

ポテンシャル問題とその数値解法

桂田 祐史

2017年6月20日, 2023年7月20日

<https://m-katsurada.sakura.ne.jp/complex2/>

複素関数論 (以下単に「関数論」という) では、Laplace 方程式の境界値問題 (ポテンシャル問題と呼ばれることがある) が良く現れる。その事情を簡単に紹介する。

ポテンシャル問題では、かなり一般的に解の存在と一意性が成り立つが、厳密解が具体的な式で表せることは稀で、多くの場合、数値計算で近似解を求めるしかない。幸い多くの数値解法が利用できる。代表的な数値解法を紹介する。

現象数理学科の学生で、この文書に載っているプログラムの動かし方が分からない人は、相談に応じます。

目次

1	はじめに	2
2	ポテンシャル問題に対する Dirichlet の原理と Poisson 方程式に対する弱解の方法	3
2.1	ポテンシャル問題に対する解の存在証明、Dirichlet の原理	3
2.2	Poisson 方程式に対する弱解の方法	4
2.2.1	1次元の場合の弱解の方法	4
2.2.2	1次元の場合の有限要素法	6
2.2.3	2次元の場合の弱解の方法	7
3	有限要素法と FreeFem++	7
3.1	FreeFem++ プログラム (その 1)	7
3.2	FreeFem++ プログラム (その 2) 流体の速度ポテンシャルを求める	9
3.3	工事中	11
3.3.1	サンプル・プログラム	13
4	等角写像, Riemann の写像定理 (工事中)	14
5	領域の等角写像, Riemann の写像定理, ポテンシャル問題との関係	17
5.1	はじめに	17
5.2	Riemann の写像定理	17
5.3	単連結領域の場合の等角写像の正規化条件	18
5.4	Jordan 領域の等角写像を求めるアルゴリズム	18
5.5	(細かい話) 多重連結領域の場合	19

6 Laplace 方程式に対する基本解の方法	21
6.1 準備	21
6.1.1 基本解とは	21
6.1.2 Poisson 方程式の特解	21
6.1.3 Green の積分公式 (続き)	21
6.2 基本解の方法	22
6.3 近似等角写像を求めるための天野の方法	23
6.4 Python による Jordan 領域の等角写像の計算プログラム	24
A ポテンシャル問題に対する Poincaré-Perron の方法	26
B Poisson 方程式に対する差分法	27
B.1 基本的な考え方	27
B.2 1次元でイントロ	28
B.2.1 差分方程式	28
B.2.2 C 言語のプログラム例	29
B.3 2次元の場合 (結果だけ紹介)	31
B.3.1 差分方程式	32
B.3.2 差分方程式 (連立1次方程式) の行列、ベクトル表現	33
B.3.3 MATLAB プログラム	34
B.4 区間でない領域における差分法	36
B.5 結び	36
C Eigen ライブラリを用いた Jordan 領域の等角写像の計算プログラム	36

1 はじめに

Laplace 方程式 $\Delta u = 0$ の境界値問題を**ポテンシャル問題**という。正則関数の実部・虚部は調和関数 (ラプラス方程式の解) であるため、関数論のあちこちの重要な場面でポテンシャル問題が登場する。例えば関数論で重要な定理の一つである Riemann の写像定理に現れる等角写像を求めるためにも、ポテンシャル問題が現れる。

(2次元渦無し非圧縮流の速度ポテンシャル ϕ は、Laplace 方程式の Neumann 境界値問題

$$\Delta \phi = 0 \quad \text{in } \Omega, \quad \frac{\partial \phi}{\partial n} = \mathbf{v} \cdot \mathbf{n} \quad \text{on } \partial \Omega$$

の解である、ということを経に紹介したが、そのような応用上の問題に止まらない、ということを経認識してもらいたい。)

ここでは少し一般化した Poisson 方程式の境界値問題

$$(1.1) \quad -\Delta u = f \quad \text{in } \Omega,$$

$$(1.2) \quad u = g_1 \quad \text{in } \Gamma_1,$$

$$(1.3) \quad \frac{\partial u}{\partial n} = g_2 \quad \text{in } \Gamma_2$$

を考える。ここで Ω は平面内の領域で、 Γ_1 と Γ_2 はその境界 $\partial \Omega$ を分解したものである ($\partial \Omega = \Gamma_1 \cup \Gamma_2$, $\Gamma_1 \cap \Gamma_2 = \emptyset$ が成り立つ)。 \mathbf{n} は Γ_2 上の点における、 Ω の外向き単位法線ベクトルである。 f, g_1, g_2 は与えられた関数である。

実は、この問題は非常に筋の良い問題であり、様々な数値計算法が適用出来る。ここでは、(1) 差分法, (2) 有限要素法, (3) 基本解の方法を紹介する。

差分法、有限要素法は、偏微分方程式に対する数値解法の、二大スタンダードと言えるもので、そういう有名な方法を紹介出来るのは有意義と考えられる。基本解の方法は、微分作用素の簡単な基本解が分かっているという、Laplace 方程式の特徴を最大限に生かす方法で、Laplace 方程式の解法としては特に優れていると言える。

2 ポテンシャル問題に対する Dirichlet の原理と Poisson 方程式に対する弱解の方法

この節の狙い: 有限要素法の話は、基本的な部分を説明するだけで、セメスターの科目になってしまうようなボリュームがあり、とても短時間で紹介出来るものではないが、FreeFem++ (これはぜひ紹介したい) のプログラムをただの呪文にしてしまわないためには、弱形式にそれなりのしっかりした背景 (弱解の方法) があることを知って欲しい、と考えた。

弱解の方法は、現代の偏微分方程式論になくってはならないもので、学ぶ価値が高い、ということもある。

2.1 ポテンシャル問題に対する解の存在証明、Dirichlet の原理

G. F. B. Riemann (1826–1866) が、今では Riemann の写像定理と呼ばれる定理を発表した際 (1851 年) に、Laplace 方程式の Dirichlet 境界値問題

$$\Delta u = 0 \quad (\text{in } \Omega), \quad u = g_1 \quad (\text{on } \partial\Omega)$$

の解 u が存在すること証明する必要性が生じた。彼はそれを以下に説明する「Dirichlet の原理¹」を用いて“証明”した。

Riemann による解の存在証明のあらすじ 境界条件 $u = g_1$ (on $\partial\Omega$) を満たす u のうちで

$$J[u] = \iint_{\Omega} (u_x^2 + u_y^2) dx dy \quad (\text{この } J \text{ を Dirichlet 積分と呼ぶ})$$

を最小にするものは、 $\Delta u = 0$ を満たす。この事実を Dirichlet の原理と呼ぶ。実際、 v を $v = 0$ (on $\partial\Omega$) を満たす任意の関数とすると、

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

は $t = 0$ で最小となる (なぜならば、 $u + tv$ も同じ境界条件を満たすので、最小性の仮定から $J[u + tv] \geq J[u]$. これを f で言い換えると、 $f(t) \geq f(0)$. これは f が $t = 0$ で最小になることを意味する。)。ところで

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

であるから、 f は 2 次関数であり、 $t = 0$ で最小となるためには

$$1 \text{ 次の係数 } \iint_{\Omega} (u_x v_x + u_y v_y) dx dy = 0$$

¹なんでも、Dirichlet 先生の講義に出て来たのだとか。

が必要十分である。Green の積分公式² を適用して

$$\iint_{\Omega} (u_{xx} + u_{yy})v \, dx \, dy = 0.$$

これが任意の v について成り立つことから、 $\Delta u = u_{xx} + u_{yy} = 0$ (厳密には、変分法の基本補題という定理を適用する)。

以上の議論から、 $J[u]$ を最小にするような u を見出せば問題が解決することが分かる。 J は常に $J \geq 0$ を満たすので、 J が下に有界であることは明らかで、従って J の下限が存在する。Riemann は、(この下限は最小値であるから) 最小値を与える u が存在する、と議論したのだが、Weierstrass は「下限は最小値である」ことに疑義を示した(「数学解析」を学んだ人は、いかにも Weierstrass がツッコミそうなところと思うかも)。■

残念ながら若くして亡くなった Riemann は、Weierstrass の批判に答えることが出来なかった。この論法による完全な証明は、約 50 年後 (1900 年頃) に D. Hilbert が解決するまで持ち越された。

本当は、Dirichlet の原理は、C. F. Gauss (1777–1855) がルーツで、物理学の世界ではすでに知られていた考え方で、それを Riemann が純粋数学に応用したのだ、という見方をする人もいる。

2.2 Poisson 方程式に対する弱解の方法

Dirichlet 原理を用いる方法は、Laplace 方程式以外の多くの微分方程式に対しても拡張され、今では「弱解の方法」と呼ばれる。

弱解の方法は、数値計算とも相性がよく、そこに基礎を置く W. Ritz による **Ritz の方法** は、1909 年に発表され次第、重要な地位を占めている。この **Ritz-Galerkin 法** は有限要素法の基礎ともなっている。

2.2.1 1次元の場合の弱解の方法

簡単のため、まず 1次元版で論じる (多次元でも本質的な違いはない)。

我々の目標は次の問題の解を求めることである。

問題 (P)

$$(2.1) \quad -u''(x) = f(x) \quad (x \in (0, 1)),$$

$$(2.2) \quad u(0) = \alpha,$$

$$(2.3) \quad u'(1) = \beta$$

解を求めるために問題の言い換えをする。

この境界値問題 (P) の解は、以下に説明する問題 (W), (V) の解でもある。

まず**弱定式化** (weak formulation) した問題 (W) を述べよう。

そのために記号 X_{g_1} , X を導入する。

$$(2.4) \quad X := \{v \in H^1(0, 1) \mid v(0) = 0\}, \quad X_{g_1} := \{v \in H^1(0, 1) \mid v(0) = \alpha\}.$$

²Green の積分公式とは、 $\iint_{\Omega} \Delta u v \, dx \, dy = \int_{\partial\Omega} \frac{\partial u}{\partial n} v \, d\sigma - \iint_{\Omega} \nabla u \cdot \nabla v \, dx \, dy$ と言うもの。これは発散定理 $\int_{\Omega} \operatorname{div} \mathbf{f} \, dx \, dy = \int_{\partial\Omega} \mathbf{f} \cdot \mathbf{n} \, d\sigma$ に、 $\mathbf{f} = v \nabla u$ を代入すれば得られる。 $\operatorname{div}(v \nabla u) = \nabla u \cdot \nabla v + v \Delta u$, $\nabla u \cdot \mathbf{n} = \frac{\partial u}{\partial n}$ であることに注意する。

(ただし $H^1(0,1)$ は 1 階の **Sobolev 空間** である。おそらくこの講義を履修する多くの人が「初耳」だとも思う。一応説明しておく、超関数の意味で 1 回微分可能で、その導関数が Lebesgue 積分の意味で自乗可積分であるような関数全体の集合である— ちんぷんかんぷんかもしれないけれど、とりあえず気にしないで進もう)。次の 2 つが大事である。

(a) X と X_{g_1} はともに関数の集合である。

(b) X と X_{g_1} に属する関数 v は、境界条件としてそれぞれ $v(0) = 0, v(1) = \alpha$ を満たす。

問題 (W)

X_{g_1} に属する u で

$$(2.5) \quad \int_0^1 u'(x)v'(x)dx = \int_0^1 f(x)v(x)dx + \beta v(1) \quad (v \in X)$$

を満たすものを求めよ。

(2.5) が問題 (W) における「方程式」と呼ぶべきものであるが、これを**弱形式** (weak form) と呼ぶ。問題 (W) の解 $u \in X_{g_1}$ を元の問題の**弱解** (weak solution) と呼ぶ。

(P) の解が (W) を満たすこと u が (P) を満たすとする。任意の $v \in X$ を (2.1) にかけて、 $[0, 1]$ で積分し、部分積分すると、

$$- [u'(x)v(x)]_0^1 + \int_0^1 u'(x)v'(x) dx = \int_0^1 f(x)v(x) dx.$$

X の定義から $v(0) = 0$, また (2.3) が成り立つので、

$$[u'(x)v(x)]_0^1 = u'(1)v(1) - u'(0)v(0) = \beta v(1).$$

ゆえに

$$\int_0^1 u'(x)v'(x) dx = \int_0^1 f(x)v(x) dx + \beta v(1).$$

すなわち u は問題 (W) の解である。■

問 1. 逆に問題 (W) の解は、 C^2 級であれば、(P) の解でもあることを示せ。

次に変分問題³(variational problem) にしたものを述べる。

問題 (V)

X_{g_1} に属する u で J を最小にするもの、すなわち

$$J[u] = \inf_{w \in X_{g_1}} J[w] \quad (\text{inf は結局は min と書いても良い})$$

を満たすものを求めよ。ただし

$$J[u] := \frac{1}{2} \int_0^1 u'(x)^2 dx - \int_0^1 f(x)v(x) dx - \beta v(1).$$

J が汎関数であることに注意しよう。

³変分法を知らない人のために説明: 汎関数 (関数を変数とする関数のこと) の最小値問題を変分問題と呼ぶ。ここでは、 $J: X_{g_1} \rightarrow \mathbb{R}$ が汎関数で、 u は J の最小値を与える点となっている。

