応用複素関数レポート課題２

桂田 祐史

2026年7月1日,2026年7月7日

目 次

1 課題２

2次元非圧縮ポテンシャル流の定常流で、流体の占める領域 Ω と、その境界 $\Gamma = \partial\Omega$ での流速の法線成分 $V_n := \boldsymbol{v} \cdot \boldsymbol{n}$ が分かっている場合に、有限要素法を用いて速度ポテンシャル ϕ を計算し、等ポテンシャル線と速度場を可視化せよ。

領域 Ω としては、(サンプル・プログラム以外のものを選ぶ意味で) 円盤領域以外のものを選ぶこと[1]。境界値 (流速の法線成分) V_n も自分で興味のあるもの、自分の都合の良いものを選んで良い (後の注意を読んでおくこと)。

以下でサンプル・プログラムを紹介する。それを理解した上で、領域 Ω と境界データ V_n を自分が決めたものに書き換えれば良い。自由度は高いので、工夫・遊び心発揮を期待する。

もし出来れば、流線も可視化せよ。流線の書き方には色々なやり方があるが、ここでは流れ関数を求めるやり方を勧めておく。

[1] ちなみに2024年度に、差分法のプログラムで、Dirichlet 境界値問題を Python で解いたレポートがあった。ほぼ無茶苦茶である (特に Dirichlet 境界値問題としているところ)。AI におかしなリクエストをしたことを疑っている。

2　〆切, 提出の仕方, 参考情報など

- 〆切は 7 月 31 日 (金曜) 22:00 (注: 定期試験は 7/29(水) に終了予定)。
- 提出方法: Oh-o! Meiji に A4 サイズの PDF で提出する。
  もし容量制限 (1 ファイル 30MB) に引っかかった場合は、複数のファイルに分割するなど工夫すること。
- 使用するプログラミング言語は、自分の MacBook で実行して見せることが可能なものであればなんでも可。
  (実際上は FreeFEM となると思われる。FreeFEM に関する質問は気軽にすること。)
- プログラムとその実行結果、実行するための情報を含めること。プログラムは別ファイルにしても良い。
- FreeFEM の使い方については、
  - `FreeFEM-documentation.pdf` (公式ドキュメント)
    https://doc.freefem.org/pdf/FreeFEM-documentation.pdf
  - 「FreeFEM の紹介」桂田が書いた紹介文書
    https://m-katsurada.sakura.ne.jp/labo/text/welcome-to-freefem/
  - 「FreeFem++ノート」桂田が書いた文法メモ
    https://m-katsurada.sakura.ne.jp/labo/text/freefem-note/
- サンプル・プログラムを用意してある。入手法をまとめて示しておく (2026/6/30 午前 まだ色々いじっています)。

```
curl -O https://m-katsurada.sakura.ne.jp/program/fem/poisson-kikuchi.edp
curl -O https://m-katsurada.sakura.ne.jp/complex2/potential2d-2026-naive.edp
curl -O https://m-katsurada.sakura.ne.jp/complex2/sample0.edp
curl -O https://m-katsurada.sakura.ne.jp/complex2/sample1.edp
curl -O https://m-katsurada.sakura.ne.jp/complex2/sample2.edp
curl -O https://m-katsurada.sakura.ne.jp/complex2/sample3.edp
```

`poisson-kikuchi.edp` は、正方形領域における Poisson 方程式の境界値問題を解くプログラムで、すでに紹介済み。これについては、「FreeFEM で菊地 [1] の Poisson 方程式の例題を解く」で説明してある。多角形領域の扱い方が分かるはず。

それ以外の `potential2d-2026.edp`, `sample` x`.edp` ($x = 0, 1, 2, 3$) は、どれも円盤領域で

$$\triangle\phi = 0 \quad (\text{in } \Omega) \tag{2.1a}$$

$$\frac{\partial\phi}{\partial \boldsymbol{n}} = V_n \quad (\text{on } \partial\Omega) \tag{2.1b}$$

を解くプログラムである。特に $x = 1, 2, 3$ は同じ問題を解いているが、境界条件のコーディングの仕方が異なる。

この境界値問題の弱形式は、すでに説明したように

$$\iint_\Omega (\phi_x v_x + \phi_y v_y)\, dx\, dy = \int_{\partial\Omega} V_n v\, d\sigma \quad (\text{任意の試験関数 } v \text{ に対して}) \tag{2.2}$$

