれば、有限要素についてはソフトに任せにして、数値計算ができる。

実は Dirichlet 原理の証明中に現れた (10) は Laplace 方程式の Dirichlet 境界値問題の弱形式である。(弱形式については、次回解説を行う。)

今回は「百聞は一見にしかず」で、 まずはプログラム (スライド 1 枚) を紹介する。

2,3 行書き換えるだけで「自分の問題」が解ける。

```
// potential2d-v0.edp ---  2 次元非圧縮ポテンシャル流
//  速度ポテンシャル，速度を求め、等ポテンシャル線，速度場を描く
border Gamma(t=0,2*pi) { x = cos(t); y = sin(t); } // 円盤領域
int m=40;
mesh Th=buildmesh(Gamma(m));
plot(Th, wait=1, ps="Th.eps");
// 次の 2 行は区分 1 次多項式を使うという意味
fespace Vh(Th,P1);
Vh phi, v, v1, v2;
// 境界条件の設定
func Vn=x+2*y; // Ωが単位円で，V=(1,2) のとき V・n=x+2y

// 速度ポテンシャルφを求め、その等高線（等ポテンシャル線）を描く
solve Laplace(phi,v) =
  int2d(Th)(dx(phi)*dx(v)+dy(phi)*dy(v)) -int1d(Th,Gamma)(Vn*v);
plot(phi,ps="contourpotential.eps",wait=1);
// ベクトル場 (v1,v2)=∇φ を描く （ちょっと雑なやり方）
v1=dx(phi); v2=dy(phi);
plot([v1,v2],ps="vectorfield.eps",wait=1);

// 等ポテンシャル線とベクトル場を同時に描く
plot([v1,v2],phi,ps="both.eps", wait=1);
```

サンプル・プログラムを読む

前回の授業で紹介した potential2d-v0.edp をまだ入手していない人は、次のようにダウンロードして、自分の使い慣れたテキスト・エディターで開いてみよう (簡単な解説はレポート課題 3に書いてある)。

```
curl -O https://m-katsurada.sakura.ne.jp/complex2/potential2d-v0.edp
```

実行はターミナルで次のように行う。

```
FreeFem++ potential2d-v0.edp
```

enter キー (↵) で次のグラフィックスに進み、p で一つ前のグラフィックに戻る。esc で終了する。

5 FreeFem++の文法

5.1 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 [1])。

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

5 FreeFem++の文法

5.1 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 [1])。
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)。

5 FreeFem++の文法

5.1 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 [1])。
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。

5 FreeFem++の文法

5.1 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 [1])。
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 ([2] の付録に書いておいた)。
参考: FreeFem++ は C++ で記述されている。

5 FreeFem++の文法

5.1 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 [1])。
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 ([2] 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 Hect[3] 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 [2] 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 [4], [5])、テキスト大塚・高石 [6] (本学学生は Maruzen eBook で読める) などがある。

5.1 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 -△ 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。

改めて数えるととても多い。

改めて数えるととても多い。

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

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 ;

改めて数えるととても多い。

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

改めて数えるととても多い。

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

改めて数えるととても多い。

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

改めて数えるととても多い。

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

改めて数えるととても多い。

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

改めて数えるととても多い。

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

改めて数えるととても多い。

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

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

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…

FreeFem++ のソース・プログラムは C++ 言語で書かれているため、「C 言語に似ている」は本当は「C++言語に似ている」かもしれない。

5.2.2 C++言語と似ているところ: cin, cout を用いた入出力

cin, cout を使ったストリーム・入出力は、C++ を知っている人には簡単であろう。C++については、付録**??**(準備中　) を適宜参考にすること。

C で printf() で出力したものは、cout << で置き換えられる。

```
cout << "Hello world" << endl; // printf("Hello, world\n");
cout << "n=" << n << ", m=" << m << endl; // printf("n=%d, m=%d\n",n,m);
cout << "x=" << x << endl; // printf("x=%f\n", x);
```

書式指定については後述する。

C で scanf() で入力したものは、cin >> 変数名 で置き換えられる。

```
cin >> a >> b; // scanf("%d %d", &a, &b);
cin >> x;      // scanf("%lf", &x);
```

外部ファイルとの入出力については、https://m-katsurada.sakura.ne.jp/labo/text/freefem-note/node31.html#section:_____ を見よ。

5.2.3 データ型

- 整数を表すための int がある (C 言語の `int` に相当)

5.2.3 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)

5.2.3 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)`

5.2.3 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` と言う値があるが、それぞれ 1, 0 の別名と考えて良い。
  例えば `plot(u,wait=true);` は `plot(u,wait=1);` と同じ。

5.2.3 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` と言う値があるが、それぞれ 1, 0 の別名と考えて良い。
  例えば `plot(u,wait=true);` は `plot(u,wait=1);` と同じ。
- 文字列を表すための string がある (C++言語の `string` に相当, 日本語不可？)。

5.2.3 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` と言う値があるが、それぞれ 1, 0 の別名と考えて良い。
  例えば `plot(u,wait=true);` は `plot(u,wait=1);` と同じ。
- 文字列を表すための string がある (C++言語の `string` に相当, 日本語不可？)。
  - 2 つの string `s1`, `s2` を、(+ 演算子を用いて) `s1+s2` で連結できる。

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

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

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

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

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

5.2.4 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。

5.2.4 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;
```

5.2.5 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、以下デフォールト (C 言語の%g) に戻る。

5.2.5 FreeFem++の real データの入出力の書式指定　例

```
// testfloat.edp
real NA = 6.022e+23;
// デフォールト %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;
```

5.3 有限要素法のための機能

5.3.1 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。

5.3.2 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。

5.3.2 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。

5.3.2 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` と言う型の変数として定義される。

5.3.2 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` と言う型の変数として定義される。
- 三角形分割は `mesh` と言う型の変数として定義される。

5.3.2 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` と言う型の変数として定義される。
- 三角形分割は `mesh` と言う型の変数として定義される。
- `buildmesh()` と言う関数は、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。

5.3.2 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` と言う型の変数として定義される。
- 三角形分割は `mesh` と言う型の変数として定義される。
- `buildmesh()` と言う関数は、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()` と言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)。

5.3.2 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` と言う型の変数として定義される。
- 三角形分割は `mesh` と言う型の変数として定義される。
- `buildmesh()` と言う関数は、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()` と言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。

5.3.2 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` と言う型の変数として定義される。
- 三角形分割は `mesh` と言う型の変数として定義される。
- `buildmesh()` と言う関数は、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()` と言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()` と言う命令で mesh 型データが作れる (参考「FreeFem++ノート」)。

5.3.2 領域の定義と領域の三角形分割　border

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");
```

5.3.2 領域の定義と領域の三角形分割　メッシュ (mesh)

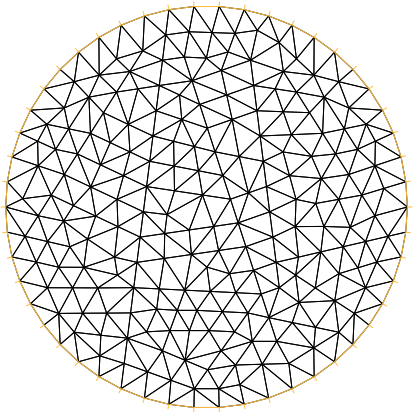


図 1: C(50)

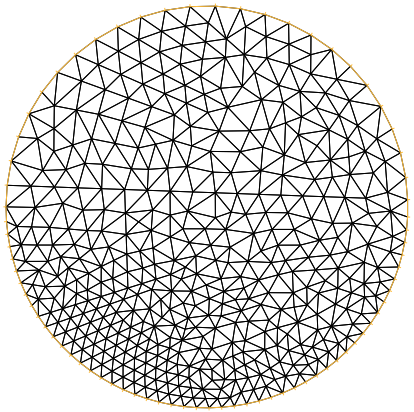


図 2: Gamma1(25)+Gamma2(50)

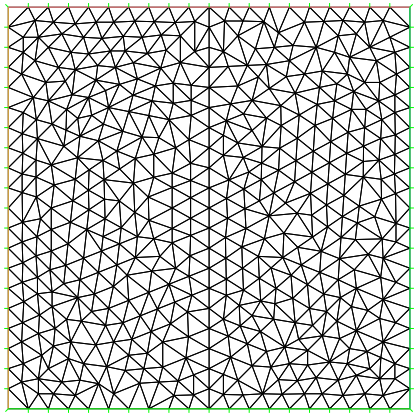


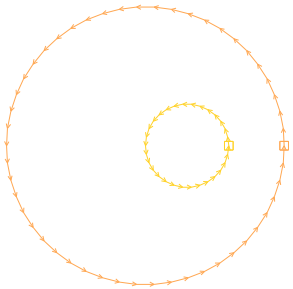
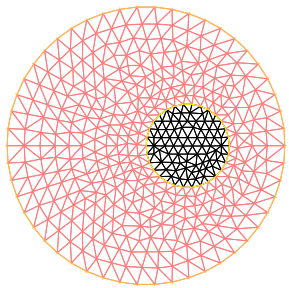
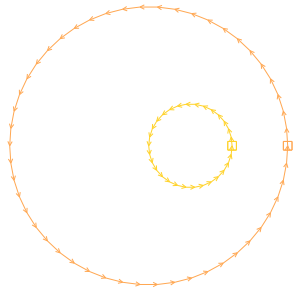
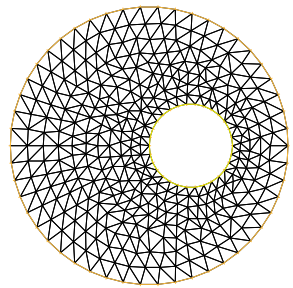
図 3: C1(20)+C2(20)+...

