FreeFem++ の紹介

桂田 祐史

2007 年 9 月 26 日〜, 2016 年 2 月 14 日, 2018 年 12 月 9 日

この文書の最新版は

http://nalab.mind.meiji.ac.jp/~mk/labo/text/welcome-to-freefem/

または

http://nalab.mind.meiji.ac.jp/~mk/labo/text/welcome-to-freefem.pdf

で閲覧可能です。

FreeFem++ の文法については、一新された FreeFem++ Documentation[1] を読むのが本筋だと思います。(「FreeFem++ノート」[2] というメモも書いてありましたが、今後どうしようか思案中です。)

1 手短な紹介

偏微分方程式の汎用数値解法には、古くから**差分法** (finite difference method, FDM) と**有限要素法** (finite element method, FEM) という二大手法があります (最近では**有限体積法** (finite volume method) も良く使われるようになっていて、「二大」は止めた方が良いでしょうか)。

有限要素法は、解析学の世界でスタンダードとなっている**弱解の方法**を数値計算に応用したもので、微分方程式の弱定式化の議論とほぼ並行した形で近似解の議論がなされ、数学者になじみやすいものです。また有限要素法は、プログラムの自動生成がしやすいという特徴を持っています[3]。このため、従来から、プログラムを自動生成するシステムを開発する試みが色々なされてきました。

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、ソースコードと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ます。

これまで 400 ページ超の “マニュアル” PDF が配布されていましたが[5]、それはなくなって、現在は WWW 上のドキュメンテーション FreeFem++ Documentation[6] がその代わりになるもの (らしい) です。

---

[1] https://doc.freefem.org/documentation/

[2] http://nalab.mind.meiji.ac.jp/~mk/labo/text/freefem-note.pdf

[3] こう書くと差分法のシンパに反発されそうな気もしますが、野次馬の観察 (岡目八目) によると、有限要素法に一日の長があるのは否定できないと思います。

[4] http://www.freefem.org/

[5] これまで「こんなのマニュアルじゃない」とブツブツ言ってきたのですが、この新しいドキュメンテーションは (まだあまり目を通していませんが) とても良いと思います。馬力のある人が翻訳プロジェクトでも始めないかな…

一方で、従来の “マニュアル” は事例集として重要なものなので、どこか分かりやすいところに置いてもらいたいですね。今だと http://www3.freefem.org/ff++/ftp/freefem++doc.pdf からダウンロードできます。

[6] https://doc.freefem.org/documentation/

FreeFem++ では、ユーザーが境界曲線を指定することで定義した 2 次元領域に対して、自動的に三角形要素分割を生成し、**弱形式**により記述された問題に対して、種々の有限要素 (関数) 空間を用いて弱解を求め、可視化することが可能になっています (最近では 3 次元領域の問題も解けるようになっているようですが、筆者はそれについてほとんど知らないので、言及するのを止めておきます。)。

問題の領域や弱形式などは専用の言語で記述した短いプログラム (ものによっては 20〜30 行程度) として与え、それを実行することでシミュレーションを行います。

すべての問題がこのソフトで扱えるわけではありませんが、扱える問題は結構幅広く、それが出来るときは多くの場合、C 言語のようなプログラミング言語で一からプログラムを書くのに比べて、桁違いに短い時間でシミュレーション実行までたどり着けます。(プログラムの行数自体は、2 桁とは行きませんが、1.5 桁くらい小さいように思います。)

大塚・高石「有限要素法で学ぶ現象と数理—FreeFem++」共立出版 (2014) という解説書が出版され、そのサポートサイト「有限要素法で学ぶ現象と数理」[7] が出来ました。有益な情報が日本語で得られるようになりました。

日本応用数理学会で、FreeFem++ のセミナーが何回か開かれています。私が参加できた直近 2 回の情報は

- 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 －上級編－[8] (2016/6/4,5)
  資料https://www.ljll.math.upmc.fr/~suzukia/FreeFempp-tutorial-JSIAM2016b/index.html
- ソフトウェアセミナー：FreeFem++で学ぶ現象と数理[9] (2016/2/11, 12)
  資料http://www.ljll.math.upmc.fr/~suzukia/FreeFempp-tutorial-JSIAM2016

幸い、2 回とも都合がついて参加でき、有益な情報が得られました。関係者のご尽力に感謝いたします。

講師の鈴木厚先生には、2007 年の九州大学で行われたチュートリアルでもお世話になりました。そのとき頂いた資料をネタにして、自分の学生に FreeFem++ を教えて来ましたが、今後は、上のセミナーの資料を咀嚼して、学生に教えていくことになると思います。

2 百聞は一見に如かず

2.1 Mac に最新版をインストールする

(Mac OS X 10.11 がインストールされている Mac に、FreeFem++ version 3.61-1 をインストールした時の記録を、http://nalab.mind.meiji.ac.jp/~mk/labo/text/inst_memo_361/ に載せておきました。)

FreeFem++ 本体をインストールするには、WWW サイト Freefem++ Home Page[10] から、`FreeFem++-3.61-1-MacOS_10.11.pkg` (3.61-1 はバージョン番号で、これは当然変化します) のようなインストーラーを入手して実行します (Ctrl+クリックして「開く」)。

ターミナルから実行するには、環境変数 PATH が設定されている必要がありますが、最近の FreeFem++ では、`/etc/paths.d/FreeFem++` というファイルを用意することによって “自動的に” 設定されます。