である。2 次元なので境界積分 $\int_{\partial\Omega} V_n v\, d\sigma$ は、線積分 $\int_{\partial\Omega} V_n v\, ds$ に等しい。

(2.2) の左辺は FreeFEM のプログラム中では、

`int2d(Th)(dx(phi)*dx(v)+dy(phi)*dy(v))`

のようにすれば良いが (変更は不要である)、(2.2) の右辺 $\int_{\partial\Omega} V_n v\ ds$ については、ケース・バイ・ケースで、工夫が必要になることがある (FreeFEM の機能が今ひとつなため)。

3 注意 (必ず読むこと)

次の (1) が非常に重要である。例年それを間違えてナンセンスなレポートを提出する人が少なくない。

(1) V_n は $\int_{\partial\Omega} V_n\ d\sigma = 0$ を満たしている必要がある (そうでないと解が存在しない)。実際、$\dfrac{\partial\phi}{\partial n}$ の定義と Gauss の発散定理から

$$\int_{\partial\Omega} V_n\ d\sigma = \int_{\partial\Omega} \frac{\partial\phi}{\partial n}\ d\sigma = \int_{\partial\Omega} \operatorname{grad}\phi\cdot\boldsymbol{n}\ d\sigma = \int_{\Omega} \operatorname{div}\operatorname{grad}\phi\ d\boldsymbol{x} = \int_{\Omega} \triangle\phi\ d\boldsymbol{x} = \int_{\Omega} 0\ d\boldsymbol{x} = 0.$$

非圧縮流 ($\operatorname{div}\boldsymbol{v} = 0$) の速度場 $\boldsymbol{v}$ が分かっている場合に、$V_n = \boldsymbol{v}\cdot\boldsymbol{n}$ と置いた場合は、この条件は (自動的に) 成り立っている ($\int_{\partial\Omega} V_n\ d\sigma = \int_{\partial\Omega} \boldsymbol{v}\cdot\boldsymbol{n}\ d\sigma = \int_{\Omega} \operatorname{div}\boldsymbol{v}\ d\boldsymbol{x} = \int_{\Omega} 0\ d\boldsymbol{x} = 0$)。

そうでない場合には、$\int_{\partial\Omega} V_n\ d\sigma = 0$ が成り立つように、注意深く V_n を選ぶ必要がある。

過去の例では、円を楕円にするような (一見) 安直な選択をしたレポートがあったが、何か工夫をしないと $\int_{\partial\Omega} V_n\ d\sigma = 0$ が確かめにくい (円弧と異なり、楕円弧の長さは計算しにくいので、たとえ V_n が定数であっても計算は面倒である)。境界曲線が線分や円弧からなる場合 (多角形領域とか) を選ぶのが無難である。

(2) 湧き出しや吸い込み、点渦など、特異点が Ω 内にあるような問題 (レポート課題 1 のテーマであったとも言える) は、この方法では解くことが出来ない。

4 FAQ (よくされる質問)

「境界値 V_n を場合分けを含む関数としたいが、FreeFEM がプログラムを受け付けてくれません。どうすればいいですか？」

(FreeFEM ってしょうがないですね…) 例えば以下の 2 つの解決策がある。

(a) 弱形式には複数の `int1d()` が指定できるので、境界 $\Gamma = \partial\Omega$ を $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \cdots$ と分割して、その部分 Γ_j $(j = 1, 2, 3, \cdots)$ ごとに (場合分けを含まない) 境界値 V_{nj} を与えるようにプログラムを書く。

```
-int1d(Th,Gamma1)(Vn1)-int1d(Th,Gamma2)(Vn2)-int1d(Th,Gamma3)(Vn3)--…
```

(b) 例えば `(x>1 && y>0)` のような条件式は、条件が成り立つならば 1, 成り立たないならば 0 という値を持つので、`if` を使わずに、式だけで場合分けを含む関数が記述できる。

次のサンプル・プログラム `potential2d-2026.edp` では、(b) を用いているが、ある程度以上複雑になった場合は、(a) を用いることを勧める。付録の `sample1.edp` を見て下さい。