5.3.2 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");
```

5.3.2 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…

5.3.2 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.nt` は三角形の数 (the number of triangles)、`Th.nv` は節点の数 (the number of vertices)、`Th.area` は領域の面積 (area) である。

5.3.2 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.nt` は三角形の数 (the number of triangles)、`Th.nv` は節点の数 (the number of vertices)、`Th.area` は領域の面積 (area) である。
- `Th`(i) は i 番目の節点 ($i = 0, 1, \ldots,$ `Th.nv` $- 1$) で、その座標は `Th`(i)`.x` と `Th`(i)`.y` である。`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。

5.3.2 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.nt` は三角形の数 (the number of triangles)、`Th.nv` は節点の数 (the number of vertices)、`Th.area` は領域の面積 (area) である。
- `Th`(i) は i 番目の節点 ($i = 0, 1, \ldots,$ `Th.nv` $- 1$) で、その座標は `Th`(i)`.x` と `Th`(i)`.y` である。`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。
- `Th[`i`]` は i 番目の三角形 ($i = 0, 1, \ldots,$ `Th.nt` $- 1$)、`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 `Th[`i`][`j`].x` と `Th[`i`][`j`].y` である。三角形の面積は `Th[`i`].area` である。

5.3.2 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.nt` は三角形の数 (the number of triangles)、`Th.nv` は節点の数 (the number of vertices)、`Th.area` は領域の面積 (area) である。
- `Th`(i) は i 番目の節点 ($i = 0, 1, \ldots,$ `Th.nv` $- 1$) で、その座標は `Th`(i)`.x` と `Th`(i)`.y` である。`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。
- `Th[`i`]` は i 番目の三角形 ($i = 0, 1, \ldots,$ `Th.nt` $- 1$)、`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 `Th[`i`][`j`].x` と `Th[`i`][`j`].y` である。三角形の面積は `Th[`i`].area` である。
- 点 (x, y) を含む三角形の番号は `Th(`x`,`y`).nuTriangle` で得られる。

5.3.2 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。

5.3.2 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・データー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)。

5.3.2 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・データー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・データーを FreeFem++ で利用する。

5.3.2 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・データー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・データーを 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。

5.3.3 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 $\widetilde{X}$ 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 と言う 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。)

5.3.3 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。

5.3.3 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。

(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。

(c) `varf`, `matrix` — 弱形式を与えて連立 1 次方程式を作る。

5.3.3 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。

(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。

(c) `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。

$$-\triangle u(x,y) = f(x,y) \quad ((x,y) \in \Omega) \tag{3a}$$

$$u(x,y) = g_1(x,y) \quad ((x,y) \in \Gamma_1) \tag{3b}$$

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

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。

$$\langle u, v \rangle = (f, v) + [g_2, v] \quad (v \in X). \tag{4}$$

ただし

$$X_{g_1} := \{w \mid w = g_1 \quad \text{on } \Gamma_1\}, \quad X := \{v \mid v = 0 \quad \text{on } \Gamma_1\}. \tag{5}$$

5.3.3 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);
```

5.3.3 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。

5.3.3 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。

5.3.3 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。

5.3.3 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。

5.3.3 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。

5.3.3 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)。

5.3.3 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 $A\boldsymbol{u} = \boldsymbol{f}$ の $A, \boldsymbol{f}$ を別々に計算して、$A^{-1}\boldsymbol{f}$ を計算することで $\boldsymbol{u}$ 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)。

5.3.3 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 $A\boldsymbol{u} = \boldsymbol{f}$ の $A, \boldsymbol{f}$ を別々に計算して、$A^{-1}\boldsymbol{f}$ を計算することで $\boldsymbol{u}$ 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 $= 10^{30}$)。

5.3.3 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 $Au = f$ の A, f を別々に計算して、$A^{-1}f$ を計算することで 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。

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

以上で 5 節を終わり、4 節「ポテンシャル問題」に戻ります。

(2023/7/11 日の授業では、弱形式 (10) の導出だけ行いました。その他も説明したいところですが、今年度はあきらめます。)

4.7 弱解の方法　4.7.1 はじめに

今回用いる有限要素法は、今では微分方程式論で常識となっている**弱解の方法** (weak formulation) を応用したものである。

4.7 弱解の方法　4.7.1 はじめに

今回用いる有限要素法は、今では微分方程式論で常識となっている**弱解の方法** (weak formulation) を応用したものである。

これは、Riemann による、Laplace 方程式の Dirichlet 境界値問題を解くために用いた論法を発展させたものである。現在では様々な微分方程式に応用されているが、ここではもっとも基本的な Poisson 方程式の境界値問題について説明する (おおむね菊地 [7] に沿った解説)。

今回用いる有限要素法は、今では微分方程式論で常識となっている**弱解の方法** (weak formulation) を応用したものである。

これは、Riemann による、Laplace 方程式の Dirichlet 境界値問題を解くために用いた論法を発展させたものである。現在では様々な微分方程式に応用されているが、ここではもっとも基本的な Poisson 方程式の境界値問題について説明する (おおむね菊地 [7] に沿った解説)。

厳密に議論するには、**広義導関数**、いわゆる **Sobolev (ソボレフ) 空間**を導入する必要がある。具体的には、後で出て来る X_{g_1} と X は、本当は Sobolev 空間の一種 $H^1(\Omega)$ を用いて

$$X_{g_1} := \left\{ w \in H^1(\Omega) \,\middle|\, w = g_1 \text{ on } \Gamma_1 \right\}, \quad X := \left\{ w \in H^1(\Omega) \,\middle|\, w = 0 \text{ on } \Gamma_1 \right\}$$

と定義すべきものである。ともあれ、ここでは関数の滑らかさに関する議論には目をつぶって議論する。

(Sobolev 空間を学ぶときは、Brezis [8], [9] をチェックしてみよう。)

4.7.2 Poisson 方程式の境界値問題 (P)

Laplace 方程式を一般化した Poisson 方程式の境界値問題を考える。

Ω は $\mathbb{R}^m$ $(m = 2, 3)$ の有界領域、$\Gamma := \partial\Omega$ は Ω の境界、Γ_1 と Γ_2 は

$$\Gamma_1 \cup \Gamma_2 = \Gamma, \quad \Gamma_1 \cap \Gamma_2 = \emptyset$$

を満たす。また、$f : \Omega \to \mathbb{R}$, $g_1 : \Gamma_1 \to \mathbb{R}$, $g_2 : \Gamma_2 \to \mathbb{R}$ が与えられているとする。$\Gamma_1 = \emptyset$ (全周 Neumann) のときは $\displaystyle\int_{\Gamma_2} g_2 \, d\sigma = 0$ を仮定する。

問題 (P)

Find u s.t.

$$-\triangle u = f \quad (\text{in } \Omega), \tag{6}$$

$$u = g_1 \quad (\text{on } \Gamma_1), \tag{7}$$

$$\frac{\partial u}{\partial n} = g_2 \quad (\text{on } \Gamma_2). \tag{8}$$

4.7.3 弱定式化 (W) と変分問題 (V)

4.7.3 弱定式化 (W) と変分問題 (V)

$$X_{g_1} := \left\{ w \;\middle|\; w \colon \overline{\Omega} \to \mathbb{R},\ w = g_1 \text{ on } \Gamma_1 \right\}, \quad X := \left\{ w \;\middle|\; w \colon \overline{\Omega} \to \mathbb{R},\ w = 0 \text{ on } \Gamma_1 \right\},$$

$$J[w] := \frac{1}{2} \int_\Omega |\nabla w|^2 \, dx - \int_\Omega f\, w \, dx - \int_{\Gamma_2} g_2\, w \, d\sigma \quad (w \in X_{g_1}). \tag{9}$$

とおく。X の要素はしばしば**試験関数** (test function) と呼ばれる。

次の 2 つの問題を考える。

4.7.3 弱定式化 (W) と変分問題 (V)

$$X_{g_1} := \left\{ w \;\middle|\; w : \overline{\Omega} \to \mathbb{R},\ w = g_1 \text{ on } \Gamma_1 \right\}, \quad X := \left\{ w \;\middle|\; w : \overline{\Omega} \to \mathbb{R},\ w = 0 \text{ on } \Gamma_1 \right\},$$

$$J[w] := \frac{1}{2} \int_\Omega |\nabla w|^2 \, dx - \int_\Omega f\, w \, dx - \int_{\Gamma_2} g_2\, w \, d\sigma \quad (w \in X_{g_1}). \tag{9}$$

とおく。X の要素はしばしば**試験関数** (test function) と呼ばれる。

次の 2 つの問題を考える。

(W)

Find $u \in X_{g_1}$ s.t.

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f\, v \, dx + \int_{\Gamma_2} g_2\, v \, d\sigma \quad (v \in X). \tag{10}$$

(条件 (10) を**弱形式** (weak form) と呼ぶ。)

4.7.3 弱定式化 (W) と変分問題 (V)

$$X_{g_1} := \left\{ w \;\middle|\; w \colon \overline{\Omega} \to \mathbb{R},\ w = g_1 \text{ on } \Gamma_1 \right\}, \quad X := \left\{ w \;\middle|\; w \colon \overline{\Omega} \to \mathbb{R},\ w = 0 \text{ on } \Gamma_1 \right\},$$

$$J[w] := \frac{1}{2} \int_\Omega |\nabla w|^2 \, dx - \int_\Omega f\, w \, dx - \int_{\Gamma_2} g_2\, w \, d\sigma \quad (w \in X_{g_1}). \tag{9}$$

とおく。X の要素はしばしば**試験関数** (test function) と呼ばれる。
次の 2 つの問題を考える。

(W)

Find $u \in X_{g_1}$ s.t.