(W) と (V) は同値な問題であり、常に一意的な解を持つことが比較的容易に分かる。

問 2. (W) と (V) が同値な問題であることを示せ。(ヒント:

$$J[u + tv] - J[u] = t \left[\iint_{\Omega} (u'(x)v'(x) - f(x)v(x)) dx dy - \beta v(1) \right] + \frac{t^2}{2} \iint_{\Omega} (v_x^2 + v_y^2) dx dy$$

が成り立つことが簡単な計算で確認できる。)

問題 (V) の解 (それは (W) の解でもある) が C^2 級であることを認めると、(P), (W), (V) は互いに同値な問題ということになる。(W) \Rightarrow (P) は、Dirichlet 原理の一般化である (Laplace 方程式の Dirichlet 境界値問題の場合、この J は Dirichlet 積分 (の 1/2 倍) に他ならない。)

そこで問題 (P) を解く代わりに、(W) あるいは (V) を解くことを目指す。

通常、変分法は、変分問題を解くために、それと同値な微分方程式の問題を導き、そちらを解くことで変分問題の解を得るのが普通であるが、ここでは逆に微分方程式の問題を解くために、それを変分問題に書き換え、それを直接解く、という手順の議論をしている。これは、変分法の**直接法**と呼ばれるものになっている。

2.2.2 1次元の場合の有限要素法

$\{x_i\}_{i=0}^N$ を

$$0 = x_0 < x_1 < \dots < x_N = 1$$

を満たす数列として、

$$\tilde{X} := \{v \in C([0, 1]) \mid v \text{ は小区間 } [x_{i-1}, x_i] \text{ では 1 次多項式と一致}\},$$

$$\hat{X} := \{v \in \tilde{X} \mid v(0) = 0\}, \quad \hat{X}_{g_1} := \{v \in \tilde{X} \mid v(0) = \alpha\}$$

とおくとき、 X を \hat{X} で、 X_{g_1} を \hat{X}_{g_1} で置き換えた問題を考える。 \tilde{X} の要素を**区分 1 次多項式**と呼ぶ。

次の 2 つの問題は同値であり、常に一意的な解 \hat{u} を持つ。それを近似解として採用する。

(\hat{W})

Find $\hat{u} \in \hat{X}_{g_1}$ s.t.

$$\int_0^1 \hat{u}'(x)v'(x)dx = \int_0^1 f(x)v(x)dx + \beta v(1) \quad (v \in \hat{X}).$$

(\hat{V})

Find $\hat{u} \in \hat{X}_{g_1}$ s.t.

$$J[\hat{u}] = \min_{w \in \hat{X}_{g_1}} J[w].$$

ϕ_i を、 $\phi_i \in \tilde{X}$,

$$\tilde{\phi}_i(x_j) = \delta_{ij}$$

を満たすものとする (この条件で ϕ_i は一意的に定まる)。任意の $\hat{u} \in \hat{X}_{g_1}$ は、

$$\hat{u}(x) = \alpha \phi_0(x) + \sum_{i=1}^N a_i \phi_i(x)$$

の形に一意的に表現出来る。係数 a_1, \dots, a_N を定めれば良いが、 u が (W) (あるいは (V)) を満たすことは、 a_1, \dots, a_N がある連立1次方程式の解であることと同値であることが分かる。

実は $\{x_i\}$ が $[0, 1]$ の N 等分点であるとき、有限要素解 \hat{u} の x_i での値は、差分解 U_i と一致する。もちろん、いつもそうなるわけではない(もしそうならば、2つの方法を考える意味がない)。

有限要素法には以下の利点がある。

- 弱形式の議論を済ませてあれば、有限要素解の厳密解への収束の議論は簡単になる。
- 多次元問題の場合に、長方形領域以外でも、それほど苦勞なく解析が可能である。
- プログラムの自動生成がしやすい。

2.2.3 2次元の場合の弱解の方法

部分積分を、その一般化である Green の積分公式に置き換えるだけで、後は1次元とほぼ同様の議論が可能である。その結果、次のような弱形式が得られる。

2次元版 問題 (W)

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

$$(2.6) \quad \int_{\Omega} \text{grad } u \cdot \text{grad } v \, dx = \int_{\Omega} f v \, dx + \int_{\Gamma_2} g_2 v \, ds \quad (v \in X).$$

ここで

$$X_{g_1} := \{w \in H^1(\Omega) \mid w = g_1 \text{ on } \Gamma_1\},$$

$$X := \{w \in H^1(\Omega) \mid w = 0 \text{ on } \Gamma_1\}.$$

Green の積分公式

$$(2.7) \quad \int_{\Omega} \Delta u \, v \, dx = \int_{\partial\Omega} \frac{\partial u}{\partial \mathbf{n}} v \, ds - \int_{\Omega} \text{grad } u \cdot \text{grad } v \, dx.$$

(1次元の場合の、 $\int_0^1 u''(x)v(x) \, dx = [u'(x)v(x)]_0^1 - \int_0^1 u'(x)v'(x) \, dx$ に相当する。)

念のため:

$$\text{grad } u = \nabla u = \begin{pmatrix} u_x \\ u_y \end{pmatrix}, \quad \text{grad } u \cdot \text{grad } v = u_x v_x + u_y v_y, \quad \frac{\partial u}{\partial \mathbf{n}} = \text{grad } u \cdot \mathbf{n}.$$

\mathbf{n} は境界 $\partial\Omega$ 上の点における、**外向き単位法線ベクトル**である。

3 有限要素法と FreeFem++

3.1 FreeFem++ プログラム (その1)

有限要素法は、弱解の方法を原理とする数値計算法である。それはプログラム作成のかなりの部分を自動化出来るため、専用のソフトウェアがいくつか開発されている。

その1つである、FreeFem++⁴ は、パリ第6大学 J. L. Lions 研究所の Frédéric Hecht, Oliver Pironneau, A. Le Hyaric, 広島国際学院大学の 大塚厚二氏らが開発した、2次元, 3次元問題を有限要素法で解くための、一種の PSE (problem solving environment) である。ソースコード、マニュアル (約 400 ページ, 幸い英文)、主なプラットホーム (Windows, Mac, Linux) 向けの 実行形式パッケージが公開されている。

細かいことは、以前書いた桂田 [1] という紹介文を見てもらうことにする。

有限要素法の定番教科書の一つである菊地 [2] に載っている Poisson 方程式の例題

$$(3.1) \quad -\Delta u = f \quad \text{in } \Omega,$$

$$(3.2) \quad u = g_1 \quad \text{in } \Gamma_1,$$

$$(3.3) \quad \frac{\partial u}{\partial n} = g_2 \quad \text{in } \Gamma_2$$

(ただし、 $\Omega = (0, 1) \times (0, 1)$, $\Gamma_1 = \{0\} \times [0, 1] \cup [0, 1] \times \{0\}$, $\Gamma_2 = \{1\} \times (0, 1] \cup (0, 1] \times \{1\}$, $f = 1$, $g_1 = 0$, $g_2 = 0$) を FreeFEM++ を用いて解くとどうなるか。

次のようなプログラムで解ける。

```
Poisson2.edp
// Poisson2.edp
int Gamma1=1, Gamma2=2;
border Gamma10(t=0,1) { x=0; y=1-t; label=Gamma1; }
border Gamma11(t=0,1) { x=t; y=0; label=Gamma1; }
border Gamma20(t=0,1) { x=1; y=t; label=Gamma2; }
border Gamma21(t=0,1) { x=1-t; y=1; label=Gamma2; }
int m=10;
mesh Th = buildmesh(Gamma10(m)+Gamma11(m)+Gamma20(m)+Gamma21(m));
plot(Th, wait=1, ps="Th.eps");
savemesh(Th, "Th.msh"); // optional
fespace Vh(Th, P1);
Vh u, v;
func f=1;
func g1=0;
func g2=0;
solve Poisson(u, v) =
  int2d(Th) (dx(u)*dx(v)+dy(u)*dy(v))
  -int2d(Th) (f*v)
  -int1d(Th, Gamma2) (g2*v)
  +on(Gamma1, u=g1); // on(Gamma10, Gamma11, u=g1)
plot(u, ps="contour.eps");
```

プログラムはテキスト・エディター (現象数理学科 Mac では、Cot Editor, VS code, Atom, emacs, テキスト・エディット⁵ など) で作成し、ターミナルから、

こんなふうにして実行

```
FreeFem++ Poisson2.edp
```

とタイプして実行できる。[return] キーを打つごとに次の図に移り、最後は [esc] キーを打って終了する。

⁴<http://www.freefem.org/ff++/>

⁵テキスト・エディットで、テキスト・ファイルを入力・編集するには、若干の設定が必要である。<https://m-katsurada.sakura.ne.jp/knowhow-2015/node1.html> を見よ。

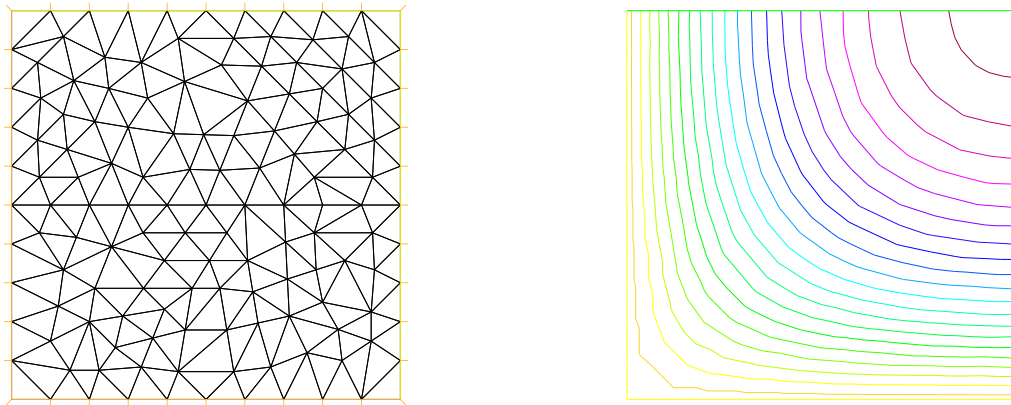


図 1: Poisson2.edp の出力 — 要素分割と解の等高線

3.2 FreeFem++ プログラム (その2) 流体の速度ポテンシャルを求める

速度ポテンシャルを求める境界値問題

$$(3.4) \quad \Delta\phi = 0 \quad (\text{in } \Omega)$$

$$(3.5) \quad \frac{\partial\phi}{\partial\mathbf{n}} = \mathbf{v} \cdot \mathbf{n} \quad (\text{on } \partial\Omega)$$

の場合は、 $\Gamma_1 = \emptyset$, $\Gamma_2 = \partial\Omega$, $f = 0$, $g_2 = \mathbf{v} \cdot \mathbf{n}$.

特に $\Omega = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 < 1\}$, $\mathbf{v} \equiv \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ のときは、 $(x, y) \in \partial\Omega$ において $\mathbf{n} = \begin{pmatrix} x \\ y \end{pmatrix}$ であるから、

$$g_2 = \mathbf{v} \cdot \mathbf{n} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = x + 2y.$$

速度ポテンシャルを求める potential2d-v0-kai.edp⁶ ← これを保存する

```
// potential2d-v0-kai.edp
// http://nalab.mind.meiji.ac.jp/~mk/complex2/potential2d-v0-kai.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
func Vn2=((x>0&&y>0) || (x<0&&y<0))*(x+2*y); // 右上と左下のみ

// 速度ポテンシャルφを求める
solve Laplace(phi,v) =
  int2d(Th)(dx(phi)*dx(v)+dy(phi)*dy(v)) -int1d(Th,Gamma)(Vn2*v);
// 平均を0にする (解の一意性がないため、時々生じる大きなズレを消す)
real mean = int2d(Th)(phi)/int2d(Th)(1);
phi = phi - mean;
// φの等高線 (等ポテンシャル線) を描く
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);
```

ターミナルでこうして入手できる

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

プログラムはテキスト・エディター (Cot Editor, テキスト・エディット⁷, Visual Studio Code, Vim, emacs など) で作成し、ターミナルから、

こんなふうにして実行

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

とタイプして実行できる。リターン・キーを打つごとに次の図に進み、最後 (ベクトル場と等ポテンシャル線を描いてお終い) は Escape キーを打って終了する。

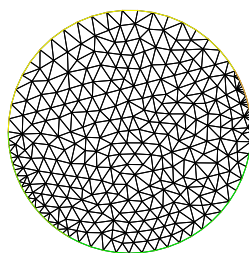


図 2: Ω の三角形分割

⁷テキストエディットで、テキスト・ファイルを入力・編集するには若干の設定が必要である。<https://m-katsurada.sakura.ne.jp/knowhow-2015/node1.html> を見よ。