5　サンプルプログラム potential2d-2026.edp 解説

potential2d-2026.edp

```
1  // potential2d-2026-naive.edp --- 2 次元非圧縮ポテンシャル流
2  //   https://nalab.mind.meiji.ac.jp/~mk/complex2/potential2d-2026-naive.edp
3  //     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
4
5
6  border Gamma(t=0,2*pi) { x = cos(t); y = sin(t); } // 円盤領域
7  int m=40;
8  mesh Th=buildmesh(Gamma(m));
9  plot(Th, wait=1, ps="Th.eps");
10 // 次の 2 行は区分 1 次多項式を使うという意味
11 fespace Vh(Th,P1);
12 Vh phi, v;
13 // 境界条件の設定
14 //func Vn=x+2*y; // これは一様流 v=(1,2) の場合、v・n=(1,2)・(x,y)=x+2y
15 func Vn=((x>0&&y>0)||(x<0&&y<0))*(x+2*y);//右上と左下のみ出入りがある
16 // 整合条件 ∫ V_n ds=0 のチェック
17 cout << "check the compatibility condition: " << int1d(Th,Gamma)(Vn) << endl;
18
19 // 有限要素法で速度ポテンシャルφを求める
20 solve Laplace(phi,v,solver=CG) =
21   int2d(Th)(dx(phi)*dx(v)+dy(phi)*dy(v)) -int1d(Th,Gamma)(Vn*v);
22
23 // 平均を 0 にする（解の一意性がないので、時々生じるずれを消す）
24 real meanphi = int2d(Th)(phi)/Th.area; // Th.area は int2d(Th)(1); でも計算可能
25 cout << "mean phi=" << meanphi << endl;
26 phi=phi-meanphi;
27
28 // 速度ポテンシャルφの等高線（等ポテンシャル線）を描く
29 plot(phi,ps="contourpotential.eps",wait=1);
30
31 // ベクトル場 (v1,v2)=∇φ を描く（ちょっと雑なやり方）
32 Vh v1, v2;
33 v1=dx(phi); v2=dy(phi);
34 plot([v1,v2],ps="vectorfield.eps",wait=1);
35
36 // 等ポテンシャル線とベクトル場を同時に描く
37 plot([v1,v2],phi,ps="both.eps", wait=1);
```

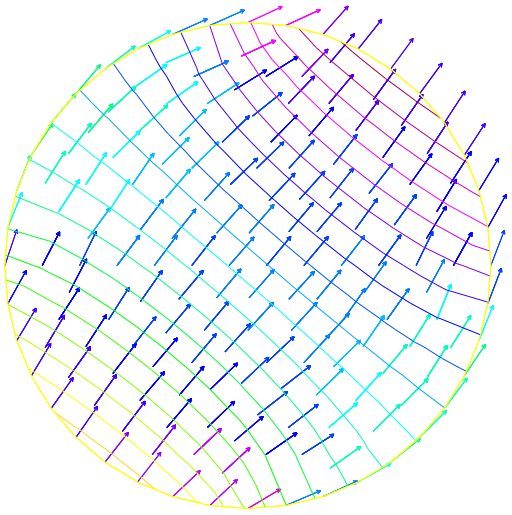

図 1: potential-2d-2026.edp の結果 (左下と右上で出入りがある)

- 6 行目で領域 Ω の境界 $\Gamma := \partial\Omega$ を指定している (それで Ω が決まる)。
- 8 行目で、Γ を m 分割して、Γ の囲む範囲を三角形分割して、それを `mesh` 型の変数 `Th` に代入している。
- 9 行目で三角形分割の様子を画面に表示する。
- 11 行目、有限要素空間 V_h を区分的 1 次関数の空間と定義している。この辺についてはこの講義では説明を省略する (知りたい人は有限要素法のテキストを読んで下さい)。ここを変更する必要はない。
- 12 行目、解 ϕ と試験関数 v を `Vh` の要素とする。
- 15 行目、境界値 V_n の設定をしている。ここでは

$$V_n(x,y) = \begin{cases} x+2y & ((x>0 \text{ かつ } y>0) \text{ または } (x<0 \text{ かつ } y<0) \text{ のとき}) \\ 0 & (\text{それ以外}) \end{cases}$$

としている。(後の例 `sample0.edp` では $V_n(x,y) = x+2y = \begin{pmatrix}1\\2\end{pmatrix} \cdot \boldsymbol{n}$ としていて、これは $\boldsymbol{v} = \begin{pmatrix}1\\2\end{pmatrix}$ という一様流が解となる。) これは確かに $\displaystyle\int_{\partial\Omega} V_n \, d\sigma = 0$ を満たしている。