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f\, v \, dx + \int_{\Gamma_2} g_2\, v \, d\sigma \quad (v \in X). \tag{10}$$

(条件 (10) を**弱形式** (weak form) と呼ぶ。)

(V)

Find $u \in X_{g_1}$ s.t.

$$J[u] = \inf_{w \in X_{g_1}} J[w]. \qquad (\text{inf は結局は min}) \tag{11}$$

4.7.4 3つの問題 (P), (W), (V) の同等性

(P), (W), (V) は、ほぼ同値な問題である。

4.7.4 3つの問題 (P), (W), (V) の同等性

(P), (W), (V) は、ほぼ同値な問題である。実際、次が成り立つ。

定理 12.1

1. u が (P) の解 $\Rightarrow$ u が (W) の解
2. u が (W) の解 $\Leftrightarrow$ u が (V) の解
3. u が (W) の解かつ u が C^2 級 $\Rightarrow$ u が (P) の解

(授業では (2) の証明を省略した。そのため、次の補題も省略した。)

4.7.4 3つの問題 (P), (W), (V) の同等性

(P), (W), (V) は、ほぼ同値な問題である。実際、次が成り立つ。

定理 12.1

1. u が (P) の解 $\Rightarrow$ u が (W) の解
2. u が (W) の解 $\Leftrightarrow$ u が (V) の解
3. u が (W) の解かつ u が C^2 級 $\Rightarrow$ u が (P) の解

(授業では (2) の証明を省略した。そのため、次の補題も省略した。)
(2) の証明のために補題を準備する (証明は単なる計算であるので省略する)。

補題 12.2

任意の $w \in X_{g_1}$, $v \in X$, $t \in \mathbb{R}$ に対して次式が成立する。

$$J[w+tv] = \frac{t^2}{2}\int_\Omega |\nabla v|^2\, d\boldsymbol{x} + t\left(\int_\Omega \nabla w\cdot\nabla v\, d\boldsymbol{x} - \int_\Omega f\, v\, d\boldsymbol{x} - \int_{\Gamma_2} g_2\, v\, d\sigma\right) + J[w]. \tag{12}$$

4.7.5 (P) の解は (W) の解 — 弱形式の導出

(1) の証明

u が (P) の解と仮定する。(6) $-\triangle u = f$ に任意の $v \in X$ をかけて Ω 上で積分すると

$$-\int_\Omega \triangle u\, v \, dx = \int_\Omega fv \, dx. \tag{13}$$

4.7.5 (P) の解は (W) の解 — 弱形式の導出

(1) の証明

u が (P) の解と仮定する。(6) $-\triangle u = f$ に任意の $v \in X$ をかけて Ω 上で積分すると

$$-\int_\Omega \triangle u\, v\, dx = \int_\Omega fv\, dx. \tag{13}$$

左辺を Green の公式 ([10]) を用いて変形してから、$v = 0$ (on Γ_1) と (8) $\frac{\partial u}{\partial n} = g_2$ を用いると

$$\begin{aligned}
-\int_\Omega \triangle u\, v\, dx &= -\int_\Gamma \frac{\partial u}{\partial n} v\, d\sigma + \int_\Omega \nabla u \cdot \nabla v\, dx \\
&= -\int_{\Gamma_1} \frac{\partial u}{\partial n} v\, d\sigma - \int_{\Gamma_2} \frac{\partial u}{\partial n} v\, d\sigma + \int_\Omega \nabla u \cdot \nabla v\, dx \\
&= -\int_{\Gamma_2} g_2\, v\, d\sigma + \int_\Omega \nabla u \cdot \nabla v\, dx.
\end{aligned}$$

4.7.5 (P) の解は (W) の解 — 弱形式の導出

(1) の証明

u が (P) の解と仮定する。(6) $-\triangle u = f$ に任意の $v \in X$ をかけて Ω 上で積分すると

$$-\int_\Omega \triangle u\, v\, dx = \int_\Omega fv\, dx. \tag{13}$$

左辺を Green の公式 ([10]) を用いて変形してから、$v = 0$ (on Γ_1) と (8) $\frac{\partial u}{\partial n} = g_2$ を用いると

$$\begin{aligned}
-\int_\Omega \triangle u\, v\, dx &= -\int_\Gamma \frac{\partial u}{\partial n} v\, d\sigma + \int_\Omega \nabla u \cdot \nabla v\, dx \\
&= -\int_{\Gamma_1} \frac{\partial u}{\partial n} v\, d\sigma - \int_{\Gamma_2} \frac{\partial u}{\partial n} v\, d\sigma + \int_\Omega \nabla u \cdot \nabla v\, dx \\
&= -\int_{\Gamma_2} g_2\, v\, d\sigma + \int_\Omega \nabla u \cdot \nabla v\, dx.
\end{aligned}$$

(13) に代入して移項すると

$$\int_\Omega \nabla u \cdot \nabla v\, dx = \int_\Omega f\, v\, dx + \int_{\Gamma_2} g_2\, v\, d\sigma.$$

すなわち、u は弱形式を満たす ((W) の解である)。 □

4.7.6 (W) と (V) は同値

(2) の証明

[u が (V) の解 $\Rightarrow$ u は (W) の解]　(デジャブ？ Dirichlet 原理の証明と似ている)

u は (V) の解とする。任意の $v \in X$ に対して、$f(t) := J[u+tv]$ $(t \in \mathbb{R})$ とおくと、

$$f(t) = J[u+tv] \geq J[u] = f(0).$$

ゆえに f は $t=0$ で最小になる。ゆえに $J[u+tv]$ の t の 1 次の項の係数は 0 である:

$$\int_\Omega \nabla u \cdot \nabla v \, dx - \int_\Omega f\, v \, dx - \int_{\Gamma_2} g_2\, v \, d\sigma = 0.$$

ゆえに u は (W) の解である。

4.7.6 (W) と (V) は同値

(2) の証明

[u が (V) の解 $\Rightarrow$ u は (W) の解]　(デジャブ？ Dirichlet 原理の証明と似ている)

u は (V) の解とする。任意の $v \in X$ に対して、$f(t) := J[u+tv]$ $(t \in \mathbb{R})$ とおくと、

$$f(t) = J[u+tv] \geq J[u] = f(0).$$

ゆえに f は $t=0$ で最小になる。ゆえに $J[u+tv]$ の t の 1 次の項の係数は 0 である:

$$\int_\Omega \nabla u \cdot \nabla v \, dx - \int_\Omega f\, v \, dx - \int_{\Gamma_2} g_2\, v \, d\sigma = 0.$$

ゆえに u は (W) の解である。[u が (W) の解 $\Rightarrow$ u は (V) の解]

u は (W) の解とする。任意の $w \in X_{g_1}$ に対して、$v := u - w$ とおくと $v \in X$ である。

$$\begin{aligned} J[w] - J[u] &= J[u + 1\cdot v] - J[u] \quad (t=1 \text{ として補題 12.2 を適用}) \\ &= \frac{1}{2}\int_\Omega |\nabla v|^2 \, dx + \left(\int_\Omega \nabla u \cdot \nabla v \, dx - \int_\Omega f\, v \, dx - \int_{\Gamma_2} g_2\, v \, d\sigma \right) \\ &= \frac{1}{2}\int_\Omega |\nabla v|^2 \, dx \geq 0. \end{aligned}$$

ゆえに $J[u]$ は $J[w]$ の最小値である。すなわち u は (V) の解である。 □

4.7.7 (W) の解が滑らかならば (P) の解

(3) の証明　u が (W) の解であり、かつ十分滑らかと仮定する。(W) の解であるから、

$$\int_\Omega \nabla u \cdot \nabla v \, d\boldsymbol{x} = \int_\Omega f \, v \, d\boldsymbol{x} + \int_{\Gamma_2} g \, v \, d\sigma \quad (v \in X)$$

を満たす。

4.7.7 (W) の解が滑らかならば (P) の解

(3) の証明　u が (W) の解であり、かつ十分滑らかと仮定する。(W) の解であるから、

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f \, v \, dx + \int_{\Gamma_2} g \, v \, d\sigma \quad (v \in X)$$

を満たす。さらに滑らかであるので、左辺に Green の公式が適用できて

$$(\star) \qquad \int_{\Gamma_2} \frac{\partial u}{\partial n} \, v \, d\sigma - \int_\Omega \triangle u \, v \, dx = \int_\Omega f \, v \, dx + \int_{\Gamma_2} g_2 \, v \, d\sigma \quad (v \in X).$$

4.7.7 (W) の解が滑らかならば (P) の解

(3) の証明　u が (W) の解であり、かつ十分滑らかと仮定する。(W) の解であるから、

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f \, v \, dx + \int_{\Gamma_2} g \, v \, d\sigma \quad (v \in X)$$

を満たす。さらに滑らかであるので、左辺に Green の公式が適用できて

$$(\star) \qquad \int_{\Gamma_2} \frac{\partial u}{\partial n} \, v \, d\sigma - \int_\Omega \triangle u \, v \, dx = \int_\Omega f \, v \, dx + \int_{\Gamma_2} g_2 \, v \, d\sigma \quad (v \in X).$$