(例えば version 3.61-1 の場合は、

[7] http://comfos.org/jp/ffempp/book/
[8] http://na.cs.tsukuba.ac.jp/acmi/?p=403
[9] http://na.cs.tsukuba.ac.jp/acmi/?p=383
[10] http://www.freefem.org/

- `/usr/local/ff++/openmpi-2.1/3.61-1/bin`
- `/usr/local/ff++/openmpi-2.1/bin`

という2つのディレクトリィに実行可能プログラムが置かれていますが、`/etc/paths.d/FreeFem++` には、前者のディレクトリィが書かれています。

(新しいバージョンをインストールしてうまく行かない場合は、`/usr/local/ff++` の下を探検してみることを勧めます[11]。)

GUI でプログラムの開発＆実行の出来る **FreeFem++-cs** という IDE (統合開発環境) があって、以前は私も身の回りの学生に推奨していましたが、最近ではオリジナルの `FreeFem++` だけで十分と考えるようになりました。

2.2 簡単な問題で数学超特急

Poisson 方程式の境界値問題で説明します (有限要素法を自習したい場合には、菊地文雄, 『有限要素法概説』, サイエンス社, という参考書がイチオシです, 桂田研の学生で希望する人は申し出て下さい)。

$\mathbb{R}^2$ 内の有界な領域 (連結開集合) Ω における Poisson 方程式の Dirichlet 問題

$$\text{(PE)} \qquad -\triangle u = f \quad (\text{in } \Omega),$$
$$\text{(DBC)} \qquad u = 0 \quad (\text{on } \Gamma := \partial\Omega)$$

を考えます (弱解の方法の例題として、もっとも簡単な、定番の問題です)。

Poisson 方程式 (PE) に、任意の $v \in C_0^\infty(\Omega)$ ($C_0^\infty(\Omega)$ は、Ω 内に compact な台を持つ C^∞ 級の関数全体) をかけて、Green の積分公式 (多次元版部分積分) を用いると、次式が得られます。

$$\iint_\Omega \left(\frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y} \right) dx\,dy = \iint_\Omega f(x,y)v(x,y)\,dx\,dy \quad (v \in C_0^\infty(\Omega)). \tag{1}$$

逆に $u = 0$ (on Γ) を満たす滑らかな u が、この条件 (1) を満たせば、u は Poisson 方程式 (PE) を満たすことも容易に分かります。

微分方程式 (PE) を満たす u を求める代りに、(1) を満たす u を見出すことを考えましょう。解の存在を容易に保証できるようにするため、関数空間を完備化するのが便利です (解を極限の論法で得よう、という解析学の常套手段を使うため)。具体的には、以下に定義する $H_0^1(\Omega)$ というソボレフ空間 (一般化された導関数を持つ関数からなる関数空間の一種) で u を探すことにします。

(PE),(DBC) の弱解の定義

$u \in H_0^1(\Omega)$ かつ

$$\iint_\Omega \left(\frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y} \right) dx\,dy = \iint_\Omega f(x,y)v(x,y)\,dx\,dy \quad (v \in H_0^1(\Omega)).$$

[11] ターミナルで、`open /usr/local/ff++` とすると、Finder で `/usr/local/ff++` が開かれます。

$H_0^1(\Omega)$ とは

$$H^1(\Omega) = \left\{ u \in L^2(\Omega) \;\middle|\; \frac{\partial u}{\partial x_j} \in L^2(\Omega)\ (j = 1, 2) \right\}$$

という関数の集合に

$$(u, v)_{H^1} := \iint_\Omega u(x, y)\overline{v(x, y)}\, dx\, dy + \iint_\Omega \nabla u(x, y) \cdot \overline{\nabla v(x, y)}\, dx\, dy$$

という内積を導入すると、$H^1(\Omega)$ は Hilbert 空間 (完備な内積空間) になる。

$$H_0^1(\Omega) := C_0^\infty(\Omega) \text{ の } H^1(\Omega) \text{ での閉包}$$

とおく。$H_0^1(\Omega)$ の要素は、$H^1(\Omega)$ の要素のうちで、ある意味で Γ 上 0 となるものとみなすことができる。

Ω を三角形の「整った」合併 T_h で近似し、T_h で連続で、各三角形上で 1 次関数であるような関数全体を V_h とします。V_h は $H^1(\Omega)$ の有限次元部分空間とみなせ、グラフが三角錐であるような関数 $\varphi_1, \ldots, \varphi_m$ を基底に取って、各元をその基底の線型結合として表現することができます:

$$V_h = \left\{ \sum_{j=1}^m c_j \varphi_j \;\middle|\; (c_j) \in \mathbb{R}^m \right\}.$$

(PE),(DBC) の弱解の定義 (書き換え)

$u \in H^1(\Omega)$ かつ

$$\iint_\Omega \left(\frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y} \right) dx\, dy = \iint_\Omega f(x, y)v(x, y)\, dx\, dy \quad (v \in H^1(\Omega),\ v = 0 \text{ on } \Gamma),$$
$$u = 0 \quad (\text{on } \Gamma).$$

有限要素解の定義

$u_h \in V_h$ かつ

$$\iint_{T_h} \left(\frac{\partial u_h}{\partial x}\frac{\partial v_h}{\partial x} + \frac{\partial u_h}{\partial y}\frac{\partial v_h}{\partial y} \right) dx\, dy = \iint_{T_h} f(x, y)v_h(x, y)\, dx\, dy \quad (v_h \in V_h,\ v_h = 0 \text{ on } \Gamma),$$
$$u_h = 0 \quad (\text{on } \Gamma).$$

