応用数値解析特論 第 8 回

～プログラミング言語 FreeFem++ (1)～

かつらだ まさし
桂田 祐史
http://nalab.mind.meiji.ac.jp/~mk/ana/

2020 年 11 月 16 日

目次

おわび

まったくの私事ですが、転んで左肩を骨折してしまいました。現在、片手でしかタイプ出来ません。そのため資料作成が遅れてしまいました。

当初の予定まで到達していませんが、授業をやらないよりはやる方が良い、と考えて、不十分な内容ではありますが、講義します。

なお、11 月 16 日 12:30–13:30 のオフィスアワーはお休みにさせてもらいます。

8 FreeFem++言語

8.1 はじめに

前回 FreeFem++ のインストール手順と簡単な解説を行った。

8 FreeFem++言語

8.1 はじめに

前回 FreeFem++ のインストール手順と簡単な解説を行った。

FreeFem++ は有限要素法によって微分方程式の数値シミュレーションを行うためのソフトウェアであり、言語処理系である。
今のところインタープリターである (その点は MATLAB や Python と似ている)。

8 FreeFem++言語

8.1 はじめに

前回 FreeFem++ のインストール手順と簡単な解説を行った。

FreeFem++ は有限要素法によって微分方程式の数値シミュレーションを行うためのソフトウェアであり、言語処理系である。
今のところインタープリターである (その点は MATLAB や Python と似ている)。

今回は、プログラミング言語としての FreeFem++を説明する (マニュアルを見ても良く分からない — 少なくとも私は)。

8 FreeFem++言語

8.1 はじめに

前回 FreeFem++ のインストール手順と簡単な解説を行った。

FreeFem++ は有限要素法によって微分方程式の数値シミュレーションを行うためのソフトウェアであり、言語処理系である。
今のところインタープリターである (その点は MATLAB や Python と似ている)。

今回は、プログラミング言語としての FreeFem++を説明する (マニュアルを見ても良く分からない — 少なくとも私は)。

FreeFem++のことを「有限要素法専用ツール」と考える人もいる。確かに有限要素法に便利な命令が組み込まれているが、それ以外の目的のプログラミングに必要な機能も十分に備わっている (実際、有限体積法や差分法のプログラムも記述可能である)。効率を度外視すれば、C のようなプログラミング言語で出来ることは FreeFem++でも出来る、と考えよう。

8 FreeFem++言語

8.1 はじめに

前回 FreeFem++ のインストール手順と簡単な解説を行った。

FreeFem++ は有限要素法によって微分方程式の数値シミュレーションを行うためのソフトウェアであり、言語処理系である。
今のところインタープリターである (その点は MATLAB や Python と似ている)。

今回は、プログラミング言語としての FreeFem++を説明する (マニュアルを見ても良く分からない — 少なくとも私は)。

FreeFem++のことを「有限要素法専用ツール」と考える人もいる。確かに有限要素法に便利な命令が組み込まれているが、それ以外の目的のプログラミングに必要な機能も十分に備わっている (実際、有限体積法や差分法のプログラムも記述可能である)。効率を度外視すれば、C のようなプログラミング言語で出来ることは FreeFem++でも出来る、と考えよう。

文法は、C++に似ている (ゆえに C にも似ている)。C しか知らない人は、C++のストリームを使った入出力 (cout, cin の利用) だけは調べておくことを勧める。
参考: FreeFem++ は C++ で記述されている。

8 FreeFem++言語

8.1 はじめに

前回 FreeFem++ のインストール手順と簡単な解説を行った。

FreeFem++ は有限要素法によって微分方程式の数値シミュレーションを行うためのソフトウェアであり、言語処理系である。
今のところインタープリターである (その点は MATLAB や Python と似ている)。

今回は、プログラミング言語としての FreeFem++を説明する (マニュアルを見ても良く分からない — 少なくとも私は)。

FreeFem++のことを「有限要素法専用ツール」と考える人もいる。確かに有限要素法に便利な命令が組み込まれているが、それ以外の目的のプログラミングに必要な機能も十分に備わっている (実際、有限体積法や差分法のプログラムも記述可能である)。効率を度外視すれば、C のようなプログラミング言語で出来ることは FreeFem++でも出来る、と考えよう。

文法は、C++に似ている (ゆえに C にも似ている)。C しか知らない人は、C++のストリームを使った入出力 (cout, cin の利用) だけは調べておくことを勧める。
参考: FreeFem++ は C++ で記述されている。

マニュアル Hect[1] は事例集の性格が強く、言語仕様は整理した形では載っていない。以下の説明は、個人的なノートである桂田 [2] に基づく。

8.1 はじめに　基本的な Poisson 方程式のプログラム

```
// poisson.edp
// 境界の定義（単位円），いわゆる正の向き
border Gamma(t=0,2*pi) { x=cos(t); y=sin(t); }
// 三角形要素分割を生成（境界を 50 に分割）
mesh Th = buildmesh(Gamma(50));
plot(Th,wait=true); // plot(Th,wait=true,ps="Th.eps");
// 有限要素空間は P1（区分的１次多項式）要素
real [int] levels =-0.012:0.001:0.012;
fespace Vh(Th,P1);
Vh u,v;
// Poisson 方程式 -△ u=f の右辺
func f = x*y;
// 問題を解く
solve Poisson(u,v)
 = int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v))-int2d(Th)(f*v)
   +on(Gamma,u=0);
// 可視化（等高線）
plot(u,wait=true);
//plot(u,viso=levels,fill=true,wait=true);
// 可視化（3 次元）--- マウスで使って動かせる
plot(u,dim=3,viso=levels,fill=true,wait=true);
```

→ 独特の命令ばかりで、汎用のプログラミング言語の機能があることは分かりにくい。

8.2 汎用のプログラミング機能

8.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

8.2 汎用のプログラミング機能

8.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

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

8.2 汎用のプログラミング機能

8.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

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

8.2 汎用のプログラミング機能

8.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

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

8.2 汎用のプログラミング機能

8.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

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

8.2 汎用のプログラミング機能

8.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

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

8.2 汎用のプログラミング機能

8.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

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

8.2 汎用のプログラミング機能

8.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

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

8.2 汎用のプログラミング機能

8.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

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

8.2 汎用のプログラミング機能

8.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

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

8.2 汎用のプログラミング機能

8.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

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

8.2 汎用のプログラミング機能

8.2.1 C 言語と良く似ているところ

改めて数えるととても多い。

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

他にもあるだろう…

8.2.2 データ型

8.2.2 データ型

- 整数を表すための int がある (C 言語の `int` に相当)

8.2.2 データ型

- 整数を表すための int がある (C 言語の `int` に相当)
- 実数を表すための real がある (C 言語の `double` に相当)

8.2.2 データ型

- 整数を表すための int がある (C 言語の `int` に相当)
- 実数を表すための real がある (C 言語の `double` に相当)
- 複素数を表すための complex がある (C 言語の `complex` に相当, 実部・虚部が `double`)

8.2.2 データ型

- 整数を表すための int がある (C 言語の `int` に相当)
- 実数を表すための real がある (C 言語の `double` に相当)
- 複素数を表すための complex がある (C 言語の `complex` に相当, 実部・虚部が `double`)
- 論理を表すための bool がある (C 言語の `bool` に相当). `true`, `false` と言う値があるが、それぞれ 1, 0 の別名と考えて良い。
  例えば `plot(u,wait=true);` は `plot(u,wait=1);` と同じ。

8.2.2 データ型

- 整数を表すための int がある (C 言語の `int` に相当)
- 実数を表すための real がある (C 言語の `double` に相当)
- 複素数を表すための complex がある (C 言語の `complex` に相当, 実部・虚部が `double`)
- 論理を表すための bool がある (C 言語の `bool` に相当). `true`, `false` と言う値があるが、それぞれ 1, 0 の別名と考えて良い。
  例えば `plot(u,wait=true);` は `plot(u,wait=1);` と同じ。
- 文字列を表すための string がある (C++言語の `string` に相当, 日本語不可？)。

8.2.2 データ型

- 整数を表すための int がある (C 言語の `int` に相当)
- 実数を表すための real がある (C 言語の `double` に相当)
- 複素数を表すための complex がある (C 言語の `complex` に相当, 実部・虚部が `double`)
- 論理を表すための bool がある (C 言語の `bool` に相当). `true`, `false` と言う値があるが、それぞれ 1, 0 の別名と考えて良い。
  例えば `plot(u,wait=true);` は `plot(u,wait=1);` と同じ。
- 文字列を表すための string がある (C++言語の `string` に相当, 日本語不可？)。
  - 2 つの string `s1`, `s2` を、(+ 演算子を用いて) `s1+s2` で連結できる。

8.2.2 データ型

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

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

8.2.2 データ型

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

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

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

8.2.3 配列型　1 次元

1 次元配列は、C 言語に (少し) 似ている。

```
real[int] a1(3); // C で double a1[3]; とするのに似ている
for (int i=0;i<3;i++)
  a1[i]=i;

real[int] a2 = [0,1,2]; // C で double a[]={0,1,2}; とするのに似てる
real[int] a3 = 0:2; // これは少し MATLAB 風

cout << "a1=" << a1 << endl;
cout << "a2=" << a2 << endl;
cout << "a3=" << a3 << endl;
```

追記: a1 の要素数は a1.n で得られる。

8.2.3 配列型　2 次元

2 次元配列はかなり違う。要素にアクセスするには 名前 (i,j) とする？

```
real[int,int] kuku(9,9);
int i,j;
for (i=0; i<kuku.n; i++) {
  for (j=0; j<kuku.m; j++) {
      kuku(i,j)=(i+1)*(j+1);
      cout << setw(3) << kuku(i,j);
  }
  cout << endl;
}
cout << kuku << endl;

real[int,int] kuku2=[[1,2,3,4,5,6,7,8,9],
                     [2,4,6,8,10,12,14,16,18],
                     [3,6,9,12,15,18,21,24,27],
                     [4,8,12,16,20,24,28,32,36],
                     [5,10,15,20,25,30,35,40,45],
                     [6,12,18,24,30,36,42,48,54],
                     [7,14,21,28,35,42,49,56,63],
                     [8,16,24,32,40,48,56,64,72],
                     [9,18,27,36,45,54,63,72,81]];
cout << kuku2 << endl;
```

8.3 有限要素法のための機能

8.3.1 有限要素法のプログラムの構成

有限要素法のプログラムと言っても色々あるが (この科目では、熱方程式や波動方程式などの発展方程式、流体力学の方程式、固有値問題などを取り上げる)、2 次元領域における Poisson 方程式の境界値問題のプログラムは基本的と考えられる。

1. 領域の定義と領域の三角形分割
2. 有限要素空間
3. 弱形式を次のいずれかで定義して解く。
   a. `solve()`
      弱形式を与えると同時にそれを解く (弱解を求める)。
   b. `problem()`
      弱形式を与えて問題を解く関数を定義する。発展問題で便利。
   c. `varf`, `matrix`
      弱形式を与えて連立 1 次方程式を作る。

8.3.2 領域の定義と領域の三角形分割

問題を考える領域を定義し、三角形分割をすることが必要である。

8.3.2 領域の定義と領域の三角形分割

問題を考える領域を定義し、三角形分割をすることが必要である。

- 2 次元 (有界) 領域の多くは、その境界曲線を定義することで定まる。

8.3.2 領域の定義と領域の三角形分割

問題を考える領域を定義し、三角形分割をすることが必要である。

- 2 次元 (有界) 領域の多くは、その境界曲線を定義することで定まる。
- 境界曲線は `border` と言う型の変数として定義される。

8.3.2 領域の定義と領域の三角形分割

問題を考える領域を定義し、三角形分割をすることが必要である。

- 2 次元 (有界) 領域の多くは、その境界曲線を定義することで定まる。
- 境界曲線は `border` と言う型の変数として定義される。
- 三角形分割は `mesh` と言う型の変数として定義される。

8.3.2 領域の定義と領域の三角形分割

問題を考える領域を定義し、三角形分割をすることが必要である。

- 2 次元 (有界) 領域の多くは、その境界曲線を定義することで定まる。
- 境界曲線は `border` と言う型の変数として定義される。
- 三角形分割は `mesh` と言う型の変数として定義される。
- `buildmesh()` と言う関数は、各 border を何等分するか指定することで、border の囲む領域を三角形分割して、mesh 型のデータを作る。

8.3.2 領域の定義と領域の三角形分割

問題を考える領域を定義し、三角形分割をすることが必要である。

- 2 次元 (有界) 領域の多くは、その境界曲線を定義することで定まる。
- 境界曲線は `border` と言う型の変数として定義される。
- 三角形分割は `mesh` と言う型の変数として定義される。
- `buildmesh()` と言う関数は、各 border を何等分するか指定することで、border の囲む領域を三角形分割して、mesh 型のデータを作る。
- `mesh` 型のデータは、`readmesh()`, `writemesh()` と言う関数を用いて入出力できる (結果はテキスト・ファイル)。

8.3.2 領域の定義と領域の三角形分割

問題を考える領域を定義し、三角形分割をすることが必要である。

- 2 次元 (有界) 領域の多くは、その境界曲線を定義することで定まる。
- 境界曲線は `border` と言う型の変数として定義される。
- 三角形分割は `mesh` と言う型の変数として定義される。
- `buildmesh()` と言う関数は、各 border を何等分するか指定することで、border の囲む領域を三角形分割して、mesh 型のデータを作る。
- `mesh` 型のデータは、`readmesh()`, `writemesh()` と言う関数を用いて入出力できる (結果はテキスト・ファイル)。
- `mesh` 型のデータは、`plot()` により可視化できる。

8.3.2 領域の定義と領域の三角形分割

問題を考える領域を定義し、三角形分割をすることが必要である。

- 2 次元 (有界) 領域の多くは、その境界曲線を定義することで定まる。
- 境界曲線は `border` と言う型の変数として定義される。
- 三角形分割は `mesh` と言う型の変数として定義される。
- `buildmesh()` と言う関数は、各 border を何等分するか指定することで、border の囲む領域を三角形分割して、mesh 型のデータを作る。
- `mesh` 型のデータは、`readmesh()`, `writemesh()` と言う関数を用いて入出力できる (結果はテキスト・ファイル)。
- `mesh` 型のデータは、`plot()` により可視化できる。
- 矩形領域 (辺が座標軸に平行な長方形) は、`square()` と言う命令で mesh 型データが作れる (参考「FreeFem++ノート」)。

8.3.2 領域の定義と領域の三角形分割　border

円周全体を C とする

```
border C(t=0,2*pi) { x=cos(t); y=sin(t); }
```

円周の上半分、下半分を別々に Gamma1, Gamma2 と定義する

```
int C=1;
...
border Gamma1(t=0,pi)    { x=cos(t); y=sin(t); label=C; }
border Gamma2(t=pi,2*pi) { x=cos(t); y=sin(t); label=C; }
```

正方形領域 $(0,1)\times(0,1)$ の 4 つの辺 C1,C2,C3,C4 を定義

```
border C1(t=0,1) { x=t; y=0; }
border C2(t=0,1) { x=1; y=t; }
border C3(t=0,1) { x=1-t; y=1; }
border C4(t=0,1) { x=0; y=1-t; }
```

8.3.2 領域の定義と領域の三角形分割　メッシュ(mesh)

```
border C(t=0,2*pi) { x=cos(t); y=sin(t); }

int C0=1;
border Gamma1(t=0,pi)    { x=cos(t); y=sin(t); label=C0; }
border Gamma2(t=pi,2*pi) { x=cos(t); y=sin(t); label=C0; }

border C1(t=0,1) { x=t; y=0; }
border C2(t=0,1) { x=1; y=t; }
border C3(t=0,1) { x=1-t; y=1; }
border C4(t=0,1) { x=0; y=1-t; }

mesh Th1=buildmesh(C(50));
mesh Th2=buildmesh(Gamma1(25)+Gamma2(50));
mesh Th3=buildmesh(C1(20)+C2(20)+C3(20)+C4(20));

plot(Th1,wait=true,ps="Th1.eps");
plot(Th2,wait=true,ps="Th2.eps");
plot(Th3,wait=true,ps="Th3.eps");
```

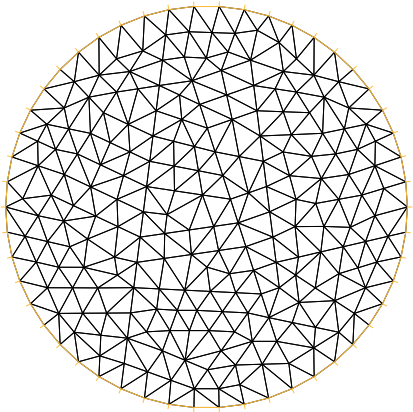


図 1: C(50)

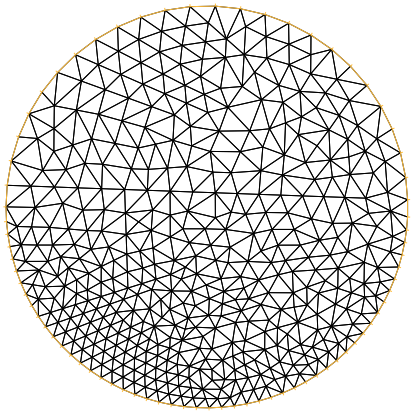


図 2: Gamma1(25)+Gamma2(50)

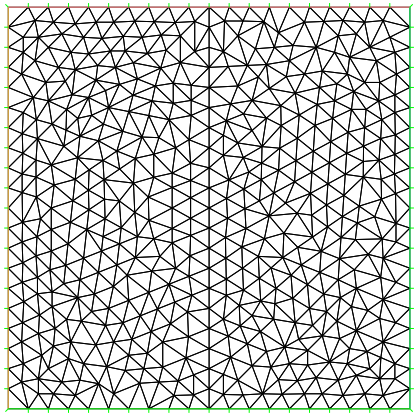


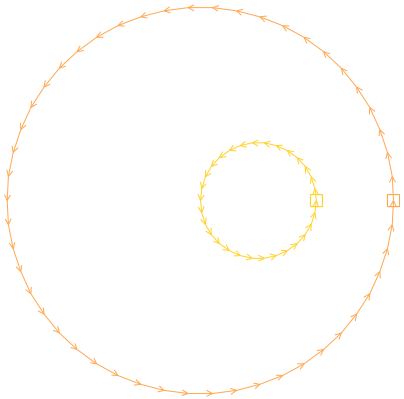
図 3: C1(20)+C2(20)+...

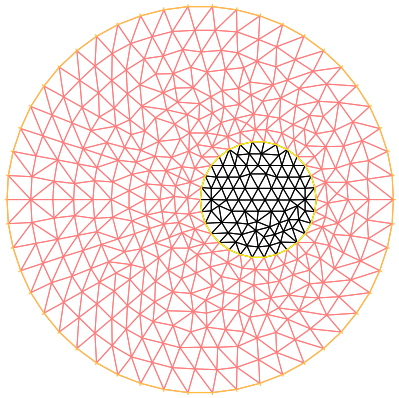
8.3.2 領域の定義と領域の三角形分割　メッシュ(mesh)

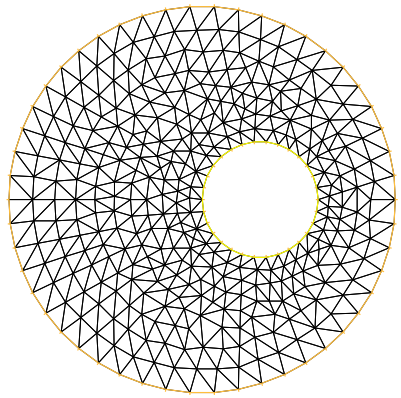
有限個の Jordan 閉曲線で囲まれた多重連結領域を三角形分割することもできる。

sampleMesh.edp

```
border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
border b(t=0,2*pi){ x=0.3+0.3*cos(t); y=0.3*sin(t);label=2;}
plot(a(50)+b(+30),wait=true,ps="border.eps");
mesh ThWithoutHole = buildmesh(a(50)+b(+30));
mesh ThWithHole = buildmesh(a(50)+b(-30));
plot(ThWithoutHole,wait=1,ps="Thwithouthole.eps");
plot(ThWithHole,wait=1,ps="Thwithhole.eps");
```







8.3.2 領域の定義と領域の三角形分割　メッシュ(mesh)

普通は mesh データの細かいことは見る必要がないかもしれないが…

8.3.2 領域の定義と領域の三角形分割　メッシュ(mesh)

普通は mesh データの細かいことは見る必要がないかもしれないが…

- `Th` をメッシュとするとき、`Th.nt` は三角形の数 (the number of triangles)、`Th.nv` は節点の数 (the number of vertices) である。

8.3.2 領域の定義と領域の三角形分割　メッシュ(mesh)

普通は mesh データの細かいことは見る必要がないかもしれないが…

- `Th` をメッシュとするとき、`Th.nt` は三角形の数 (the number of triangles)、`Th.nv` は節点の数 (the number of vertices) である。
- `Th`(i) は i 番目の節点 ($i = 0, 1, \ldots,$ `Th.nv` $- 1$) で、その座標は `Th`(i)`.x` と `Th`(i)`.y` である。`Th`(i)`.label` はその接点が領域内部にあるか (0)、境界にあるか (0 以外)、境界のどの部分にあるかを示す。

8.3.2 領域の定義と領域の三角形分割　メッシュ(mesh)

普通は mesh データの細かいことは見る必要がないかもしれないが…

- `Th` をメッシュとするとき、`Th.nt` は三角形の数 (the number of triangles)、`Th.nv` は節点の数 (the number of vertices) である。
- `Th`(i) は i 番目の節点 ($i = 0, 1, \ldots,$ `Th.nv` $- 1$) で、その座標は `Th`(i)`.x` と `Th`(i)`.y` である。`Th`(i)`.label` はその接点が領域内部にあるか (0)、境界にあるか (0 以外)、境界のどの部分にあるかを示す。
- `Th[`i`]` は i 番目の三角形 ($i = 0, 1, \ldots,$ `Th.nt` $- 1$)、`Th[`i`][`j`]` は i 番目の三角形の j 番目の節点 ($j = 0, 1, 2$) の全体節点番号、その節点の座標は `Th[`i`][`j`].x` と `Th[`i`][`j`].y` である。三角形の面積は `Th[`i`].area` である。

8.3.2 領域の定義と領域の三角形分割　メッシュ(mesh)

普通は mesh データの細かいことは見る必要がないかもしれないが…

- `Th` をメッシュとするとき、`Th.nt` は三角形の数 (the number of triangles)、`Th.nv` は節点の数 (the number of vertices) である。
- `Th`(i) は i 番目の節点 ($i = 0, 1, \ldots,$ `Th.nv` $- 1$) で、その座標は `Th`(i)`.x` と `Th`(i)`.y` である。`Th`(i)`.label` はその接点が領域内部にあるか (0)、境界にあるか (0 以外)、境界のどの部分にあるかを示す。
- `Th[`i`]` は i 番目の三角形 ($i = 0, 1, \ldots,$ `Th.nt` $- 1$)、`Th[`i`][`j`]` は i 番目の三角形の j 番目の節点 ($j = 0, 1, 2$) の全体節点番号、その節点の座標は `Th[`i`][`j`].x` と `Th[`i`][`j`].y` である。三角形の面積は `Th[`i`].area` である。
- 点 (x, y) を含む三角形の番号は `Th(`x`,`y`).nuTriangle` で得られる。

8.3.2 領域の定義と領域の三角形分割　メッシュ(mesh)

次のような場合に `readmesh()`, `writemesh()` は有効である。

8.3.2 領域の定義と領域の三角形分割　メッシュ(mesh)

次のような場合に `readmesh()`, `writemesh()` は有効である。

1. FreeFem++ を用いて三角形分割を行い、得られたメッシュ・データーを外部のプログラムで利用する (有限要素法の計算は自作プログラムで行う等)。

8.3.2 領域の定義と領域の三角形分割　メッシュ(mesh)

次のような場合に `readmesh()`, `writemesh()` は有効である。

1. FreeFem++ を用いて三角形分割を行い、得られたメッシュ・データーを外部のプログラムで利用する (有限要素法の計算は自作プログラムで行う等)。
2. 自作のプログラムで三角形分割を行い、そのメッシュ・データーを FreeFem++ で利用する。

8.3.2 領域の定義と領域の三角形分割　メッシュ(mesh)

次のような場合に `readmesh()`, `writemesh()` は有効である。

1. FreeFem++ を用いて三角形分割を行い、得られたメッシュ・データーを外部のプログラムで利用する (有限要素法の計算は自作プログラムで行う等)。
2. 自作のプログラムで三角形分割を行い、そのメッシュ・データーを FreeFem++ で利用する。

`readmesh()`, `writemesh()` で入出力されるデータのフォーマットについては、「FreeFem++ノート §6.2 mesh ファイルの構造」を見よ。

8.3.3 有限要素空間

既に定義しておいた mesh 型データと、要素の種類を表す名前 (P1, P2, ...) を用いて、有限要素空間 (この講義では $\widetilde{X}$ のように表したが、V_h などの記号で表すことが多い) を定義する。
fespace 型の変数は関数空間を表すことになる。

例えば Th と言う mesh 型の変数があるとき、

```
fespace Vh(Th,P1);
```

とすると有限要素空間 Vh が定義される。
これは型名で

```
Vh u,v;
```

として変数 u, v が定義できる。これらが個々の関数を表す。
(数学語では $u, v \in V_h$ という調子)

(**注**　これまでの授業で、三角形要素分割して、区分的 1 次多項式 (P1 要素) しか紹介しなかったが、Poisson 方程式の境界値問題以外では、他の要素 (P2, P2Morley,...) が必要になることがある。)

参考文献

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

[2] 桂田祐史：FreeFEM++ ノート, http://nalab.mind.meiji.ac.jp/~mk/labo/text/freefem-note.pdf (2012〜).