特に $v \in C_0^\infty(\Omega)$ の場合を考えると (境界上の積分が 0 なので)

$$-\int_\Omega \triangle u \, v \, dx = \int_\Omega f \, v \, dx \quad (v \in C_0^\infty(\Omega)).$$

4.7.7 (W) の解が滑らかならば (P) の解

(3) の証明　u が (W) の解であり、かつ十分滑らかと仮定する。(W) の解であるから、

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f \, v \, dx + \int_{\Gamma_2} g \, v \, d\sigma \quad (v \in X)$$

を満たす。さらに滑らかであるので、左辺に Green の公式が適用できて

$$(\star) \qquad \int_{\Gamma_2} \frac{\partial u}{\partial n} \, v \, d\sigma - \int_\Omega \triangle u \, v \, dx = \int_\Omega f \, v \, dx + \int_{\Gamma_2} g_2 \, v \, d\sigma \quad (v \in X).$$

特に $v \in C_0^\infty(\Omega)$ の場合を考えると (境界上の積分が 0 なので)

$$-\int_\Omega \triangle u \, v \, dx = \int_\Omega f \, v \, dx \quad (v \in C_0^\infty(\Omega)).$$

変分法の基本補題より

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

4.7.7 (W) の解が滑らかならば (P) の解

(3) の証明　u が (W) の解であり、かつ十分滑らかと仮定する。(W) の解であるから、

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f \, v \, dx + \int_{\Gamma_2} g \, v \, d\sigma \quad (v \in X)$$

を満たす。さらに滑らかであるので、左辺に Green の公式が適用できて

$$(\star) \qquad \int_{\Gamma_2} \frac{\partial u}{\partial n} \, v \, d\sigma - \int_\Omega \triangle u \, v \, dx = \int_\Omega f \, v \, dx + \int_{\Gamma_2} g_2 \, v \, d\sigma \quad (v \in X).$$

特に $v \in C_0^\infty(\Omega)$ の場合を考えると (境界上の積分が 0 なので)

$$-\int_\Omega \triangle u \, v \, dx = \int_\Omega f \, v \, dx \quad (v \in C_0^\infty(\Omega)).$$

変分法の基本補題より

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

これを $(\star)$ に代入すると

$$\int_{\Gamma_2} \frac{\partial u}{\partial n} \, v \, d\sigma = \int_{\Gamma_2} g_2 \, v \, d\sigma \quad (v \in X).$$

4.7.7 (W) の解が滑らかならば (P) の解

(3) の証明　u が (W) の解であり、かつ十分滑らかと仮定する。(W) の解であるから、

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f \, v \, dx + \int_{\Gamma_2} g \, v \, d\sigma \quad (v \in X)$$

を満たす。さらに滑らかであるので、左辺に Green の公式が適用できて

$$(\star) \qquad \int_{\Gamma_2} \frac{\partial u}{\partial n} \, v \, d\sigma - \int_\Omega \triangle u \, v \, dx = \int_\Omega f \, v \, dx + \int_{\Gamma_2} g_2 \, v \, d\sigma \quad (v \in X).$$

特に $v \in C_0^\infty(\Omega)$ の場合を考えると (境界上の積分が 0 なので)

$$-\int_\Omega \triangle u \, v \, dx = \int_\Omega f \, v \, dx \quad (v \in C_0^\infty(\Omega)).$$

変分法の基本補題より

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

これを $(\star)$ に代入すると

$$\int_{\Gamma_2} \frac{\partial u}{\partial n} \, v \, d\sigma = \int_{\Gamma_2} g_2 \, v \, d\sigma \quad (v \in X).$$

再び変分法の基本補題より、$\dfrac{\partial u}{\partial n} = g_2$ (on Γ_2). ゆえに u は (P) の解である。 証明終

…… 以上、Dirichlet 原理の一般化

4.7.8 補足　変分法の基本補題

「任意の」(実際には「何かの条件を満たすすべての」) 関数 φ について $\int_\Omega f(x)\varphi(x)\,dx = 0$ が成り立つならば、$f = 0$ (in Ω), という形の命題を**変分法の基本補題** (fundamental lemma of calculs of variations) という。

色々なバージョンがあるが、次の形のもので用が足りることが多い。

命題 12.3 (変分法の基本補題)

Ω は $\mathbb{R}^n$ の開集合、$f : \Omega \to \mathbb{C}$ は局所可積分関数で

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

を満たすならば

$$f = 0 \quad (\text{a.e. on } \Omega).$$

Ω で局所可積分とは、Ω に含まれる任意のコンパクト集合上で Lebesgue 積分可能ということ。

$f = 0$ (a.e on Ω) とは、Ω に含まれるある測度 0 の集合 N を除いて $f = 0$ ということ。

$C_0^\infty(\Omega)$ については、次のページで説明する。

4.7.8 補足　$C_0^\infty(\Omega)$

$C_0^\infty(\Omega)$ という記号は、解析学で頻出する (知っておくと良い)。

Ω を $\mathbb{R}^n$ の開集合とするとき、$C_0^\infty(\Omega)$ を

$$C_0^\infty(\Omega) := \left\{ v \in C^\infty(\Omega) \,\middle|\, \operatorname{supp} v \text{ はコンパクトで、} \Omega \text{ に含まれる} \right\}$$

で定める。ここで $\operatorname{supp} v$ は v の台 (the support of v) と呼ばれる集合で、次式で定められる。

$$\operatorname{supp} v := \overline{\{x \in \Omega \mid v(x) \neq 0\}}.$$

ここで $\overline{}$ は、$\mathbb{R}^n$ における閉包を意味する。

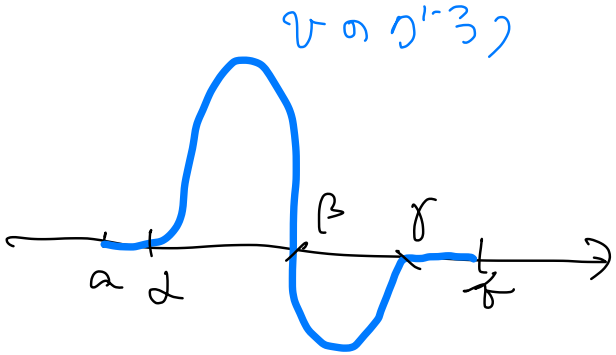
コンパクトとは、$\mathbb{R}^n$ の部分集合 K については、K が有界な閉集合ということである。

$v \in C_0^\infty(\Omega)$ とは、v が C^∞ 級で、$\partial\Omega$ のある近傍では $v = 0$ を満たすことを意味する。

4.7.8 補足　$C_0^\infty(\Omega)$

$\mathrm{supp}\, v$ とは どんなもの.

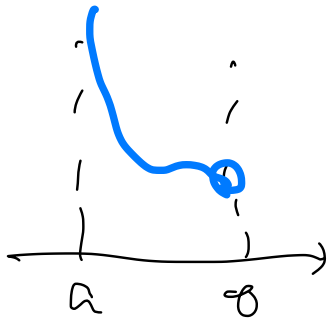
$\Omega = (a, b)$ にて



$\{x \in \Omega \mid v(x) \neq 0\} = (\alpha, \beta) \cup (\beta, \gamma)$

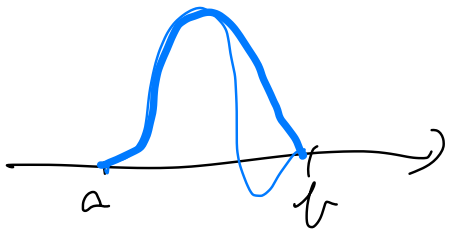
$\mathrm{supp}\, v = [\alpha, \gamma] \subset \Omega$
コンパクト

$\therefore \quad v \in C_0^\infty(\Omega)$



$\mathrm{supp}\, v = [a, b] \not\subset \Omega$

$\therefore \quad v \notin C_0^\infty(\Omega)$



$\mathrm{supp}\, v = [a, b]$

$\partial\Omega = \{a, b\}$ のまわりで $v = 0$

4.7.9 定理の使い道

(P) の解を求めたり、一意的な存在を示したいわけであるが、代わりに (W) (あるいは (V)) を考える。

4.7.9 定理の使い道

(P) の解を求めたり、一意的な存在を示したいわけであるが、代わりに (W) (あるいは (V)) を考える。

(W) の解の一意的な存在を示すのは比較的容易である。また (W) の近似解を求めるのも簡単である (有限要素法がまさにそれをしてくれる)。

4.7.9 定理の使い道

(P) の解を求めたり、一意的な存在を示したいわけであるが、代わりに (W) (あるいは (V)) を考える。

(W) の解の一意的な存在を示すのは比較的容易である。また (W) の近似解を求めるのも簡単である (有限要素法がまさにそれをしてくれる)。

(W) の解が本当に (P) の解であるか？が問題になる。言い換えると

(W) の解 u は滑らかだろうか？

4.7.9 定理の使い道

(P) の解を求めたり、一意的な存在を示したいわけであるが、代わりに (W) (あるいは (V)) を考える。

(W) の解の一意的な存在を示すのは比較的容易である。また (W) の近似解を求めるのも簡単である (有限要素法がまさにそれをしてくれる)。

(W) の解が本当に (P) の解であるか？が問題になる。言い換えると

(W) の解 u は滑らかだろうか？

この問は、一見細かいことのようだが、実はとても重要である。

4.7.9 定理の使い道

(P) の解を求めたり、一意的な存在を示したいわけであるが、代わりに (W) (あるいは (V)) を考える。

(W) の解の一意的な存在を示すのは比較的容易である。また (W) の近似解を求めるのも簡単である (有限要素法がまさにそれをしてくれる)。

(W) の解が本当に (P) の解であるか？が問題になる。言い換えると

(W) の解 u は滑らかだろうか？

この問は、一見細かいことのようだが、実はとても重要である。

良く知られた (部分的な) 解答

- Ω が C^2 級であれば (どういう意味？) Yes.
- Ω が多角形の場合、凸ならば Yes, 凸でないならば一般には No.