次のようなプログラム `poisson.edp` を用意します (テキスト・エディターを使って作成します。現象数理学科学生は、`mi`, `emacs`, `Atom` などを習っているはずです。)。

poisson.edp[12]

```
// poisson.edp
// http://nalab.mind.meiji.ac.jp/~mk/program/fem/poisson.edp
// http://nalab.mind.meiji.ac.jp/~mk/labo/text/welcome-to-freefem/node4.html

// 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;
// 現在時刻をメモ
real start = clock();
// 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,wait=true);
plot(u,dim=3,viso=levels,fill=true,wait=true);
// 計算時間を表示
cout << " CPU time= " << clock() - start << endl;
```

ターミナルで poisson.edp を実行 ($ はプロンプトなので入力しない)

```
$ FreeFem++ poisson.edp
```

このプログラムを実行すると、円盤領域の三角形分割を表示したところで停止します (plot() に wait=true としてあるため)。先に進めるには enter (return) を入力します。

等高線が表示されたところでも、先に進めるには enter (return) を入力します。

最後にグラフを描画しますが、そのウィンドウを閉じるには esc を入力します。

キーボードからのコマンド

[Enter]	次のプロットへ
[ESC]	グラフィックスを閉じる
+	ズーム（イン）する
−	ズームアウトする
=	ズームの状態を元に戻す
c	ベクトルの矢印の大きさを短くする
C	ベクトルの矢印の大きさを長くする
r	リフレッシュする
f	カラーを塗りつぶすかどうか
v	
?	help

plot(u,wait=true); のところを plot(u,wait=true,ps="poisson-disk.eps"); のようにすると、画面に表示するだけでなく、PostScript 形式でファイル出力できます (TEX 文書へ

の取り込みが簡単です)。

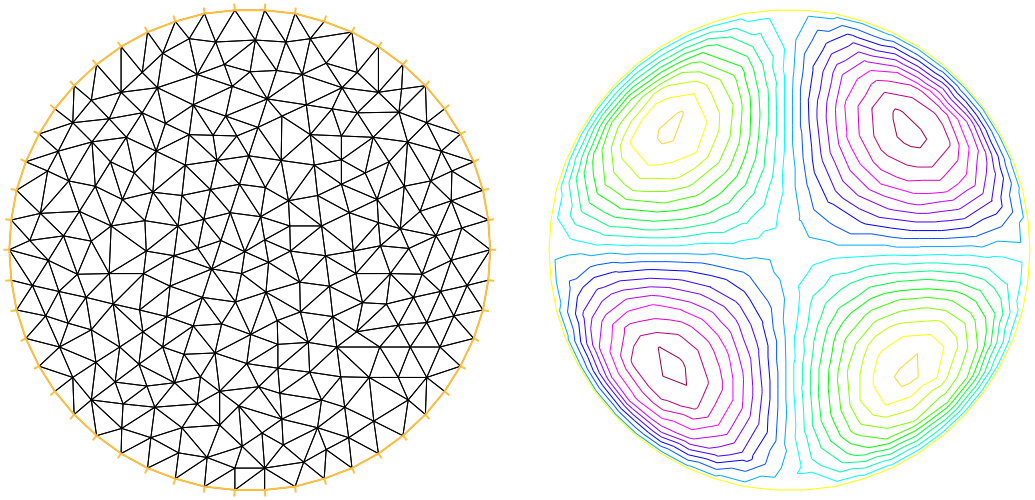


図 1: 円盤領域で Poisson 方程式 $-\triangle u(x,y) = xy$ の同次 Dirichlet 問題を解く (要素分割と解の等高線)

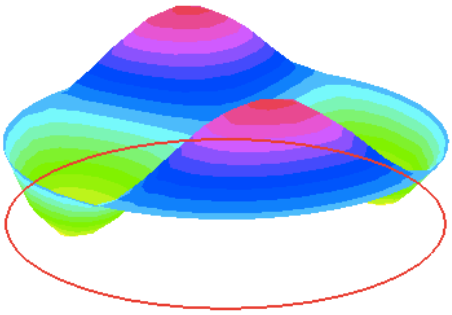


図 2: `dim=3` でグラフの鳥瞰図を描く

3 おまけ: gnuplot で可視化

従来、FreeFem++ では、等高線描画やベクトル場の表示などは出来るが、グラフの鳥瞰図描画などはサポートしていなかった (今は `plot(...,dim=3,...);` とするだけで出来る)。そこで gnuplot を使ってグラフを描く方法を紹介する。

poisson-g.edp[13]

```
// poisson-g.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,ps="Th.eps",wait=true);
// 有限要素空間は P1（区分的 1 次多項式）要素
fespace Vh(Th,P1);
Vh u,v;
// Poisson 方程式 -△ u=f の右辺
func f = x*y;
// 現在時刻を記録
real start = clock();
// 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
plot(u,ps="poisson-disk.eps");
// 計算結果をファイルに出力
{
  ofstream ug("u-g.txt");
  for (int i=0; i<Th.nt; i++) {
    for (int j=0; j<3; j++) {
      ug << Th[i][j].x << " " << Th[i][j].y << " " << u[][Vh(i,j)]<<endl;
    }
    ug << Th[i][0].x << " " << Th[i][0].y << " " << u[][Vh(i,0)]<<"\n\n\n";
   }
}
// 計算時間を表示
cout << " CPU time= " << clock() - start << endl;
```

“u-g.txt” は次のようなデータである (3 次元空間における三角形が並んでいる)。

u-g.txt[14]

```
0.84068 0.327399 0.00414118
0.794983 0.411712 0.00532531
0.751583 0.286674 0.00623707
0.84068 0.327399 0.00414118


0.870933 0.153872 0.00249565
0.84068 0.327399 0.00414118
0.751583 0.286674 0.00623707
0.870933 0.153872 0.00249565

(中略)
......
0.1092 0.735914 0.00288613
0.140979 0.864349 0.00225845
0.0271079 0.868281 0.000433488
0.1092 0.735914 0.00288613


-0.0867634 0.872212 -0.00146171
-0.203275 0.85648 -0.00323151
-0.110736 0.746398 -0.00290419
-0.0867634 0.872212 -0.00146171
```

このデータを gnuplot で可視化するには、例えば次のようにする (図 3)。

poisson-g.g

```
set hidden3d
set palette rgbformulae 33,13,10
splot "u-g.txt" with lines palette
```

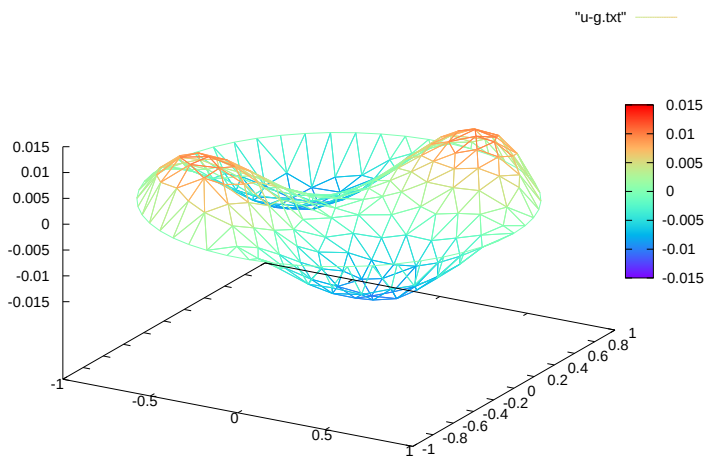


図 3: Poisson 方程式の解のグラフ表示 (gnuplot による)

参考文献

[1] 菊地文雄：有限要素法概説, サイエンス社 (1980), 新訂版 1999.

A 参考: FreeFem++ で菊地 [1] の Poisson 方程式の例題を解く

菊地 [1] に載っている Poisson 方程式の例題

$$-\triangle u = f \quad \text{in } \Omega, \tag{2}$$

$$u = g_1 \quad \text{in } \Gamma_1, \tag{3}$$

$$\frac{\partial u}{\partial n} = g_2 \quad \text{in } \Gamma_2 \tag{4}$$

(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。

poisson-kikuchi.edp[15]

```
// poisson-kikuchi.edp
// http://nalab.mind.meiji.ac.jp/~mk/program/fem/poisson-kikuchi.edp
// 菊地文雄，有限要素法概説，サイエンス社

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=true,ps="Th.eps");
savemesh(Th,"Th.msh");
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) とも書ける。
plot(u,wait=1,ps="poisson-kikuchi.eps");

//3 次元鳥瞰図
//real [int] levels =0.0:0.01:1.0;
//plot(u,dim=3,viso=levels,fill=true,wait=true);
```

§2.2 のサンプル・プログラムを叩き台にする。

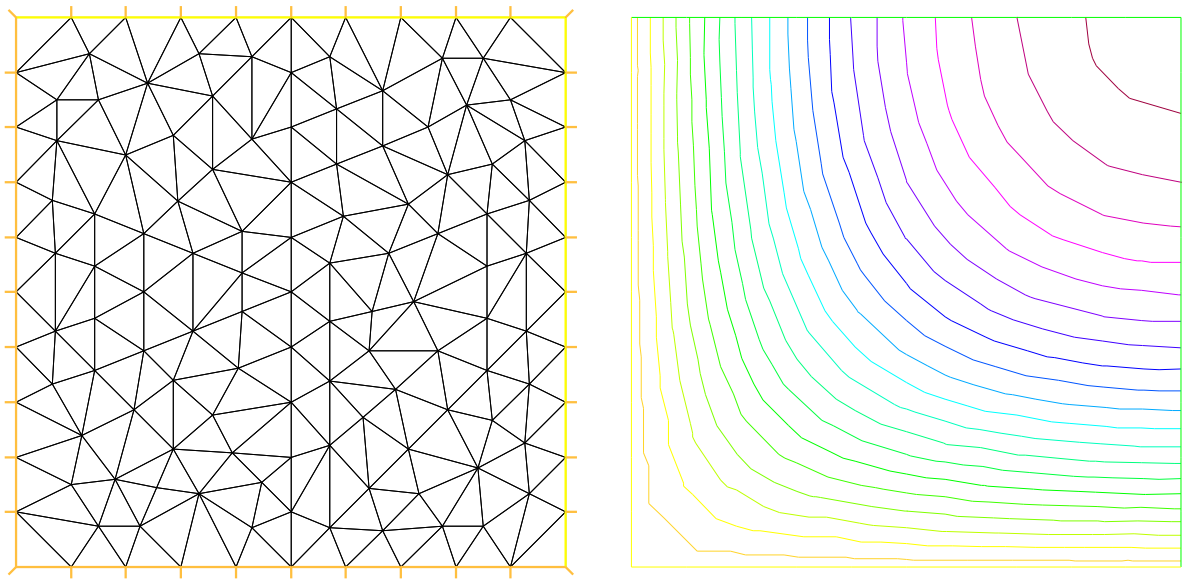


図 4: poisson-kikuchi.edp — 要素分割と解の等高線

プログラムで生成される、三角形要素分割のデータを出力できるが、それは次のようなものである (しばしば、数値実験の三角形要素分割を、FreeFem++ の出力を拝借している研究者がいる)。

$m = 2$ のときの “Th.msh”

```
9 8 8
0 0 1
0 1 2
0 0.5 1
0.5 0 1
1 0 2
0.499999999488 0.499999999488 0
0.5 1 2
1 0.5 2
1 1 2
7 8 9 0
1 4 3 0
4 5 8 0
6 8 7 0
4 6 3 0
4 8 6 0
2 3 7 0
7 3 6 0
9 7 2
7 2 2
5 8 2
8 9 2
1 4 1
4 5 1
2 3 1
3 1 1
```

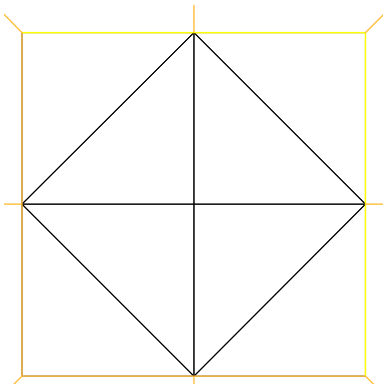


図 5: $m = 2$ の時の三角形分割の様子 (Th.msh に対応)

マニュアルで確認していないが、多分次のようなフォーマットになっているのであろう。

FreeFem++ のメッシュファイルのフォーマット

総節点数 n　総要素数 m　　境界に属する辺の数 N
節点 P_1 の x,y 座標とラベル (0 は内点)
　⋮
節点 P_n の x,y 座標とラベル (0 は内点)
要素 e_1 の 3 節点の節点番号と 0
要素 e_2 の 3 節点の節点番号と 0
　⋮
要素 e_m の 3 節点の節点番号と 0
境界に属する辺 Q_1 とラベル
　⋮
境界に属する辺 Q_N とラベル

なお、`square()`[16] を使うと、分割まで込めて菊地 [1] の例と同じように出来る。

poissno-kikuchi-square.edp[17]

```
// poisson-kikuchi-square.edp
// http://nalab.mind.meiji.ac.jp/~mk/program/fem/poisson-kikuchi-square.edp
// 菊地文雄，有限要素法概説，サイエンス社

int m=10;
mesh Th=square(m,m);
// plot(Th,wait=true,ps="Th-square.eps");
savemesh(Th,"Th-square.msh");
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,2,3)(g2*v)
 +on(1,4,u=g1);
plot(u,wait=1,ps="poisson-kikuchi-square.eps");

//3 次元鳥瞰図
//real [int] levels =0.0:0.01:1.0;
//plot(u,dim=3,viso=levels,fill=true,wait=true);
```

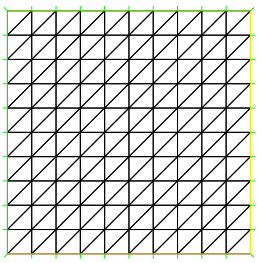


図 6: $m = 10$ の時の三角形分割の様子

[16]square は正方形という意味であるが、`square(m,n,[a+(b-a)*x,c+(d-c)*y])` とすると、長方形 $[a, b] \times [c, d]$ を分割出来る。

B　2次元 Laplacian の固有値問題を FreeFem++ で解く

(ここは書きなおす必要があることが判明したけれど、しばらくは実行する時間が取れない…書いてあることを信用しないように。)

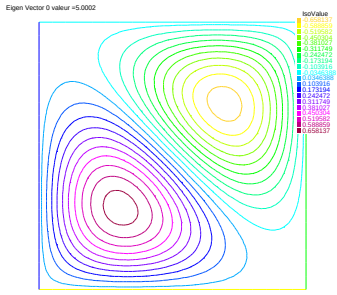
F. Hecht, O. Pironneau, FreeFem++ a software to solve PDE[18] に分かりやすい解説がある(リンク切れ…と思ったら、マニュアルに入った…と安心していたらなくなった)。次に掲げるプログラムはそこに載っているものである。

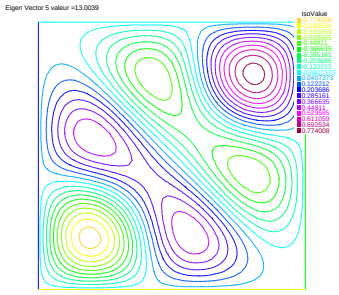
LaplacianEigenvalues.edp

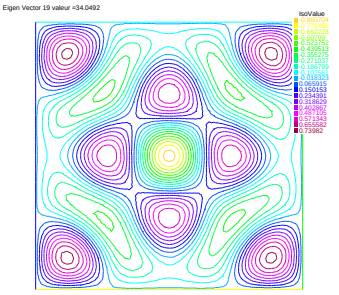
```
verbosity=10 ;
mesh Th=square(20,20,[pi*x,pi*y]);
fespace Vh(Th,P2);
Vh u1,u2;
real sigma = 20;  // value of the shift
                  // OP = A - sigma B; // the shifted matrix
varf op(u1,u2)= int2d(Th)( dx(u1)*dx(u2) + dy(u1)*dy(u2) - sigma* u1*u2 )
             + on(1,2,3,4,u1=0) ; // Boundary condition
varf b([u1],[u2]) = int2d(Th)( u1*u2 ); // no Boundary condition
matrix OP= op(Vh,Vh,solver=Crout,factorize=1);
                              // crout solver because the matrix in not positive
matrix B= b(Vh,Vh,solver=CG,eps=1e-20);
// important remark:
// the boundary condition is make with exact penalisation:
// we put 1e30=tgv on the diagonal term to lock the degre of freedom.
// So take dirichlet boundary condition just on a variationnal form
// and not on b variationnanl form.
// because we solve w=OP^-1*B*v

int nev=20;         //   number of computed eigen valeu close to sigma
real[int] ev(nev); //   to store the nev eigenvalue
Vh[int] eV(nev);   //   to store the nev eigenvector

int k=EigenValue(OP,B,sym=true,sigma=sigma,value=ev,vector=eV,
                    tol=1e-10,maxit=0,ncv=0) ;
                              // return the number of computed eigenvalue
for (int i=0;i<k;i++) {
  u1=eV[i];
  real gg = int2d(Th)(dx(u1)*dx(u1) + dy(u1)*dy(u1));
  real mm= int2d(Th)(u1*u1);
  cout<<"----"<< i<<""<<ev[i]<<"err="
      <<int2d(Th)(dx(u1)*dx(u1) + dy(u1)*dy(u1) - (ev[i])*u1*u1)
      << " --- "<<endl;
  plot(eV[i],cmm="Eigen Vector "+i+" valeur =" + ev[i] ,wait=1,value=1);
}
```







なお、8 行目の `+on(1,2,3,4,u=0)` を削除すると、Dirichlet 境界条件の代わりに、自然境界条件である Neumann 境界条件となる。

[18] https://www.ljll.math.upmc.fr/hecht/ftp/CIMPA/CIMPA-Guadeloupe-FF.pdf

$$-\triangle u = \lambda u \quad \text{in } \Omega, \quad u = 0 \quad (\text{on } \partial\Omega)$$

の両辺に試験関数 v をかけて積分し、部分積分すると、

$$\iint_\Omega \nabla u \cdot \nabla v \, dx\, dy = \lambda \int_\Omega uv \, dx\, dy.$$

すなわち

$$\iint_\Omega (u_x v_x + u_y v_y) \, dx\, dy = \lambda \iint_\Omega uv \, dx\, dy.$$

これを有限要素法で離散化すると、(行列の) 一般化固有値問題と呼ばれる

$$A\boldsymbol{u} = \lambda B\boldsymbol{u}$$

という形をした方程式が得られる。ここで B は正値対称行列、A は実対称行列である。

「シフト法で解く」と簡単に書いてあるだけで、詳しいことは書いてない。次のようなことかと推測する。一般化固有値に近いと考えられる実数 σ を与えたとき、$A\boldsymbol{u} = \lambda B\boldsymbol{u}$ の両辺から $\sigma B\boldsymbol{u}$ を引くと、

$$(A - \sigma B)\boldsymbol{u} = (\lambda - \sigma) B\boldsymbol{u}.$$

ゆえに

$$(A - \sigma B)^{-1} B\boldsymbol{u} = \frac{1}{\lambda - \sigma}\boldsymbol{u}.$$

これは $\boldsymbol{u}$ が、行列 $(A - \sigma B)^{-1}$ の、固有値 $\dfrac{1}{\lambda - \sigma}$ に属する固有ベクトルであることを示す。ゆえに行列 $(A - \sigma B)^{-1} B$ について冪乗法を実行すれば、絶対値が最大であるような固有値 $\mu = \dfrac{1}{\lambda - \sigma}$ が得られる。

細かいことだが、重要な注意がある。ここでは square() を用いてメッシュ分割をしているが、それを用いずに、自分で境界曲線を定義してメッシュ分割をすると、真の固有値が重複 (縮重) する場合であっても、数値計算で得られる近似固有値は重複 (縮重) しない。これは自動メッシュ分割で得られる三角形分割は、正方形の二面体群 (合同変換全体のなす群) D_4 の対称性を持たないため、近似固有値が重複しないためである。

C FreeFem++ のマニュアル§3.9 から: Stokes方程式の cavity flow

```
// stokes.edp (in the manual, chapter 3)
int n=3;
mesh Th=square(10*n,10*n);
plot(Th, wait=1, ps="stokes-p1b-mesh.eps");
fespace Uh(Th,P1b); Uh u,v,uu,vv;
fespace Ph(Th,P1); Ph p,pp;
solve stokes([u,v,p],[uu,vv,pp]) =
int2d(Th)(dx(u)*dx(uu)+dy(u)*dy(uu) + dx(v)*dx(vv)+ dy(v)*dy(vv)
+ dx(p)*uu + dy(p)*vv + pp*(dx(u)+dy(v)))
+ on(1,2,4,u=0,v=0) + on(3,u=1,v=0);
plot([u,v],p,wait=1,ps="stokes-p1b-solution.eps");
```

$$\boldsymbol{u}_h = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \leftarrow \begin{pmatrix} \mathtt{u} \\ \mathtt{v} \end{pmatrix}, \quad \boldsymbol{v}_h = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \leftarrow \begin{pmatrix} \mathtt{uu} \\ \mathtt{vv} \end{pmatrix}, \quad p_h \leftarrow \mathtt{p}, \quad q_h \leftarrow \mathtt{pp}, \quad x_1 \leftarrow \mathtt{x}, \quad x_2 \leftarrow \mathtt{y}$$

とすると、

$$\iint_{\Omega_h} \left(\frac{\partial u_1}{\partial x_1}\frac{\partial v_1}{\partial x_1} + \frac{\partial u_1}{\partial x_2}\frac{\partial v_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1}\frac{\partial v_2}{\partial x_1} + \frac{\partial u_2}{\partial x_2}\frac{\partial v_2}{\partial x_2} \right) dx\,dy$$
$$+ \iint_{\Omega_h} \left(\frac{\partial p_h}{\partial x_1} v_1 + \frac{\partial p_h}{\partial x_2} v_2 \right) dx\,dy + \iint_{\Omega_h} q_h \left(\frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2} \right) dx\,dy = 0,$$

すなわち

$$a(\boldsymbol{u}_h, \boldsymbol{v}_h) + (\nabla p_h, \boldsymbol{v}_h) + (q_h, \nabla \cdot \boldsymbol{u_h}) = 0 \quad (\forall (\boldsymbol{v}_h, q_h)).$$

これは

$$\begin{cases} a(\boldsymbol{u}_h, \boldsymbol{v}_h) + (\nabla p_h, \boldsymbol{v}_h) = 0 \quad (\forall \boldsymbol{v}_h), \\ (q_h, \nabla \cdot \boldsymbol{u_h}) = 0 \quad (\forall q_h) \end{cases}$$

と同値である。

P1b/P1 を採用した結果は、図 7 となる。

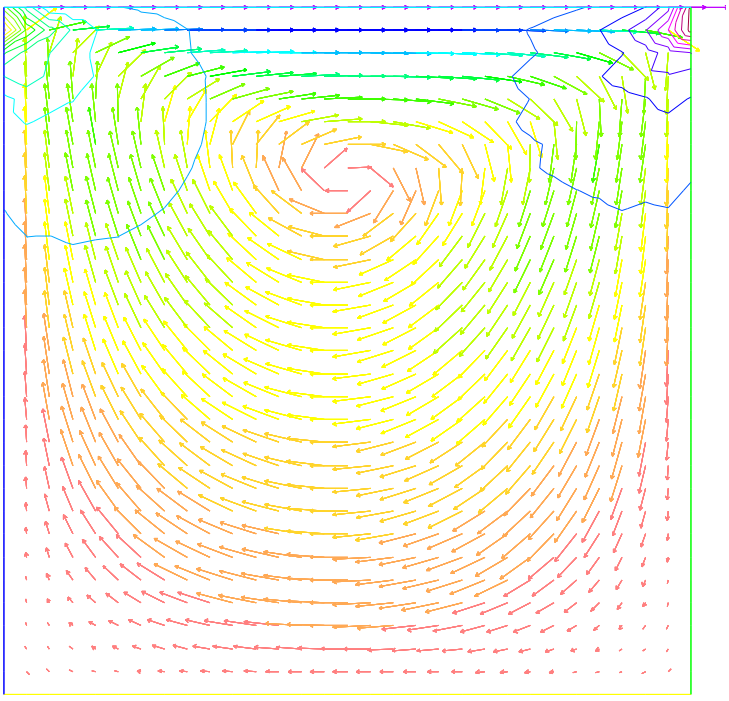


図 7: stokes-p1b.edp の結果

P1/P1 にすると上手く計算できないことは常識とされているが、自分でプログラムを書く場合、なかなかそこまで試す気にはなれない。しかし FreeFem++ でそれをするのは簡単である (図 8)。

P2/P1 は、良く知られているようにきちんと解ける (図 9)。

他にも、悪い冗談のようだが P2/P2 などが気軽に試せる。

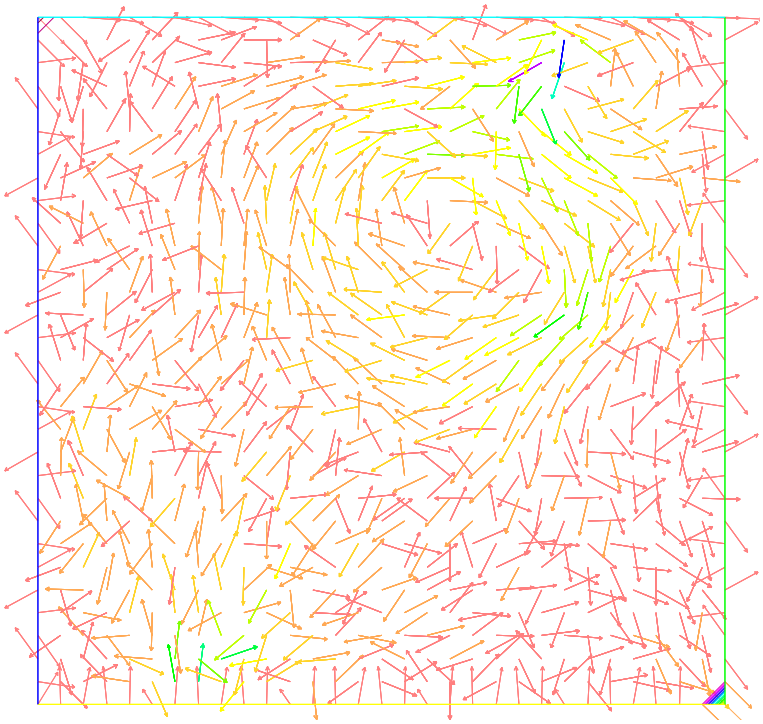


図 8: もしも P1/P1 とすると…なんだ、これは ?!

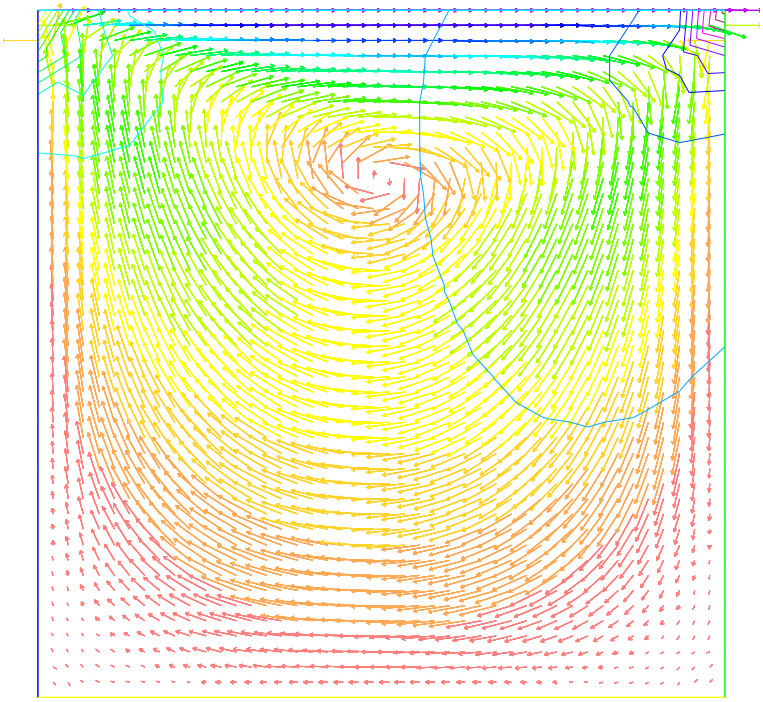


図 9: P2/P1 (分割は粗くした)

D　熱方程式に対する後退 Euler 法

内部で熱が発生する (or 熱が吸収される) 場合の熱方程式の初期値境界値問題

$$u_t(x,t) = \triangle u(x,t) + f(x) \quad ((x,t) \in \Omega \times (0,\infty)), \tag{5a}$$

$$u(x,t) = g_1(x) \quad ((x,t) \in \Gamma_1 \times (0,\infty)), \tag{5b}$$

$$\frac{\partial u}{\partial n}(x,t) = g_2(x) \quad ((x,t) \in \Gamma_2 \times (0,\infty)), \tag{5c}$$

$$u(x,0) = u_0(x) \quad (x \in \overline{\Omega}) \tag{5d}$$

を考える。

変数 t については (後退) 差分近似する。すなわち $t_n := n\Delta t$, $u^n := u(\cdot, t_n)$ とおいて、

$$\frac{\partial u}{\partial t}(\cdot, t_n) \fallingdotseq \frac{u^n - u^{n-1}}{\Delta t}$$

という近似を採用する。

弱形式は

$$\left(\frac{u^n - u^{n-1}}{\Delta t}, v\right) + \langle u^n, v\rangle - (f, v) - [g_2, v] = 0.$$

すなわち

$$\big(u^{n+1}, v\big) - (u^n, v) + \Delta t\langle u^{n+1}, v\rangle - \Delta t(f, v) - \Delta t[g_2, v] = 0.$$

既に u^n が “分かっている” とき、これは u^{n+1} についての Poisson 方程式もどきに対する弱形式であり、これまでとほぼ同様にして解くことが出来る。

heatB.edp[19]

```
// heatB.edp
// http://nalab.mind.meiji.ac.jp/~mk/program/fem/heatB.edp
// 菊地文雄，有限要素法概説，サイエンス社の Poisson 方程式の問題の非定常版
// 時刻については後退 Euler 法（陰解法）
int i,m=10;
real Tmax=10, tau=0.01, t;
// コメント・アウトすると、m, tau, theta が実行時に変更できるようになる
// cout << "m dt theta: ";
// cin >> m >> tau >> theta;
// cout << "m=" << m << ", tau=" << tau << ", theta=" << theta << endl;

mesh Th=square(m,m);
plot(Th,wait=true);
fespace Vh(Th,P1);
func f=1;
func g1=0;
func g2=0;
func u0=sin(pi*x)*sin(pi*y);
Vh u=u0,uold,v;
plot(u,cmm="t=0",wait=1);
problem heat(u,v,init=i)=
  int2d(Th)(u*v+tau*(dx(u)*dx(v)+dy(u)*dy(v)))
 -int2d(Th)(tau*f*v)
 -int1d(Th,2,3)(tau*g2*v)
 -int2d(Th)(uold*v)
 +on(1,4,u=g1);
for (i=0;i<Tmax/tau;i++) {
  uold=u;
  t=(i+1)*tau;
  heat;
  plot(u,cmm="t="+t,wait=0); // ps="heat"+i+".ps" として保存することも可能
}
plot(u,wait=1);
```

ループの制御変数を i という変数にして、problem に ,init=i と書き足すのがミソ。最初は i が 0 であるので、init が False であるが、それ以降は $i \neq 0$ であるので、init が True である、つまり LU 分解済みだ、と指示するわけである。

時間が経過すると、定常解に収束する (Poisson 方程式の解に近く) 様子が見える。