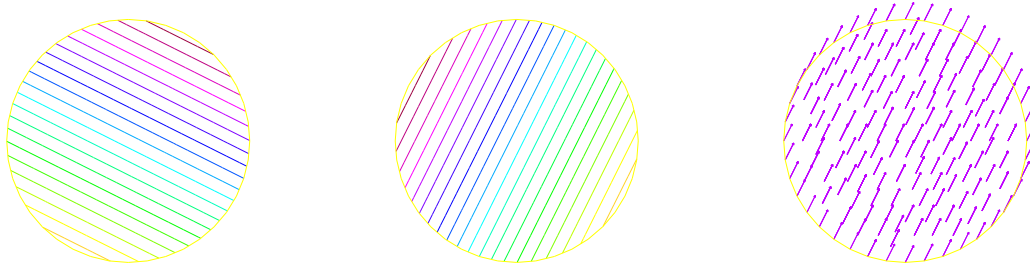


図 3: 一様流の等ポテンシャル線, 流線, ベクトル場

このプログラムは、図の PostScript データ “counterpotential.eps”, “vectorfield.eps”, “both.eps” を出力している。

(実は、上の弱形式は解の一意性がないので ($\partial\Omega$ 全体で Neumann 境界条件だから)、このプログラムは少し危ういところがある。)

3.3 工事中

(2022/8/12 追記。2023 年度の授業で用いるため。しばらく工事中。内容の正しさは保証しない。)

流れ関数をどのように求めるか。Jordan 領域であれば、境界上の流れ関数の値が

$$\psi(\mathbf{x}) = \int_{C_{\mathbf{x}}} \mathbf{v} \cdot \mathbf{n} \, ds$$

と線積分で求められるので ($C_{\mathbf{x}}$ は境界上に取った定点を始点として、境界に沿って \mathbf{x} まで達する曲線である)、Dirichlet 境界値問題を解いて求められる。

ところで、もしも境界上で \mathbf{v} が既知ならば、実は Neumann 境界データ $\frac{\partial\psi}{\partial\mathbf{n}}$ も求められる。
 $\frac{\partial\psi}{\partial\mathbf{n}} = -\mathbf{v} \cdot \mathbf{t}$ である。

速度ポテンシャルと同様にして流れ関数が求められることになる。以上は実はこれまで気づいていなかった (ちょっと恥ずかしい)。まだ忙しくて整理する時間がないが、散逸するとまずいので、とりあえず書いたメモをここに移しておく。

どちらかという「複素関数と流体力学」に書くことかな？

- 2次元または3次元の速度場 \mathbf{v} が速度ポテンシャルを持つためには、流れが渦なしであること、すなわち

$$\text{rot } \mathbf{v} = 0$$

が必要であり、またこの条件が成り立つとき、単連結領域では速度ポテンシャル ϕ が存在する。実際、任意の $\mathbf{x} \in \partial\Omega$ に対して、 \mathbf{a} から \mathbf{x} に至る曲線 $C_{\mathbf{x}}$ を取り

$$\phi(\mathbf{x}) = \int_{C_{\mathbf{x}}} \mathbf{v} \cdot d\mathbf{r}$$

が速度ポテンシャルとなる。

2次元3次元での渦なしの流れでは速度ポテンシャルが存在することが期待できる

- 速度場 \mathbf{v} の速度ポテンシャル ϕ が存在するとき、 ϕ は

$$\Delta\phi = \text{div } \mathbf{v} \quad \text{in } \Omega, \quad \frac{\partial\phi}{\partial\mathbf{n}} = \mathbf{v} \cdot \mathbf{n} \quad (\text{on } \partial\Omega)$$

を満たす。特に流れが非圧縮の場合、 ϕ は調和関数であるので、 $\partial\Omega$ 上での $\mathbf{v} \cdot \mathbf{n}$ が既知とすると、これは Laplace 方程式の Neumann 境界値問題である。

$$\int_{\partial\Omega} \frac{\partial\phi}{\partial\mathbf{n}} ds = \int_{\partial\Omega} \mathbf{v} \cdot \mathbf{n} ds = \int_{\Omega} \operatorname{div} \mathbf{v} dx = \int_{\Omega} 0 dx = 0$$

であるから解が存在し、定数差を除いて一意的に定まる。

- ψ が2次元速度場 \mathbf{v} の流れ関数であるとは

$$\psi_x = -v, \quad \psi_y = u$$

が成り立つことと定義される。存在するためには

$$\operatorname{rot} \begin{pmatrix} -v \\ u \end{pmatrix} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = \operatorname{div} \mathbf{v} = 0$$

であること、すなわち非圧縮な流れであることが必要である。この条件が満たされてい、単連結領域であれば

$$(\heartsuit) \quad \psi(\mathbf{x}) := \int_{C_{\mathbf{x}}} \begin{pmatrix} -v \\ u \end{pmatrix} \cdot d\mathbf{r} = \int_{C_{\mathbf{x}}} \mathbf{v} \cdot \mathbf{n} ds$$

で定義された ψ は流れ関数である。注意すべきは単連結でなくても、この式により ψ が well-defined になる場合があることである。例えば互いに交わらない有限個の Jordan 領域 D_1, \dots, D_r があり、 $\Omega = \mathbb{R}^2 \setminus \bigcup_{k=1}^r \overline{D}_k$ となっていて、各境界 ∂D_k ($1 \leq k \leq r$) に沿っての線積分が0:

$$\int_{\partial D_k} \mathbf{v} \cdot \mathbf{n} ds = 0$$

という条件が成り立つならば、 (\heartsuit) は流れ関数を定める。

2次元非圧縮な流れでは流れ関数が存在することが期待できる

流れ関数 ψ が存在するならば

$$\Delta\psi = -\operatorname{rot} \mathbf{v}$$

が成り立つ。特にポテンシャル流であれば渦なしであるから、 ψ は調和関数である。

- Ω が Jordan 領域である場合、 $\partial\Omega$ 上の点 \mathbf{a} を固定して、任意の $\mathbf{x} \in \partial\Omega$ に対して、 \mathbf{a} から \mathbf{x} に至る曲線 $C_{\mathbf{x}}$ を取り

$$\Psi(\mathbf{x}) := \int_{C_{\mathbf{x}}} \mathbf{v} \cdot \mathbf{n} ds$$

とおいて $\Psi: \partial\Omega \rightarrow \mathbb{R}$ が定義できる。

$$\Delta\psi = -\operatorname{rot} \mathbf{v} \quad \text{in } \Omega, \quad \psi = \Psi \quad \text{on } \partial\Omega$$

という Dirichlet 境界値問題を解けば ψ が得られる？これは渦なしの場合に限ったりするかな。

- $\partial\Omega$ 上で \mathbf{v} を知っていれば、Neumann 境界値問題の解としても求められるかな？

$$\frac{\partial\psi}{\partial\mathbf{n}} = \nabla\psi \cdot \mathbf{n} = \begin{pmatrix} -v \\ u \end{pmatrix} \cdot \mathbf{n} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} -v \\ u \end{pmatrix} \cdot \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \mathbf{n} = -\mathbf{v} \cdot \mathbf{t}.$$

$$\Delta\psi = 0 \quad \text{in } \Omega, \quad \frac{\partial\psi}{\partial\mathbf{n}} = -\mathbf{v} \cdot \mathbf{t} \quad \text{on } \partial\Omega.$$

解が存在するためには

$$\int_{\partial\Omega} \mathbf{v} \cdot \mathbf{t} \, ds = 0$$

であることが必要？ Stokes の定理？ これはポテンシャルが存在すれば満たされる。

- おっと、思いの外深く進んだ。
- 角 $\pi/2$, 反時計回りの回転を R とする:

$$R = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}.$$

これを用いると Cauchy-Riemann 方程式 $u_x = v_y, u_y = -v_x$ は

$$\nabla v = R \nabla u$$

と書ける。

$$\frac{\partial\psi}{\partial\mathbf{n}} = \nabla\psi \cdot \mathbf{n} = R \nabla\phi \cdot \mathbf{n} = \nabla\phi \cdot R^{-1}\mathbf{n} = \nabla\phi \cdot (-\mathbf{t}) = -\mathbf{v} \cdot \mathbf{t}.$$

- 2次元非圧縮流では、流れ関数 ψ が存在しそうで、その場合

$$-\Delta\psi = \text{rot } \mathbf{v} \quad \text{in } \Omega, \quad \frac{\partial\psi}{\partial\mathbf{n}} = -\mathbf{v} \cdot \mathbf{t} \quad \text{on } \partial\Omega.$$

\mathbf{v} のポテンシャルが存在する場合は、 $\text{rot } \mathbf{v} = 0$ であり、また

$$\int_{\partial\Omega} \frac{\partial\psi}{\partial\mathbf{n}} \, ds = - \int_{\partial\Omega} \mathbf{v} \cdot \mathbf{t} \, ds = - \oint_{\partial\Omega} \nabla\phi \cdot \mathbf{t} \, ds = 0$$

3.3.1 サンプル・プログラム

potential2d-v2.edp というサンプル・プログラムを用意した。これは単位円盤領域 Ω の境界 $\partial\Omega$ を

$$\Gamma_1 : (x, y) = (\cos \theta, \sin \theta) \quad (\theta \in [0, \pi/6]),$$

$$\Gamma_2 : (x, y) = (\cos \theta, \sin \theta) \quad (\theta \in [\pi/6, \pi]),$$

$$\Gamma_3 : (x, y) = (\cos \theta, \sin \theta) \quad (\theta \in [\pi, 4\pi/3]),$$

$$\Gamma_4 : (x, y) = (\cos \theta, \sin \theta) \quad (\theta \in [4\pi/3, 2\pi])$$

という4つの部分に分け、

$$v_n = \begin{cases} -2 & (\text{on } \Gamma_1) \\ 1 & (\text{on } \Gamma_3) \\ 0 & (\text{on } \Gamma_2 \cup \Gamma_4) \end{cases}$$

とした場合のシミュレーション・プログラムである。

potential2d-v2.edp

```
// potential2d-v2.edp
// http://nalab.mind.meiji.ac.jp/~mk/ana/potential2d-v2.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 Gamma1(t=0,pi/6) { x = cos(t); y = sin(t); }
border Gamma2(t=pi/6,pi) { x = cos(t); y = sin(t); }
border Gamma3(t=pi,4*pi/3) { x = cos(t); y = sin(t); }
border Gamma4(t=4*pi/3,2*pi) { x = cos(t); y = sin(t); }
int m=20;
mesh Th=buildmesh(Gamma1(m)+Gamma2(m)+Gamma3(m)+Gamma4(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, psi, v, v1, v2;
// 境界条件の設定
func Vn1=-2.0; // Gamma1 では勢いよく注水
func Vn3=1.0; // Gamma3 からゆるく排水
func Vn=(x>0)*(-2.0)+(x<0)*1.0; // xの符号でGamma1, Gamma3を区別してまとめる

// 速度ポテンシャルφを求める
solve Laplace(phi,v) =
  int2d(Th)(dx(phi)*dx(v)+dy(phi)*dy(v))
  -int1d(Th,Gamma1)(Vn1*v)-int1d(Th,Gamma3)(Vn3*v);
// -int1d(Th,Gamma1,Gamma3)(Vn*v);
// 平均を0にする (解の一意性がないため、時々生じる大きなズレを消す)
real mean=int2d(Th)(phi)/int2d(Th)(1);
phi=phi-mean;
// φの等高線 (等ポテンシャル線) を描く
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);

// 流れ関数を求める
func angle=(atan2(y,x)>=0)*atan2(y,x)+(atan2(y,x)<0)*(atan2(y,x)+2*pi);
solve LaplaceD(psi,v) =
  int2d(Th)(dx(psi)*dx(v)+dy(psi)*dy(v))
  +on(Gamma1,psi=-2*angle(x,y))
  +on(Gamma2,psi=-pi/3)
  +on(Gamma3,psi=angle(x,y)-4*pi/3)
  +on(Gamma4,psi=0);
plot(psi,ps="streamline.eps",wait=1);
plot([v1,v2],phi,psi,ps="all.eps", wait=1);
```

4 等角写像, Riemann の写像定理 (工事中)

(この文書は、2018/6/18 の講義のためのメモ 08-20180618 応用複素関数.pdf の主要部分を打ち込んだものである。次の節と適当にマージする。)

定義 4.1 (1) U は \mathbb{C} の領域で、 $f: U \rightarrow \mathbb{C}$ とするとき、 f が等角写像 (conformal mapping) であるとは、 f が正則で、かつ

$$(\forall z \in U) \quad f'(z) \neq 0$$

を満たすことをいう。

(2) U, V は \mathbb{C} の領域で、 $f: U \rightarrow V$ とするとき、 f が双正則 (biholomorphic) であるとは、 f が全単射で、 f と f^{-1} がともに正則であることをいう。

$z \in U$ で交わる滑らかな2曲線 C_1, C_2 があるとき、 C_1 と C_2 がなす角は、 C_1 と C_2 を f で写した2曲線のなす角に等しい。実際 $t = 0$ で z を通る曲線 $\varphi: (-\varepsilon, \varepsilon) \rightarrow U$ に対して、

$$\left. \frac{d}{dt} f(\varphi(t)) \right|_{t=0} = f'(\varphi(0))\varphi'(0) = f'(z)\varphi'(0)$$