4.7.9 定理の使い道

(P) の解を求めたり、一意的な存在を示したいわけであるが、代わりに (W) (あるいは (V)) を考える。

(W) の解の一意的な存在を示すのは比較的容易である。また (W) の近似解を求めるのも簡単である (有限要素法がまさにそれをしてくれる)。

(W) の解が本当に (P) の解であるか？が問題になる。言い換えると

(W) の解 u は滑らかだろうか？

この問は、一見細かいことのようだが、実はとても重要である。

良く知られた (部分的な) 解答

- Ω が C^2 級であれば (どういう意味？) Yes.
- Ω が多角形の場合、凸ならば Yes, 凸でないならば一般には No.

(FreeFem++ の例で、L 字型の領域や、立方体から小さい立方体を除いた領域がしばしば登場するが、このあたりのこと (領域の凸性) を問題にしているわけである。)

2022/6/20 の授業は、4.7 の説明までで終わりました。

次回以降に、以下の 4.8 を講義するかどうかは、まだ決めていません。

4.8 ポテンシャル問題の数値解法 (2) 基本解の方法
4.8.1 $-\triangle$ の基本解

次の関数 E は、$-\triangle$ の**基本解** (fundamental solution) と呼ばれる。

$$E(x) := \begin{cases} -\dfrac{1}{2\pi}\log|x| & (2\text{ 次元の場合, } x \in \mathbb{R}^2 \setminus \{0\}) \\ \dfrac{1}{4\pi}\dfrac{1}{|x|} & (3\text{ 次元の場合, } x \in \mathbb{R}^3 \setminus \{0\}). \end{cases} \tag{14}$$

4.8 ポテンシャル問題の数値解法 (2) 基本解の方法
4.8.1 $-\triangle$ の基本解

次の関数 E は、$-\triangle$ の**基本解** (fundamental solution) と呼ばれる。

$$E(x) := \begin{cases} -\dfrac{1}{2\pi}\log|x| & (2\text{ 次元の場合, } x \in \mathbb{R}^2 \setminus \{0\}) \\ \dfrac{1}{4\pi}\dfrac{1}{|x|} & (3\text{ 次元の場合, } x \in \mathbb{R}^3 \setminus \{0\}). \end{cases} \tag{14}$$

一体何者か？

4.8 ポテンシャル問題の数値解法 (2) 基本解の方法
4.8.1 $-\triangle$ の基本解

次の関数 E は、$-\triangle$ の**基本解** (fundamental solution) と呼ばれる。

$$E(x) := \begin{cases} -\dfrac{1}{2\pi}\log|x| & (2\text{ 次元の場合, } x \in \mathbb{R}^2 \setminus \{0\}) \\ \dfrac{1}{4\pi}\dfrac{1}{|x|} & (3\text{ 次元の場合, } x \in \mathbb{R}^3 \setminus \{0\}). \end{cases} \tag{14}$$

一体何者か？

数学的な解答　次を満たす。ここで δ は **Dirac のデルタ超関数**。

$$-\triangle E = \delta \qquad (\triangle \overset{\text{def.}}{=} \sum_{j=1}^{n} \frac{\partial^2}{\partial x_j^2}). \tag{15}$$

4.8 ポテンシャル問題の数値解法 (2) 基本解の方法
4.8.1 $-\triangle$ の基本解

次の関数 E は、$-\triangle$ の**基本解** (fundamental solution) と呼ばれる。

$$E(x) := \begin{cases} -\dfrac{1}{2\pi}\log|x| & (2\text{ 次元の場合, } x \in \mathbb{R}^2 \setminus \{0\}) \\ \dfrac{1}{4\pi}\dfrac{1}{|x|} & (3\text{ 次元の場合, } x \in \mathbb{R}^3 \setminus \{0\}). \end{cases} \tag{14}$$

一体何者か？

数学的な解答　次を満たす。ここで δ は **Dirac のデルタ超関数**。

$$-\triangle E = \delta \qquad (\triangle \overset{\text{def.}}{=} \sum_{j=1}^{n} \frac{\partial^2}{\partial x_j^2}). \tag{15}$$

物理的な解答 (解釈)　E は原点に置かれた単位点電荷の作る電場のポテンシャル (電位) である。

4.8 ポテンシャル問題の数値解法 (2) 基本解の方法
4.8.1 $-\triangle$ の基本解

次の関数 E は、$-\triangle$ の**基本解** (fundamental solution) と呼ばれる。

$$E(x) := \begin{cases} -\dfrac{1}{2\pi}\log|x| & (2\text{ 次元の場合, } x \in \mathbb{R}^2 \setminus \{0\}) \\ \dfrac{1}{4\pi}\dfrac{1}{|x|} & (3\text{ 次元の場合, } x \in \mathbb{R}^3 \setminus \{0\}). \end{cases} \tag{14}$$

一体何者か？

数学的な解答　次を満たす。ここで δ は **Dirac のデルタ超関数**。

$$-\triangle E = \delta \qquad (\triangle \overset{\text{def.}}{=} \sum_{j=1}^{n} \frac{\partial^2}{\partial x_j^2}). \tag{15}$$

物理的な解答 (解釈)　E は原点に置かれた単位点電荷の作る電場のポテンシャル (電位) である。

(Cf. 密度 ρ で分布する場合のポテンシャル u は $-\triangle u = \rho$ を満たす。)

4.8.1 $-\triangle$ の基本解

なぜ基本解は重要か？

4.8.1 $-\triangle$ の基本解

なぜ基本解は重要か？重ね合わせることで “任意” の電荷分布 ρ のポテンシャルが得られる。

定理 (のようなもの)　Poisson 方程式の特解

$\Omega \subset \mathbb{R}^n$, $\rho\colon \Omega \to \mathbb{R}$ が与えられたとき

$$u(x) := \int_\Omega E(x-y)\rho(y)\,dy \quad (x \in \Omega) \tag{16}$$

とおくと次式が成り立つ。

$$-\triangle u = \rho \quad (\text{in } \Omega). \tag{17}$$

4.8.1 $-\triangle$ の基本解

なぜ基本解は重要か？重ね合わせることで “任意” の電荷分布 ρ のポテンシャルが得られる。

定理 (のようなもの)　Poisson 方程式の特解

$\Omega \subset \mathbb{R}^n$, $\rho\colon \Omega \to \mathbb{R}$ が与えられたとき

$$u(x) := \int_\Omega E(x-y)\rho(y)\,dy \quad (x \in \Omega) \tag{16}$$

とおくと次式が成り立つ。

$$-\triangle u = \rho \quad (\text{in } \Omega). \tag{17}$$

E には特異性があるので、(17) を証明するのは少し難しい (神保 [11] など)。物理的には次のように納得できる。

4.8.1 $-\triangle$ の基本解

なぜ基本解は重要か？重ね合わせることで “任意” の電荷分布 ρ のポテンシャルが得られる。

定理 (のようなもの)　Poisson 方程式の特解

$\Omega \subset \mathbb{R}^n$, $\rho\colon \Omega \to \mathbb{R}$ が与えられたとき

$$u(x) := \int_\Omega E(x-y)\rho(y)\,dy \quad (x \in \Omega) \tag{16}$$

とおくと次式が成り立つ。

$$-\triangle u = \rho \quad (\text{in } \Omega). \tag{17}$$

E には特異性があるので、(17) を証明するのは少し難しい (神保 [11] など)。

物理的には次のように納得できる。

微小体積 dy に存在する電荷は $\rho(y)\,dy$ で、それが作る電場のポテンシャルは (基本解を平行移動したものの電荷量倍で) $E(x-y)\rho(y)\,dy$. それを Ω 全体でトータルした u がポテンシャルになる。実際、$\boldsymbol{E} := -\operatorname{grad} u$ は電場で、Maxwell の方程式の 1 つ $\operatorname{div} \boldsymbol{E} = \rho$ から、$-\operatorname{div}\operatorname{grad} u = \rho$ が得られる。すなわち (17) が成り立つ。

ポテンシャル問題の数値解法 (近似解法) への応用「**基本解の方法** (the method of fundamental solutions)」を紹介する。

Dirichlet 境界値問題を考えよう (Neumann 境界値問題でも同様)。

$$\triangle u = 0 \quad (\text{in } \Omega) \tag{18}$$

$$u = g \quad (\text{on } \partial\Omega). \tag{19}$$

ここで Ω は $\mathbb{R}^n$ ($n = 2$ or $n = 3$) の領域である。

4.8.2 基本解の方法のアルゴリズム (電荷の作る電場の電位でポテンシャルを近似)

ポテンシャル問題の数値解法 (近似解法) への応用「**基本解の方法** (the method of fundamental solutions)」を紹介する。

Dirichlet 境界値問題を考えよう (Neumann 境界値問題でも同様)。

$$\triangle u = 0 \quad (\text{in } \Omega) \tag{18}$$

$$u = g \quad (\text{on } \partial\Omega). \tag{19}$$

ここで Ω は $\mathbb{R}^n$ ($n = 2$ or $n = 3$) の領域である。

Ω の外部に、Ω を取り囲むように、有限個の点 $y_1, \cdots, y_N$ を取り、各 y_k に電荷量 Q_k の電荷を置く。

4.8.2 アルゴリズム (電荷の作る電場の電位でポテンシャルを近似)

それらの電荷の作る電場のポテンシャルは

$$u^{(N)}(x) := \sum_{k=1}^{N} Q_k E(x - y_k). \tag{20}$$

4.8.2 アルゴリズム (電荷の作る電場の電位でポテンシャルを近似)

それらの電荷の作る電場のポテンシャルは

$$u^{(N)}(x) := \sum_{k=1}^{N} Q_k E(x - y_k). \tag{20}$$

$\triangle E = 0$ (in $\mathbb{R}^n \setminus \{0\}$) であるから、$Q_k$ の取り方によらず

$\triangle u^{(N)}(x) = 0 \quad (x \in \mathbb{R}^n \setminus \{y_1, \cdots, y_N\})$. 特に $\triangle u^{(N)} = 0$ (in Ω).