- 17 行目 $\displaystyle\int_{\partial\Omega} V_n \, d\sigma$ を計算して表示する。これが 0 になるか (0 に十分近いか) どうか。このようにプログラムでチェックすることを勧める。
- 20〜21 行目、弱形式を定義すると同時に有限要素解を求めている。係数行列が特異なので、連立 1 次方程式は Gauss の消去法のような直接法では解けない。ここでは CG 法 (共役勾配法) を指定しているが、この辺は変更する必要はない。
- 24〜26 行目、Neumann 境界値問題の解はは定数だけの不定性を持つため (ϕ が解ならば ϕ + 定数 も解)、ごく稀に大きなズレが生じ、解の描画などで問題が生じる。平均値 $\dfrac{\iint_\Omega \phi(x,y)dx\,dy}{\iint_\Omega dx\,dy}$ を引くことで、平均 = 0 にしておく。やらなくても動く場合が多い。
- 29 行目、解 ϕ の等高線 (等ポテンシャル線) を描く。
- 32〜34 行目、速度場 $\mathrm{grad}\,\phi$ を (矢印で) 表示する。
- 37 行目、解 ϕ の等高線と速度場 $\mathrm{grad}\,\phi$ を一つの図に描く。

注意 5.1 (解の一意性と連立 1 次方程式の解法について) 領域の境界全体で Neumann 境界条件を課す場合、解に一意性はない (1 つの解に任意の定数を加えたものは解になる。逆にすべての解はこうして得られる。) そのことが原因となって、ここで現れる連立 1 次方程式の係数行列は特異である (逆行列は存在しない)。そのため (もし丸め誤差がなければ) 連立 1 次方程式の解法として、Gauss の消去法のような方法を用いると (ピボットに 0 が現れて) 解が求まらない。何か工夫が必要である。ここでは CG 法 (共役勾配法) を採用している。

このプログラムは一応動くのだが、より複雑な問題になると、丸め誤差の影響でうまく動かなくなる。この後のサンプル・プログラム `sample` x`.edp` ($x = 0, 1, 2, 3$) では、連立 1 次方程式 $Au = f$ の A と f を求めて、f から定数成分 (本来 0 のはずだが、誤差の影響でわずかに

混入している) を除去する (Image A へ射影する) ことにより、この困難を回避している。以上は数値線形代数の話題であるが、今回はブラックボックスとして用いれば良い (真似すれば良い) ので、詳細は省略する。 ■

6 流線を描くために: 流れ関数 ψ を求める

これは少し難しいので、レポートの必要条件とはしない。しかし流線があると流れの様子がよく分かるので、描く価値は高い。以下は頑張ろうという人向けのヒントである。講義で

$$\psi(\boldsymbol{x}) = \int_{C_{\boldsymbol{x}}} \psi_x \, dx + \psi_y \, dy = \int_{C_{\boldsymbol{x}}} \begin{pmatrix} -v \\ u \end{pmatrix} \cdot d\boldsymbol{r} = \int_{C_{\boldsymbol{x}}} \begin{pmatrix} -v \\ u \end{pmatrix} \cdot \boldsymbol{t} \, ds \quad (\boldsymbol{x} \in \overline{\Omega})$$

という式を紹介した ($C_{\boldsymbol{x}}$ は定点 $\boldsymbol{a}$ と $\boldsymbol{x}$ を結ぶ曲線)。領域が Jordan 領域 (1 つの単純閉曲線で囲まれた領域) であれば、$\boldsymbol{x}$ が境界上の点である場合、定点 $\boldsymbol{a}$ と $C_{\boldsymbol{x}}$ を境界上に選ぶことで、($\begin{pmatrix} -v \\ u \end{pmatrix} \cdot \boldsymbol{t} = \boldsymbol{v} \cdot \boldsymbol{n}$. また境界上での $\boldsymbol{v} \cdot \boldsymbol{n} = V_n$ は知っていると仮定しているので)

$$\psi(\boldsymbol{x}) = \int_{C_{\boldsymbol{x}}} \boldsymbol{v} \cdot \boldsymbol{n} \, ds = \int_{C_{\boldsymbol{x}}} V_n \, ds \quad (\boldsymbol{x} \in \partial\Omega).$$

これから、境界上の点 $\boldsymbol{x}$ における ψ の値 $g(\boldsymbol{x}) := \psi(\boldsymbol{x})$ $(x \in \partial\Omega)$ が得られる。それを用いて

$$\triangle\psi = 0 \quad (\text{in } \Omega), \quad \psi(\boldsymbol{x}) = g(\boldsymbol{x}) \quad (\text{on } \partial\Omega)$$