であるから

$$\arg \left. \frac{d}{dt} f(\varphi(t)) \right|_{t=0} = \arg f'(z) + \arg \varphi'(0).$$

接線の方法はつねに

命題 4.2 双正則ならば等角である。

証明 $w = f(z)$ が双正則ならば、 $f^{-1}(f(z)) = z$. 両辺を微分して、 $(f^{-1})'(w)f'(z) = 1$. これから $f'(z) \neq 0$. ■

注意 4.3 上の命題の逆は真でない (等角写像は必ずしも双正則ではない) が、「 f は等角」と言っている場合、考えている領域 U で f は単射であり、 $V := f(U)$ とおくと、 $f: U \rightarrow V$ は双正則となっていることが多い。そもそも「双正則」という言葉は数学書以外ではほとんど現れないので、そういうことを思いつかないのかもしれない。■

例 4.4 (単位円盤の等角写像) $z_0 \in D_1, \varepsilon \in \mathbb{C}, |\varepsilon| = 1$ とするとき、

$$\varphi(z) = \varepsilon \frac{z - z_0}{1 - \bar{z}_0 z} \quad (z \in D_1)$$

で定まる $\varphi_1: D_1 \rightarrow D_1$ は双正則である。これは容易に確かめられるが、逆に任意の双正則関数 $\varphi: D_1 \rightarrow D_1$ に対して、

$$(\exists z_0 \in D_1)(\exists \varepsilon \in \mathbb{C} : |\varepsilon| = 1)(\forall z \in D_1) \quad \varphi(z) = \varepsilon \frac{z - z_0}{1 - \bar{z}_0 z}$$

が成り立つ (証明には、Schwarz の補題という定理を用いる。詳しいことは付録に回す)。■

例 4.5 $H := \{z \in \mathbb{C} \mid \text{Im } z > 0\}$ を上半平面と呼ぶ。

$$\varphi(z) = \frac{z - i}{z + i} \quad (z \in H)$$

とおくとき、 $|\varphi(z)| < 1$ が成り立つ。それで $\varphi: H \rightarrow D_1$ が定義出来るが、これは双正則である。この φ を Cayley 変換と呼ぶ。■

例 4.6 (Schwarz-Cristoffel の公式) (準備中) ■

数学の多くの分野で「同型写像」と呼ばれる写像があるが、双正則写像は複素関数論における同型写像と言って良いであろう。 $f: U \rightarrow V$ が双正則であるとき、複素関数論的には U と V は同じ、ということである。

これは応用上も意味があることで、例えば U (V) における渦無し非圧縮流は、 f (f^{-1}) によって V (U) における渦無し非圧縮流にうつる。 U と V の一方で流れの問題が解ければ、他方でも解けたことになる、等々。

同型写像というものを考えると、標準的なものに移すことが問題になるが、複素関数論では、次の定理が基礎的である。

記号の紹介 複素平面の原点中心、半径 1 の開円盤を D_1 で表す: $D_1 = D(0; 1) = \{w \in \mathbb{C} \mid |w| < 1\}$.

定理 4.7 (Riemann の写像定理) U は \mathbb{C} 内の単連結領域で、 \mathbb{C} とは異なるものとする。このとき、 U から単位円盤 D_1 への双正則写像 $\varphi: U \rightarrow D_1$ が存在する。

φ のことを U の等角写像、 U の写像関数と呼ぶことがある。(要するに、「 U の等角写像」とは、 U から D_1 への双正則な関数のことである。)

注意 4.8 (1) 条件 $U \neq \mathbb{C}$ は必要である。実際、 \mathbb{C} から D_1 への双正則写像 $\varphi: \mathbb{C} \rightarrow D_1$ は存在しない。もし存在したとすると、 $|\varphi(z)| < 1$ が成り立つので、 φ は有界な整関数であり、Liouville の定理によって、定数関数である。 \mathbb{C} 上の定数関数は単射ではないから、矛盾である。

(2) 単連結でない場合にも、相応の結果が成り立つ。 n 重連結領域 (直観的には、 $n-1$ 個の穴の空いた領域) から、ある標準的な n 重連結な領域への双正則写像が存在する。例えば、任意の 2 重連結領域 U に対して、 U から円環領域 $A(0; \rho, 1)$ への双正則写像が存在する。 ρ は U から定まる 1 未満の正数である (U の modulus と呼ばれる)。■

上の定理の仮定のもとで、双正則写像 φ は一意的には定まらないが、次の定理が成り立つ。

命題 4.9 (等角写像の正規化条件) U は \mathbb{C} 内の単連結領域で、 \mathbb{C} とは異なるものとする。このとき、任意の $z_0 \in \Omega$ に対して

$$(4.1) \quad \varphi(z_0) = 0 \quad \text{かつ} \quad \varphi'(z_0) > 0$$

を満たす双正則写像 $\varphi: U \rightarrow D_1$ が一意的に存在する。

条件 (5.1) を **正規化条件** と呼ぶ。

複素平面 \mathbb{C} 内の任意の単純閉曲線 (Jordan 曲線) C は、ある有界領域 U を「囲む」が、 U は単連結である。こういう U を **Jordan 領域** と呼ぶ。この場合は、任意の双正則関数 $\varphi: U \rightarrow D_1$ に対して、同相写像であるような拡張 $\tilde{\varphi}: \bar{U} \rightarrow \bar{D}_1$ が存在することが知られている (**Carathéodry の定理**)。

その場合は、ポテンシャル問題を解くことで φ が求まることを説明しよう。

$\varphi(z_0) = 0$ となる $z_0 \in \Omega$ を取ると、関数 $f(z) := \frac{\varphi(z)}{z - z_0}$ は (z_0 が除去可能特異点であるので) U で正則である。

$$f(z) = \lim_{z \rightarrow z_0} \frac{\varphi(z)}{z - z_0} = \varphi'(z_0) \neq 0$$

であることに注意すると、 $f(z) \neq 0$ 。 U は単連結であるから、 $\log \frac{\varphi(z)}{z - z_0}$ の一価正則な分枝が存在する。その実部・虚部を u, v としよう:

$$u(z) := \operatorname{Re} \log \frac{\varphi(z)}{z - z_0}, \quad v(z) := \operatorname{Im} \log \frac{\varphi(z)}{z - z_0}.$$

正則関数の実部であることから

$$(4.2) \quad \Delta u = 0 \quad (\text{in } U).$$

さらに

$$(4.3) \quad u(z) = -\log |z - z_0| \quad (z \in \partial U)$$

が成り立つ。実際、 $z \in \partial U$ のとき、 $\varphi(z) \in \partial D_1$, すなわち $|\varphi(z)| = 1$ であるから

$$u(z) = \operatorname{Re} \log \frac{\varphi(z)}{z - z_0} = \log \left| \frac{\varphi(z)}{z - z_0} \right| = \log \frac{1}{|z - z_0|} = -\log |z - z_0|.$$

すなわち

$$(4.4) \quad u(z) = -\log |z - z_0| \quad (z \in \partial U).$$

これはポテンシャル問題であり、解が一意的に存在する。こうして u が定まるが、 v は u の共役調和関数として定数差を除き定まる。特に $v(z_0) = 0$ を満たすものを選ぶ。

このとき

$$\varphi(z) := (z - z_0) \exp [u(z) + iv(z)]$$

は U から D_1 への双正則関数である。

5 領域の等角写像, Riemann の写像定理, ポテンシャル問題との関係

(等角写像の定義なども書くべきだ。2017/6/28 の講義のメモ、「応用複素関数 講義ノート」の §6 などを見て適当に整理すること。)

5.1 はじめに

関数論には、**領域の等角写像 (写像関数)** という重要な概念がある。与えられた領域 Ω に対して、円盤領域のような簡単な標準領域 D に双正則に写す正則な関数 $\Phi: \Omega \rightarrow D$ が存在するとき、 Φ を Ω の等角写像、あるいは写像関数と呼ぶ。

ここで $\Phi: \Omega \rightarrow D$ が双正則であるとは、正則であり、逆関数が存在し、逆関数 $\Phi^{-1}: D \rightarrow \Omega$ も正則であることをいう。

「 \mathbb{C} 内の単連結領域 Ω で、 \mathbb{C} とは異なるものに対して、 Ω の写像関数 Φ が存在する」という Riemann の写像定理を紹介する。

特に Ω が Jordan 閉曲線で囲まれた領域の場合に、 Φ はあるポテンシャル問題の解を使って表せることを紹介する。ポテンシャル問題の数値解法と合わせると、写像関数の数値計算法が得られる。

5.2 Riemann の写像定理

(講義でそれなりに説明するつもり…その要点をこちらに写したい)

U, V を \mathbb{C} の領域とする。 $\varphi: U \rightarrow V$ が**双正則**であるとは、 φ が正則でかつ全単射で、 φ^{-1} も正則であることをいう。

Ω を \mathbb{C} の単連結領域で、 \mathbb{C} とは異なるものとするとき、双正則写像

$$\varphi: \Omega \rightarrow D_1 = D(0; 1)$$

が存在する (**Riemann の写像定理**, 1851 年)。

この φ のことを領域 Ω の**等角写像**, あるいは**写像関数**と呼ぶ⁸。

問題となる領域の等角写像はしばしば役に立つ。そのため、その計算方法は重要視され、古くから研究されてきた。多角形領域の場合の **Schwarz-Christoffel mapping** などは、時間の関係で、「複素関数」、「応用複素関数」ではスルーしているが、複素関数論の定番のメニューと言える。

5.3 単連結領域の場合の等角写像の正規化条件

Ω を \mathbb{C} の単連結領域で、 \mathbb{C} とは異なるものとする。Riemann の写像定理により、 Ω の等角写像

$$\varphi: \Omega \rightarrow D_1 = D(0; 1)$$

が存在する。この写像は一意的には定まらないが、次が成り立つ。

命題 5.1 Ω を \mathbb{C} とは異なる \mathbb{C} の単連結領域で、 z_0 を Ω の任意の点とする。このとき

$$(5.1) \quad \varphi(z_0) = 0, \quad \varphi'(z_0) > 0$$

という条件を満たす双正則写像 $\varphi: \Omega \rightarrow D_1$ は、存在すれば一意的である。

証明 φ_1, φ_2 がその条件を満たすとする。 $\psi := \varphi_2 \circ \varphi_1^{-1}$ とおくと、 $\psi: D_1 \rightarrow D_1$ は双正則写像である。そして、

$$\psi(0) = \varphi_2(\varphi_1^{-1}(0)) = \varphi_2(z_0) = 0$$

であるから、

$$(\exists \varepsilon \in \mathbb{C} : |\varepsilon| = 1)(\forall z \in D_1) \quad \psi(z) = \varepsilon \frac{z - 0}{1 - \bar{0}z} = \varepsilon z.$$

ところが、

$$\varepsilon = \psi'(0) = \varphi_2'(z_0) \frac{1}{\varphi_1'(z_0)} > 0.$$

であるから、 $\varepsilon = 1$ 。ゆえに $\psi(z) = z$ 。ゆえに $\varphi_2 = \varphi_1$ 。■

5.4 Jordan 領域の等角写像を求めるアルゴリズム

典型的な単連結領域に **Jordan 領域**がある。

Jordan 曲線定理

\mathbb{C} 内の Jordan 閉曲線 C に対して、 C の“囲む”領域 Ω が定まり、 Ω は有界かつ単連結で、その境界は C の像に等しい。

この定理で存在が保証される領域を、 C の定める Jordan 領域と呼ぶ。Jordan 領域 Ω は単連結領域であるから、Riemann の写像定理によって、 Ω の等角写像 φ が存在するが、以下に示すように、 φ は、あるポテンシャル問題を解くことによって求めることができる。

⁸関数論において等角写像とは、いたるところ $\varphi' \neq 0$ を満たす正則関数のことを指す。一対一の等角写像 $\varphi: U \rightarrow \mathbb{C}$ があるとき、終域を $V := \varphi(U)$ で置き換えた、 $\tilde{\varphi}: z \mapsto \varphi(z) \in V$ は双正則である ($\tilde{\varphi}'(z) \neq 0$ が簡単に導かれる)。しばしば、等角写像という言葉、双正則な関数の意味に使うことがある。d

このとき、 $\varphi: \Omega \rightarrow D_1$ が双正則写像で、(5.1) を満たすとしよう。 Ω の閉包から閉円盤への同相写像 $\tilde{\varphi}: \bar{\Omega} \rightarrow \bar{D}_1$ に拡張できる (Carathéodory の定理)。以下 $\tilde{\varphi}$ のことも φ と書くことにする。

関数 $\frac{\varphi(z)}{z - z_0}$ は Ω で正則であり、かつ 0 という値を取らない。 Ω が単連結であるから、 $\log \frac{\varphi(z)}{z - z_0}$ の一価正則な分枝が取れる。その実部、虚部をそれぞれ u, v とおく：

$$u(z) + iv(z) := \log \frac{\varphi(z)}{z - z_0}.$$

両辺の実部を取ると、

$$u(z) = \log \frac{|\varphi(z)|}{|z - z_0|}.$$

$z \in \partial\Omega$ のとき、 $\varphi(z) \in \partial D_1$, すなわち $|\varphi(z)| = 1$ であるから、

$$(5.2) \quad u(z) = -\log |z - z_0| \quad (z \in \partial\Omega).$$

一方

$$(5.3) \quad \Delta u(z) = 0 \quad (z \in \Omega).$$

(5.3), (5.2) は、Laplace 方程式の Dirichlet 境界値問題である。これを解いて u を求め、 v を u の共役調和関数で、 $v(z_0) = 0$ を満たすものとする、 φ は次のように求まる。

$$\varphi(z) = (z - z_0) \exp [u(z) + iv(z)].$$

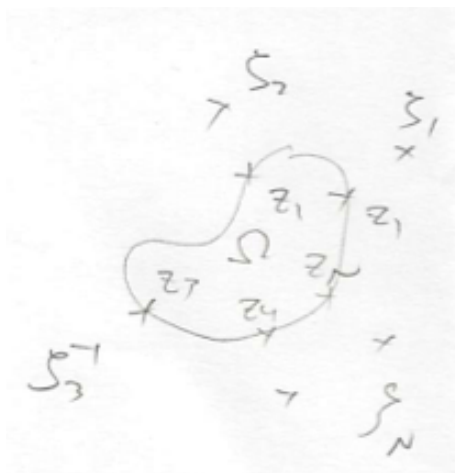


図 4:

5.5 (細かい話) 多重連結領域の場合

Ω が単連結でないときは？ D_1 は単連結であるから、 $\varphi: \Omega \rightarrow D_1$ は双正則ではありえない。

\mathbb{C} の領域 Ω は、 $\hat{\mathbb{C}} \setminus \Omega$ が $n - 1$ 個の連結成分からなるとき、 n **重連結領域** であるという。例えば $\mathbb{C} \setminus \{0\}$ や $\mathbb{C} \setminus \bar{D}_1$ は**二重連結**、 $\mathbb{C} \setminus \{0, 1\}$ は**三重連結**である。(単連結は、1連結に相当して、補集合 $\hat{\mathbb{C}} \setminus \Omega$ は1個の連結成分からなる—つまり連結である。)

Ω が2重連結領域である場合、1より小さい r と、 Ω から円環領域 $A(0; r, 1) = \{w \in \mathbb{C} \mid r < |w| < 1\}$ の上への双正則写像 $\varphi: \Omega \rightarrow A(0; r, 1)$ が存在する。

三重連結以上の場合も、調べられている。

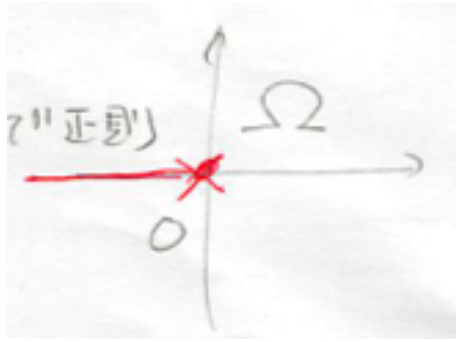


图 5:

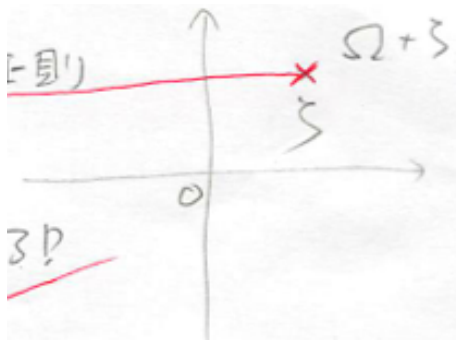


图 6:

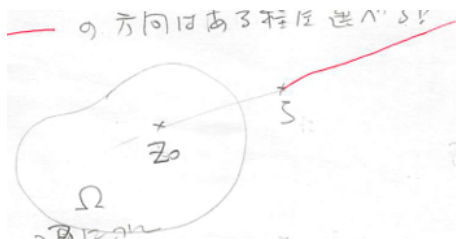


图 7:

6 Laplace 方程式に対する基本解の方法

差分法、有限要素法は、偏微分方程式に対する定番の数値解法であるが、ポテンシャル問題の場合は、その特性を生かした優秀な解法がいくつかある。ここでは**基本解の方法**を紹介する。

6.1 準備

6.1.1 基本解とは

3次元の場合 $E(\mathbf{x}) = \frac{1}{4\pi|\mathbf{x}|}$, 2次元の場合 $E(\mathbf{x}) = -\frac{1}{2\pi} \log|\mathbf{x}|$ を $-\Delta$ の**基本解** (the fundamental solution) と呼ぶ。

E は原点以外では調和関数であることは簡単な計算で分かる:

$$\Delta E(\mathbf{x}) = 0 \quad (x \in \mathbb{R}^n \setminus \{0\}).$$

さらに、超関数の言葉で言うと (原点まで込めて)

$$(6.1) \quad -\Delta E = \delta$$

が成り立つ。ここで δ は Dirac のデルタ関数である。

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

6.1.2 Poisson 方程式の特解

$$U(\mathbf{x}) := \int_{\Omega} E(\mathbf{x} - \mathbf{y}) f(\mathbf{y}) d\mathbf{y}$$

とおくと、

$$-\Delta U = f$$

が成り立つ (証明は結構難しい)。つまり U は Poisson 方程式の特解である。

$v := u - U$ とおくと、 $\Delta v = 0$ が成り立つので、 $f = 0$ の場合の問題が一般に解ければ良いことになる。(実際には U を計算することは難しいことが多く、数値計算向きではないかもしれない。)

6.1.3 Green の積分公式 (続き)

Green の第2積分公式

$$\int_{\partial\Omega} \left(v \frac{\partial u}{\partial n} - u \frac{\partial v}{\partial n} \right) d\sigma = \int_{\Omega} (v \Delta u - u \Delta v) d\mathbf{x}$$

を利用して、次の Green の第3積分公式を得る (証明の詳細は略するが、関数論の Cauchy の積分公式の証明のように、 x を中心とする球を除いた領域で Green の公式を適用してから、球の半径を 0 に近づける。詳しくは桂田 [3] の §3.5 を見よ。)

$$-\int_{\Omega} E(\mathbf{x}-\mathbf{y}) \Delta u(\mathbf{y}) d\mathbf{y} + \int_{\partial\Omega} E(\mathbf{x}-\mathbf{y}) \frac{\partial u}{\partial n_y}(\mathbf{y}) d\sigma_y - \int_{\partial\Omega} u(\mathbf{y}) \frac{\partial}{\partial n_y} E(\mathbf{x}-\mathbf{y}) d\sigma_y = \begin{cases} u(\mathbf{x}) & (x \in \Omega) \\ \frac{1}{2}u(\mathbf{x}) & (x \in \partial\Omega) \\ 0 & (x \in \mathbb{R}^m \setminus \bar{\Omega}). \end{cases}$$

($x \in \partial\Omega$ の場合は左辺第3項は主値積分である。)

(a) u が $\Delta u = 0$ を満たすならば、 $x \in \Omega$ に対して、

$$u(x) = \int_{\partial\Omega} E(x-y) \frac{\partial u}{\partial n_y}(y) d\sigma_y - \int_{\partial\Omega} u(y) \frac{\partial}{\partial n_y} E(x-y) d\sigma_y.$$

すなわち、 u が調和関数であるとき、 u , $\partial u / \partial n$ の $\partial\Omega$ での値が分かれば、 $u(x)$ の値がこの式で求まることになる (正則関数の Cauchy の積分公式に似ていて、使いでのある公式)。境界条件から半分は分かっているので、もう半分求めれば良いことになる。

以下は細かい話になるが: 例えば $\Gamma_N = \emptyset$ のとき、

$$\frac{1}{2}g_1(x) = \int_{\partial\Omega} E(x-y) \frac{\partial u}{\partial n_y}(y) d\sigma_y - \int_{\partial\Omega} g_1(y) \frac{\partial}{\partial n_y} E(x-y) d\sigma_y.$$

これから $\partial\Omega$ 上で $\partial u / \partial n$ を求めることが出来る。

(b) u が $\partial\Omega$ の近傍で 0 ならば、 $x \in \Omega$ に対して、

$$- \int_{\Omega} E(x-y) \Delta u(y) d\mathbf{y} = u(x).$$

この事実を超関数解釈すると $-\Delta E(x-\cdot) = \delta(\cdot-x)$ となる。

6.2 基本解の方法

簡単のため、 $\Gamma_N = \emptyset$ の場合の Laplace 方程式の境界値問題

$$\begin{aligned} \Delta u(x) &= 0 \quad (x \in \Omega), \\ u(x) &= g_1(x) \quad (x \in \Gamma_D = \partial\Omega) \end{aligned}$$

を取り上げる。

Ω が滑らかな境界を持つ有界領域の場合に、この境界値問題が一意的な解を持つことは知られている。

Ω が (円盤とか、長方形とか) 特別な形をしている場合に、解 u を求める公式はいくつか知られているが、ここでは多くの場合に使える数値解法を紹介する。一見素朴であるが、多くの場合に良好な近似解を得ることが出来る。

Ω の外部に Ω を「囲むように」点 $\{y_k\}_{k=1}^N$ を取り、

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

とおく (もちろん、 E は Laplacian の基本解である)。ここで Q_1, Q_2, \dots, Q_N は未定係数である。これらが何であっても

$$\Delta u^{(N)}(x) = 0 \quad (x \in \mathbb{R}^n \setminus \{y_1, y_2, \dots, y_N\})$$

が成り立つ。もし $u^{(N)}(x) = g_1(x)$ ($x \in \partial\Omega$) が成り立てば $u^{(N)}$ は解である。さすがにそんな都合の良いことはめったにおこらないが、多くの場合、境界 $\partial\Omega$ 上で選んだ点 x_1, \dots, x_N に対して

$$u^{(N)}(x_j) = g_1(x_j) \quad (j = 1, 2, \dots, N)$$

を満たすように Q_k を決めることが出来る。このとき

$$u^{(N)} \doteq u$$

が成り立つことが期待できる。この近似解法を、the method of fundamental solutions (**基本解の方法**, fundamental solution method), あるいは代用電荷法 (**charge simulation method**) と呼ぶ⁹。

次のような利点がある。

- しばしば**誤差の指数関数的減少**

$$(\exists C \in \mathbb{R})(\exists \rho \in (0, 1))(\forall N \in \mathbb{N}) \quad \|u - u^{(N)}\| \leq C\rho^N$$

が成り立つ(この場合は、差分法や有限要素法と比較して、高精度の解が少ない計算量で得られる)。

- $u^{(N)}$ 自身が調和関数であり、特に

$$\text{grad } u^{(N)}(x) = \begin{cases} -\frac{1}{2\pi} \sum_{j=1}^N Q_j \frac{x - y_j}{|x - y_j|^2} & (2 \text{次元の場合}) \\ -\frac{1}{4\pi} \sum_{j=1}^N Q_j \frac{x - y_j}{|x - y_j|^3} & (3 \text{次元の場合}) \end{cases}$$

のように grad が直接計算できる(ポテンシャルの grad が必要な場合が多いので、非常に便利である)。

基本解の方法以外に、基本解を利用する方法として、**境界要素法** (boundary element method, BEM) がある。

6.3 近似等角写像を求めるための天野の方法

天野要は、§5.4 で述べた等角写像の求め方と、基本解の方法を組み合わせた、効率的なアルゴリズムを提唱した ([4])。それを解説する。

§5.4 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」 $\mathbb{C} \setminus \bar{\Omega}$ から選び、

$$(6.2) \quad u^{(N)}(z) := \sum_{k=1}^N Q_k \log |z - \zeta_k|$$

とおく。ここで Q_k は未知の実定数である ($k = 1, \dots, N$)。 $\{z_j\}_{j=1}^N$ を $\partial\Omega$ から選び、collocation equation

$$(6.3) \quad u^{(N)}(z_j) = -\log |z_j - z_0| \quad (j = 1, \dots, N)$$

で $\{Q_k\}$ を定める。

天下りになるが、

$$(6.4) \quad f^{(N)}(z) := Q_0 + \sum_{k=1}^N Q_k \text{Log} \frac{z - \zeta_k}{z_0 - \zeta_k}, \quad Q_0 := \sum_{k=1}^N Q_k \log |z_0 - \zeta_k|$$

⁹その理由は、 y_1, \dots, y_N それぞれに電荷量 Q_1, \dots, Q_N の電荷を置いたとき、それら電荷の作る電場のポテンシャルが $u^{(N)}$ である、という事実に基づく。

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)。

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

であり、

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

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

天野のアルゴリズム

- (1) (6.2), (6.3) で $\{Q_k\}$ を定める。
- (2) (6.4) 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 $\varphi: \Omega \rightarrow D_1$ の近似として採用する。

6.4 Python による Jordan 領域の等角写像の計算プログラム

$\Omega = D_1$ の等角写像を考えよう。 Ω と D_1 が等しいので、 Ω から D_1 への双正則写像として、恒等写像 $\text{id}(z) = z$ が取れるのは自明である。

正規化条件によっては、それ以外のものが得られる。実際、 $z_0 \in \Omega$ として、 Ω の写像関数 $\varphi: \Omega \rightarrow D_1$ のうち、正規化条件

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

を満たすものは、実は 1 次分数変換

$$(6.5) \quad \varphi(z) = \frac{z - z_0}{1 - \bar{z}_0 z}$$

であることが知られている。 $z_0 = 0$ のとき、 $\varphi = \text{id}$ となる訳である。

ここでは、前項で説明した天野の方法で Ω の近似等角写像を求めて、真の等角写像 (6.5) と比べてみることにする。

(答えの分かっている問題を解くことによって、アルゴリズムやプログラムの正しさをチェックする、というしばしば使われるテクニック。)

天野の方法では、 $\{\zeta_k\}_{k=1}^N, \{z_j\}_{j=1}^N$ を適切に選ぶ必要があるが、今の場合は Ω は円盤領域であるから、 $\{\zeta_k\}_{k=1}^N$ として

$$\zeta_k = R \exp \frac{2\pi i k}{N} \quad (k = 1, \dots, N) \quad (R \text{ は } 1 \text{ より大きい数}),$$

$\{z_j\}_{j=1}^N$ として

$$z_j = \exp \frac{2\pi i j}{N} \quad (j = 1, \dots, N).$$

を採用するのが自然であろう。 R として、とりあえず $R = 2$ で計算してみる。

```

conformalmap.py
import numpy as np
import matplotlib.pyplot as plt

def amanomap(z,z0,n,Q0,Q,zeta):
    s=Q0
    for k in range(n):
        s += Q[k]*np.log((z-zeta[k])/(z0-zeta[k]))
    return (z-z0) * np.exp(s)

z0=0.6
R=2
N=80
dt=2*np.pi/N
I=1j
thetas=np.linspace(0.0,2.0*np.pi,N,endpoint=False)
zetas=R*np.exp(I*thetas)
zs=np.exp(I*thetas)
a=np.empty((N,N))
for j in (range(N)):
    for k in (range(N)):
        a[j,k]=np.log(abs(zs[j]-zetas[k]))
b=-np.log(abs(zs-z0))
# Qk (k=1,...N) と Q0 を求める
Q=np.linalg.solve(a,b)
Q0=0
for k in (range(N)):
    Q0 += Q[k]*np.log(abs(z0-zetas[k]))

# 計算の方はまあまあだけど、可視化はどうするものかな
plt.clf()
fig=plt.figure()
fig.set_size_inches(5, 5)
plt.axes().set_aspect('equal')
m=20
nr=m
rs=np.linspace(0.0,1.0,nr+1)
nt=4*nr
ts=np.linspace(0.0,2.0*np.pi,nt+1)
#zs=np.empty((nr+1,nt+1),dtype=np.complex128)
ws=np.empty((nr+1,nt+1),dtype=np.complex128)
for i in range(nr+1):
    for j in range(nt+1):
        zs=rs[i]*np.exp(1j*ts[j])
        ws[i,j]=amanomap(zs,z0,N,Q0,Q,zetas)
for i in range(nr+1):
    plt.plot(ws[i,:].real,ws[i,:].imag)
for j in range(nt+1):
    plt.plot(ws[:,j].real,ws[:,j].imag)

plt.show()

```

```

conformalmap-v2.py
# conformalmap-v2.py

import numpy as np
import matplotlib.pyplot as plt

def amanomap(z,z0,n,Q0,Q,zeta):
    s = Q0+np.dot(Q,np.log(np.divide(z-zeta,z0-zeta)))
    return (z-z0) * np.exp(s)

z0=0.6
R=2
N=80
I=1j
thetas=np.linspace(0.0,2.0*np.pi,N,endpoint=False)
zetas=R*np.exp(I*thetas)
zs=np.exp(I*thetas)
a=np.empty((N,N))
for j in (range(N)):
    for k in (range(N)):
        a[j,k]=np.log(abs(zs[j]-zetas[k]))
b=-np.log(abs(zs-z0))
# Qk (k=1,...N) と Q0 を求める
Q=np.linalg.solve(a,b)
Q0=np.dot(Q,np.log(abs(z0-zetas)))

# 計算の方はまあまあだけど、可視化はどうするものかな
plt.clf()
fig=plt.figure()
fig.set_size_inches(5, 5)
plt.axes().set_aspect('equal')
m=20
nr=m
rs=np.linspace(0.0,1.0,nr+1)
nt=4*nr
ts=np.linspace(0.0,2.0*np.pi,nt+1)
#zs=np.empty((nr+1,nt+1),dtype=np.complex128)
ws=np.empty((nr+1,nt+1),dtype=np.complex128)
for i in range(nr+1):
    for j in range(nt+1):
        zs=rs[i]*np.exp(1j*ts[j])
        ws[i,j]=amanomap(zs,z0,N,Q0,Q,zetas)
for i in range(nr+1):
    plt.plot(ws[i,:].real,ws[i,:].imag)
for j in range(nt+1):
    plt.plot(ws[:,j].real,ws[:,j].imag)

plt.show()

```

A ポテンシャル問題に対する Poincaré-Perron の方法

(準備中)

順序のからむ解法は、ひょつとすると精度保証付き数値計算の役に立つかもしれない。

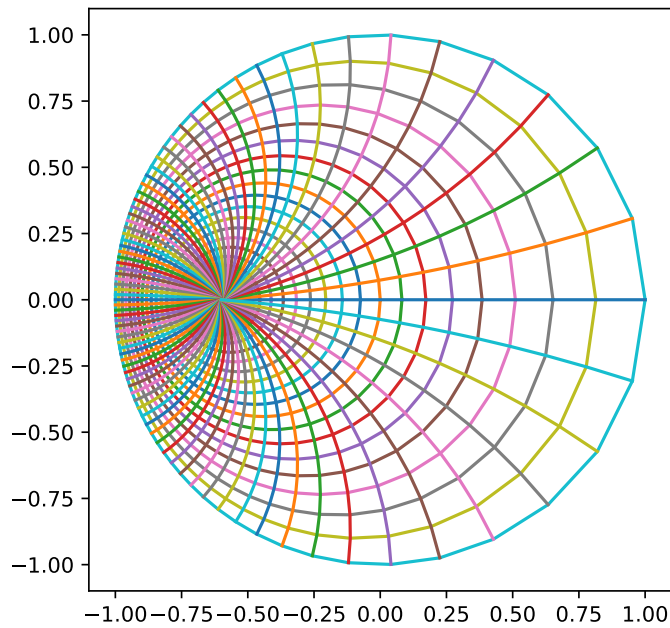


図 8: 天野の方法による近似等角写像

B Poisson 方程式に対する差分法

(差分法はポピュラーな方法であるが、ポテンシャル問題に関してはそれほど便利でもないので、付録に入れておくに止める。)

有名な差分法を紹介しよう。簡単のために領域は区間 (1次元では線分、2次元では長方形) とする、

B.1 基本的な考え方

差分法 (finite difference method, FDM) では、次の2つの考え方をを用いる。

- 微分方程式に含まれる導関数を差分商で置き換えた差分方程式の解を近似解に採用する。
- 領域を格子に区切って、格子点上の値を求めることを目標にする。

常微分方程式の初期値問題に対する Euler 法, Runge-Kutta 法などを学んだことがあれば、理解しやすいであろう。

$$(B.1) \quad f'(x) = \frac{f(x+h) - f(x)}{h} + O(h) \quad (h \rightarrow 0).$$

$$(B.2) \quad f'(x) = \frac{f(x) - f(x-h)}{h} + O(h) \quad (h \rightarrow 0).$$

$$(B.3) \quad f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2) \quad (h \rightarrow 0).$$

$$(B.4) \quad f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2) \quad (h \rightarrow 0).$$

$$(B.5) \quad f'''(x) = \frac{f(x+2h) - 2f(x+h) + 2f(x-h) - f(x-2h)}{2h^3} + O(h^2) \quad (h \rightarrow 0).$$

$$(B.6) \quad f''''(x) = \frac{f(x+2h) - 4f(x+h) + 6f(x,y) - 4f(x-h) + f(x-2h)}{h^4} + O(h^2) \quad (h \rightarrow 0).$$

多変数関数の偏導関数はこれらを適当に組み合わせて近似する。例えば

$$\Delta u(x, y) = \frac{u(x+h_x, y) - 2u(x, y) + u(x-h_x, y)}{h_x^2} + \frac{u(x, y+h_y) - 2u(x, y) + u(x, y-h_y)}{h_y^2} + O(h_x^2 + h_y^2).$$

B.2 1次元でイントロ

$\Omega = (0, 1)$, $\Gamma_1 = \{0\}$, $\Gamma_2 = \{1\}$ の場合、(1.1), (5.2), (1.3) は、次のように書き換えられる。

$$(B.7) \quad -u''(x) = f(x) \quad (x \in (0, 1)),$$

$$(B.8) \quad u(0) = \alpha,$$

$$(B.9) \quad u'(1) = \beta.$$

B.2.1 差分方程式

$$h = \Delta x = \frac{1}{N}, \quad x_i = ih \quad (i = 0, 1, \dots, N, N+1),$$

$$(B.10) \quad -\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} = f(x_i) \quad (i = 1, 2, \dots, N),$$

$$(B.11) \quad u_0 = \alpha,$$

$$(B.12) \quad \frac{u_{N+1} - u_{N-1}}{2h} = \beta$$

結果として次の連立1次方程式が得られる:

$$\begin{pmatrix} 2 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ 0 & & & -2 & 2 \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_{N-1} \\ U_N \end{pmatrix} = h^2 \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \\ f_N \end{pmatrix} + \begin{pmatrix} \alpha \\ 0 \\ \vdots \\ 0 \\ 2\beta h \end{pmatrix}.$$

あるいは係数行列を対称化して (両辺の最後の成分に 1/2 をかけて)

$$\begin{pmatrix} 2 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ 0 & & & -1 & 1 \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_{N-1} \\ U_N \end{pmatrix} = h^2 \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \\ \frac{1}{2}f_N \end{pmatrix} + \begin{pmatrix} \alpha \\ 0 \\ \vdots \\ 0 \\ \beta h \end{pmatrix}.$$

この係数行列は既約優対角行列なので、正則である。ゆえに連立1次方程式は一意的な解を持つ。また、この係数行列は、対角線とその両隣り以外は0となっている、いわゆる三重対角行列であり (連立1次方程式自体は三項方程式と呼ばれる)、Gaussの消去法で効率的に ($O(N)$ の計算量で) 解くことができる。

B.2.2 C 言語のプログラム例

```
/*
 * poisson1d.c --- 1次元 Poisson 方程式
 *
 *   -  $u''(x)=f(x)$       ( $0<x<1$ )
 *   -  $u(0)=\alpha$ 
 *   -  $u'(1)=\beta$ 
 *
 * を差分法で解くプログラム。cglsc コマンドでコンパイル&実行出来る。
 *
 *  $u(x)=(x-1/3)^2$  の場合、 $\alpha=1/9$ ,  $\beta=4/3$  --- 2次関数なので厳密に解ける
 */

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

#define MAXN (100);

// 三重対角行列の Gauss の消去法による LU 分解
void trilu1(int, double *, double *, double *);
// LU 分解を用いて三項方程式を解く
void trisol1(int, double *, double *, double *, double *);

// Poisson 方程式の右辺
double f(double x)
{
    return - 2.0;
}

// 厳密解
double solution(double x)
{
    return (x - 1.0 / 3) * (x - 1.0 / 3);
}

int main(void)
{
    double alpha, beta;
    int i, n;
    double *x, *al, *ad, *au, *u;
    double h, h2;
    alpha = 1.0 / 9.0;
    beta = 4.0 / 3.0;

    printf("n="); scanf("%d", &n);

    x = malloc(sizeof(double) * (n + 1));
    al = malloc(sizeof(double) * (n + 1));
    ad = malloc(sizeof(double) * (n + 1));
    au = malloc(sizeof(double) * (n + 1));
    u = malloc(sizeof(double) * (n + 1));
    if (ad == NULL || al == NULL || au == NULL || u == NULL) {
        fprintf(stderr, "メモリの割当てに失敗しました。\\n");
        exit(1);
    }
    h = 1.0 / n;
    for (i = 0; i <= n; i++)
        x[i] = i * h;
    h2 = h * h;
    for (i = 1; i <= n; i++) {
```



```

    ad[i] = 2;
    au[i] = al[i] = -1;
    u[i] = h2 * f(i * h);
}
/* 第N成分は 1/2 倍する */
ad[n] = 1;
u[n] *= 0.5;
/* 非同次境界条件 */
u[1] += alpha;
u[n] += beta * h;
/* 連立1次方程式を解く */
trilu1(n, al, ad, au);
trisol1(n, al, ad, au, u);

/* グラフの準備 */
u[0] = alpha;
g_init("POISSON1D", 170.0, 170.0);
g_device(G_BOTH);
g_def_scale(0,
            -0.2, 1.2, -0.2, 1.2,
            10.0, 10.0, 150.0, 150.0);
g_sel_scale(0);
g_move(-0.2, 0.0); g_plot(1.2, 0.0);
g_move(0.0, -0.2); g_plot(0.0, 1.2);

/* 厳密解を描く */
g_move(x[0], solution(x[0]));
for (i = 1; i <= n; i++)
    g_plot(x[i], solution(x[i]));
printf("クリックして下さい\n");
g_sleep(-1.0);
/* 差分解を赤く描く */
g_line_color(G_RED);
g_move(x[0], u[0]);
for (i = 1; i <= n; i++)
    g_plot(x[i], u[i]);

printf("x=1 での厳密解 %f, 差分解 %f, 誤差 %e\n",
       solution(1.0), u[n], fabs(solution(1.0) - u[n]));
g_sleep(-1.0);
g_term();

return 0;
}

/* 3項方程式（係数行列が三重対角である連立1次方程式のこと）Ax=b を解く
*
* 入力
*   n: 未知数の個数
*   al,ad,au: 連立1次方程式の係数行列
*   (al: 対角線の下側 i.e. 下三角部分 (lower part)
*   ad: 対角線 i.e. 対角部分 (diagonal part)
*   au: 対角線の上側 i.e. 上三角部分 (upper part)
*   つまり
*
*
*   ad[1] au[1]  0  ..... 0
*   al[2] ad[2] au[2]  0  ..... 0
*   0    al[3] ad[3] au[3]  0  ..... 0
*
*   .....
*   al[n-1] ad[n-1] au[n-1]
*   0      al[n]   ad[n]
*
*   al[i] = A_{i,i-1}, ad[i] = A_{i,i}, au[i] = A_{i,i+1},

```

```

*      al[1], au[n] は意味がない)
*
*      b: 連立 1 次方程式の右辺の既知ベクトル
*      (添字は 1 から。i.e. b[1],b[2],...,b[n] にデータが入っている。)
* 出力
*      al,ad,au: 入力した係数行列を LU 分解したもの
*      b: 連立 1 次方程式の解
* 能書き
*      一度 call すると係数行列を LU 分解したものが返されるので、
*      以後は同じ係数行列に関する連立 1 次方程式を解くために、
*      関数 trisol1() が使える。
* 注意
*      Gauss の消去法を用いているが、ピボットの選択等はしていな
*      いので、ピボットの選択をしていないので、係数行列が正定値である
*      などの適切な条件がない場合は結果が保証できない。
*/

void trid1(int n, double *al, double *ad, double *au, double *b)
{
    trilu1(n,al,ad,au);
    trisol1(n,al,ad,au,b);
}

/* 三重対角行列の LU 分解 (pivoting なし) */
void trilu1(int n, double *al, double *ad, double *au)
{
    int i;
    /* 前進消去 (forward elimination) */
    for (i = 1; i < n; i++) {
        al[i + 1] /= ad[i];
        ad[i + 1] -= au[i] * al[i + 1];
    }
}

/* LU 分解済みの三重対角行列を係数に持つ 3 項方程式を解く */
void trisol1(int n, double *al, double *ad, double *au, double *b)
{
    int i;
    /* 前進消去 (forward elimination) */
    for (i = 1; i < n; i++) b[i + 1] -= b[i] * al[i + 1];
    /* 後退代入 (backward substitution) */
    b[n] /= ad[n];
    for (i = n - 1; i >= 1; i--) b[i] = (b[i] - au[i] * b[i + 1]) / ad[i];
}

```

B.3 2次元の場合 (結果だけ紹介)

長方形領域 $\Omega = (0, W) \times (0, H)$ の場合は、1次元と大筋で同様にして扱える。簡単のため Dirichlet 境界値問題、すなわち

$$\Gamma_1 = \partial\Omega, \quad \Gamma_2 = \emptyset \quad (\text{Neumann 境界が空集合})$$

の場合に説明する。この場合は、(1.1), (5.2), (1.3) は、

$$-\Delta u = f \quad (\text{in } \Omega), \quad u = g_1 \quad (\text{on } \partial\Omega)$$

と書ける。

B.3.1 差分方程式

$N_x, N_y \in \mathbb{N}$ に対して、

$$h_x = \Delta x := \frac{W}{N_x}, \quad h_y = \Delta y := \frac{H}{N_y},$$

$$x_i = i\Delta x \quad (0 \leq i \leq N_x), \quad y_j = j\Delta y \quad (0 \leq j \leq N_y)$$

によって格子点 (x_i, y_j) を定める。

領域内部にある格子点のインデックスの集合を

$$\omega := \{(i, j) \mid 1 \leq i \leq m, 1 \leq j \leq n\}, \quad m := N_x - 1, \quad n := N_y - 1,$$

領域の境界上にある格子点のインデックスの集合を

$$\gamma := \{(i, 0) \mid 0 \leq i \leq N_x\} \cup \{(i, N_y) \mid 0 \leq i \leq N_x\} \cup \{(0, j) \mid 0 \leq j \leq N_y\} \cup \{(N_x, j) \mid 0 \leq j \leq N_y\}$$

とおく。個数を調べておこう。

$$\#\omega = (N_x - 1)(N_y - 1) = mn, \quad \#\gamma = 2(N_x + N_y), \quad \#(\omega \cup \gamma) = (N_x + 1)(N_y + 1).$$

$$(B.13) \quad -\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{\Delta x^2} - \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{\Delta y^2} = f(x_i, y_j) \quad ((i, j) \in \omega),$$

$$(B.14) \quad U_{i,j} = g_1(x_i, y_j) \quad ((i, j) \in \gamma),$$

という差分方程式の解 U_{ij} を $u(x_i, y_j)$ の近似値とするのが良いと期待できる。

これは $(N_x + 1)(N_y + 1)$ 個の未知数 $U_{i,j}$ ($(i, j) \in \omega \cup \gamma$) についての、 $(N_x + 1)(N_y + 1)$ 個の1次方程式である。 $U_{i,j}$ ($(i, j) \in \gamma$) は (B.14) から分かるので、それを (B.13) に代入して消去すると、 $(N_x - 1)(N_y - 1)$ 個の未知数 $U_{i,j}$ ($(i, j) \in \omega$) についての、

$$(B.15) \quad N := (N_x - 1)(N_y - 1)$$

個の1次方程式が得られる。

未知数の個数と方程式の個数が等しいので、正方行列を係数とする連立1次方程式の形に表せるはずである。実際にそうするためには、 $U_{i,j}$ を並べて1つのベクトルにする必要がある。

以下このことを実行するが、自分でプログラムを書く必要が生じるまで、読む必要はないであろう。

B.3.2 差分方程式 (連立 1 次方程式) の行列、ベクトル表現

簡単のため、同次 Dirichlet 境界条件 ($g_1 \equiv 0$) の場合に説明する。

$$\mathbf{U} = \begin{pmatrix} U_{11} \\ U_{21} \\ \vdots \\ U_{N_x-1,1} \\ \hline U_{12} \\ U_{22} \\ \vdots \\ U_{N_x-1,2} \\ \hline \vdots \\ \hline U_{1,N_y-1} \\ U_{2,N_y-1} \\ \vdots \\ U_{N_x-1,N_y-1} \end{pmatrix}, \quad \mathbf{F} = \begin{pmatrix} f(x_1, y_1) \\ f(x_2, y_1) \\ \vdots \\ f(x_{N_x-1}, y_1) \\ \hline f(x_1, y_2) \\ f(x_2, y_2) \\ \vdots \\ f(x_{N_x-1}, y_2) \\ \hline \vdots \\ \hline f(x_1, y_{N_y-1}) \\ f(x_2, y_{N_y-1}) \\ \vdots \\ f(x_{N_x-1}, y_{N_y-1}) \end{pmatrix}$$

とおくと、差分方程式から次のような連立 1 次方程式が得られる:

$$\begin{aligned}
 & \mathbf{A}\mathbf{U} = \mathbf{F}, \\
 \text{(B.16)} \quad & \mathbf{A} := \left(I_{N_y-1} \otimes \frac{1}{h_x^2} (2I_{N_x-1} - J_{N_x-1}) + \frac{1}{h_y^2} (2I_{N_y-1} - J_{N_y-1}) \otimes I_{N_x-1} \right).
 \end{aligned}$$

ここで \otimes は行列のテンソル積を表す記号であり、 I_k は k 次の単位行列、 J_k は次の形の k 次正方行列であるとする。

$$J_k = \begin{pmatrix} 0 & 1 & & & \mathbf{0} \\ 1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & & 1 & 0 & 1 \\ \mathbf{0} & & & & 1 & 0 \end{pmatrix},$$

(詳しくは桂田 [5] を見よ。)

プログラムを書くときのために、 \mathbf{U} , \mathbf{F} の成分 U_ℓ , F_ℓ を式で表しておく。

$$\text{(B.17)} \quad U_\ell = U_{i,j}, \quad F_\ell = f(x_i, y_j),$$

$$\text{(B.18)} \quad \ell = i + (j - 1)(N_x - 1).$$

つまり、領域内部の格子点 (x_i, y_j) を 1 次元的に並べて番号をつけた¹⁰、ということである。並べ方は一通りではなく、(B.18) の代わりに

$$\text{(B.19)} \quad \ell = j + (i - 1)(N_y - 1)$$

というものも良く使われる。(B.18) を row first, (B.19) を column first と呼んで区別する。次に紹介する MATLAB プログラムでは、(B.18) を採用してある。

¹⁰ $(i, j) \mapsto \ell$ の逆変換は、 ℓ を $N_x - 1$ で割った余りと商を $i - 1, j - 1$ とすれば良い。

B.3.3 MATLAB プログラム

MATLAB¹¹ は定評のあるシミュレーション・ソフトウェアである (桂田 [6])。

明治大学ではライセンス契約をしているので、学生は申請すれば使えるようになる (MATLAB TAH ライセンス¹²)。

(もちろん研究テーマによるが) 卒業研究等で有効に利用できる可能性があり、学ぶ価値は高い。この授業で MATLAB の解説をする余裕はないが、ここでは既に MATLAB を知っている人向けにサンプル・プログラムを紹介しておく。(なお、熱方程式を解くための MATLAB プログラムについては、桂田 [7], [8] を見よ。)

(B.16) の係数行列 A は次のプログラムで計算出来る。

```
poisson_coef.m
function A=poisson_coef(W, H, nx, ny)
% 長方形領域 (0,W) × (0,H) における Poisson 方程式の Dirichlet 境界値問題
% Laplacian を差分近似した行列を求める。
% 長方形を nx × ny 個の格子に分割して差分近似する。
% MATLAB では
% (1) 行列は Fortran と同様の column first であり、
% (2) mesh(), contour() による「行列描画」は Z(j,i) と添字の順が普通と逆なので、
% l=i+(j-1)*(nx-1) と row first となるように 1 次元の番号付けする
hx=W/nx;
hy=H/ny;
m=nx-1;
n=ny-1;
ex=ones(nx,1);
ey=ones(ny,1);
Lx=spdiags([-ex,2*ex,-ex],-1:1,m,m)/(hx*hx);
Ly=spdiags([-ey,2*ey,-ey],-1:1,n,n)/(hy*hy);
A=kron(speye(m,m),Ly)+kron(Lx,speye(n,n));
```

$$W = 3, \quad H = 2, \quad f \equiv 1, \quad N_x = 30, \quad N_y = 20$$

の場合、次のプログラムで計算出来る。

¹¹<https://jp.mathworks.com/>

¹²<http://www.meiji.ac.jp/isc/matlab-tah/>

```
poisson2d_f1.m
```

```
% 長方形領域 (0,W) × (0,H) で Poisson 方程式の同次 Dirichlet 境界値問題を解く
W=3.0;
H=2.0;
nx=30;
ny=20;
m=nx-1;
n=ny-1;
% 連立 1 次方程式を作成する。
A=poisson_coef(W, H, nx, ny);
% 描画用のメッシュ・グリッドを用意
x=linspace(0,W,nx+1); % x=[x_0,x_1,...,x_nx]
y=linspace(0,H,ny+1); % y=[y_0,y_1,...,y_ny]
[X,Y]=meshgrid(x,y); % これについては meshgrid() の説明を見よ。
% f ≡ 1 の場合
F=ones(m*n,1); % ここを頑張ると一般の非同次項の問題が解ける。
%
u=zeros(ny+1,nx+1);
u(2:ny,2:nx)=reshape(A\F,ny-1,nx-1);
%
% グラフの鳥瞰図
clf
colormap hsv
subplot(1,2,1);
mesh(X,Y,u);
colorbar
% 等高線
subplot(1,2,2);
contour(X,Y,u);
%
disp(' 図を保存する ');
print -dpdf poisson2d.pdf % 利用できるフォーマットは doc print で分かる
```

問 3. プログラム poisson2d_f1.m は、 $f \equiv 1$ の場合に問題を解くプログラムであるが、一般の f の場合に問題を解くプログラムを作成せよ。

以下の図は、 $f(x,y) = -2(x^2 - 3x + y^2 - 2y)$ の場合 (厳密解は $u(x,y) = x(3-x)y(2-y)$) の解を可視化したものである。

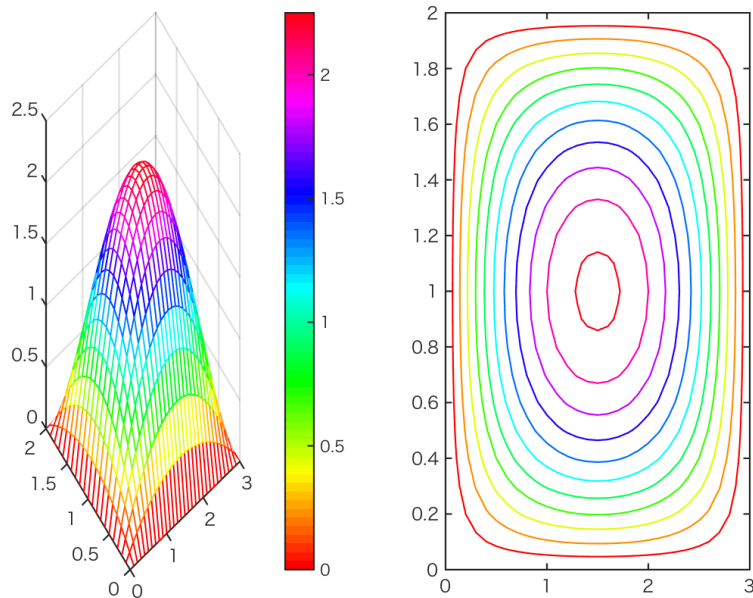


図 9: $-\Delta u = -2(x^2 - 3x + y^2 - 2y)$

問 4. プログラム poisson2d_f1.m は、同次 Dirichlet 境界条件 ($u = 0$ on $\partial\Omega$) の場合に問題を解くプログラムであるが、非同次 Dirichlet 境界条件 ($u = g_1$ on $\partial\Omega$) の場合に問題を解くプログラムを作成せよ。

B.4 区間でない領域における差分法

(準備中)

B.5 結び

- 差分方程式の導出は比較的分かりやすい (他人が作った差分方程式が、元の微分方程式の近似になっていることは分かりやすい) が、差分解の収束や、差分スキームの安定性などの証明は、もとの偏微分方程式の性質の証明と関連がある (結局勉強をサボるのは難しい)。
- 多次元の場合、境界が曲がっている領域を扱うには工夫が必要になる。

C Eigen ライブラリを用いた Jordan 領域の等角写像の計算プログラム

本文中に Python を用いたサンプル・プログラムを載せた。以前は C++ を用いていた。参考までそれも載せておく。

個人的には、この手の計算には MATLAB を使うのが好みであるが、ここでは、C++ でベクトル、行列に関する演算をするために便利なクラス・ライブラリである、Eigen¹³ を利用してみる (なお、C++ では、複素数を使うのも簡単である)。

¹³<http://eigen.tuxfamily.org/>

WWW サイトから、eigen-eigen-5a0156e40feb.tar.gz のような名前のソース・ファイル一式を入手して、以下のようにインストールする。

MacBook のターミナルで次のようにインストール

```
tar xzf eigen-eigen-5a0156e40feb.tar.gz
cd eigen-eigen-5a0156e40feb
ls
```

(Eigen があることを確認)

```
cp -pr Eigen /include   あるいは   cp -pr Eigen /usr/local/include
アーカイブ・ファイルに含まれていた、Eigen というディレクトリを、~/include
や/usr/local/include の下にコピーしている。
```

(脱線になるけれど、Eigen を用いた、Runge-Kutta 法のサンプル・プログラム <https://m-katsurada.sakura.ne.jp/complex2/ball-bound.cpp> を紹介しておく。コンパイルの仕方、実行の仕方は、注釈に書いてある。)

$\Omega = D_1 = D(0; 1)$, $z_0 = 1/2$ の場合を解いてみよう。つまり双正則な

$$\varphi: \Omega \rightarrow D_1$$

で、

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

を満たすものを求める。この場合は、1次分数変換

$$\varphi(z) = \frac{z - z_0}{1 - \bar{z}_0 z}$$

が解であることが分かっている。

conformalmap.cpp¹⁴ — 例えばターミナルで

```
curl -O https://m-katsurada.sakura.ne.jp/complex2/conformalmap.cpp
```

としてダウンロード出来る。

```
/*
 * conformalmap.cpp --- 等角写像の数値計算例 (基本解の方法の天野要氏バージョン)
 *
 * g++ -I/opt/X11/include -I/usr/local/include conformalmap.cpp
 * -L/usr/local/lib -lglsd -L/opt/X11/lib -lX11
 * これは /opt/X11 に X 関係のファイルが、
 * /usr/local に Eigen, GLSC 関係のファイルが
 * あると仮定している。
 *
 * 現象数理学科 Mac では、GLSC は ~/include, ~/lib にインストールして
 * あるだろうから、次のようにするのか。
 * g++ -I/opt/X11/include -I ~/include conformalmap.cpp -L ~/lib -lglsd -L/opt/X11/lib -lX11
 * その場合は、Eigen も ~/include にインストールすると良い。
 * まあ、適宜やって下さい。
 */

#include <iostream>
#include <complex>
```

¹⁴<https://m-katsurada.sakura.ne.jp/complex2/conformalmap.cpp>

```

#include <Eigen/Dense>

extern "C" {
#include <glsc.h>
};

using namespace std;
using namespace Eigen;

// MatrixXd は大きさが実行時に指定できて、成分が double の行列
// d(double) の代わりに i(int), f(float), cf, cd (complex<double>) も OK

//  $\phi(z_0)=0$ ,  $\phi'(z_0)>0$  を満たす等角写像
complex<double> phi(complex<double> z,
    complex<double> z0)
{
    complex<double> w;
    w = (z-z0)/(1.0-conj(z0)*z);
    return w;
}

// 天野の方法による等角写像
complex<double> f(complex<double> z,
    complex<double> z0,
    int N,
    double Q0,
    VectorXd Q, VectorXcd zeta)
{
    int k;
    complex<double> s;
    s = Q0;
    for (k = 0; k < N; k++)
        s += Q(k) * log((z-zeta(k))/(z0-zeta(k)));
    return (z-z0) * exp(s);
}

int main(void)
{
    int i, j, k, N, m;
    double theta, pi, dt, R, Q0, r, dr;
    // 虚数単位と原点に移る点 z0=1/2
    complex<double> I(0,1), z0(0.6,0.0), zp, w, w2;

    R = 2;
    N = 80;
    VectorXcd zeta(N), z(N);
    VectorXd b(N), Q(N);
    MatrixXd a(N,N);

    pi = 4.0 * atan(1.0);
    dt = 2 * pi / N;
    // 円周 |z|=2 上の等分点
    for (k = 0; k < N; k++)
        zeta(k) = R * exp(I * (k * dt));
    // cout << "zeta=" << zeta << endl;
    // 単位円周 |z|=1 上の等分点
    for (j = 0; j < N; j++)
        z(j) = exp(I * (j * dt));

    // 係数行列の準備
    for (j = 0; j < N; j++)
        for (k = 0; k < N; k++) {
            a(j,k) = log(abs(z(j)-zeta(k)));
        }
}

```

```

    }
// 右辺
for (j = 0; j < N; j++) {
    b(j) = - log(abs(z(j)-z0));
}
// 電荷を求める
Q = a.partialPivLu().solve(b);

// Q0 を求める
Q0 = 0;
for (k = 0; k < N; k++) {
    Q0 += Q(k) * log(abs(z0-zeta(k)));
}

g_init((char *)"GRAPH", 150.0, 150.0);
g_device(G_BOTH);
g_def_scale(0, - 1.2, 1.2, -1.2, 1.2, 10.0, 10.0, 140.0, 140.0);
g_sel_scale(0);

m = 20;
// 同心円の像
dr = 1.0 / m;
dt = 2 * pi / (4 * m);
for (i = 1; i <= m; i++) {
    r = i * dr;
    for (j = 0; j <= 4 * m; j++) {
        zp = r * exp(I * (j * dt));
        w=f(zp,z0,N,Q0,Q,zeta);
        if (j == 0)
            g_move(w.real(), w.imag());
        else
            g_plot(w.real(), w.imag());
    }
}
// 放射線の像
for (j = 0; j <= 4 * m; j++) {
    theta = j * dt;
    for (i = 0; i <= m; i++) {
        r = i * dr;
        zp = r * exp(I * theta);
        w=f(zp,z0,N,Q0,Q,zeta);
        if (i == 0)
            g_move(w.real(), w.imag());
        else
            g_plot(w.real(), w.imag());
    }
}
g_sleep(-1.0);
g_term();
return 0;
}

```

こんな風にコンパイル

```
g++ -I/opt/X11/include -I ~/include conformalmap.cpp -L ~/lib -lglscd -L/opt/X11/lib -lX11
```

(GLSC を利用していて、現象数理学科 Mac では、~/include, ~/lib にインストールされていることを想定している。上の手順で、Eigen も ~/include にインストールしておいた。)

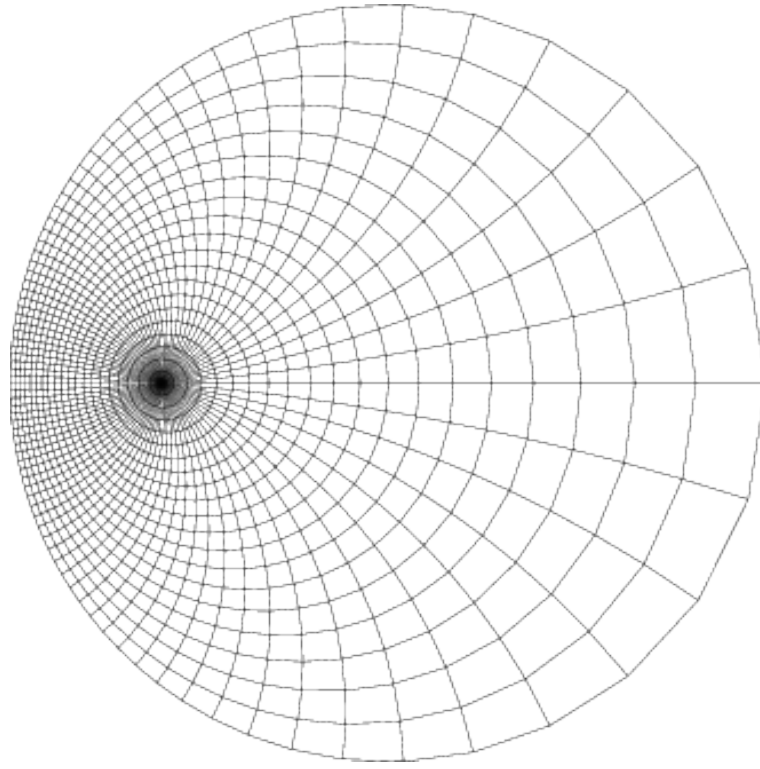


図 10: $w = \frac{z - z_0}{1 - \bar{z}_0 z}$, $z_0 = 0.6$ による z 平面の同心円、放射線の像を描いた。

参考文献

- [1] 桂田祐史：FreeFEM++ の紹介, <https://m-katsurada.sakura.ne.jp/labo/text/welcome-to-freefem/> (2007～).
- [2] 菊地文雄：有限要素法概説, サイエンス社 (1980), 新訂版 1999.
- [3] 桂田祐史：微分方程式 2 講義ノート (旧「応用解析 II」), <https://m-katsurada.sakura.ne.jp/lecture/pde/pde2013.pdf> (1997 年～).
- [4] 天野^{かなめ} 要：代用電荷法に基づく等角写像の数値計算法, 情報処理学会論文誌, Vol. 28, No. 27, pp. 697–704 (1987).
- [5] 桂田祐史：Poisson 方程式に対する差分法, <https://m-katsurada.sakura.ne.jp/labo/text/poisson.pdf> (2000 年?～).
- [6] 桂田祐史：MATLAB 使い方入門, <https://m-katsurada.sakura.ne.jp/labo/text/matlab.pdf> (HTML 版もある) (2005 年～).
- [7] 桂田祐史：長方形領域における熱方程式に対する差分法 — MATLAB を使って数値計算 —, <https://m-katsurada.sakura.ne.jp/labo/text/heat2d.pdf> (2015 年～).
- [8] 桂田祐史：Neumann 境界条件下の熱方程式に対する差分法 — MATLAB を使って数値計算 —, <https://m-katsurada.sakura.ne.jp/labo/text/heat2n.pdf> (2015 年～).