4.8.2 アルゴリズム (電荷の作る電場の電位でポテンシャルを近似)

それらの電荷の作る電場のポテンシャルは

$$u^{(N)}(x) := \sum_{k=1}^{N} Q_k E(x - y_k). \tag{20}$$

$\triangle E = 0$ (in $\mathbb{R}^n \setminus \{0\}$) であるから、$Q_k$ の取り方によらず

$\triangle u^{(N)}(x) = 0 \quad (x \in \mathbb{R}^n \setminus \{y_1, \cdots, y_N\})$. 特に $\triangle u^{(N)} = 0$ (in Ω).

後は Q_k をうまく選んで、境界条件 (19) $u = g$ (on $\partial\Omega$) を近似的に満たすようにする。

4.8.2 アルゴリズム (電荷の作る電場の電位でポテンシャルを近似)

それらの電荷の作る電場のポテンシャルは

$$u^{(N)}(x) := \sum_{k=1}^{N} Q_k E(x - y_k). \tag{20}$$

$\triangle E = 0$ (in $\mathbb{R}^n \setminus \{0\}$) であるから、$Q_k$ の取り方によらず

$\triangle u^{(N)}(x) = 0 \quad (x \in \mathbb{R}^n \setminus \{y_1, \cdots, y_N\})$. 特に $\triangle u^{(N)} = 0$ (in Ω).

後は Q_k をうまく選んで、境界条件 (19) $u = g$ (on $\partial\Omega$) を近似的に満たすようにする。

一つのやり方として、$\partial\Omega$ 上に N 個の点 $x_1, \cdots, x_N$ を取って

$$u^{(N)}(x_j) = g(x_j) \quad (j = 1, \cdots, N). \tag{21}$$

4.8.2 アルゴリズム (電荷の作る電場の電位でポテンシャルを近似)

それらの電荷の作る電場のポテンシャルは

$$u^{(N)}(x) := \sum_{k=1}^{N} Q_k E(x - y_k). \tag{20}$$

$\triangle E = 0$ (in $\mathbb{R}^n \setminus \{0\}$) であるから、$Q_k$ の取り方によらず

$\triangle u^{(N)}(x) = 0 \quad (x \in \mathbb{R}^n \setminus \{y_1, \cdots, y_N\})$. 特に $\triangle u^{(N)} = 0$ (in Ω).

後は Q_k をうまく選んで、境界条件 (19) $u = g$ (on $\partial\Omega$) を近似的に満たすようにする。

一つのやり方として、$\partial\Omega$ 上に N 個の点 $x_1, \cdots, x_N$ を取って

$$u^{(N)}(x_j) = g(x_j) \quad (j = 1, \cdots, N). \tag{21}$$

これで Q_k $(k = 1, \cdots, N)$ が定まることはすぐ分かる (次のスライド)。

4.8.2 アルゴリズム (電荷の作る電場の電位でポテンシャルを近似)

それらの電荷の作る電場のポテンシャルは

$$u^{(N)}(x) := \sum_{k=1}^{N} Q_k E(x - y_k). \tag{20}$$

$\triangle E = 0$ (in $\mathbb{R}^n \setminus \{0\}$) であるから、$Q_k$ の取り方によらず

$\triangle u^{(N)}(x) = 0 \quad (x \in \mathbb{R}^n \setminus \{y_1, \cdots, y_N\})$. 特に $\triangle u^{(N)} = 0$ (in Ω).

後は Q_k をうまく選んで、境界条件 (19) $u = g$ (on $\partial\Omega$) を近似的に満たすようにする。

一つのやり方として、$\partial\Omega$ 上に N 個の点 $x_1, \cdots, x_N$ を取って

$$u^{(N)}(x_j) = g(x_j) \quad (j = 1, \cdots, N). \tag{21}$$

これで Q_k $(k = 1, \cdots, N)$ が定まることはすぐ分かる (次のスライド)。

非常に素朴な感じがするが、とてもうまく行くことが多い。

4.8.2 アルゴリズム (電荷の作る電場の電位でポテンシャルを近似)

(21) は次の連立 1 次方程式と同値である。

$$\begin{pmatrix} E(x_1-y_1) & E(x_1-y_2) & \cdots & E(x_1-y_N) \\ E(x_2-y_1) & E(x_2-y_2) & & E(x_2-y_N) \\ \vdots & & \ddots & \vdots \\ E(x_N-y_1) & E(x_N-y_2) & \cdots & E(x_N-y_N) \end{pmatrix} \begin{pmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_N \end{pmatrix} = \begin{pmatrix} g(x_1) \\ g(x_2) \\ \vdots \\ g(x_N) \end{pmatrix}.$$

Gauss の消去法などを用いて、Q_k $(k = 1, \cdots, N)$ が求められる。

4.8.2 アルゴリズム (電荷の作る電場の電位でポテンシャルを近似)

(21) は次の連立 1 次方程式と同値である。

$$\begin{pmatrix} E(x_1-y_1) & E(x_1-y_2) & \cdots & E(x_1-y_N) \\ E(x_2-y_1) & E(x_2-y_2) & & E(x_2-y_N) \\ \vdots & & \ddots & \vdots \\ E(x_N-y_1) & E(x_N-y_2) & \cdots & E(x_N-y_N) \end{pmatrix} \begin{pmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_N \end{pmatrix} = \begin{pmatrix} g(x_1) \\ g(x_2) \\ \vdots \\ g(x_N) \end{pmatrix}.$$

Gauss の消去法などを用いて、Q_k $(k = 1, \cdots, N)$ が求められる。

(いわゆる密行列であるが、それほど大きな N は必要ないので、難しくない。)

4.8.3 基本解の方法の特徴

4.8.3 基本解の方法の特徴

1. ある ρ $(0 < \rho < 1)$, C が存在して

$$\left\| u - u^{(N)} \right\| \leq C\rho^N \quad (\|\cdot\| \text{ は適当なノルム})$$

が成り立つ (**誤差の指数関数的減少**, 次のスライドで数値例を示す)。

4.8.3 基本解の方法の特徴

1. ある ρ $(0 < \rho < 1)$, C が存在して

$$\left\| u - u^{(N)} \right\| \leq C\rho^N \quad (\|\cdot\| \text{ は適当なノルム})$$

が成り立つ (**誤差の指数関数的減少**, 次のスライドで数値例を示す)。しばしば、**高精度の解が非常に少ない計算量で得られる**ことが期待できる。

4.8.3 基本解の方法の特徴

1. ある ρ $(0 < \rho < 1)$, C が存在して

$$\left\| u - u^{(N)} \right\| \leq C\rho^N \quad (\|\cdot\| \text{ は適当なノルム})$$

が成り立つ (**誤差の指数関数的減少**, 次のスライドで数値例を示す)。しばしば、**高精度の解が非常に少ない計算量で得られる**ことが期待できる。

Cf. 差分法, 有限要素法では、典型的な場合に $\left\| u - u^{(N)} \right\| \leq \frac{C}{N^2}$.

4.8.3 基本解の方法の特徴

1. ある ρ $(0 < \rho < 1)$, C が存在して

$$\left\| u - u^{(N)} \right\| \leq C\rho^N \quad (\|\cdot\| \text{ は適当なノルム})$$

が成り立つ (**誤差の指数関数的減少**, 次のスライドで数値例を示す)。しばしば、**高精度の解が非常に少ない計算量で得られる**ことが期待できる。
Cf. 差分法, 有限要素法では、典型的な場合に $\left\| u - u^{(N)} \right\| \leq \frac{C}{N^2}$.

2. $u^{(N)}$ は調和関数である。特に $\operatorname{grad} u^{(N)}$ の計算が簡単:

$$\operatorname{grad} u^{(N)}(x) = -\frac{1}{2\pi} \sum_{k=1}^{N} Q_k \frac{x - y_k}{|x - y_k|^2} \quad (2 \text{ 次元の場合}).$$

(例えばポテンシャル流の計算を思い浮かべると、超便利と分かる。)

4.8.3 基本解の方法の特徴

1. ある ρ $(0 < \rho < 1)$, C が存在して

$$\left\| u - u^{(N)} \right\| \leq C\rho^N \quad (\|\cdot\| \text{ は適当なノルム})$$

が成り立つ (**誤差の指数関数的減少**, 次のスライドで数値例を示す)。しばしば、**高精度の解が非常に少ない計算量で得られる**ことが期待できる。

Cf. 差分法, 有限要素法では、典型的な場合に $\left\| u - u^{(N)} \right\| \leq \frac{C}{N^2}$.