という Laplace 方程式の Dirichlet 境界値問題を解くことで流れ関数 ψ が得られる。

A 一様流のプログラム `sample0.edp`

(これは場合分けの不要な単純なプログラムである。)

サンプルプログラム `sample0.edp` では、単位円盤領域 $\Omega = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 < 1\}$ における一様流 $\boldsymbol{v} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ の場合のプログラムである。単位円盤なので $\boldsymbol{n} = \begin{pmatrix} x \\ y \end{pmatrix}$ (ここは良く考えること。例えば楕円の場合は $\boldsymbol{n}$ はかなり複雑な式になる。). これから

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

プログラムでは、`func Vn=x+2*y;` で `Vn` を定義して、弱形式中で `int1d(Th,Gamma)(Vn*v)` と使っている。

($\boldsymbol{v}$ が Ω で定数関数なので) $\operatorname{div} \boldsymbol{v} = 0$ であるから、当然 $\displaystyle\int_{\partial\Omega} V_n \, d\sigma = 0$ も成り立つ。

sample0.edp

```
// sample0.edp
//   https://m-katsurada.sakura.ne.jp/complex2/sample0.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=64;
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;
// 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
// 整合条件 ∫ V_n ds=0 のチェック
cout << "check the compatibility condition: " << int1d(Th,Gamma)(Vn) << endl;

// 有限要素法で連立1次方程式 A u=f を導く
varf Laplace(phi,v) =
  int2d(Th)(dx(phi)*dx(v)+dy(phi)*dy(v)) -int1d(Th,Gamma)(Vn*v);

matrix A = Laplace(Vh, Vh);
set(A,solver=CG);
real[int] f = Laplace(0, Vh); f = -f;
// f の定数成分（丸め誤差によるゴミ）を除く（Image A に射影する）
real[int] one(Vh.ndof);
one = 1.0;
one = one / sqrt(one' * one);
f=f-one'*f*one;
cout << "Does f belong to image space? (1, f)="<< (one' * f) << endl;

// 有限要素解を求める
phi[]=A^-1*f;

real meanphi=int2d(Th)(phi)/Th.area; // Th.area は int2d(Th)(1.0)
cout << "mean phi=" << meanphi << endl;
phi=phi-meanphi;
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）
Vh v1, v2;
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);
```

B sample1.edp, sample2.edp, sample3.edp

以下に紹介するすべてのプログラムは、単位円版領域において

$$\triangle\phi = 0 \quad \text{in } \Omega, \quad \frac{\partial\phi}{\partial n} = V_n \quad \text{on } \partial\Omega$$

を解くプログラムである。境界値 V_n も同じである (すぐ後で説明する)。コーディングの仕方だけが違っている。

単位円盤領域の境界である単位円周上で、$-\pi/8 \le \theta \le \pi/8, \pi/8 \le \theta \le \pi/2, \pi/2 \le \theta \le 3\pi/2,$

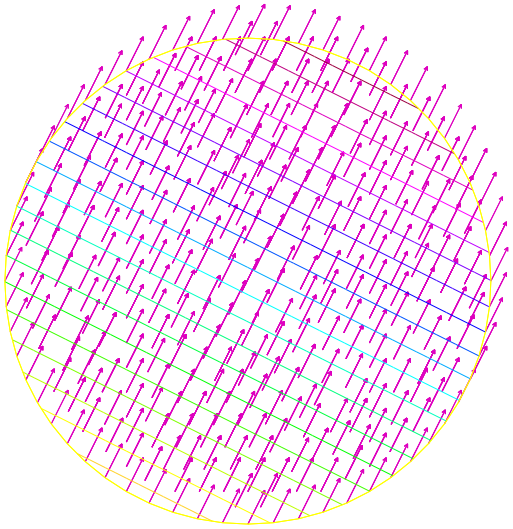

図 2: 一様流 (`sample0.edp`)

$3\pi/2 \le \theta \le 15\pi/8$ の範囲をそれぞれ $\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4$ とする。すなわち

$$\begin{aligned}
\Gamma_1 &= \{(\cos\theta, \sin\theta) \mid -\pi/8 \le \theta \le \pi/8\}, \\
\Gamma_2 &= \{(\cos\theta, \sin\theta) \mid \pi/8 \le \theta \le \pi/2\}, \\
\Gamma_3 &= \{(\cos\theta, \sin\theta) \mid \pi/2 \le \theta \le 3\pi/2\}, \\
\Gamma_4 &= \{(\cos\theta, \sin\theta) \mid 3\pi/2 \le \theta \le 15\pi/8\}.
\end{aligned}$$

V_n は次式で定める:

$$V_n = \begin{cases} 4\cos(4\theta) & (\text{on } \Gamma_1) \\ 0 & (\text{on } \Gamma_2) \\ \cos\theta & (\text{on } \Gamma_3) \\ 0 & (\text{on } \Gamma_4). \end{cases}$$

(Γ_3 ではゆっくり入ってくるが、狭い Γ_1 では勢いよく出る。Γ_2, Γ_4 では出入りがない。)

念のため $\displaystyle\int_{\partial\Omega} V_n \, d\sigma = 0$ の確認

$$\begin{aligned}
\int_{\partial\Omega} V_n \, d\sigma &= \int_{\Gamma_1} V_n \, d\sigma + \int_{\Gamma_2} V_n \, d\sigma + \int_{\Gamma_3} V_n \, d\sigma + \int_{\Gamma_4} V_n \, d\sigma = \int_{\Gamma_1} V_n \, d\sigma + \int_{\Gamma_3} V_n \, d\sigma \\
&= 4\int_{-\pi/8}^{\pi/8} \cos(4\theta) d\theta + \int_{\pi/2}^{3\pi/2} \cos\theta \, d\theta = 4 \cdot \frac{1}{4} \int_{-\pi/2}^{\pi/2} \cos\theta' d\theta' + \int_{\pi/2}^{3\pi/2} \cos\theta \, d\theta = 0.
\end{aligned}$$

デカルト座標で表すと

$$\cos\theta = x,$$

$$\cos(4\theta) = \cos^4\theta - 6\cos^2\theta\sin^2\theta + \sin^4\theta = x^4 - 6x^2y^2 + y^4.$$

あるいは

$$\cos(4\theta) = \mathrm{Re}\left[e^{4i\theta}\right] = \mathrm{Re}\left[(\cos\theta + i\sin\theta)^4\right] = x^4 - 6x^2y^2 + y^4.$$

(繰り返し) `sample` x`.edp` $(x = 1, 2, 3)$ はどれもこの問題を解くプログラムである。

もしも、プログラム中で V_n がうまく定義できれば、`sample0.edp` と同様に

```
solve Laplace(phi,v) =
   int2d(Th)(dx(phi)*dx(v)+dy(phi)*dy(v)) -int1d(Th,Gamma)(Vn*v);
```

というコードが使える。

しかし V_n を 1 行の関数として定義するのはあまり簡単でない。一応出来なくはなくて、sample2.edp ではそうしているが、ここでは sample1.edp のようにすることを勧める。

要点は

$$\int_{\partial\Omega} V_n v \, d\sigma = \sum_{j=1}^{4} \int_{\Gamma_j} V_n v \, d\sigma = \int_{\Gamma_1} V_n v \, d\sigma + \int_{\Gamma_3} V_n v \, d\sigma$$

と分解することである (Γ_2, Γ_4 では $V_n = 0$ であるから、$\int_{\Gamma_j} V_n v \, d\sigma$ $(j = 2, 4)$ は必要ない)。

Γ_1, Γ_3 における V_n を、それぞれ Vn1, Vn3 と定義すれば

```
solve Laplace(phi,v) =
   int2d(Th)(dx(phi)*dx(v)+dy(phi)*dy(v))
  -int1d(Th,Gamma1)(Vn1*v)-int1d(Th,Gamma3)(Vn3*v);
```

とできる。こうするためには、もちろんプログラム中で $\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4$ を用意する必要があるが、それは難しくない (境界を 4 つに分けた正方形領域のプログラム poisson-kikuchi.edp と同様のことをすれば良い)。以上の考察から、次のプログラムが得られる。

sample1.edp

```
// sample1.edp
//   https://m-katsurada.sakura.ne.jp/complex2/sample1.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
//   sample1.edp と同じ問題を解く。境界値の指定法が異なる。

border Gamma1(t=-pi/8,pi/8)     { x = cos(t); y = sin(t); }// 勢いよく出る
border Gamma2(t=pi/8,pi/2)      { x = cos(t); y = sin(t); }
border Gamma3(t=pi/2,3*pi/2)    { x = cos(t); y = sin(t); }// ゆっくり入る
border Gamma4(t=3*pi/2,15*pi/8) { x = cos(t); y = sin(t); }
int m=4;
mesh Th=buildmesh(Gamma1(2*m)+Gamma2(3*m)+Gamma3(8*m)+Gamma4(3*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;
// 境界条件の設定
// Gamma1 上で 4 cos(4 θ)=4(x^4-6x^2y^2+y^4), Gamma3 上で cos θ=x
func Vn1 = 4*(x^4-6*x^2*y^2+y^4);
func Vn3 = x;
// 整合条件 ∫ V_n ds=0 のチェック
cout << "\int Vn ds=" << int1d(Th,Gamma1)(Vn1)+int1d(Th,Gamma3)(Vn3) << endl;

// 有限要素法で連立 1 次方程式 A u=f を導く
varf Laplace(phi,v) =
  int2d(Th)(dx(phi)*dx(v)+dy(phi)*dy(v))
  -int1d(Th,Gamma1)(Vn1*v)-int1d(Th,Gamma3)(Vn3*v);

matrix A = Laplace(Vh, Vh);
set(A,solver=CG);
real[int] f = Laplace(0, Vh); f = -f;
// f の定数成分 (丸め誤差によるゴミ) を除く (Image A に射影する)
real[int] one(Vh.ndof);
one = 1.0;
```

```
one = one / sqrt(one' * one);
f=f-one'*f*one;
cout << "Does f belong to image space? (1, f)="<< (one' * f) << endl;

// 有限要素解を求める
phi[]=A^-1*f;

real meanphi=int2d(Th)(phi)/Th.area; // Th.area は int2d(Th)(1.0)
cout << "mean phi=" << meanphi << endl;
phi=phi-meanphi;
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)
Vh v1, v2;
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);
```

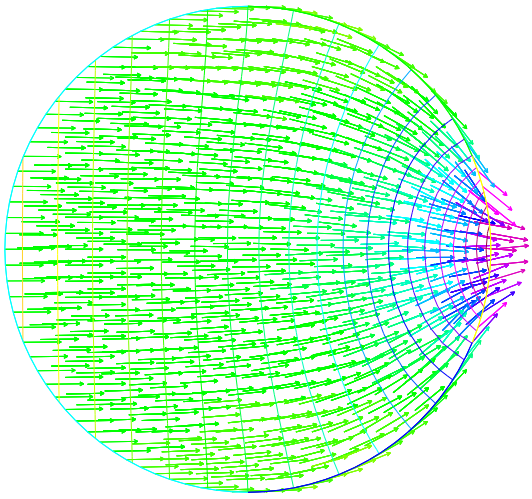

図 3: sample1.edp の結果 ($\theta \in [\pi/2, 3\pi/2]$ でゆっくり入り、$\theta \in [-\pi/8, \pi/8]$ で勢い良く出る)

以下、参考までに、V_n を 1 命令 (長いので 2 行になっている) で書いたプログラム sample2.edp を紹介しておく。いわゆる偏角の主値 (値が $(-\pi, \pi]$ に属する) を返す関数 atan2(y,x) を使うが、やや不自然な感じがするのは否めない (角度の範囲が $[0, 2\pi)$ でないことにも注意が必要である)。

強引に 1 行関数を作る

```
func Vn=(atan2(y,x)>=-pi/8 && atan2(y,x)<=pi/8) * 4.0 * (x^4-6*x*x*y*y+y^4)
       + (x<=0) * x;
```

sample2.edp

```
// sample2.edp
//   https://m-katsurada.sakura.ne.jp/complex2/sample2.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
//   sample1.edp と同じ問題を解く。境界値の指定法が異なる。

border Gamma(t=0,2*pi) { x = cos(t); y = sin(t); } // 円盤領域
int m=64;
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;
// 境界条件の設定 (atan2(y,x) は [-pi,pi) の範囲の値を返す)
// Gamma1 上で 4 cos(4 θ)=4(x^4-6x^2y^2+y^4), Gamma3 上で cos θ=x
func Vn=(atan2(y,x)>=-pi/8 && atan2(y,x)<=pi/8) * 4.0 * (x^4-6*x*x*y*y+y^4)
       + (x<=0) * x;
// 整合条件 ∫ V_n ds=0 のチェック
cout << "\int Vn ds=" << int1d(Th,Gamma)(Vn) << endl;

// 有限要素法で連立 1 次方程式 A u=f を導く
varf Laplace(phi,v) =
  int2d(Th)(dx(phi)*dx(v)+dy(phi)*dy(v)) -int1d(Th,Gamma)(Vn*v);

matrix A = Laplace(Vh, Vh);
set(A,solver=CG);
real[int] f = Laplace(0, Vh); f = -f;
// f の定数成分 (丸め誤差によるゴミ) を除く (Image A に射影する)
real[int] one(Vh.ndof);
one = 1.0;
one = one / sqrt(one' * one);
f=f-one'*f*one;
cout << "Does f belong to image space? (1, f)="<< (one' * f) << endl;

// 有限要素解を求める
phi[]=A^-1*f;

real meanphi=int2d(Th)(phi)/Th.area; // Th.area は int2d(Th)(1.0)
cout << "mean phi=" << meanphi << endl;
phi=phi-meanphi;
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)
Vh v1, v2;
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);
```

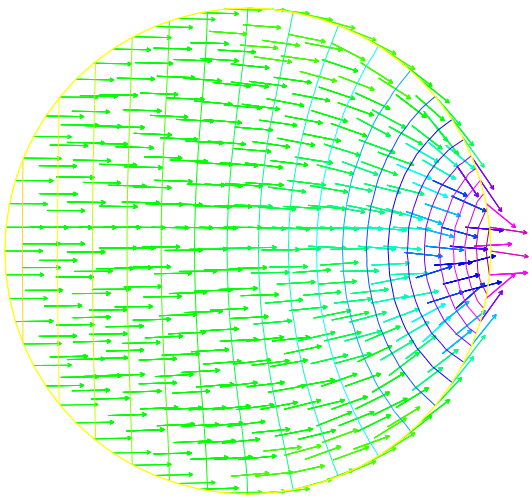

図 4: sample2.edp の結果 ($\theta \in [\pi/2, 3\pi/2]$ でゆっくり入り、$\theta \in [-\pi/8, \pi/8]$ で勢い良く出る)

普通のプログラミング言語であれば、V_n 程度の場合分けを含む関数は、if 文を使って書くのだろう。そのように関数 Vn(x,y) を作ってみよう。しかしこの Vn() を弱形式にそのまま

使うことは出来ないので (FreeFEM の文法の仕様上の制限)、fespace の変数 Vn2 を用意して Vn2=Vn(x,y) と代入して、Vn2 を弱形式中で用いる (領域内部での値を計算しているが、それはめちゃくちゃな値で、後の計算で利用もしていない。プログラムは読みやすい)。

sample3.edp

```
// sample3.edp
//   https://m-katsurada.sakura.ne.jp/complex2/sample3.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
//   sample1.edp と同じ問題を解く。境界値の指定法が異なる。

border Gamma(t=0,2*pi) { x = cos(t); y = sin(t); } // 円盤領域
int m=64;
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;

// 境界条件の設定
func real Vn(real x, real y)
{
  real theta=atan2(y,x);
  if (theta >= - pi / 8 && theta <= pi / 8)
    return 4.0 * (x^4 - 6*x^2*y^2 + y^4);
  else if (x <= 0)
    return x;
  else
    return 0.0;
}

// 境界値 (弱形式の中で func Vn() は直接指定できないので値を求めておく)
Vh Vn2;
Vn2 = Vn(x,y);
cout << "int Vn ds=" << int1d(Th,Gamma)(Vn2) << endl;
// 有限要素法で連立 1 次方程式 A u=f を導く
varf Laplace(phi,v) =
  int2d(Th)(dx(phi)*dx(v)+dy(phi)*dy(v)) -int1d(Th,Gamma)(Vn2*v);

matrix A = Laplace(Vh, Vh);
set(A,solver=CG);
real[int] f = Laplace(0, Vh); f = -f;
// f の定数成分 (丸め誤差によるゴミ) を除く (Image A に射影する)
real[int] one(Vh.ndof);
one = 1.0;
one = one / sqrt(one' * one);
f=f-one'*f*one;
cout << "Does f belong to image space? (1, f)="<< (one' * f) << endl;

// 有限要素解を求める
phi[]=A^-1*f;

real meanphi=int2d(Th)(phi)/Th.area; // Th.area は int2d(Th)(1.0)
cout << "mean phi=" << meanphi << endl;
phi=phi-meanphi;
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)
Vh v1, v2;
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);
```