2. $u^{(N)}$ は調和関数である。特に $\operatorname{grad} u^{(N)}$ の計算が簡単:

$$\operatorname{grad} u^{(N)}(x) = -\frac{1}{2\pi} \sum_{k=1}^{N} Q_k \frac{x - y_k}{|x - y_k|^2} \quad (2 \text{ 次元の場合}).$$

(例えばポテンシャル流の計算を思い浮かべると、超便利と分かる。)

しかも $\left\| \operatorname{grad} u - \operatorname{grad} u^{(N)} \right\|$ も指数関数的に減少する。

Cf. 差分法や有限要素法では、微分が難しかったり、精度が下がったりする。

4.8.3 基本解の方法の特徴

1. ある ρ $(0 < \rho < 1)$, C が存在して

$$\left\| u - u^{(N)} \right\| \leq C\rho^N \quad (\|\cdot\| \text{ は適当なノルム})$$

が成り立つ (**誤差の指数関数的減少**, 次のスライドで数値例を示す)。しばしば、**高精度の解が非常に少ない計算量で得られる**ことが期待できる。

Cf. 差分法, 有限要素法では、典型的な場合に $\left\| u - u^{(N)} \right\| \leq \frac{C}{N^2}$.

2. $u^{(N)}$ は調和関数である。特に $\operatorname{grad} u^{(N)}$ の計算が簡単:

$$\operatorname{grad} u^{(N)}(x) = -\frac{1}{2\pi} \sum_{k=1}^{N} Q_k \frac{x - y_k}{|x - y_k|^2} \quad (2 \text{ 次元の場合}).$$

(例えばポテンシャル流の計算を思い浮かべると、超便利と分かる。)

しかも $\left\| \operatorname{grad} u - \operatorname{grad} u^{(N)} \right\|$ も指数関数的に減少する。

Cf. 差分法や有限要素法では、微分が難しかったり、精度が下がったりする。

3. 理論的な基礎づけは、差分法、有限要素法と比べて不十分である。
4. 同次方程式にしか適用できない、具体的な基本解が必要 → 汎用性は低い。

4.8.3 基本解の方法の特徴

1. ある ρ $(0 < \rho < 1)$, C が存在して

$$\left\| u - u^{(N)} \right\| \leq C\rho^N \quad (\|\cdot\| \text{ は適当なノルム})$$

が成り立つ (**誤差の指数関数的減少**, 次のスライドで数値例を示す)。しばしば、**高精度の解が非常に少ない計算量で得られる**ことが期待できる。

Cf. 差分法, 有限要素法では、典型的な場合に $\left\| u - u^{(N)} \right\| \leq \frac{C}{N^2}$.

2. $u^{(N)}$ は調和関数である。特に $\operatorname{grad} u^{(N)}$ の計算が簡単:

$$\operatorname{grad} u^{(N)}(x) = -\frac{1}{2\pi} \sum_{k=1}^{N} Q_k \frac{x - y_k}{|x - y_k|^2} \quad (2 \text{ 次元の場合}).$$

(例えばポテンシャル流の計算を思い浮かべると、超便利と分かる。)

しかも $\left\| \operatorname{grad} u - \operatorname{grad} u^{(N)} \right\|$ も指数関数的に減少する。

Cf. 差分法や有限要素法では、微分が難しかったり、精度が下がったりする。

3. 理論的な基礎づけは、差分法、有限要素法と比べて不十分である。
4. 同次方程式にしか適用できない、具体的な基本解が必要 → 汎用性は低い。

汎用性低いが、使えるときは、差分法・有限要素法に性能で勝る場合が多い。

4.8.3 基本解の方法の特徴　数値例

Ω が円盤 $\left\{\boldsymbol{x} \in \mathbb{R}^2 \,\middle|\, |\boldsymbol{x}| < 1\right\}$ の場合に、原点中心半径 $R = 2$ の円周上に一様に電荷点 y_k を配置した場合の近似解の精度を示す。

左が $g(\boldsymbol{x}) = \mathrm{Re}\,[(x+iy)^m]$, 右が $g(\boldsymbol{x}) = \log|\boldsymbol{x} - \boldsymbol{p}|$ $(\boldsymbol{p} = (p, 0))$ の場合。

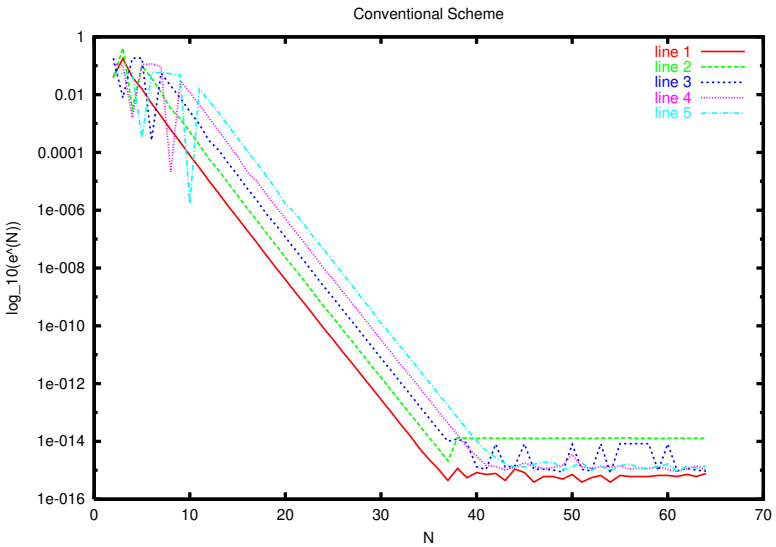


図 4: $m = 1, \cdots, 5$

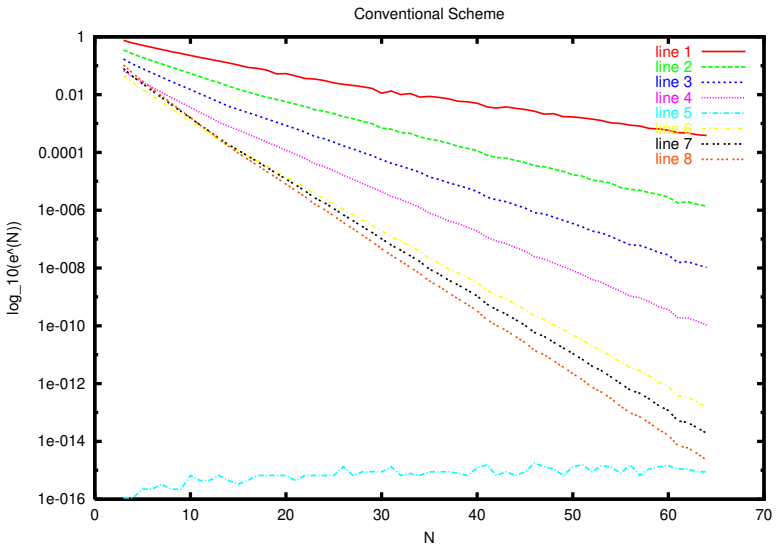


図 5: $p = 1.2, 1.4, \cdots, 2.6$. p が大きい (特異点が遠い) ほど速く減衰

誤差の減少は非常に速い！

4.8.3 基本解の方法の特徴　数値例

Ω が円盤 $\left\{\boldsymbol{x} \in \mathbb{R}^2 \,\middle|\, |\boldsymbol{x}| < 1\right\}$ の場合に、原点中心半径 $R = 2$ の円周上に一様に電荷点 y_k を配置した場合の近似解の精度を示す。

左が $g(\boldsymbol{x}) = \mathrm{Re}\,[(x+iy)^m]$, 右が $g(\boldsymbol{x}) = \log|\boldsymbol{x} - \boldsymbol{p}|$ $(\boldsymbol{p} = (p, 0))$ の場合。

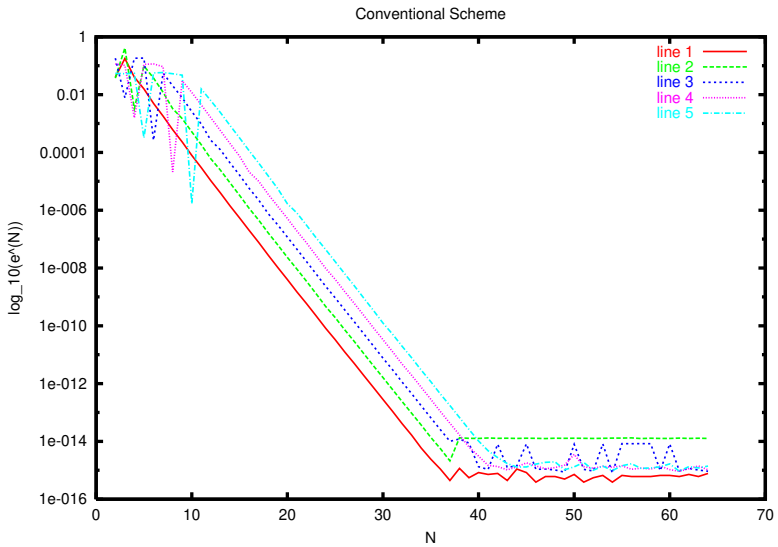


図 4: $m = 1, \cdots, 5$

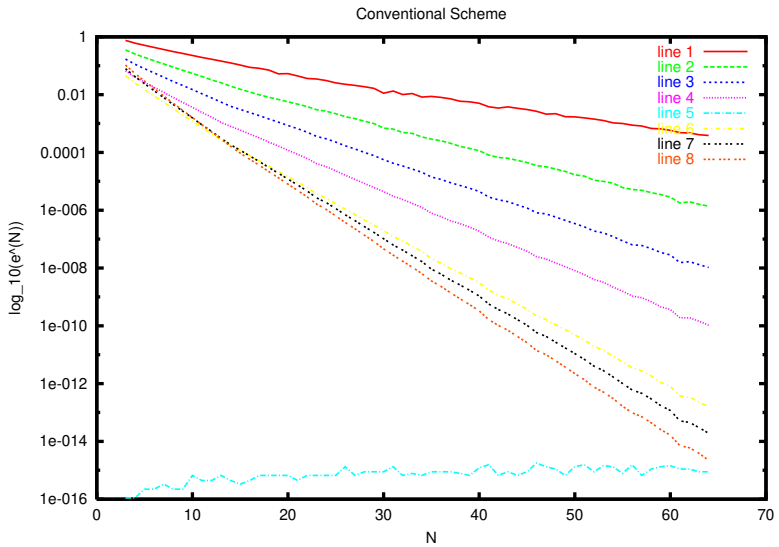


図 5: $p = 1.2, 1.4, \cdots, 2.6$. p が大きい (特異点が遠い) ほど速く減衰

誤差の減少は非常に速い！

片側対数目盛で、直線上にのることから、誤差が指数関数的に減少している。減少の速さについては研究されていて、ある程度まで説明可能である。

4.8.4 数値等角写像に対する天野の方法

天野要は、§4.4 で述べた等角写像の求め方 (定理 8.6) と、基本解の方法を組み合わせた、数値等角写像 (領域の写像関数を数値的に求めること) の効率的なアルゴリズムを提唱した (天野 [12])。それを解説する。

§4.4 で導入した記号を用いる。

u の近似 $u^{(N)}$ を基本解の方法で求めよう。$N \in \mathbb{N}$ に対して、$\{\zeta_k\}_{k=1}^N$ を「Ω を取り囲むように」 $\mathbb{C} \setminus \overline{\Omega}$ から選び、

$$u^{(N)}(z) := \sum_{k=1}^{N} Q_k \log|z - \zeta_k| \tag{22}$$

とおく。ここで Q_k $(k = 1, \ldots, N)$ は未知の実定数である。

$\{z_j\}_{j=1}^N$ を $\partial\Omega$ から選び、連立 1 次方程式

$$u^{(N)}(z_j) = -\log|z_j - z_0| \quad (j = 1, \ldots, N) \tag{23}$$

を解いて Q_k $(k = 1, \ldots, N)$ が求められる。

4.8.4 数値等角写像に対する天野の方法

$u^{(N)}$ の共役調和関数 $v^{(N)}$ を求めたい ($u^{(N)}$ を実部に持つ正則関数を求めたい)。

4.8.4 数値等角写像に対する天野の方法

$u^{(N)}$ の共役調和関数 $v^{(N)}$ を求めたい ($u^{(N)}$ を実部に持つ正則関数を求めたい)。天下りになるが、

$$f^{(N)}(z) := Q_0 + \sum_{k=1}^{N} Q_k \operatorname{Log} \frac{z-\zeta_k}{z_0-\zeta_k}, \qquad Q_0 := \sum_{k=1}^{N} Q_k \log|z_0-\zeta_k| \tag{24}$$

とおく。ここで Log は主値を表すとする ($\mathbb{C} \setminus (-\infty, 0]$ を定義域とする)。

4.8.4 数値等角写像に対する天野の方法

$u^{(N)}$ の共役調和関数 $v^{(N)}$ を求めたい ($u^{(N)}$ を実部に持つ正則関数を求めたい)。天下りになるが、

$$f^{(N)}(z) := Q_0 + \sum_{k=1}^{N} Q_k \operatorname{Log} \frac{z-\zeta_k}{z_0-\zeta_k}, \qquad Q_0 := \sum_{k=1}^{N} Q_k \log|z_0-\zeta_k| \tag{24}$$

とおく。ここで Log は主値を表すとする ($\mathbb{C} \setminus (-\infty, 0]$ を定義域とする)。

$$\operatorname{Re} f^{(N)}(z) = \sum_{k=1}^{N} Q_k \log|z_0-\zeta_k| + \sum_{k=1}^{N} Q_k \log\left|\frac{z-\zeta_k}{z_0-\zeta_k}\right| = \sum_{k=1}^{N} Q_k \log|z-\zeta_k| = u^{(N)}(z)$$

である。さらに

$$f^{(N)}(z_0) = Q_0 + \sum_{k=1}^{N} Q_k \operatorname{Log} \frac{z_0-\zeta_k}{z_0-\zeta_k} = Q_0 + \sum_{k=1}^{N} 0 = Q_0 \in \mathbb{R}.$$

言い換えると $\operatorname{Im} f^{(N)}(z_0) = 0$. この $f^{(N)}$ は、$f = u + iv$ の良い近似であると考えられる。

4.8.4 数値等角写像に対する天野の方法

以上をまとめると、次のアルゴリズムが得られる。

(再掲 22) $$u^{(N)}(z) := \sum_{k=1}^{N} Q_k \log|z-\zeta_k|,$$

(再掲 23) $$u^{(N)}(z_j) = -\log|z_j - z_0| \quad (j=1,\dots,N),$$

(再掲 24) $$f^{(N)}(z) := Q_0 + \sum_{k=1}^{N} Q_k \operatorname{Log}\frac{z-\zeta_k}{z_0-\zeta_k}, \quad Q_0 := \sum_{k=1}^{N} Q_k \log|z_0-\zeta_k|$$

天野のアルゴリズム

0. $\{\zeta_k\}_{k=1}^{N} \subset \mathbb{C}\setminus\overline{\Omega}$, $\{z_j\}_{j=1}^{N} \subset \partial\Omega$ を適当に選ぶ。
1. (22), (23) で $\{Q_k\}$ を求める。
2. (24) で $f^{(N)}$ を定める。
3. $\varphi^{(N)}(z) := (z-z_0)\exp f^{(N)}(z)$ で定義される $\varphi^{(N)}$ を、等角写像 $\varphi\colon \Omega \to D_1$ の近似として採用する。

4.8.5 Jordan 領域の等角写像の計算プログラム

以下の Python プログラム `conformalmap-v2.py` では

$$\Omega = D_1 \overset{\text{def.}}{=} \{z \in \mathbb{C} \mid |z| < 1\}, \quad z_0 = \frac{1}{2}$$

の場合の Ω の写像関数、すなわち双正則な $\varphi\colon \Omega \to D_1$ で

$$\varphi(z_0) = 0, \quad \varphi(z_0) > 0$$

を満たすものを求める。この場合、実は次の 1 次分数変換が解である。

$$\varphi(z) = \frac{z - z_0}{1 - \overline{z_0} z}.$$

プログラム入手 — ターミナルで次を実行

```
curl -O http://nalab.mind.meiji.ac.jp/~mk/complex2/conformalmap-v2.py
```

ターミナルで次のように実行

```
python conformalmap-v2.py
```

4.8.5 Jordan 領域の等角写像の計算プログラム

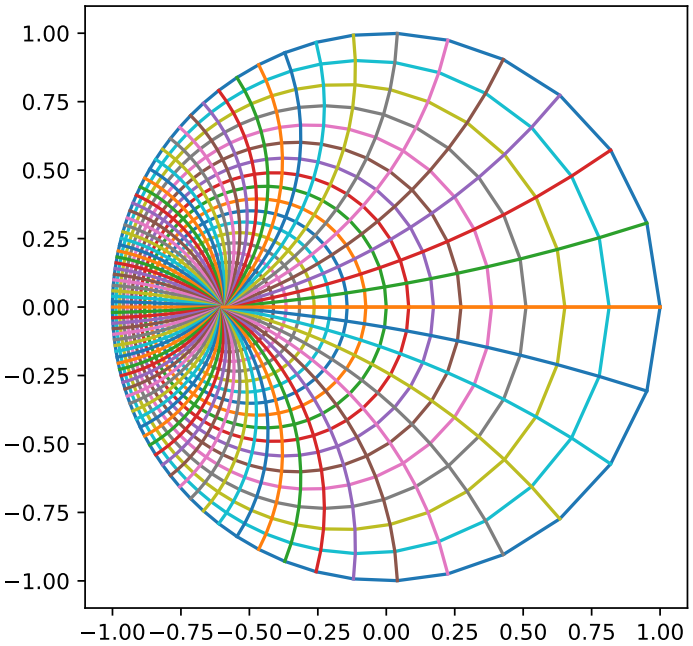


図 6: $z_0 = 0.6$ の場合. $w = \varphi(z)$ による z 平面の原点中心の同心円、単位円の半径の像を描いた。

参考文献

[1] 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.

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

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

[4] Suzuki, A.: Finite element programming by FreeFem++ —intermediate course, 日本応用数理学会「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).

[5] Suzuki, A.: Finite element programming by FreeFem++ —advanced course, 日本応用数理学会「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: