FreeFem++ Manual

version 1.45-5 (Under construction)

http://www.freefem.org
http://www.ann.jussieu.fr/~hecht/freefem++.htm

F. Hecht [1], O. Pironneau [2],
Université Pierre et Marie Curie,
Laboratoire Jacques-Louis Lions,
175 rue du Chevaleret ,PARIS XIII

K. Ohtsuka [3],
Hiroshima Kokusai Gakuin University, Hiroshima, Japan

February 24, 2005

[1]mailto:hecht@ann.jussieu.fr
[2]mailto:pironneau@ann.jussieu.fr
[3]mailto:ohtsuka@barnard.cs.hkg.ac.jp

Contents

Chapter 1

Introduction

A partial differential equation is a relation between a function of several variables and its (partial) derivatives. Many problems in physics, engineering, mathematics and even banking are modeled by one or several partial differential equations.

`Freefem` is a software to solve these equations numerically. As its name says, it is a free software (see copyright for full detail) based on the Finite Element Method. This software runs on all unix OS (with g++ 2.95.2 or better and X11R6) , on Window95, 98, 2000, NT, XP, on MacOS 9 and X.

Many phenomena involve several different fields. Fluid-structure interactions, Lorenz forces in liquid aluminium and ocean-atmosphere problems are three such systems. These require approximations of different degrees, possibly on different meshes. Some algorithms such as Schwarz' domain decomposition method also require data interpolation on different meshes within one program. `freefem++` can handle these difficulties, i.e. *arbitrary finite element spaces on arbitrary unstructured and adapted meshes.*

The characteristics of `freefem++` are:

- A large variety of finites elements : linear and quadratic Lagrangian elements, discontinuous P1 and Raviart-Thomas elements, elements of a non-scalar type, mini-element, ...
- Automatic interpolation of data on different meshes to an over mesh, store the interpolation matrix.
- Linear problems description (real or complex) thanks to a formal variational form, with access to the vectors and the matrix if needed.
- Includes tools to define discontinuous Galerkin formulations (please refer to the following keywords: “jump”, “average”, “intalledges”).
- Analytic description of boundaries. When two boundaries intersect, the user must specify the intersection points.

- Automatic mesh generator, based on the Delaunay-Voronoï algorithm. Inner points density is proportional to the density of points on the boundary [6].
- Metric-based anisotropic mesh adaptation. The metric can be computed automatically from the Hessien of a solution [7].
- Solvers : LU, Cholesky, Crout, CG, GMRES, UMFPACK linear solver, eigenvalue and eigenvector computation.
- Online graphics, C++-like syntax.
- Many examples: Navier-Stokes, elasticity, Fluid structure, Schwarz's domain decomposition method, Eigen value problem, residual error indicator, ...
- Parallel version using `mpi`

1.1 History

The project has evolved from `MacFem, PCfem`, written in Pascal. The first C version was `freefem 3.4`; it offered mesh adaptation on a single mesh. A thorough rewriting in C++ led to `freefem+ 1.2.10`, which also included interpolation over multiple meshes (functions defined on one mesh can be used on any other mesh). Implementing the interpolation from one unstructured mesh to another was not easy because it had to be fast and non-diffusive; for each point, one had to find the containing triangle. This is one of the basic problems of computational geometry (see Preparata & Shamos[13] for example). Doing it in a minimum number of operations was a challenge. Our implementation was $O(n \log n)$ and based on a quadtree.

We are now introducing `freefem++` , an entirely new program written in C++ and based on `bison` for a more adaptable freefem language.

The freefem language allows for a quick specification of any partial differential system of equations. The language syntax of `freefem++` is the result of a new design which makes use of the STL [21], templates and `bison` for its implementation. The outcome is a versatile software in which any new finite element can be included in a few hours; but a recompilation is then necessary. The library of finite elements available in `freefem++` will therefore grow with the version number. So far we have linear and quadratic Lagrangian elements, discontinuous P1 and Raviart-Thomas elements.

1.2 Getting Started

We explain how `freefem++` solve the problem **Poisson**; For a given function $f(x,y)$, find a function $u(x,y)$ satisfying

$$\begin{aligned} -\Delta u(x,y) &= f(x,y) \quad \text{if } (x,y) \text{ is in } \Omega, \qquad \Delta u = \partial^2 u/\partial x^2 + \partial^2 u/\partial y^2, && (1.1) \\ u(x,y) &= 0 \quad \text{if } (x,y) \text{ is on } \partial\Omega && (1.2) \end{aligned}$$

The following example shows `freefem++` program solving u when $f(x,y) = xy$ (see 5th line) and Ω is the unit disk. The boundary $C = \partial\Omega$ is

$$C = \{(x,y) |\ x = \cos(t),\ y = \sin(t),\ 0 \le t \le 2\pi\} \quad \text{(see 1st line)}$$

The domain will be on the left side of the oriented boundary by the parameter t. As illustrated in Fig. 1.2, we can see the isovalue of u by using **`plot`** (see 13th line).

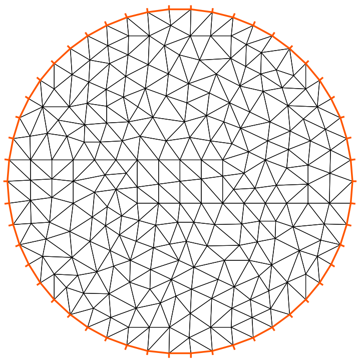


Figure 1.1: mesh `Th` by `build(C(50))`

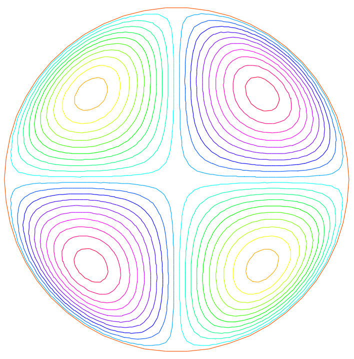


Figure 1.2: isovalue by `plot(u)`

Example 1

```
 1: border C(t=0,2*pi){x=cos(t); y=sin(t);label=1;}
 2: mesh Th = buildmesh (C(50));
 3; fespace Vh(Th,P2);
 4: Vh u,v;
 5: func f= x*y;
 6: problem Poisson(u,v,solver=LU) =
 7:    int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v))             //    bilinear part
 8:    - int2d(Th)( f*v)                               //    right hand side
 9:    + on(1,u=0)  ;                       //    Dirichlet boundary condition
10:
11: real cpu=clock();
12: Poisson;                                            //    SOLVE THE PDE
13: plot(u);
14: cout << " CPU time = " << clock()-cpu << endl;
```

1.2.1 FEM by `freefem++`

The example shows `freefem++` covers easily all standard step in FEM (finite element method). We explain how they are done by `freefem++` in a step-by-step manner.

Step1: Mesh Generation

1st line: the boundary Γ are described analytically (by opposition to CSG) as stated before. In the case $\Gamma = \sum_{j=0}^{J} \Gamma_j$ with curves Γ_j, then the user must specify the intersection points in case two boundaries intersect. By the use of the keyword "label" such as

```
border Gamma1(t=a1,b1) { x=···; y=··· ;label=1; }
  ⋮          ⋮              ⋮        ⋮
border GammaJ(t=aJ,bJ) { x=···; y=···;label=1; }
```

the user can refer to Γ by the number "1". (examples are in Section 3.9).
2nd line: the triangulation $\mathcal{T}_h$ of Ω is automatically generated by "`buildmesh`(C(50))" giving 50 points on C as in Fig. 1.1. Automatic mesh generation is based on the Delaunay-Voronoi algorithm. Refinement of the mesh are done by increasing the points on Γ, for example, "`buildmesh`(C(100))", because inner vertices are determined by the density of points on the boundary.
The symbol $\mathcal{T}_h$ (`Th` in `freefem++`) shows a family $\{T_k\}_{k=1,\cdots,n_t}$ of triangles in Fig. 1.1 with the size h of the mesh. Here n_t stands for the number of triangules in $\mathcal{T}_h$. If Ω is not polygonal domain, a "skin" remains between the exact domain Ω and its approximation $\Omega_h = \cup_{k=1}^{n_t} T_k$. However, we notice that all corners of $\Gamma_h = \partial\Omega_h$ are on Γ.

Step2: Making finite element space

3rd line: "`fespace` Vh(Th,P2)" makes the continuous Finite Element SPACE

$$V_h(\mathcal{T}_h, P_2) = \left\{ w(x,y) \;\middle|\; w(x,y) = \sum_{k=0}^{M-1} w_k\phi_k(x,y),\ w_k \text{ are real numbers} \right\} \tag{1.3}$$

where P_2 indicate ϕ_k is a polynomial of degree ≤ 2, that is, in each T_k,

$$\phi_k(x,y) = \alpha_1 + \alpha_2 x + \alpha_3 y + \alpha_4 x^2 + \alpha_5 xy + \alpha_6 y^2$$

and the constants $\alpha_1, \cdots, \alpha_6$ are defined by its values at the vertices of T_k and their middle points that continuous in Ω. Here w_k are called the degree of freedom of w and M the number of the degree of freedom. Already `freefem++` implemented P0, P1, P2, RT0, P1nc, P1dc, P2dc, P1b, P2b elements, . The user can easily add a part of arbitrary degree elements to `freefem++` , so the available finite elements will differ with the version.

Step3: Setting the problem

4th line: "`Vh u`" declare that u is approximated through the use of the basis functions ϕ_k in V_h, that is,

$$u(x,y) \simeq u_h(x,y) = \sum_{k=0}^{M-1} u_k\phi_k(x,y)$$

5th line: the given function f is defined analytically using the keyword `func`.
6th–9th lines: the formulation of (1.1) and (1.2) are done.
Multiplying (1.1) by $v(x,y)$ and integrating the result over Ω, we have

$$-\int_\Omega v\Delta u\, dxdy = \int_\Omega vf\, dxdy$$

Then, by Green's formula, the problem Poisson is translated into finding u such that

$$a(u,v) - \ell(f,v) = 0 \qquad \text{for all } v \tag{1.4}$$

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v\, dxdy \quad \ell(f,v) = \int_\Omega fv\, dxdy \tag{1.5}$$

satisfying $v = 0$ on $\partial\Omega$. In `freefem++` the problem **Poisson** is declared by

```
Vh u,v; problem Poisson(u,v) =
```

and (1.4) is expressed by symbols **dx**(u) $= \partial u/\partial x$, **dy**(u) $= \partial u/\partial y$ and

$$\int_\Omega \nabla u \cdot \nabla v \, dxdy \longrightarrow \texttt{int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v))}$$

$$\int_\Omega f v \, dxdy \longrightarrow \texttt{int2d(Th)(f*v)} \qquad \text{(notice here, } u \text{ is unused)}$$

In `freefem++` the first and second formulas just above must be distinguished each other. Because, the linear system to be solved are created from substituting u_h for u and ϕ_i for v in (1.4),

$$A_{ij}u_j - F_i = 0 \quad i,j = 0, \cdots, M-1; \qquad F_i = \int_\Omega f\phi_i \, dxdy \tag{1.6}$$

and the solution $u_h = \sum_{j=0}^{M-1} u_j\phi_j$ must satisfy "$u_h = 0$ on $\Gamma_h \simeq C$". The matrix $A = (A_{ij})$ is called *stiffness matrix* and is modified from

$$A^0 = \{A^0_{ij}\}, \, A^0_{ij} = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \, dxdy \quad i.j = 0, \cdots, M-1 \tag{1.7}$$

to ensure $u = 0$ on C by "+**on**{1,u=0}" in 9th line. If you want use the symbol "C" such as "+**on**{C,u=0}" (9th line), then *the user do not use the keyword "label"*.
we can create directly the stiffness matrix A in (1.6) by

```
varf a(u,v) = int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v))
                    + on(C,u=0) ;
matrix A=a(Vh,Vh);                  //    stiffness matrix, see (1.7)
```

(see Example 2). The vector B in (1.6) is called *load matrix* , and also we get it:

```
varf b([v],[f]) = int2d(Th)(v*f);
matrix B=b(Vh,Vh);
```

The linear system (1.6) is solved by a Gauss LU factorisation. You can declare the solver of (1.6), for example,

```
Vh u,v; problem Poisson(u,v,solver=CG) =
```

means that (1.6) will be solved by Conjugate Graduent method.

Step4: Solving and visualization

11th line: the current time is stored into the real-valued variable `cpu`.
12th line: the problem is solved by calling its name.
13th line: the visualization is done as illustrated in Fig. 1.2 (see Section 5.1 for zoom, postscript and other commands).
14th line: the time in calculation is outputed into your console (= default of standard output) using C++-like syntax. The user need not study C++ for using `freefem++` , because C++-like syntax is used for input/output, loops, flow-controls etc.

1.2.2 Features of `freefem++`

The language it defines is typed, polymorphic and reentrant with macro generation (see 7.11). Every variable must be typed and declared in a statement; each statement separated from the next by a semicolon '`;`'.
For purposes of explanation, we used $\mathcal{T}_h$ (`Th`), V_h (`Vh`), unknown function u (`u`), test functions v (`v`) and the problem **Poisson**, etc. (the term inside the parentheses are symbols in `freefem++` programming), but you can use *any name except reserved words and names already used*. Reserved words are shown in blue. `pi, x, y, label, solver` are reserved variables. It is allowed (although not advisable) to redefine these variables, so they will not be highlighted again in the following example programs.

In each step, the independence in `freefem++` programming is very high as stated below.

- For example, by changing 1st and 2nd lines as following, we can solve (1.1) and (1.2) in L-shape domain with $\Gamma = \sum_{j=1}^{6} \Gamma_j$.

```
border G1(t=0,1){x=t;y=0;label=1;}                     //    Γ1
border G2(t=0,0.5){x=1;y=t;label=1;}                   //    Γ2
border G3(t=0,0.5){x=1-t;y=0.5;label=1;}               //    Γ3
border G4(t=0.5,1){x=0.5;y=t;label=1;}                 //    Γ4
border G5(t=0.5,1){x=1-t;y=1;label=1;}                 //    Γ5
border G6(t=0,1){x=0;y=1-t;label=1;}                   //    Γ6
mesh Th = buildmesh ( G1(6)+G2(4)+G3(4)+G4(4)+G5(4)+G6(6));
```

- In Step 3, you can control where the solution will be approximated. If you write "`Vh(Th,P1);`" in 3rd line, you can get P_1-approximation. The machine time by P_1-element will be faster than P_2-element and the storage is less.

- In Step 4, you can change the equation and boundary conditions easily. For example, if you want solve

$$\begin{aligned} -\mathrm{div}(k(x,y)\nabla u(x,y)) &= f(x,y) \quad \text{in } \Omega \\ u(x,y) &= 0 \qquad \text{if } (x,y) \text{ on } \Gamma \end{aligned}$$

  you only write as follows

```
func f= ···;
func k= 1+sin(2*pi*x)*cos(2*pi*y);
problem Poisson(u,v) =
   int2d(Th)(k*(dx(u)*dx(v) + dy(u)*dy(v)))
   - int2d(Th)( f*v)
   + on(1,u=0)  ;
```

  The user can use FE-function as the given function f (see Section 2.6.2), for example, obtained function `u` in Example 1. In `freefem++` programming, the easy reuse of the obtained results is important feature.

- The user can easily compare between mathematical modelling and `freefem++` program.

1.3 Projection or Interpolation

For a finite element space V_h, P_1-projection Π_h is defined by

$$\Pi_h f = f(q^1)\phi_1 + f(q^2)\phi_2 + \cdots + f(q^{n_v})\phi_{n_v}$$

for all continuous functions f. In `freefem++` we can easily create the projection $f_h (= \texttt{fh})$ by

    Vh fh = $f(x,y)$;

Π_h is also called P_1-interpolate.

1.4 Matrix and Vector

Here, we show how to get the stiffness matrix using `freefem++` The first command `varf` is to define the *variational formula*.

Example 2 *Here we solve the same problem (1.1) and (1.2) using matrix. For purposes of explanation, we chage mesh size and use P_1-element:*

```
border C(t=0,2*pi) { x = cos(t); y = sin(t); }
mesh Th = buildmesh(C(7));        //    changed from Example 1
fespace Vh(Th,P1);
Vh u,v,f,F;
varf a(u,v) = int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v))
             + on(C,u=0) ;                       //    see (1.7)
varf b([v],[f]) = int2d(Th)(v*f);

f = x*y;                     //    interpolate (x,y) <-- x*y function
matrix A=a(Vh,Vh);             //    stiffness matrix, see (1.6)
matrix B=b(Vh,Vh);
F[]=B*f[];                       //    load vector, see (1.6)
cout << "F=" << F[] << endl;
cout <<"A="<< A << endl;
u[]=A^-1*F[];                  //    solve AU_h = F, see (1.6)
plot(u);
```

We get the mesh $\mathcal{T}_h = \{T_1, \cdots, T_7\}$ (see Fig. 1.3).

Note 1 *In what follows, we denote the vertices by q^i, $i = 1, \cdots, 8$, the number of vertices by n_v, the number of triangles by n_t. For each triangle $T_k \in \mathcal{T}_h$, we index the vertices by q^{k_1}, q^{k_2}, q^{k_3} and denote the edges by $[q^{k_1}, q^{k_2}]$, $[q^{k_2}, q^{k_3}]$, $[q^{k_3}, q^{k_i}]$, that is, $[q^i, q^j]$ is the segment connecting q^i and q^j. We denote the number of edges $[q^i, q^j]$ by n_e for all q^i, $q^j\partial\Omega_h$, $\Omega_h = \sum_{k=1}^{7} T_k$. Here $n_v = 8$, $n_t = 7$, $n_e = 7$.*

The function v in "`Vh`" is expressed

$$v(x) = v_1\varphi_1(x) + \cdots + v_{n_v}\varphi_{n_v}(x)$$

using the hat functions *φ_j, $j = 1, \cdots, n_v$ (see Fig. 1.4). Here the j-th hat function φ_j associated with j-th vertex q^j is defined in the following way:*

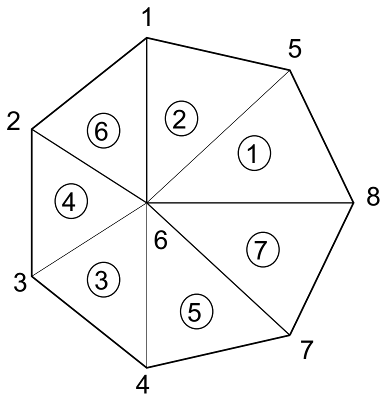


Figure 1.3: mesh `Th`

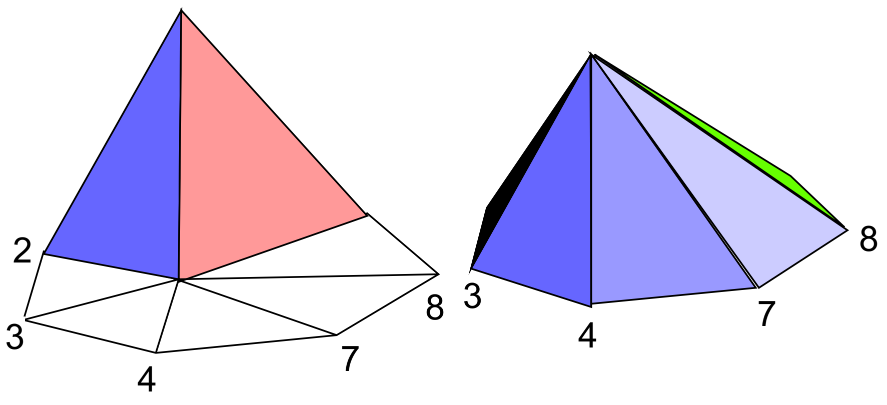


Figure 1.4: Graph of ϕ_1 (left hand side) and ϕ_6

1. *φ_j is continuous function on Ω_h.*
2. *φ_j is linear on each triangle T_k, $k = 1, \cdots, n_t$ of "`Th`".*
3. *$\varphi_j(q^i) = \delta_{ji}$ where q^i denotes the i-th vertex, for all $i = 1, \cdots, n_v$.*

Here δ_{ij} is the Kronecker symbol.

Note 2 *Other finite element spaces in* ***`freefem++`*** *are explained in Section 4.*

Note 3 *For an element $v = v_1\phi_1 + \cdots + v_M\phi_M$ in a finite element space V_h, we get the* column vector $\{v\}$

$$\{v\} = \begin{bmatrix} v_1 \\ \vdots \\ v_M \end{bmatrix} \qquad \{v\} = \texttt{v[]} \quad \textit{in } \texttt{freefem++}$$

Theoretically, it is natural to use the finite element space

$$H^1_{0h} = \left\{ v \in V_h(\mathcal{T}_h, P_1) \,\middle|\, \phi_i(x) = 0 \quad \text{if } q^i \in \partial\Omega_h \right\}$$

Let I_Ω be the set of indices i of all internal vertices of the mesh `Vh`*. In this example, $I_\Omega = \{6\}$. The stiffness matrix A in 10th line is:*

$$\begin{array}{c|rrrrrrrr|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 10^{30} & -0.31 & 0 & 0 & -0.46 & -0.51 & 0 & 0 \\
2 & -0.31 & 10^{30} & -0.23 & 0 & 0 & -0.71 & 0 & 0 \\
3 & 0 & -0.23 & 10^{30} & -0.31 & 0 & -0.71 & 0 & 0 \\
4 & 0 & 0 & -0.31 & 10^{30} & 0 & -0.51 & -0.46 & 0 \\
5 & -0.46 & 0 & 0 & 0 & 10^{30} & -0.35 & 0 & -0.54 \\
6 & -0.51 & -0.71 & -0.71 & -0.51 & -0.35 & 3.47 & -0.35 & -0.30 \\
7 & 0 & 0 & 0 & -0.46 & & -0.35 & 10^{30} & -0.54 \\
8 & 0 & 0 & 0 & 0 & -0.54 & -0.30 & -0.54 & 10^{30}
\end{array} \tag{1.8}$$

that is

$$A_{ij} = \int_{\Omega_h} \nabla u_j \cdot \nabla u_i \quad \text{if } i \neq j,\ i = j \in I_\Omega \tag{1.9}$$

$$A_{ij} = E \qquad (E = 10^{30}) \quad \text{if } j \in I_\Omega. \tag{1.10}$$

The load matrix F^T is:

$$\begin{pmatrix} -0.020 & -0.037 & 0.037 & 0.020 & 0.064 & 0 & -0.064 & 1. \times 10^{-17} \end{pmatrix}$$

For $i \notin I_\Omega$,

$$Eu_i + \sum_{i \neq j} A_{ij} u_j = b_i$$

which means that

$$u_i = (b_i - \sum_{i \neq j} A_{ij} u_j) \times E^{-1} \simeq 10^{-30} \simeq 0$$

Mathematical results indicate that the Poisson equation with Neaumann boundary condition has not unique solution, whose weak form is same to (1.1) except the boundary condition:

$$\int_\Omega \nabla u \cdot \nabla v = \int_\Omega f v dx$$

without pernarization E. Then the stiffness matrix is created by

```
varf a(u,v) = int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v))
matrix A=a(Vh,Vh);                          //    stiffness matrix
```

and the obtained stiffness matrix is the following

$$\begin{array}{c|cccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 1.29 & -0.31 & 0 & 0 & -0.46 & -0.51 & 0 & 0 \\
2 & -0.31 & 1.26 & -0.23 & 0 & 0 & -0.71 & 0 & 0 \\
3 & 0 & -0.23 & 1.26 & -0.31 & 0 & -0.71 & 0 & 0 \\
4 & 0 & 0 & -0.31 & 1.29 & 0 & -0.51 & -0.46 & 0 \\
5 & -0.46 & 0 & 0 & 0 & 1.35 & -0.35 & 0 & -0.54 \\
6 & -0.51 & -0.71 & -0.71 & -0.51 & -0.35 & 3.47 & -0.35 & -0.30 \\
7 & 0 & 0 & 0 & -0.46 & 0 & -0.35 & 1.35 & -0.54 \\
8 & 0 & 0 & 0 & 0 & -0.54 & -0.30 & -0.54 & 1.38
\end{array}$$

The determinant of this matrix is $-1.7082274230870981 \times 10^{-9} \approx 0$ (The matrix here differ from original one by omitting from third decimal decimal point).

1.4.1 Non-homogeneous Dirichlet Condition

If we want solve the problem

$$-\Delta u = f \quad \text{in } \Omega; \qquad u = g \quad \text{on } \partial\Omega$$

We rewrite Example 1 as

```
5: func f= x*y; func g = sin(pi*x)*cos(pi*y);
6: problem Poisson(u,v,solver=LU) =
7:    int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v))    //    bilinear part
8:    - int2d(Th)( f*v)                      //    right hand side
9:    + on(1,u=g)  ;                  //    Non-homogeneous Dirichlet
```

This make the following linear system, for $i \notin I_\Omega$,

$$Eu_i + \sum_{i \neq j} A_{ij} u_j = b_i + Eg(q^i)$$

which means that

$$u_i = g(q^i) + (b_i - \sum_{i \neq j} A_{ij} u_j) \times E^{-1} \simeq g(q^i) + O(1/E)$$

Note 4 *To solve non-homogeneous Dirichlet, we rewrite Example 2 as*

```
 4: Vh u,v,f,F,g,bc; g = sin(pi*x)*cos(pi*y);
 5: varf a(u,v) = int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v))
 6:                + on(C,u=1) ;                    //    see (1.7)
10: matrix A=a(Vh,Vh); bc[]=a(0,Vh);
12: F[]=B*f[];  F[] += bc[] .* g[];
```

Here "`bc[]=a(0,Vh)`" create the vector $[bc_1, bc_2, \cdots, bc_M]$*,* $bc_i = 0$ *if* $i \in I_\Omega$ *and* $bc_i = E(= 10^{30})$ *if* $i \notin I_\Omega$*. If the finite approximation of* g *is* $g \approx g_1\phi_1 + \cdots + g_M\phi_M$

$$\texttt{bc[] .* g[]} = \sum_{j=1}^{M} bc_j g_j \tag{1.11}$$

1.4.2 Matrix Operations

The multiplicative operators *, /, and % group left to right.

- `'` is unary right transposition of array, matrix
- `.*` is the term to term multiply operator.
- `./` is the term to term divide operator.

there are some compound operator:

- `^-1` is for solving the linear system (example: `b = A^-1 x`)
- `' *` is the compound of transposition and matrix product, so it is the dot product (example `real DotProduct=a'*b`)

Example 3

```
mesh Th = square(2,1);
fespace Vh(Th,P1);
Vh f,g;
f = x*y;
g = sin(pi*x);
Vh<complex> ff,gg;              //    a complex valued finite element function
ff= x*(y+1i);
gg = exp(pi*x*i);
varf mat(u,v) =
  int2d(Th)(1*dx(u)*dx(v)+2*dx(u)*dy(v)+3*dy(u)*dx(v)+4*dy(u)*dy(v))
  + on(1,2,3,4,u=1);
varf mati(u,v) =
  int2d(Th)(1*dx(u)*dx(v)+2i*dx(u)*dy(v)+3*dy(u)*dx(v)+4*dy(u)*dy(v))
  + on(1,2,3,4,u=1);
matrix A = mat(Vh,Vh);
matrix<complex> AA = mati(Vh,Vh);                //    a complex sparce matrix

Vh m0; m0[] = A*f[];
Vh m01; m01[] = A'*f[];
Vh m1; m1[] = f[].*g[];
Vh m2; m2[] = f[]./g[];
cout << "f = " << f[] << endl;
cout << "g = " << g[] << endl;
cout << "A = " << A << endl;
cout << "m0 = " << m0[] << endl;
cout << "m01 = " << m01[] << endl;
cout << "m1 = "<< m1[] << endl;
cout << "m2 = "<< m2[] << endl;
cout << "dot Product = "<< f[]'*g[] << endl;
cout << "hermitien Product = "<< ff[]'*gg[] << endl;
```

This produce the following:

$$A = \begin{array}{c|cccccc|} & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 10^{30} & 10.5 & 0 & 30. & 4-2.5 & 0 \\ 2 & 0. & 10^{30} & 0.5 & 0 & 0.5 & -2.5 \\ 3 & 0 & 0. & 10^{30} & 0 & 0 & 0.5 \\ 4 & 0.5 & 0 & 0 & 10^{30} & 0. & 0 \\ 5 & -2.5 & 0.5 & 0 & 0.5 & 10^{30} & 0. \\ 6 & 0 & -2.5 & 0. & 0 & 0.5 & 10^{30} \end{array}$$

$$\begin{aligned}
\{v\} = \texttt{f[]} &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0.5 & 1 \end{pmatrix}^T \\
\{w\} = \texttt{g[]} &= \begin{pmatrix} 0 & 1 & 1.2\times10^{-16} & 0 & 1 & 1.2\times10^{-16} \end{pmatrix}^T \\
\texttt{A*f[]} &= \begin{pmatrix} -1.25 & -2.25 & 0.5 & 0 & 5\times10^{29} & 10^{30} \end{pmatrix}^T \quad (= A\{v\}) \\
\texttt{A'*f[]} &= \begin{pmatrix} -1.25 & -2.25 & 0 & 0.25 & 5\times10^{29} & 10^{30} \end{pmatrix}^T \quad (= A^T\{v\}) \\
\texttt{f[].*g[]} &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0.5 & 1.2\times10^{-16} \end{pmatrix}^T \quad = (v_1w_1 \quad \cdots \quad v_Mw_M)^T \\
\texttt{f[]./g[]} &= \begin{pmatrix} -nan & 0 & 0 & -nan & 0.5 & 8.1\times10^{15} \end{pmatrix}^T \quad = (v_1/w_1 \cdots v_M/w_M)^T \\
\texttt{f[]'*g[]} &= 0.5 \quad (= \{v\}\{w\}^T = \{v\}\cdot\{w\})
\end{aligned}$$

Note 5 *The operators* `^-1` *cannot create the matrix by themselves. Indeed, the following occur errors*

```
matrix AAA = A^-1;
```

1.5 Modeling–Edit–Run–Visualize–Revise

`freefem++` provide many examples and its documentation, so you can easily calculate mathematical models by FEM (finite element method) and study them. Explanations for these examples are given in this book. If you are a beginner of FEM, you start from Quick Tour of `freefem++` . The numerical simulation of scientific problem will be done as follows.

Modeling: Make a mathematical model describing scientific problems. Mathematical modeling is a deep and fruitful one, with many important implications for scientific problems (refer to Chapter 7).

Programming: Translate the mathematical model to `freefem++` source code, which is easy because `freefem++` includes many clever techniques in FEM with mathematical writing.

Run: Next step is to run it to see if it works. If we provisionally give the name of the source code to "something.edp", we can execute it by the typing

```
% freefem++ something.edp
```

An important part in programming is to keep aware of collections of programs that are available, and this manual contains many examples you can use freely. So we hope you to run these examples and their representing mathematical models, which are contained in the package in `freefem++` .

Visualization: The numerical calculation by FEM make huge data, so the easy way to check the obtained result is their visualization. `freefem++` can display the mesh and the contour lines of obtained functions. If you want to use these visualization after execution, you add the filename of PostScript to the commands "plot" (see Section 5.1).

Debugging: If the boolean value of "wait" is true (default is 'false'), then `freefem++` will stop at the information in visual form. Write the following, execute it and make a change the line "`wait=true`" to "`wait=false`".

```
bool wait = true;             //   set "true" if you want see each
plotting
mesh Th = square(10,10,[-1+2*x,-1+2*y]);          //    ]-1,1[^2
plot(Th);                                      //   plot the mesh
fespace Vh(Th,P2);
Vh f = sin(pi*x)*cos(pi*y);
plot(f,wait=wait);                         //   plot the function f
Vh g = sin(pi*x + cos(pi*y));
plot(g,wait=wait);                         //   plot the function g
```

If there is a fatal error in your source code, freefem++ will end and cause an error message to appear. In MS-Windows, freefem++ will open the message file by notepad. For example, if you forget parenthesis as in

```
mesh Th = square(10,10;
plot(Th);
```

then you will get the following message from freefem++,

```
mesh Th = square(10,10;
 Error line number 0, in file xxxxxx.edp, before  token ;
parse error
Compile error : parse error
        line number :0, ;
 at exec line  0
error Compile error : parse error
        line number :0, ;
```

If you use the same symbol twice as in

```
real aaa =1;
real aaa;
```

then you will get the message

```
real aaa =1;
    1 : real aaa; The identifier aaa exist
```

Notice that the line number start from 0. If you find that the program isn't doing what you want it to do, then you check the line number and try to figure out what's wrong. We give two techniques; One is *trace* by **plot** for *meshes* and (FE-)functions with **wait=true**, and by **cout** for scalar, vectors and matrices. Another is to *comment out* by "//". If you find a doubtful line in your source code, you comment out as follows,

```
real aaa =1;
                                              //    real aaa;
```

1.6 Installation

There are binary packages available for Microsoft Windows and Apple Mac OS. For all other platforms, freefem++ must be compiled and installed from the source archive. This archive is available from:
http://www.ann.jussieu.fr/~hecht/ftp/freefem/freefem++.tgz.
To extract files from the compressed archive freefem++.tgz into a directory called freefem++-X.X
(where X.XX is the version number) enter the following commands in a shell window :

```
tar zxvf freefem++.tgz
cd freefem++-X.XX
```

To compile and install freefem++ , just follow the INSTALL and README files. The following programs are produced, depending on the system you are running (Linux, Windows, MacOS) :

1. `FreeFem++`, standard version, with a graphical interface based on X11, Win32 or MacOS

2. `FreeFem++-nw`, postscript plot output only (batch version, no windows)

3. `FreeFem++-mpi`, parallel version, postscript output only

4. `FreeFem++-glx`, graphics using OpenGL and X11

5. `FreeFem++-cs`, integrated development environment (please see chapter "Graphical User Interface" for more details).

6. `/Applications/FreeFem++.app`, Drag and Drop CoCoa MacOs Application

7. `FreeFem++-CoCoa`, MacOS Shell script for MacOS OpenGL version (MacOS 10.2 or better) (note: it uses /Applications/FreeFem++.app)

As an installation test, go into the directory `examples++-tutorial` and run `freefem++` on the example script `LaplaceP1.edp` with the command :

```
FreeFem++ LaplaceP1.edp
```

Chapter 2

Syntax

2.1 Data Types

Basically `freefem++` is a compiler, the language is typed, polymorphic and reentrant. Every variable must be typed, declared in a statement; each statement separated from the next by a semicolon '`;`'. The language allows the manipulation of basic types integers (`int`), reals (`real`), strings (`string`), arrays (example: `real[int]`), bidimensional (2D) finite element meshes (`mesh`), 2D finite element spaces (`fespace`) , definition of functions (`func`), arrays of finite element functions (`func`[*basic_type*]), linear and bilinear operators, sparse matrices, vectors , etc. For instance

```
int i,n=20;                                          //    i,n are integer.
real[int] xx(n),yy(n);                            //    two array of size n
for (i=0;i<=20;i++)              //    which can be used in statements such as
 { xx[i]= cos(i*pi/10); yy[i]= sin(i*pi/10); }
```

The life of a variable is the current block $\{\ldots\}$, except the `fespace` variable, and the in variables local to a block are destroyed at the end of the block as follows.

Example 4

```
real r= 0.01;
mesh Th=square(10,10);                         //    unit square mesh
fespace Vh(Th,P1);         //    P1 lagrange finite element space
Vh u = x+ exp(y);
func f = z * x + r * log(y);
plot(u,wait=true);
{                                                      //    new block
  real r = 2;                                       //    not the same r
  fespace Vh(Th,P1);      //    error because Vh is a global name
}                                                   //    end of block
                                           //    here r back to 0.01
```

The type declarations are compulsory in `freefem++` because it is easy to make bugs in a language with many types. The variable name is just an alphanumeric string, the underscore character "_" is not allowed, because it will be used as an operator in the future.

2.2 List of major types

bool is used for logical expression and flow-control.

int declare an integer.

string declare the varible to store a text enclosed within double quates, such as:

```
"This is a string in double quotes."
```

real declare the variable to store a number such as "12.345".

complex Complex numbers, such as $1 + 2i$, $i = \sqrt{-1}$.

```
complex a =  1i, b = 2 + 3i;
cout << "a + b = " << a + b << endl;
cout << "a - b = " << a + b << endl;
cout << "a * b = " << a * b << endl;
cout << "a / b = " << a / b << endl;
```

Here's the output;

```
a + b = (2,4)
a - b = (-2,-2)
a * b = (-3,2)
a / b = (0.230769,0.153846)
```

ofstream make a output file and its functions.

ifstream make a input file and its functions.

real[int] declare a variable that store multiple real numbers with integer index.

```
real[int] a(5);
a[0] = 1; a[1] = 2; a[2] = 3.3333333; a[3] = 4; a[4] = 5;
cout << "a = " << a  << endl;
```

This produces the output;

```
a = 5   :
  1       2     3.33333   4       5
```

real[string] declare a variable that store multiple real numbers with string index.

string[string] declare a variable that store multiple strings with string index.

`func` define a function without argument, if independent variables are `x, y`. For example

```
func f=cos(x)+sin(y) ;
```

Remark the function's type is given by the expression's type. The power of functions are given in `freefem++` such as `x^1`, `y^0.23`.

`mesh` create the triangulation, see Section 3.

`fespace` define a new type of finite element space, see Section Section 4.

problem declare the weak form of a partial differential problem without solving.

solve declare a problem and solve it.

varf define a full variational form.

matrix define a sparce matrix.

2.3 Global Variables

The names `x,y,z,label,region,P,N,nu_triangle` are used to link the language to the finite element tools:

x expresses x coordinate of current point (real value)

y expresses y coordinate of current point (real value)

z expresses z coordinate of current point (real value) , but is reserved for future use.

label show the label number of boundary if the current point is on a boundary, otherwise 0 (int value).

region returns the region number of the current point (x,y) (int value).

P give the current point ($\mathbb{R}^2$ value.). By `P.x`, `P.y`, we can get the x, y components of `P` . Also `P.z` is reserved.

N give the outward unit normal vector at the current point is on the curve define by `border` ($\mathbb{R}^3$ value). `N.x` and `N.y` are x and y components of the normal vector. `N.z` is reserved.
.

lenEdge give the length of the current edge

$$\texttt{lenEdge} = |q^i - q^j| \quad \text{if the current edge is } [q^i, q^j]$$

hTriangle give the size of the current triangle

nuTriangle give the index of the current triangle (integer).

nuEdge give the index of the current edge in the triangle (integer).

nTonEdge give the number of adjacent triangle of the current edge (integer).

area give the area of the current triangle (real).

cout is the standard output device (default is console). On MS-Windows, the standard output is only to console, in this time. `ostream`

cin is the standard input device (default is keyboard). (`istream`). On MS-Windows, this don't work.

endl give the end of line in the input/output devices.

true means “true” in `bool` value.

false means “false” in `bool` value.

pi is the `real` approximation value of π.

Here is how to show all the types, and all the operator and functions.

```
 dumptable(cout);
```

To execute a system command in the string (not implemented on Carbon MacOs)

```
  exec("shell command");
```

On MS-Windows, we need the full path. For example, if there is the command “ls.exe” in the subdirectory “`c:\cygwin\bin\`”, then we must write

```
  exec("c:\\cygwin\\bin\\ls.exe");
```

2.4 Arithmetic

In integers, $+$, $-$, $*$ express the usual arithmetic summation (plus), subtraction (minus) and multiplication (times), respectively. The operators $/$ and $\%$ yield the quotient and the remainder from the division of the first expression by the second. If the second number of $/$ or $\%$ is zero the behavior is undefined. The *maximum* or *minimum* of two integers a, b are obtained by `max(`a`,`b`)` of `min(`a`,`b`)`. The power a^b of two integers a, b is calculated by writing `a^b`.

Example 5 *Calculations with the integers*

```
int a = 12, b = 5;
cout <<"plus, minus of "<<a<<" and "<<b<<" are "<<a+b<<", "<<a-b<<endl;
cout <<"multiplication, quotient of them are "<<a*b<<", "<<a/b<<endl;
cout <<"remainder from division of "<<a<<" by "<<b<<" is "<<a%b<<endl;
cout <<"the minus of "<<a<<" is "<< -a << endl;
cout <<a<<" plus -"<<b<<" need bracket:"<<a<<"+(-"<<b<<")="<<a+(-b)<<endl;
cout <<"max and min of "<<a<<" and "<<b<<" is "<<max(a,b)<<","<<min(a,b)<< endl;
cout <<b<<"th power of "<<a<<" is "<<a^b<< endl;
b=0;
cout <<a<<"/0"<<" is "<< a/b << endl;
cout <<a<<"%0"<<" is "<< a%b << endl;
```

produce the following result:

```
plus, minus of 12 and 5 are 17, 7
multiplication, quotient of them are 60, 2
remainder from division of 12 by 5 is 2
the minus of 12 is -12
12 plus -5 need bracket :12+(-5)=7
max and min of 12 and 5 is 12,5
```

```
5th power of 12 is 248832
12/0 : long long long
Fatal error : ExecError  Div by 0 at exec line  9
Exec error : exit
```

By the relation *integer* $\subset$ *real*, the operators "+, $-$, $*$, /, %" and "`max, min,` ^" are also applicable in real-type. However, % calculates the remainder of the integral parts of two real numbers.
The following example similar to Example 5

```
real a=sqrt(2.), b = pi;
cout <<"plus, minus of "<<a<<" and "<<pi<<" are "<< a+b <<", "<<
a-b << endl;
cout <<"multiplication, quotient of them are "<<a*b<<", "<<a/b<<
endl;
cout <<"remainder from division of "<<a<<" by "<<b<<" is "<< a%b
<< endl;
cout <<"the minus of "<<a<<" is "<< -a << endl;
cout <<a<<" plus -"<<b<<" need bracket :"<<a<<"+(-"<<b<<")="<<a
+ (-b) << endl;
```

gives the following output:

```
plus, minus of 1.41421 and 3.14159 are 4.55581, -1.72738
multiplication, quotient of them are 4.44288, 0.450158
remainder from division of 1.41421 by 3.14159 is 1
the minus of 1.41421 is -1.41421
1.41421 plus -3.14159 need bracket :1.41421+(-3.14159)=-1.72738
```

By the relation

$$bool \subset int \subset real \subset complex,$$

the operators "+, $-$, $*$, /" and "^" are also applicable in complex-type, but "%, max, min" fall into disuse. Complex number such as `5+9i`, i= $\sqrt{-1}$, can be a little tricky. For real variables `a=2.45, b=5.33`, we must write the complex numbers `a+b*i` and `a+sqrt(2.0)*i` as

```
complex z1 = a+b*1i, z2=a+sqrt(2.0)*1i;
```

The imaginary and real parts of complex number `z` is obtained by `imag` and `real`. The conjugate of $a + bi$ (a, b are real) is defined by $a - bi$, which is denoted by `conj(a+b*1i)` in `freefem++` .
The complex number $z = a + ib$ is considered as the pair (a, b) of real numbers a, b. Now we draw the point (a, b) in the plane (Cartesian rectangular system of axes) and mark on the x-axis the real numbers in the usual way, on the y-axis the imaginary numbers with i as unit. By changing Cartesian coordinate (a, b) to the polar coordinate (r, ϕ), the complex number z has another expression $z = r(\cos\phi + i\sin\phi)$, $r = \sqrt{a^2 + b^2}$ and $\phi = \tan^{-1}(b/a)$; r is called the *absolute value* and ϕ the *argument* of z. In the following example, we shall show them using `freefem++` programming, and *de Moivre's formula* $z^n = r^n(\cos n\phi + i\sin n\phi)$.

Example 6

```
real a=2.45, b=5.33;
complex  z1=a+b*1i, z2 = a+sqrt(2.)*1i;
```

```
func string pc(complex z)          //    printout complex to (real)+i(imaginary)
{
   string r = "("+real(z);
   if (imag(z)>=0) r = r+"+";
   return r+imag(z)+"i)";
}
                              //    printout complex to |z|*(cos(arg(z))+i*sin(arg(z)))
func string toPolar(complex z)
{
   return abs(z)+"*(cos("+arg(z)+")+i*sin("+arg(z)+"))";
}
cout <<"Standard output of the complex "<<pc(z1)<<" is the pair "
     <<z1<<endl;
cout <<"Plus, minus of "<<pc(z1)<<" and "<<pc(z2)<<" are "<< pc(z1+z2)
     <<", "<< pc(z1-z2) << endl;
cout <<"Multiplication, quotient of them are "<<pc(z1*z2)<<", "
     <<pc(z1/z2)<< endl;
cout <<"Real/imaginary part of "<<pc(z1)<<" is "<<real(z1)<<", "
     <<imag(z1)<<endl;
cout <<"Absolute of "<<pc(z1)<<" is "<<abs(z1)<<endl;
cout <<pc(z2)<<" = "<<toPolar(z2)<<endl;
cout <<"  and polar("<<abs(z2)<<","<<arg(z2)<<") = "
     << pc(polar(abs(z2),arg(z2)))<<endl;
cout <<"de Moivre's formula: "<<pc(z2)<<"^3 = "<<toPolar(z2^3)<<endl;
cout <<"conjugate of "<<pc(z2)<<" is "<<pc(conj(z2))<<endl;
cout <<pc(z1)<<"^"<<pc(z2)<<" is "<< pc(z1^z2) << endl;
```

Here's the output from Example 6

```
Standard output of the complex (2.45+5.33i) is the pair
(2.45,5.33)
Plus, minus of (2.45+5.33i) and (2.45+1.41421i) are
(4.9+6.74421i), (0+3.91579i)
Multiplication, quotient of them are (-1.53526+16.5233i),
(1.692+1.19883i)
Real/imaginary part of (2.45+5.33i) is 2.45, 5.33
Absolute of (2.45+5.33i) is 5.86612
(2.45+1.41421i) = 2.82887*(cos(0.523509)+i*sin(0.523509))
  and polar(2.82887,0.523509) = (2.45+1.41421i)
de Moivre's formula: (2.45+1.41421i)^3
                         = 22.638*(cos(1.57053)+i*sin(1.57053))
conjugate of (2.45+1.41421i) is (2.45-1.41421i)
(2.45+5.33i)^(2.45+1.41421i) is (8.37072-12.7078i)
```

2.5 One Variable Functions

Fundamental functions are built into `freefem++` . The *power function* `x^`$\alpha(= x^\alpha)$; the *exponent function* **`exp`**`(x)` $(= e^x)$; the *logarithmic function* **`log`**`(x)`$(= \ln x)$ or **`log`**`10(x)` $(= \log_{10} x)$; the *trigonometric functions* **`sin`**`(x)`, **`cos`**`(x)`, **`tan`**`(x)` depending on angles measured by *radian*; the inverse of $\sin x$, $\cos x$, $\tan x$ called *circular function* or *inverse* *trigonometric function* **asin**(x)(=$\arcsin x$), **acos**(x)(=$\arccos x$), **atan**(x)(=$\arctan x$); the *hyperbolic function*,

$$\sinh x = \left(e^x - e^{-x}\right)/2, \qquad \cosh x = \left(e^x + e^{-x}\right)/2.$$

and $\tanh x = \sinh x/\cosh x$ written by **sinh**(x), **cosh**(x), **asinh**(x) and **acosh**(x).

$$\sinh^{-1} x = \ln\left[x + \sqrt{x^2+1}\right], \qquad \cosh^{-1} x = \ln\left[x + \sqrt{x^2-1}\right].$$

Elementary Functions is the class of functions consisting of the functions in this section (polynomials, exponential, logarithmic, trigonometric, circular) and the functions obtained from those listed by the four arithmetic operations

$$f(x)+g(x),\ f(x)-g(x),\ f(x)g(x),\ f(x)/g(x)$$

and by superposition $f(g(x))$, in which four arithmetic operarions and superpositions are permitted finitely many times. In freefem++ , we can create all elementary functions. The derivative of an elementary function is also elementary. However, the indefinite integral of an elementary function cannot always be expressed in terms of elementary functions.

Example 7 *The following give the example to make the boundary using elementary functions.* Cardioid

```
real b = 1.;
real a = b;
func real phix(real t)
{
   return (a+b)*cos(t)-b*cos(t*(a+b)/b);
}
func real phiy(real t)
{
   return (a+b)*sin(t)-b*sin(t*(a+b)/b);
}
border C(t=0,2*pi) { x=phix(t); y=phiy(t); }
mesh Th = buildmesh(C(50));
```

Taking the principal value, we can define $\log z$ for $z \neq 0$ by

$$\ln z = \ln|z| + \arg z.$$

Using freefem++ , we calculated **exp**(1+4i), **sin**(pi+1i), **cos**(pi/2-1i) and **log**(1+2i), we then have

$$-1.77679 - 2.0572i, \quad 1.8896710^{-16} - 1.1752i,$$
$$9.4483310^{-17} + 1.1752i, \quad 0.804719 + 1.10715i.$$

2.6 Two Variable Functions

2.6.1 Formula

The general form of real functions with two independent variables x, y is usually written as $z = f(x, y)$. In freefem++ , x and y are reserved word in Section 2.3. When two independent variables are x and y, we can define a function without argument, for example

```
func f=cos(x)+sin(y) ;
```

Remark the function's type is given by the expression's type. The power of functions are given in freefem++ such as x^1, y^0.23. In func, we can write an elementary function as follows

```
func f = sin(x)*cos(y);
func g = (x^2+3*y^2)*exp(1-x^2-y^2);
func h = max(-0.5,0.1*log(f^2+g^2));
```

Complex valued function create functions with 2 variables x, y as follows,

```
mesh Th=square(20,20,[-pi+2*pi*x,-pi+2*pi*y]);      //    ]-π,π[^2
fespace Vh(Th,P2);
func z=x+y*1i;                                      //    z = x + iy
func f=imag(sqrt(z));                               //    f = ℑ√z
func g=abs( sin(z/10)*exp(z^2/10) );       //    g = |sin z/10 exp z^2/10|
Vh fh = f; plot(fh);                       //    contour lines of f
Vh gh = g; plot(gh);                       //    contour lines of g
```

We call also by *two variable elementary function* functions obtained from elementary functions $f(x)$ or $g(y)$ by the four arithmetic operations and by superposition in finite times.

2.6.2 FE-function

Arithmetic built-in functions are able to construct a new function by the four arithmetic operations and superposition of them (see *elementary functions*), which are called *formulas* to distinguish from FE-functions. We can add *new formulas* easily, if we want. Here, FE-function is an element of finite element space (real or complex) (see Section Section 4). Or to put it another way: *formulas* are the mathematical expressions combining its numerical analogs, but it is independent of meshes (triangulations).
Also, in freefem++, we can give an arbitrary symbol to FE-function combining numerical calculation by FEM. The projection of a formula f to FE-space is done as in

```
func f=x^2*(1+y)^3+y^2;
mesh Th = square(20,20,[-2+2*x,-2+2*y]);      //    square ]-2,2[^2
fespace Vh(Th,P1);
Vh fh=f;      //    fh is the projection of f to Vh (real value)
func zf=(x^2*(1+y)^3+y^2)*exp(x+1i*y);
Vh<complex> zh = zf;                  //    zh is the projection of zf
                                      //    to complex value Vh space
```

The construction of fh ($=f_h$) is explained in Section 4.

Note 6 *The command* plot *is valid only for real FE-functions.*

Complex valued function create functions with 2 variables `x, y` as follows,

```
mesh Th=square(20,20,[-pi+2*pi*x,-pi+2*pi*y]);       //     ]-π,π[^2
fespace Vh(Th,P2);
func z=x+y*1i;                                       //      z = x + iy
func f=imag(sqrt(z));                                //      f = ℑ√z
func g=abs( sin(z/10)*exp(z^2/10) );      //      g = |sin z/10 exp z^2/10|
Vh fh = f; plot(fh);                 //     Fig.  2.1 isovalue of f
Vh gh = g; plot(gh);                 //     Fig.  2.2 isovalue of g
```

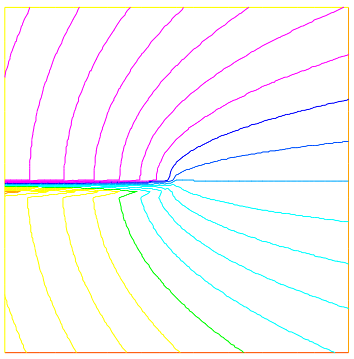


Figure 2.1: $\Im\sqrt{z}$ has branch

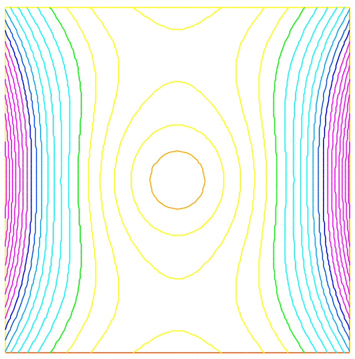


Figure 2.2: $|\sin(z/10)\exp(z^2/10)|$

2.7 Array

An *array* stores multiple objects, and there are 2 kinds of arrays: The first is the *vector* that is arrays with *integer indices* and arrays with *string indices*.

In the first case, the size of this array must be know at the execution time, and the implementation is done with the `KN<>` class so all the vector operator of `KN<>` are implemented. The sample

```
real [int] tab(10), tab1(10);               //     2 array of 10 real
real [int] tab2;                        //     bug array with no size
tab = 1.03;                         //     set all the array to 1.03
tab[1]=2.15;
cout << tab[1] << " " << tab[9] << " size of tab = "
     << tab.n << " min: " << tab.min << "  max:" << tab.max
     << " sum : "   << tab.sum <<   endl;                       //
tab.resize(12);                   //     change the size of array tab
                            //     to 12 with preserving first value
tab(10:11)=3.14;                                //     set unset value
cout <<" resize tab: " <<  tab << endl;
```

produce the output

```
2.15 1.03 size of tab = 10 min: 1.03  max:2.15 sum : 11.42
 resize tab: 12
        1.03    2.15    1.03    1.03    1.03
        1.03    1.03    1.03    1.03    1.03
        3.14    3.14
```

It is also possible to make an array of FE function, with the same syntax, and we can treat them as *vector valued function* if we need them.

Example 8 *In the following example, Poisson's equation is solved under 3 different given functions* $f = 1$, $\sin(\pi x)\cos(\pi y)$, $|x-1||y-1|$, *whose solutions are stored in an array of FE function.*

```
mesh Th=square(20,20,[2*x,2*y]);
fespace Vh(Th,P1);
Vh u, v, f;
problem Poisson(u,v) =
    int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v))
     + int2d(Th)( -f*v ) + on(1,2,3,4,u=0) ;
Vh[int] uu(3);                           //    an array of FE function
f=1;                                                        //    problem1
Poisson; uu[0] = u;
f=sin(pi*x)*cos(pi*y);                                      //    problem2
Poisson; uu[1] = u;
f=abs(x-1)*abs(y-1);                                        //    problem3
Poisson; uu[2] = u;
for (int i=0; i<3; i++)                      //    plots all solutions
  plot(uu[i], wait=true);
```

For the second case, it is just a map of the STL[1][?] so no vector operation except the selection of an item is allowed .
The transpose operator is ′ like MathLab or SciLab, so the way to compute the dot product of two array `a,b` is `real ab= a'*b`.

```
int i;
real [int] tab(10), tab1(10);                              //    2 array of 10 real
    real [int] tab2;                                    //    bug array with no size
tab = 1;                                                //    set all the array to 1
tab[1]=2;
cout << tab[1] << " " << tab[9] << " size of tab = "
     << tab.n << " " << tab.min << " " << tab.max << " " <<  endl;
tab1=tab;
tab=tab+tab1;
tab=2*tab+tab1*5;
tab1=2*tab-tab1*5;
tab+=tab;
cout << " dot product " << tab'*tab << endl;                          //    ^t tab tab
cout << tab << endl;
cout << tab[1] << " " << tab[9] <<  endl;
real[string] map;                                            //    a dynamique array
for (i=0;i<10;i=i+1)
```

[1] Standard template Library, now part of standard C++

```
  {
    tab[i] = i*i;
    cout << i << " " << tab[i] << "\n";
  };

map["1"]=2.0;
map[2]=3.0;                       //    2 is automatically cast to the string "2"

cout << " map[\"1\"] = " << map["1"] << "; "<< endl;
cout << " map[2] = " << map[2] << "; "<< endl;
```

2.8 Loops

The `for` and `while` loops are implemented with `break` and `continue` keywords.
In for-loop, there are three parameters; the INITIALIZATION of a control variable, the CONDITION to continue, the CHANGE of the control variable. While `CONDITION` is true, for-loop continue.

```
   for (INITIALIZATION; CONDITION; CHANGE)
        { BLOCK of calculations }
```

The sum from 1 to 10 is calculated by (the result is in `sum`),

```
   int sum=0;
   for (int i=1; i<=10; i++)
      sum += i;
```

The while-loop

```
   while (CONDITION) {
      BLOCK of calculations or change of control variables
   }
```

is executed repeatedly until CONDITION become false. The sum from 1 to 10 is also written by `while` as follows,

```
   int i=1, sum=0;
   while (i<=10) {
     sum += i; i++;
   }
```

We can exit from a loop in midstream by `break`. The `continue` statement will pass the part from *continue* to the end of the loop.

Example 9

```
for (int i=0;i<10;i=i+1)
    cout << i << "\n";
real eps=1;
while (eps>1e-5)
 { eps = eps/2;
   if( i++ <100) break;
   cout << eps << endl;}

for (int j=0; j<20; j++) {
   if (j<10) continue;
   cout << "j = " << j << endl;
}
```

2.9 Input/Output

The syntax of input/output statements is similar to `C++` syntax. It uses **`cout`**, **`cin`**, **`endl`**, `<<`,`>>`.
To write to (resp. read from) a file, declare a new variable `ofstream ofile("filename");` or `ofstream ofile("filename",append);` (resp. `ifstream ifile("filename");`) and use `ofile` (resp. `ifile`) as `cout` (resp. `cin`). The word `append` in `ofstream ofile("filename",append);` means openning a file in append mode.

Note 7 *The file is closed at the exit of the enclosing block,*

Example 10

```
int i;
cout << " std-out" << endl;
cout << " enter i= ? ";
cin >> i ;
{
  ofstream f("toto.txt");
  f << i << "coucou'\n";
};                          //    close the file f because the variable f is delete

{
  ifstream f("toto.txt");
   f >> i;
}
{
  ofstream f("toto.txt",append);
                                    //    to append to the existing file "toto.txt"
  f << i << "coucou'\n";
};                          //    close the file f because the variable f is delete

  cout << i << endl;
```

Chapter 3

Mesh Generation

3.1 Commands for Mesh Generation

In Step1 in Section 1.2, the keywords **border, buildmesh** are explained.
All the examples in this section come from the files `mesh.edp` and `tablefunction.edp`.

3.1.1 Square

For easy and simple testing, there is the command "**square**". The following

```
mesh Th = square(4,5);
```

generate a 4×5 grid in the unit squre $[0,1]^2$ whose labels are shown in Fig. 3.1. If you want

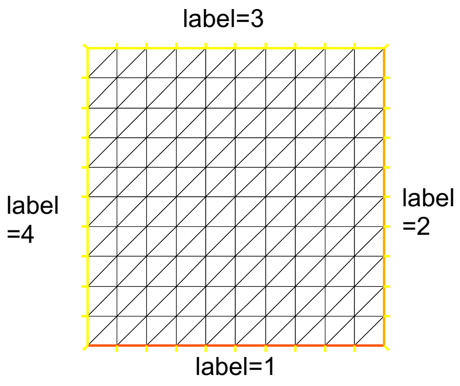


Figure 3.1: Boundary labels of the mesh by `square(10,10)`

constructs a $n \times m$ grid in the rectangle $[x_0, x_1] \times [y_0, y_1]$, you can write

```
real x0=1.2,x1=1.8;
real y0=0,y1=1;
int n=5,m=20;
mesh Th=square(n,m,[x0+(x1-x0)*x,y0+(y1-y0)*y]);
```

3.1.2 Border

A domain is defined as being on the left (resp right) of its parameterized boundary

$$\Gamma_j = \{(x,y) \mid x = \varphi_x(t),\, y = \varphi_y(t),\, a_j \le t \le b_j\}$$

We can easily check the orientation by drawing the curve $t \mapsto (\varphi_x(t), \varphi_y(t))$, $t_0 \le t \le t_1$. If the figure become like to Fig. 3.2, then the domain lie on the shaded area, otherwise it lie on opposite side (see also the examples enclosed with the box). The boundaries Γ_j can only intersect at their end points.

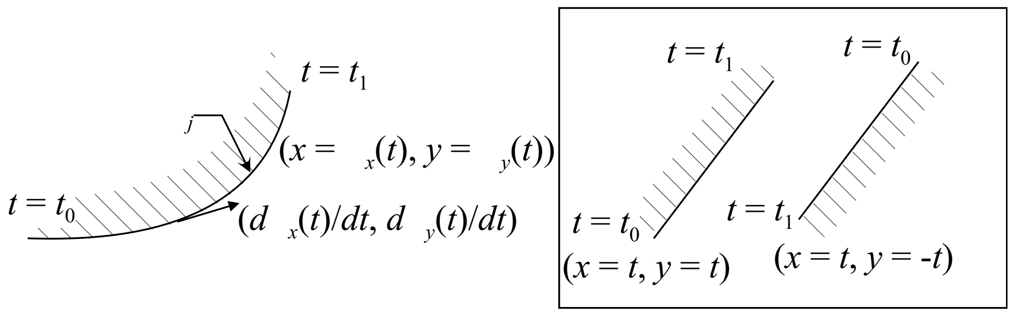


Figure 3.2: Orientation of the boundary defined by $(\phi_x(t), \phi_y(t))$

The general expression of the triangulation is

```
mesh   Mesh_Name = buildmesh(Γ1(m1) + ··· + ΓJ(mj));
```

where m_j are numbers of marked points on Γ_j, $\Gamma = \cup_{j=1}^{J} \Gamma_J$. We can change the orientation of boundaries by changing the sign of m_j. The following example shows how to change the orientation. The example generates the unit disk with a small circular hole, and assign "1" to the unit disk ("2" to the circle inside). The boundary label must be non-zero, however we can omit the label if we want use only the symbol.

```
1: border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
2: border b(t=0,2*pi){ x=0.3+0.3*cos(t); y=0.3*sin(t);label=2;}
3: plot(a(50)+b(+30)) ;                 //    to see a plot of the border mesh
4: mesh Thwithouthole= buildmesh(a(50)+b(+30));
5: mesh Thwithhole    = buildmesh(a(50)+b(-30));
6: plot(Thwithouthole,wait=1,ps="Thwithouthole.eps");          //   figure 3.3
7: plot(Thwithhole,wait=1,ps="Thwithhole.eps");               //    figure 3.4
```

Note 8 *You must notice that the orientation is changed by "`b(-30)`" in 5th line. In 7th line, `ps="fileName"` is used to generate a postscript file identification that is shown on screen.*

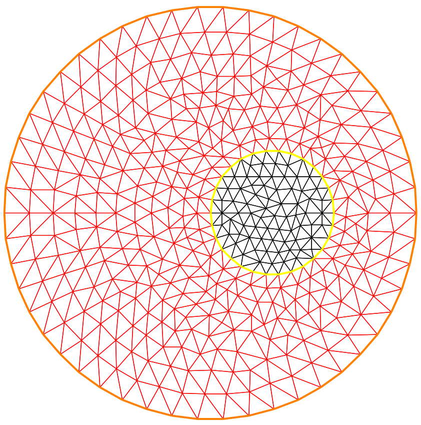


Figure 3.3: mesh without hole

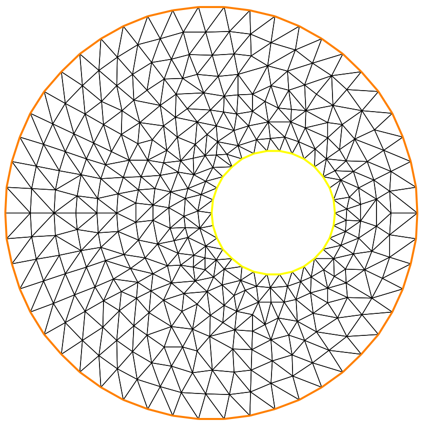


Figure 3.4: mesh with hole

3.1.3 Data Structure of Mesh and Reading/Writing a Mesh

Some user asked us that they want use the triangulation made from other tools or hand-made mesh. The example

```
border C(t=0,2*pi) { x=cos(t); y=sin(t); }
mesh Th = buildmesh(C(10));
savemesh("mesh_sample.msh");
```

make the mesh as in Fig. 3.5. The informations about `Th` are save in the file "mesh_sample.msh". We can read from Fig. 3.5 and "mesh_sample.msh" as in Table 3.1 where n_v is the number of vertices, n_t number of triangles and n_s the number of edges on boundary. For each vertex q^i, $i = 1, \cdots, n_v$, we denote by (q^i_x, q^i_y) the x-coordinate and y-coordinate.
Each triangle $T_k, k = 1, \cdots, 10$ have three vertices q^{k_1}, q^{k_2}, q^{k_3} that are oriented in counter-clockwise. The boundary consists of 10 lines L_i, $i = 1, \cdots, 10$ whose tips are q^{i_1}, q^{i_2}.

Contents of file	Explanation
14 16 10	n_v n_t n_e
-0.309016994375 0.951056516295 1	q^1_x q^1_y boundary label=1
0.309016994375 0.951056516295 1	q^2_x q^2_y boundary label=1
$\cdots$ $\cdots$ $\vdots$	
-0.309016994375 -0.951056516295 1	q^{14}_x q^{14}_y boundary label=1
9 12 10 0	1_1 1_2 1_3 region label=0
5 9 6 0	2_1 2_2 2_3 region label=0
$\cdots$	
9 10 6 0	16_1 16_2 16_3 region label=0
6 5 1	1_1 1_2 boundary label=1
5 2 1	2_1 2_2 boundary label=1
$\cdots$	
10 6 1	10_1 10_2 boundary label=1

Table 3.1: The structure of "mesh_sample.msh"

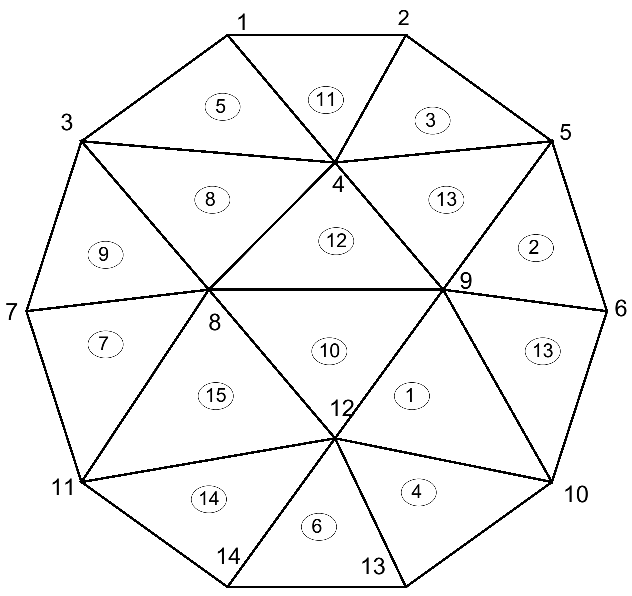


Figure 3.5: mesh by `buildmesh(C(10))`

In the left figure, we have the following.

$n_v = 14$, $n_t = 16$, $n_s = 10$

$q^1 = (-0.309016994375, 0.951056516295)$

$\vdots \quad \vdots \quad \vdots$

$q^{14} = (-0.309016994375, -0.951056516295)$

The vertices of T_1 are q^9, q^{12}, q^{10}.

$\vdots \quad \vdots \quad \vdots$

The vertices of T_{16} are q^9, q^{10}, q^6.

The edge of 1st side L_1 are q^6, q^5.

$\vdots \quad \vdots \quad \vdots$

The edge of 10th side L_{10} are q^{10}, q^6.

There are many mesh file formats available for communication with other tools such as emc2, modulef.. (see Section 10), The extension of a file gives the chosen type. More details can be found in the article by F. Hecht ”bamg : a bidimentional anisotropic mesh generator” available from the FreeFem web site.

The following give the example write and read files of generated mesh, Freefem can read and write files. A mesh file can be read back into `freefem++` but the names of the borders are lost. So these borders have to be referenced by the number which corresponds to their order of appearance in the program, unless this number is forced by the keyword ”label”.

```
border floor(t=0,1){ x=t; y=0; label=1;};                //    the unit square
border right(t=0,1){ x=1; y=t; label=5;};
border ceiling(t=1,0){ x=t; y=1; label=5;};
border left(t=1,0){ x=0; y=t; label=5;};
int n=10;
mesh th= buildmesh(floor(n)+right(n)+ceiling(n)+left(n));
savemesh(th,"toto.am_fmt");                  //    "formatted Marrocco" format
savemesh(th,"toto.Th");                                //    "bamg"-type mesh
savemesh(th,"toto.msh");                                 //    freefem format
savemesh(th,"toto.nopo");                      //    modulef format  see [8]
mesh th2 = readmesh("toto.msh");                           //    read the mesh
```

The following example explains methods to obtain mesh information.

```
{                                    //    get mesh information (version 1.37)
  mesh Th=square(2,2);
                                                    //    get data of the mesh
  int nbtriangles=Th.nt;
  for (int i=0;i<nbtriangles;i++)
    for (int j=0; j <3; j++)
      cout << i << " " << j << " Th[i][j] = "
```

```
           << Th[i][j] << "  x = "<< Th[i][j].x  << " , y= "<< Th[i][j].y
           << ",  label=" << Th[i][j].label << endl;
}
```

the output is:

```
0 0 Th[i][j] = 0  x = 0 , y= 0,  label=4
0 1 Th[i][j] = 1  x = 0.5 , y= 0,  label=1
0 2 Th[i][j] = 4  x = 0.5 , y= 0.5,  label=0
1 0 Th[i][j] = 0  x = 0 , y= 0,  label=4
1 1 Th[i][j] = 4  x = 0.5 , y= 0.5,  label=0
1 2 Th[i][j] = 3  x = 0 , y= 0.5,  label=4
.......
5 2 Th[i][j] = 6  x = 0 , y= 1,  label=4
6 0 Th[i][j] = 4  x = 0.5 , y= 0.5,  label=0
6 1 Th[i][j] = 5  x = 1 , y= 0.5,  label=2
6 2 Th[i][j] = 8  x = 1 , y= 1,  label=3
7 0 Th[i][j] = 4  x = 0.5 , y= 0.5,  label=0
7 1 Th[i][j] = 8  x = 1 , y= 1,  label=3
7 2 Th[i][j] = 7  x = 0.5 , y= 1,  label=3
```

Example 11 (Readmesh.edp) `border floor(t=0,1){ x=t; y=0; label=1;};  //    the unit square`

```
border right(t=0,1){ x=1; y=t; label=5;};
border ceiling(t=1,0){ x=t; y=1; label=5;};
border left(t=1,0){ x=0; y=t; label=5;};
int n=10;
mesh th= buildmesh(floor(n)+right(n)+ceiling(n)+left(n));
savemesh(th,"toto.am_fmt");                        //    format "formated Marrocco"
savemesh(th,"toto.Th");                        //    format database db mesh "bamg"
savemesh(th,"toto.msh");                                       //    format freefem
savemesh(th,"toto.nopo");                               //    modulef format  see [8]
mesh th2 = readmesh("toto.msh");
fespace femp1(th,P1);
femp1 f = sin(x)*cos(y),g;
{                                                                //    save solution
ofstream file("f.txt");
file << f[] << endl;
}                                                    //    close the file (end block)
{                                                                          //    read
ifstream file("f.txt");
file >> g[] ;
}                                                //    close reading file (end block)
fespace Vh2(th2,P1);
Vh2 u,v;
plot(g);
solve pb(u,v) =
    int2d(th)( u*v - dx(u)*dx(v)-dy(u)*dy(v) )
  + int2d(th)(-g*v)
  + int1d(th,5)( g*v)
  + on(1,u=0) ;
plot (th2,u);
```

3.1.4 Triangulate

`freefem++` is able to build a triangulation from a set of points. This triangulation is a Delaunay mesh of the convex hull of the set of points. It can be useful to build a mesh form a table function.
The coordinates of the points and the value of the table function are defined separately with rows of the form: `x y f(x,y)` in a file such as:

```
0.51387 0.175741 0.636237
0.308652 0.534534 0.746765
0.947628 0.171736 0.899823
0.702231 0.226431 0.800819
0.494773 0.12472 0.580623
0.0838988 0.389647 0.456045
...............
```

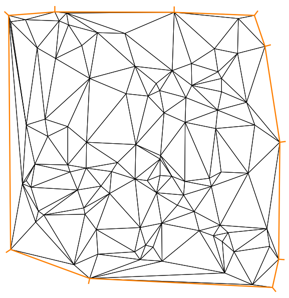


Figure 3.6: Delaunay mesh of the convex hull of point set in file xyf

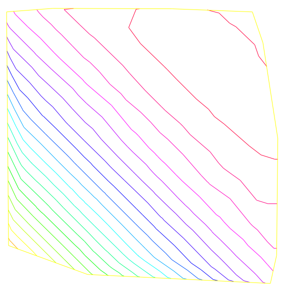


Figure 3.7: Isovalue of table function

The third column of each line is left untouched by the `triangulate` command. But you can use this third value to define a table function with rows of the form: `x y f(x,y)`.
The following example shows how to make mesh from the file "xyf" with the format stated just above. The command `triangulate` command use only use 1st and 2nd rows.

```
mesh Thxy=triangulate("xyf"); //    build the Delaunay mesh of the convex hull
                          //    points are defined by the first 2 columns of file xyf
plot(Thxy,ps="Thxyf.ps");                                     //    (see figure 3.6)

fespace Vhxy(Thxy,P1);                             //    create a P1 interpolation
Vhxy fxy;                                                     //    the function

                                       //    reading the 3rd row to define the function
{ ifstream file("xyf");
   real xx,yy;
   for(int i=0;i<fxy.n;i++)
   file >> xx >>yy >> fxy[][i];                         //    to read third row only.
                                                      //    xx and yy are just skipped
}
plot(fxu,ps="xyf.eps");                      //    plot the function (see figure 3.7)
```

3.2 build empty mesh

When you want to define Finite Element space on boundary, we come up with the idea of a mesh with no internal points (call empty mesh). It can be useful when you have a Lagrange multiplier definied on the border.
So the function emptymesh remove all the internal point of a mesh expect if the point is on internal boundary.

```
{                                   //    new stuff 2004 emptymesh (version 1.40)
                                     //    -- useful to build Multiplicator space
                                        //    build a mesh without internal point
                                                     //    with the same boundary
                                                                      //    -----
  assert(version>=1.40);
  border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
  mesh Th=buildmesh(a(20));
   Th=emptymesh(Th);
  plot(Th,wait=1,ps="emptymesh-1.eps");                        //   see figure 3.8
}
```

or it is also possible to build a empty mesh of peusdo subregion with `emptymesh(Th,ssd)` with the set of edges of the mesh `Th` a edge e is in this set if the two adjacent triangles $e = t1 \cap t2$ and $ssd[T1] \neq ssd[T2]$ where `ssd` defined the peusdo region numbering of triangle, when they are stored in `int[int]` array of size the number of triangles.

```
{  //    new stuff 2004 emptymesh (version 1.40)
 //    -- useful to build Multiplicator space
 //    build a mesh without internal point
 //    of peusdo sub domain
 //    -----
  assert(version>=1.40);
  mesh Th=square(10,10);
  int[int] ssd(Th.nt);
  for(int i=0;i<ssd.n;i++)                //    build the peusdo region numbering
   {  int iq=i/2;                               //    because 2 traingle per quad
      int ix=iq%10;                                                          //
      int iy=iq/10;                                                          //
    ssd[i]= 1 + (ix>=5) +  (iy>=5)*2;
   }
  Th=emptymesh(Th,ssd);                                  //    build emtpy with
                                          //    all edge e = T1 ∩ T2 and ssd[T1] ≠ ssd[T2]
  plot(Th,wait=1,ps="emptymesh-2.eps");                        //   see figure 3.9
  savemesh(Th,"emptymesh-2.msh");
}
```

3.3 Remeshing

3.3.1 Movemesh

After solving the elasticity, we want watch the deformation $\Omega \mapsto \Phi(\Omega)$, $\mathbf{\Phi}(x,y) = (\Phi_1(x,y), \Phi_2(x,y))$ of shape. In free boundary value problems, the shape of domain will vary.

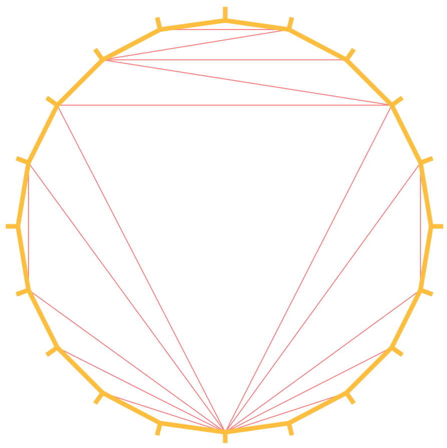


Figure 3.8: The empty mesh with boundary

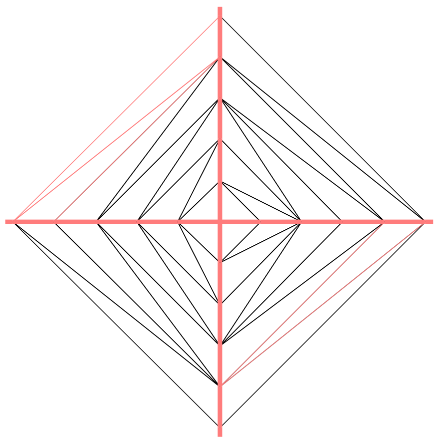


Figure 3.9: An empty mesh defined from a peusdo region numbering of triangle

If Ω is triangulated already – dubbed $T_h(\Omega)$, then we want $\Phi(\Omega)$ is also triangulated automatically. This want is satisfied by

```
    mesh   Th=movemesh(Th,[Φ1,Φ2]);
```

where $\Phi = (\Phi_1, \Phi_2)$ and Φ_i, $i = 1, 2$ are functions. Sometimes the moved mesh is invalid because some triangle becomes reversed (with a negative area). This is why we check the minimum triangle area in the transformed mesh with `checkmovemesh` before any real transformation.

Example 12 $\Phi_1(x, y) = x + k * \sin(y * \pi)/10)$, $\Phi_2(x, y) = y + k * \cos(y\pi)/10)$ *for a big number* $k > 1$.

```
verbosity=4;
border a(t=0,1){x=t;y=0;label=1;};
border b(t=0,0.5){x=1;y=t;label=1;};
border c(t=0,0.5){x=1-t;y=0.5;label=1;};
border d(t=0.5,1){x=0.5;y=t;label=1;};
border e(t=0.5,1){x=1-t;y=1;label=1;};
border f(t=0,1){x=0;y=1-t;label=1;};
func uu= sin(y*pi)/10;
func vv= cos(x*pi)/10;

mesh Th = buildmesh ( a(6) + b(4) + c(4) +d(4) + e(4) + f(6));
plot(Th,wait=1,fill=1,ps="Lshape.eps");                        //    see figure 3.10
real coef=1;
real minT0= checkmovemesh(Th,[x,y]);                     //    the min triangle area
while(1)                                               //    find a correct move mesh
{
  real minT=checkmovemesh(Th,[x+coef*uu,y+coef*vv]);//   the min triangle area
  if (minT > minT0/5) break ;                                //    if big enough
  coef=/1.5;
}
```

```
Th=movemesh(Th,[x+coef*uu,y+coef*vv]);
plot(Th,wait=1,fill=1,ps="movemesh.eps");                     //    see figure 3.11
```

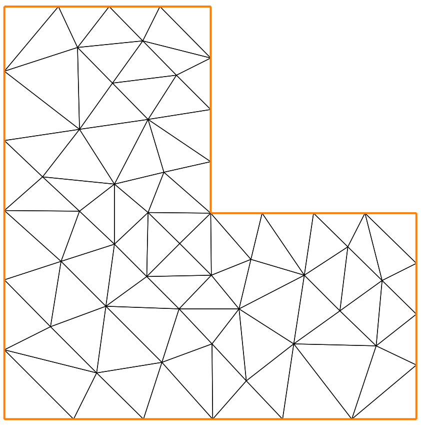


Figure 3.10: L-shape

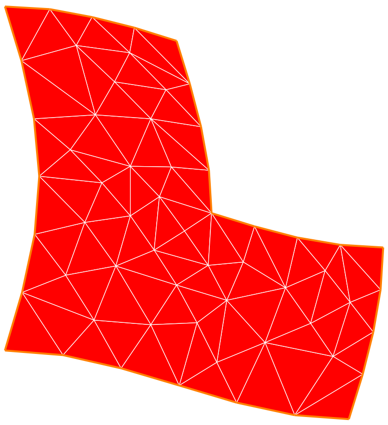


Figure 3.11: moved L-shape

Note 9 *Consider a function u defined on a mesh* `Th`*. A statement like* `Th=movemesh(Th...)` *does not change u and so the old mesh still exists. It will be destroyed when no function use it. A statement like $u = u$ redefines u on the new mesh* `Th` *with interpolation and therefore destroys the old* `Th` *if u was the only function using it.*

Example 13 (movemesh.edp) *Now, we given an example of moving mesh with lagrangian function u defined on the moving mesh.*

```
                                                    //    simple movemesh example
mesh Th=square(10,10);
fespace Vh(Th,P1);
real t=0;
                                                                     //    ---
                         //    the problem is how to build data without interpolation
                //    so the data u is moving with the mesh as you can see in the plot
                                                                     //    ---
Vh u=y;
for (int i=0;i<4;i++)
{
 t=i*0.1;
 Vh f= x*t;
 real minarea=checkmovemesh(Th,[x,y+f]);
 if (minarea >0 )                                        //    movemesh will be ok
   Th=movemesh(Th,[x,y+f]);

 cout << " Min area  " << minarea << endl;

 real[int] tmp(u[].n);
 tmp=u[];                                                    //    save the value
 u=0;                   //    to change the FEspace and mesh associated with u
 u[]=tmp;                     //    set the value of u without any mesh update
 plot(Th,u,wait=1);
};
//    In this program, since u is only defined on the last mesh, all the
//    previous meshes are deleted from memory.
//    --------
```

3.4 Regular Triangulation

For a set S, we define the diameter of S by

$$\mathrm{diam}(S) = \sup\{|\boldsymbol{x}-\boldsymbol{y}|;\ \boldsymbol{x},\,\boldsymbol{y}\in S\}$$

The sequence $\{\mathcal{T}_h\}_{h\downarrow 0}$ of Ω is called *regular* if they satisfy the following:

1. $$\lim_{h\downarrow 0}\max\{\mathrm{diam}(T_k)|\ T_k\in\mathcal{T}_h\}$$

2. There is a number $\sigma>0$ independent of h such that

$$\frac{\rho(T_k)}{\mathrm{diam}(T_k)}\geq\sigma \qquad \text{for all } T_k\in\mathcal{T}_h$$

   where $\rho(T_k)$ are the diameter of the inscribed circle of T_k.

We put $h(\mathcal{T}_h)=\max\{\mathrm{diam}(T_k)|\ T_k\in\mathcal{T}_h\}$, which is obtained by

```
   mesh Th = ......;
   fespace Ph(Th,P0);
   Ph h = hTriangle;
   cout << "size of mesh = " << h[].max << endl;
```

3.5 Adaptmesh

The function

$$f(x,y)=10.0x^3+y^3+\tan^{-1}[\epsilon/(\sin(5.0y)-2.0x)] \qquad \epsilon=0.0001$$

sharply vary its value. However, the initial mesh given by the command in Section 3.1 cannot reflect its sharp variation.

Example 14

```
real eps =  0.0001;
real h=1;
real hmin=0.05;
func f = 10.0*x^3+y^3+h*atan2(eps,sin(5.0*y)-2.0*x);
```

```
mesh Th=square(5,5,[-1+2*x,-1+2*y]);
fespace Vh(Th,P1);
Vh fh=f;
plot(fh);
for (int i=0;i<2;i++)
 {
   Th=adaptmesh(Th,fh);
   fh=f;                                              //    old mesh is deleted
   plot(Th,fh,wait=1);
 }
```

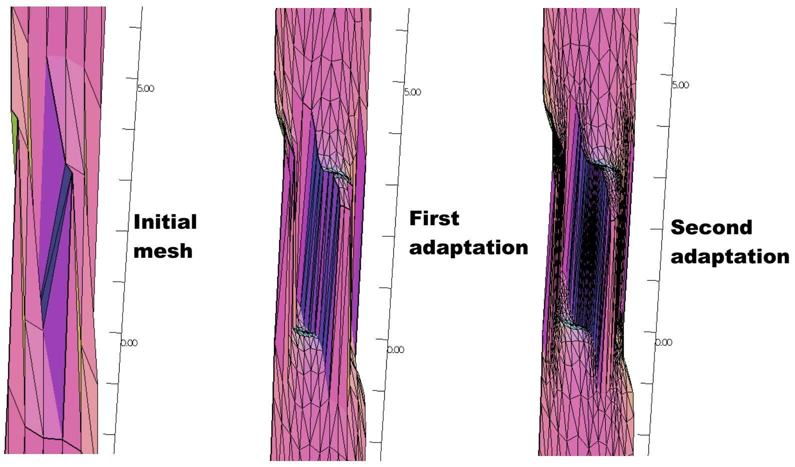


Figure 3.12: 3D graph under the initial mesh and after of 1st and 2nd adaptation

`freefem++` uses a variable metric/Delaunay automatic meshing algorithm. The command

```
   mesh ATh = adaptmesh(Th, f);
```

create the new mesh `ATh` by the Hessian

$$D^2 f = (\partial^2 f/\partial x^2,\, \partial^2 f/\partial x\partial y, \partial^2 f/\partial y^2)$$

of a function (formula or FE-function). Mesh adaptation is a very powerful tool when the solution of a problem vary locally and sharply.
Here we solve the problem (1.1)-(1.2), when $f = 1$ and Ω is L-shape domain.

Example 15 (Adapt.edp) *The solution has the singularity $r^{3/2}$, $r = |x - \gamma|$ at the point γ of the intersection of two lines bc and bd (see Fig. 3.13).*

```
border ba(t=0,1.0){x=t;   y=0;  label=1;};
border bb(t=0,0.5){x=1;   y=t;  label=1;};
border bc(t=0,0.5){x=1-t; y=0.5;label=1;};
border bd(t=0.5,1){x=0.5; y=t;  label=1;};
border be(t=0.5,1){x=1-t; y=1;  label=1;};
border bf(t=0.0,1){x=0;   y=1-t;label=1;};
mesh Th = buildmesh ( ba(6)+bb(4)+bc(4)+bd(4)+be(4)+bf(6) );
fespace Vh(Th,P1);                                      //    set FE space
Vh u,v;                                    //    set unknown and test function
```

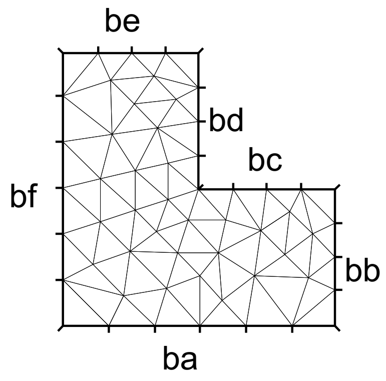


Figure 3.13: L-shape domain and its boundary name

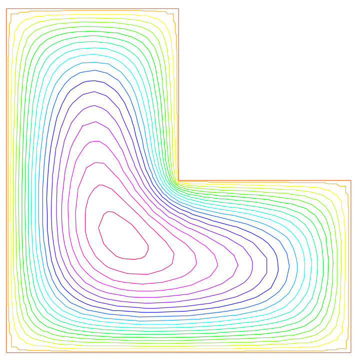


Figure 3.14: Final solution after 4-times adaptation

```
func f = 1;
real error=0.1;                                                    //    level of error
problem Poisson(u,v,solver=CG,eps=1.0e-6) =
    int2d(Th)(  dx(u)*dx(v) + dy(u)*dy(v))
  - int2d(Th) ( f*v )
  + on(1,u=0)  ;
for (int i=0;i< 4;i++)
{
  Poisson;
  Th=adaptmesh(Th,u,err=error);
  error = error/2;
} ;
plot(u);
```

To speed up the adaptation we change by hand a default parameter `err` of `adaptmesh`, which specifies the required precision, so as to make the new mesh finer. The problem is coercive and symmetric, so the linear system can be solved with the conjugate gradient method (parameter `solver=CG` with the stopping criteria on the residual, here `eps=1.0e-6`). By `adaptmesh`, we get good slope of the final solution near the point of intersection of *bc* and *bd* as in Fig. 3.14.
This method is described in detail in [7]. It has a number of default parameters which can be modified :

`hmin=` Minimum edge size. (`val` is a real. Its default is related to the size of the domain to be meshed and the precision of the mesh generator).

`hmax=` Maximum edge size. (`val` is a real. It defaults to the diameter of the domain to be meshed)

`err=` P^1 interpolation error level (0.01 is the default).

`errg=` Relative geometrical error. By default this error is 0.01, and in any case it must be lower than $1/\sqrt{2}$. Meshes created with this option may have some edges smaller than the `-hmin` due to geometrical constraints.

nbvx= Maximum number of vertices generated by the mesh generator (9000 is the default).

nbsmooth= number of iterations of the smoothing procedure (5 is the default).

nbjacoby= number of iterations in a smoothing procedure during the metric construction, 0 means no smoothing (6 is the default).

ratio= ratio for a prescribed smoothing on the metric. If the value is 0 or less than 1.1 no smoothing is done on the metric (1.8 is the default).

If `ratio` > 1.1, the speed of mesh size variations is bounded by $log(\texttt{ratio})$. Note: As `ratio` gets closer to `1`, the number of generated vertices increases. This may be useful to control the thickness of refined regions near shocks or boundary layers .

omega= relaxation parameter for the smoothing procedure (1.0 is the default).

iso= If true, forces the metric to be isotropic (false is the default).

abserror= If false, the metric is evaluated using the criterium of equi-repartion of relative error (false is the default). In this case the metric is defined by

$$\mathcal{M} = \left(\frac{1}{\texttt{err coef}^2} \quad \frac{|\mathcal{H}|}{max(\texttt{CutOff}, |\eta|)} \right)^p \tag{3.1}$$

otherwise, the metric is evaluated using the criterium of equi-distribution of errors. In this case the metric is defined by

$$\mathcal{M} = \left(\frac{1}{\texttt{err coef}^2} \quad \frac{|\mathcal{H}}{sup(\eta) - inf(\eta)} \right)^p . \tag{3.2}$$

cutoff= lower limit for the relative error evaluation (1.0e-6 is the default).

verbosity= informational messages level (can be chosen between 0 and ∞). Also changes the value of the global variable verbosity (obsolete).

inquire= To inquire graphicaly about the mesh (false is the default).

splitpbedge= If true, splits all internal edges in half with two boundary vertices (true is the default).

maxsubdiv= Changes the metric such that the maximum subdivision of a background edge is bound by `val` (always limited by 10, and 10 is also the default).

rescaling= if true, the function with respect to which the mesh is adapted is rescaled to be between 0 and 1 (true is the default).

keepbackvertices= if true, tries to keep as many vertices from the original mesh as possible (true is the default).

isMetric= if true, the metric is defined explicitly (false is the default). If the 3 functions m_{11}, m_{12}, m_{22} are given, they directly define a symmetric matrix field whose Hessian is computed to define a metric. If only one function is given, then it represents the isotropic mesh size at every point.

For example, if the partial derivatives `fxx` $(= \partial^2 f/\partial x^2)$, `fxx` $(= \partial^2 f/\partial x \partial y)$, `fyy` $(= \partial^2 f/\partial y^2)$ are given, we can set

```
Th=adaptmesh(Th,fxx,fxy,fyy,IsMetric=1,nbvx=10000,hmin=hmin);
```

power= exponent power of the Hessian used to compute the metric (1 is the default).

thetamax= minimum corner angle in degrees (default is 0).

splitin2= boolean value. If true, splits all triangles of the final mesh into 4 sub-triangles.

metric= an array of 3 real arrays to set or get metric data information. The size of these three arrays must be the number of vertices. So if `m11,m12,m22` are three P1 finite elements related to the mesh to adapt, you can write: `metric=[m11[],m12[],m22[]]` (see file convect-apt.edp for a full example)

nomeshgeneration= If true, no adapted mesh is generated (useful to compute only a metric).

periodic= As writing `periodic=[[4,y],[2,y],[1,x],[3,x]];` it builds an adapted periodic mesh. The sample build a biperiodic mesh of a square. (see periodic finite element spaces 4, and see `sphere.edp` for a full example)

3.6 Trunc

A small operator to create a truncated mesh from a mesh with respect to a boolean function. The two named parameter

label= sets the label number of new boundary item (one by default)

split= sets the level n of triangle splitting. each triangle is splitted in $n \times n$ (one by default).

To create the mesh `Th3` where alls triangles of a mesh `Th` are splitted in 3×3 , just write:

```
  mesh Th3 = trunc(Th,1,split=3);
```

The `truncmesh.edp` example construct all "trunc" mesh to the support of the basic function of the space `Vh` (cf. `abs(u)>0`), split all the triangles in 5×5, and put a label number to 2 on new boundary.

```
mesh Th=square(3,3);
fespace Vh(Th,P1);
Vh u;
```

```
int i,n=u.n;
u=0;
for (i=0;i<n;i++)                                    //    all degre of freedom
 {
  u[][i]=1;                                          //    the basic function i
  plot(u,wait=1);
  mesh Sh1=trunc(Th,abs(u)>1.e-10,split=5,label=2);
  plot(Th,Sh1,wait=1,ps="trunc"+i+".eps");              //    plot the mesh of
                                                    //    the function's support
  u[][i]=0;                                                        //    reset
 }
```

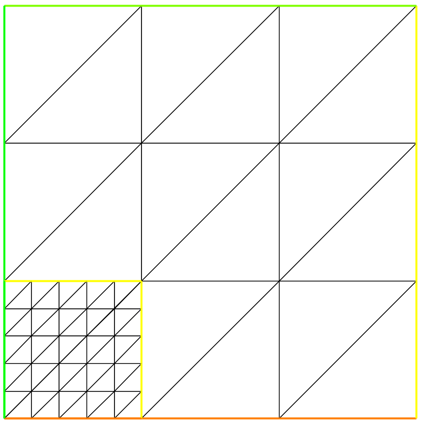


Figure 3.15: mesh of support the function P1 number 0, splitted in 5 × 5

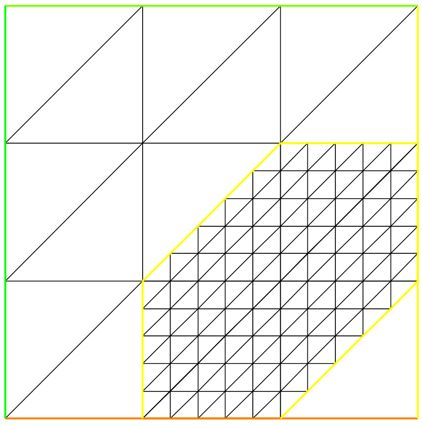


Figure 3.16: mesh of support the function P1 number 6, splitted in 5 × 5

3.7 splitmesh

A other way to split mesh triangle:

```
{                                    //    new stuff 2004 splitmesh (version 1.37)
  assert(version>=1.37);
  border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
  plot(Th,wait=1,ps="nosplitmesh.eps");                  //    see figure 3.17
  mesh Th=buildmesh(a(20));
  plot(Th,wait=1);
  Th=splitmesh(Th,1+5*(square(x-0.5)+y*y));
  plot(Th,wait=1,ps="splitmesh.eps");                    //    see figure 3.18
}
```

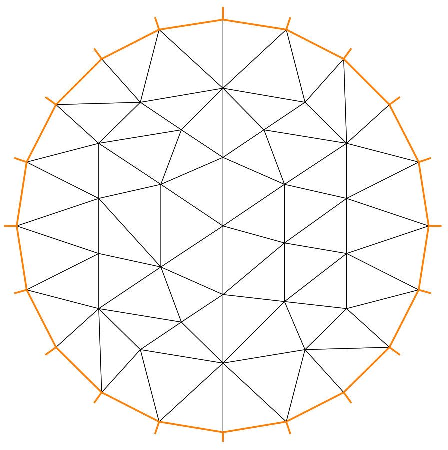


Figure 3.17: initial mesh

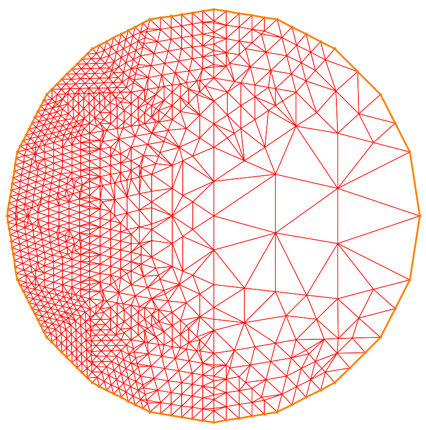


Figure 3.18: all left mesh triangle is split conformaly in `int(1+5*(square(x-0.5)+y*y)^2` triangles.

3.8 A Fast Finite Element Interpolator

In practice one may discretize the variational equations by the Finite Element method. Then there will be one mesh for Ω_1 and another one for Ω_2. The computation of integrals of products of functions defined on different meshes is difficult. Quadrature formulae and interpolations from one mesh to another at quadrature points are needed. We present below the interpolation operator which we have used and which is new, to the best of our knowledge. Let $\mathcal{T}_h^0 = \cup_k T_k^0, \mathcal{T}_h^1 = \cup_k T_k^1$ be two triangulations of a domain Ω. Let

$$V(\mathcal{T}_h^i) = \{C^0(\Omega_h^i) \ : \ f|_{T_k^i} \in P^1\}, \quad i = 0, 1$$

be the spaces of continuous piecewise affine functions on each triangulation.
Let $f \in V(\mathcal{T}_h^0)$. The problem is to find $g \in V(\mathcal{T}_h^1)$ such that

$$g(q) = f(q) \quad \forall q \text{ vertex of } \ \mathcal{T}_h^1$$

Although this is a seemingly simple problem, it is difficult to find an efficient algorithm in practice. We propose an algorithm which is of complexity $N^1 \log N^0$, where N^i is the number of vertices of $\mathcal{T}_h^i$, and which is very fast for most practical 2D applications.

Algorithm
The method has 5 steps. First a quadtree is built containing all the vertices of mesh $\mathcal{T}_h^0$ such that in each terminal cell there are at least one, and at most 4, vertices of $\mathcal{T}_h^0$.
For each q^1, vertex of $\mathcal{T}_h^1$ do:

Step 1 Find the terminal cell of the quadtree containing q^1.

Step 2 Find the the nearest vertex q_j^0 to q^1 in that cell.

Step 3 Choose one triangle $T_k^0 \in \mathcal{T}_h^0$ which has q_j^0 for vertex.

Step 4 Compute the barycentric coordinates $\{\lambda_j\}_{j=1,2,3}$ of q^1 in T_k^0.

- – if all barycentric coordinates are positive, go to Step 5
- – else if one barycentric coordinate λ_i is negative replace T_k^0 by the adjacent triangle opposite q_i^0 and go to Step 4.
- – else two barycentric coordinates are negative so take one of the two randomly and replace T_k^0 by the adjacent triangle as above.

Step 5 Calculate $g(q^1)$ on T_k^0 by linear interpolation of f:

$$g(q^1) = \sum_{j=1,2,3} \lambda_j f(q_j^0)$$

End

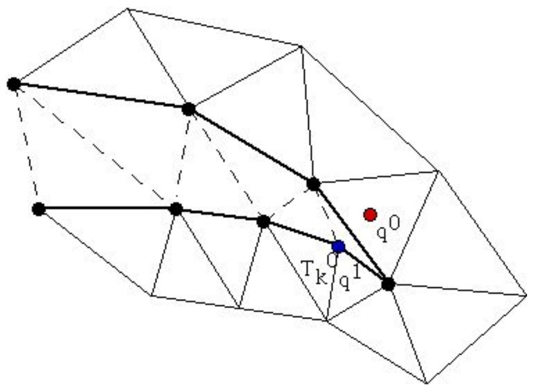


Figure 3.19: To interpolate a function at q^0 the knowledge of the triangle which contains q^0 is needed. The algorithm may start at $q^1 \in T_k^0$ and stall on the boundary (thick line) because the line q^0q^1 is not inside Ω. But if the holes are triangulated too (doted line) then the problem does not arise.

Two problems needs to solved:

- *What if q^1 is not in Ω_h^0 ?* Then Step 5 will stop with a boundary triangle. So we add a step which test the distance of q^1 with the two adjacent boundary edges and select the nearest, and so on till the distance grows.

- *What if Ω_h^0 is not convex and the marching process of Step 4 locks on a boundary?* By construction Delaunay-Voronoi mesh generators always triangulate the convex hull of the vertices of the domain. So we make sure that this information is not lost when $\mathcal{T}_h^0, \mathcal{T}_h^1$ are constructed and we keep the triangles which are outside the domain in a special list. Hence in step 5 we can use that list to step over holes if needed.

Note 10 *Step 3 requires an array of pointers such that each vertex points to one triangle of the triangulation.*

3.9 Meshing Examples

Example 16 (Two rectangles touching by a side)

```
border a(t=0,1){x=t;y=0;};
border b(t=0,1){x=1;y=t;};
border c(t=1,0){x=t ;y=1;};
border d(t=1,0){x = 0; y=t;};
border c1(t=0,1){x=t ;y=1;};
border e(t=0,0.2){x=1;y=1+t;};
border f(t=1,0){x=t ;y=1.2;};
border g(t=0.2,0){x=0;y=1+t;};
int n=1;
mesh th = buildmesh(a(10*n)+b(10*n)+c(10*n)+d(10*n));
mesh TH = buildmesh ( c1(10*n) + e(5*n) + f(10*n) + g(5*n) );
plot(th,TH,ps="TouchSide.esp");                             //   Fig.  3.20
```

Example 17 (NACA0012 Airfoil)

```
border upper(t=0,1) { x = t;
     y = 0.17735*sqrt(t)-0.075597*t
  - 0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4); }
border lower(t=1,0) { x = t;
     y= -(0.17735*sqrt(t)-0.075597*t
  -0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4)); }
border c(t=0,2*pi) { x=0.8*cos(t)+0.5;  y=0.8*sin(t); }
mesh Th = buildmesh(c(30)+upper(35)+lower(35));
plot(Th,ps="NACA0012.eps",bw=1);                            //   Fig.  3.21
```

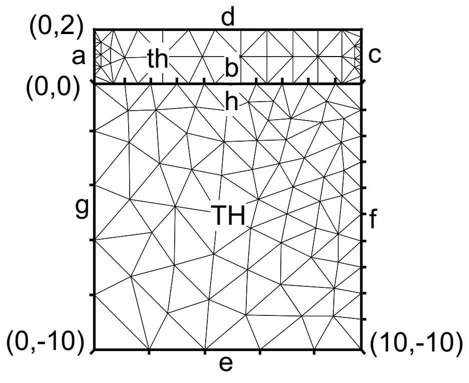


Figure 3.20: Two rectangles touching by a side

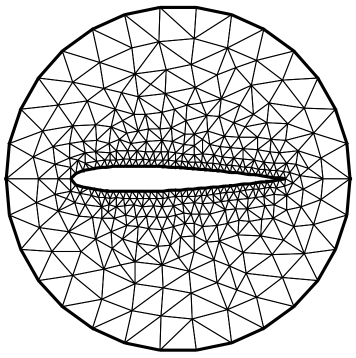


Figure 3.21: NACA0012 Airfoil

Example 18 (Cardioid)

```
real b = 1, a = b;
border C(t=0,2*pi) { x=(a+b)*cos(t)-b*cos((a+b)*t/b);
```

```
                           y=(a+b)*sin(t)-b*sin((a+b)*t/b); }
mesh Th = buildmesh(C(50));
plot(Th,ps="Cardioid.eps",bw=1);                              //    Fig.  3.22
```

Example 19 (Cassini Egg)

```
border C(t=0,2*pi) { x=(2*cos(2*t)+3)*cos(t);
                      y=(2*cos(2*t)+3)*sin(t); }
mesh Th = buildmesh(C(50));
plot(Th,ps="Cassini.eps",bw=1);                               //    Fig.  3.23
```

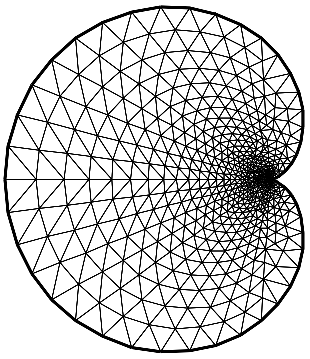


Figure 3.22: Domain with Cardioid curve boundary

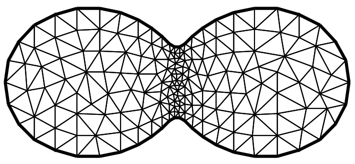


Figure 3.23: Domain with Cassini Egg curve boundary

Example 20 (By cubic Bezier curve)

```
      //    A cubic Bezier curve connecting two points with two control points
func real bzi(real p0,real p1,real q1,real q2,real t)
{
  return p0*(1-t)^3+q1*3*(1-t)^2*t+q2*3*(1-t)*t^2+p1*t^3;
}

real[int] p00=[0,1], p01=[0,-1], q00=[-2,0.1], q01=[-2,-0.5];
real[int] p11=[1,-0.9], q10=[0.1,-0.95], q11=[0.5,-1];
real[int] p21=[2,0.7], q20=[3,-0.4], q21=[4,0.5];
real[int] q30=[0.5,1.1], q31=[1.5,1.2];
border G1(t=0,1) { x=bzi(p00[0],p01[0],q00[0],q01[0],t);
                   y=bzi(p00[1],p01[1],q00[1],q01[1],t); }
border G2(t=0,1) { x=bzi(p01[0],p11[0],q10[0],q11[0],t);
                   y=bzi(p01[1],p11[1],q10[1],q11[1],t); }
border G3(t=0,1) { x=bzi(p11[0],p21[0],q20[0],q21[0],t);
                   y=bzi(p11[1],p21[1],q20[1],q21[1],t); }
border G4(t=0,1) { x=bzi(p21[0],p00[0],q30[0],q31[0],t);
                   y=bzi(p21[1],p00[1],q30[1],q31[1],t); }
int m=5;
mesh Th = buildmesh(G1(2*m)+G2(m)+G3(3*m)+G4(m));
plot(Th,ps="Bezier.eps",bw=1);                                  //    Fig 3.24
```

Example 21 (Section of Engine)

```
real a= 6., b= 1., c=0.5;
border L1(t=0,1) { x= -a; y= 1+b - 2*(1+b)*t; }
border L2(t=0,1) { x= -a+2*a*t; y= -1-b*(x/a)*(x/a)*(3-2*abs(x)/a );}
border L3(t=0,1) { x= a; y=-1-b + (1+ b )*t; }
border L4(t=0,1) { x= a - a*t;   y=0; }
border L5(t=0,pi) { x= -c*sin(t)/2; y=c/2-c*cos(t)/2; }
border L6(t=0,1) { x= a*t;  y=c; }
border L7(t=0,1) { x= a;  y=c + (1+ b-c )*t; }
border L8(t=0,1) { x= a-2*a*t; y= 1+b*(x/a)*(x/a)*(3-2*abs(x)/a); }
mesh Th = buildmesh(L1(8)+L2(26)+L3(8)+L4(20)+L5(8)+L6(30)+L7(8)+L8(30));
plot(Th,ps="Engine.eps",bw=1);                          //    Fig.  3.25
```

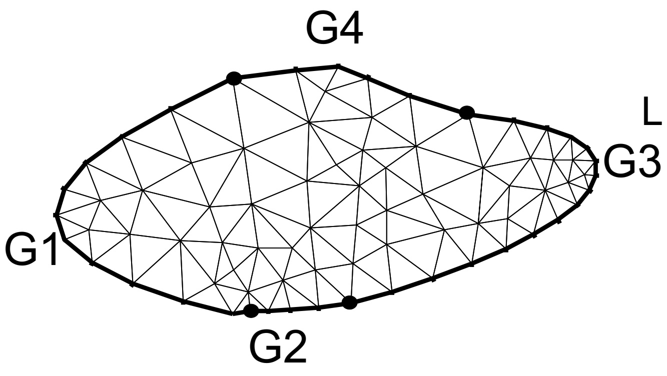


Figure 3.24: Boundary drawed by Bezier curves

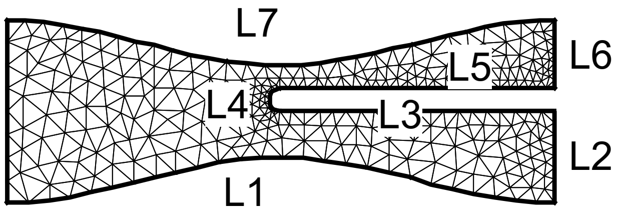


Figure 3.25: Section of Engine

Example 22 (Domain with U-shape channel)

```
real d = 0.1;                                        //    width of U-shape
border L1(t=0,1-d) { x=-1; y=-d-t; }
border L2(t=0,1-d) { x=-1; y=1-t; }
border B(t=0,2) { x=-1+t; y=-1; }
border C1(t=0,1) { x=t-1; y=d; }
border C2(t=0,2*d) { x=0; y=d-t; }
border C3(t=0,1) { x=-t; y=-d; }
border R(t=0,2) { x=1; y=-1+t; }
border T(t=0,2) { x=1-t; y=1; }
int n = 5;
mesh Th = buildmesh (L1(n/2)+L2(n/2)+B(n)+C1(n)+C2(3)+C3(n)+R(n)+T(n));
plot(Th,ps="U-shape.eps",bw=1);                          //    Fig 3.26
```

Example 23 (Domain with V-shape cut)

```
real dAg = 0.01;                                     //    angle of V-shape
border C(t=dAg,2*pi-dAg) { x=cos(t); y=sin(t); };
real[int] pa(2), pb(2), pc(2);
```

```
pa[0] = cos(dAg); pa[1] = sin(dAg);
pb[0] = cos(2*pi-dAg); pb[1] = sin(2*pi-dAg);
pc[0] = 0; pc[1] = 0;
border seg1(t=0,1) { x=(1-t)*pb[0]+t*pc[0]; y=(1-t)*pb[1]+t*pc[1]; };
border seg2(t=0,1) { x=(1-t)*pc[0]+t*pa[0]; y=(1-t)*pc[1]+t*pa[1]; };
mesh Th = buildmesh(seg1(20)+C(40)+seg2(20));
plot(Th,ps="V-shape.eps",bw=1);                            //    Fig.  3.27
```

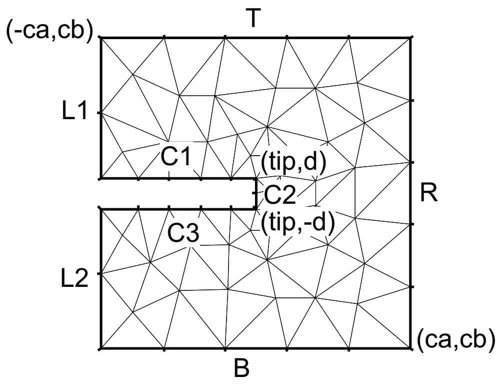


Figure 3.26: Domain with U-shape channel changed by d

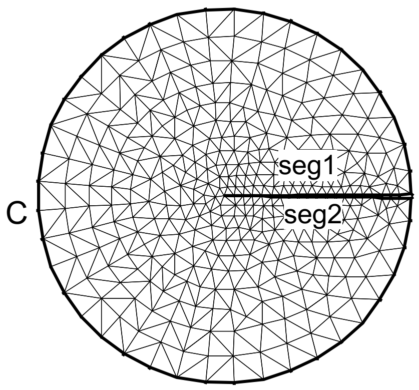


Figure 3.27: Domain with V-shape cut changed by dAg

Example 24 (Smiling face)

```
real d=0.1;
int m=5;
real a=1.5, b=2, c=0.7, e=0.01;
border F(t=0,2*pi) { x=a*cos(t); y=b*sin(t); }
border E1(t=0,2*pi) { x=0.2*cos(t)-0.5; y=0.2*sin(t)+0.5; }
border E2(t=0,2*pi) { x=0.2*cos(t)+0.5; y=0.2*sin(t)+0.5; }
func real st(real t) {
   return sin(pi*t)-pi/2;
}
border C1(t=-0.5,0.5) { x=(1-d)*c*cos(st(t)); y=(1-d)*c*sin(st(t)); }
border C2(t=0,1){x=((1-d)+d*t)*c*cos(st(0.5));y=((1-d)+d*t)*c*sin(st(0.5));}
border C3(t=0.5,-0.5) { x=c*cos(st(t)); y=c*sin(st(t)); }
border C4(t=0,1) { x=(1-d*t)*c*cos(st(-0.5)); y=(1-d*t)*c*sin(st(-0.5));}

border C0(t=0,2*pi) { x=0.1*cos(t); y=0.1*sin(t); }
mesh Th=buildmesh(F(10*m)+C1(2*m)+C2(3)+C3(2*m)+C4(3)
                  +C0(m)+E1(-2*m)+E2(-2*m));
plot(Th,ps="SmileFace.eps",bw=1);                      //    see Fig.  3.28
}
```

Example 25 (3point bending)

```
               //    Square for Three-Point Bend Speicmens fixed on Fix1, Fix2
                                                //    It will be loaded on Load.
real a=1, b=5, c=0.1;
int n=5, m=b*n;
border Left(t=0,2*a) { x=-b; y=a-t; }
border Bot1(t=0,b/2-c) { x=-b+t; y=-a; }
border Fix1(t=0,2*c) { x=-b/2-c+t; y=-a; }
border Bot2(t=0,b-2*c) { x=-b/2+c+t; y=-a; }
border Fix2(t=0,2*c) { x=b/2-c+t; y=-a; }
border Bot3(t=0,b/2-c) { x=b/2+c+t; y=-a; }
border Right(t=0,2*a) { x=b; y=-a+t; }
border Top1(t=0,b-c) { x=b-t; y=a; }
border Load(t=0,2*c) { x=c-t; y=a; }
border Top2(t=0,b-c) { x=-c-t; y=a; }
mesh Th = buildmesh(Left(n)+Bot1(m/4)+Fix1(5)+Bot2(m/2)+Fix2(5)+Bot3(m/4)
                    +Right(n)+Top1(m/2)+Load(10)+Top2(m/2));
plot(Th,ps="ThreePoint.eps",bw=1);                              //    Fig.  3.29
```

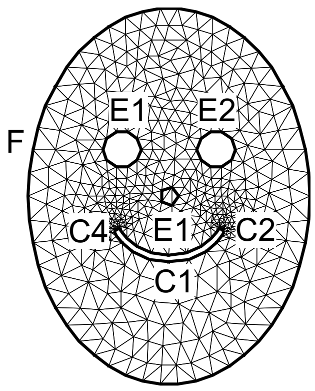


Figure 3.28: Smiling face (Mouth is changeable)

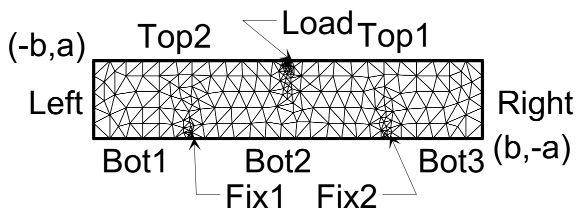


Figure 3.29: Domain for three-point bending test

Chapter 4

Finite Elements

As stated in Step2 in Section 1.2. FEM make approximations all functions w to

$$w(x,y) \simeq w_0\phi_0(x,y) + w_1\phi_1(x,y) + \cdots + w_{M-1}\phi_{M-1}(x,y)$$

with finite basis functions $\phi_k(x,y)$ and numbers w_k ($k = 0, \cdots, M-1$). The functions $\phi_k(x,y)$ is constructed from the triangle T_{i_k}, so $\phi_k(x,y)$ is called *shape function*. The finite element space

$$V_h = \{w \mid w_0\phi_0 + w_1\phi_1 + \cdots + w_{M-1}\phi_{M-1},\ w_i \in \mathbb{R}\}$$

is easily created by

```
        fespace IDspace(IDmesh,<IDFE>) ;
```

or with ℓ pair of periodic boundary condition

```
        fespace IDspace(IDmesh,<IDFE>,
                        periodic=[[la_1,sa_1],[lb_1,sb_1],
                                 ...
                                 [la_k,sa_k],[lb_k,sb_ℓ]]);
```

where `IDspace` is the name of the space (e.g. `Vh`), `IDmesh` is the name of the associated mesh and `<IDFE>` is a identifier of finite element type. In a pair of periodic boundary condition, if `[la`$_i$`,sa`$_i$`],[lb`$_i$`,sb`$_i$`]` is a pair of `int`, this expressions the 2 labels `la`$_i$ and `lb`$_i$ of the piece of the boundary to be equivalence; If `[la`$_i$`,sa`$_i$`],[lb`$_i$`,sb`$_i$`]` is a pair of `real`, this expressions `sa`$_i$ and `sb`$_i$ give two common abscissa on the two boundary curve, and two points are identify if the two abscissa are equal.

As of today, the known types of finite element are:

P0 piecewise constante discontinuous finite element

$$P0_h = \left\{ v \in L^2(\Omega) \,\middle|\, \text{for all } K \in \mathcal{T}_h \text{ there is } \alpha_K \in \mathbb{R} : \quad v_{|K} = \alpha_K \right\} \tag{4.1}$$

P1 piecewise linear continuous finite element

$$P1_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_1 \right\} \tag{4.2}$$

P1dc piecewise linear discontinuous finite element

$$P1dc_h = \left\{ v \in L^2(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_1 \right\} \tag{4.3}$$

P1b piecewise linear continuous finite element plus bubble

$$P1b_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_1 \oplus Span\{\lambda_0^K \lambda_1^K \lambda_2^K\} \right\} \tag{4.4}$$

where $\lambda_i^K, i = 0,1,2$ are the 3 area coordinate functions of the triangle K

P2 piecewise P_2 continuous finite element,

$$P2_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_2 \right\} \tag{4.5}$$

where P_2 is the set of polynomials of $\mathbb{R}^2$ of degrees ≤ 2.

P2b piecewise P_2 continuous finite element plus bubble,

$$P2_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_2 \oplus Span\{\lambda_0^K \lambda_1^K \lambda_2^K\} \right\} \tag{4.6}$$

P2dc piecewise P_2 discontinuous finite element,

$$P2dc_h = \left\{ v \in L^2(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_2 \right\} \tag{4.7}$$

RT0 Raviart-Thomas finite element

$$RT0_h = \left\{ \mathbf{v} \in H(\mathrm{div}) \,\middle|\, \forall K \in \mathcal{T}_h \quad \mathbf{v}_{|K}(x,y) = \left| \begin{smallmatrix} \alpha_K \\ \beta_K \end{smallmatrix} \right. + \gamma_K \left| \begin{smallmatrix} x \\ y \end{smallmatrix} \right. \right\} \tag{4.8}$$

where by writing div $\mathbf{w} = \partial w_1/\partial x + \partial w_2/\partial y$, $\mathbf{w} = (w_1, w_2)$,

$$H(\mathrm{div}) = \left\{ \mathbf{w} \in L^2(\Omega)^2 \,\middle|\, \mathrm{div}\ \mathbf{w} \in L^2(\Omega) \right\}$$

and $\alpha_K, \beta_K, \gamma_K$ are real numbers.

P1nc piecewise linear element continuous at the middle of edge only.

If we get the finite element spaces

$$X_h = \{ v \in H^1(]0,1[^2) |\ \forall K \in \mathcal{T}_h \quad v_{|K} \in P_1 \}$$

$$X_{ph} = \{ v \in X_h |\ v(\left| \begin{smallmatrix} 0 \\ \cdot \end{smallmatrix} \right.) = v(\left| \begin{smallmatrix} 1 \\ \cdot \end{smallmatrix} \right.), v(\left| \begin{smallmatrix} \dot{0} \end{smallmatrix} \right.) = v(\left| \begin{smallmatrix} \mathrm{i} \end{smallmatrix} \right.) \}$$

$$M_h = \{ v \in H^1(]0,1[^2) |\ \forall K \in \mathcal{T}_h \quad v_{|K} \in P_2 \}$$

$$R_h = \{ \mathbf{v} \in H^1(]0,1[^2)^2 |\ \forall K \in \mathcal{T}_h \quad \mathbf{v}_{|K}(x,y) = \left| \begin{smallmatrix} \alpha_K \\ \beta_K \end{smallmatrix} \right. + \gamma_K \left| \begin{smallmatrix} x \\ y \end{smallmatrix} \right. \}$$

when $\mathcal{T}_h$ is a mesh 10×10 of the unit square $]0,1[^2$, we only write in `freefem++` as follows:

```
mesh Th=square(10,10);
fespace Xh(Th,P1);                                    //    scalar FE
fespace Xph(Th,P1,
        periodic=[[2,y],[4,y],[1,x],[3,x]]);//   bi-periodic FE
fespace Mh(Th,P2);                                    //    scalar FE
fespace Rh(Th,RT0);                                 //    vectorial FE
```

where `Xh,Mh,Rh` expresses finite element spaces (called FE spaces) X_h, M_h, R_h, respectively. If we want use FE-functions $u_h, v_h \in X_h$ and $p_h, q_h \in M_h$ and $U_h, V_h \in R_h$, we write in `freefem++`

```
    Xh uh,vh;
    Xph uph,vph;
    Mh ph,qh;
    Rh [Uxh,Uyh],[Vxh,Vyh];
    Xh[int] Uh(10);                 //    array of 10 function in Xh
    Rh[int] [Wxh,Wyh](10);        //    array of 10 functions in Rh.
```

The functions U_h, V_h have two components so we have

$$U_h = \left| \begin{smallmatrix} Uxh \\ Uyh \end{smallmatrix} \right. \quad \text{and} \quad V_h = \left| \begin{smallmatrix} Vxh \\ Vyh \end{smallmatrix} \right.$$

4.1 Lagrange finite element

4.1.1 P0-element

For each triangule T_k, the basis function ϕ_k in `Vh(Th,P0)` is given by

$$\phi_k(x,y) = 1 \text{ if } (x,y) \in T_k, \qquad \phi_k(x,y) = 0 \text{ if } (x,y) \not\in T_k$$

If we write

```
  Vh(Th,P0);  Vh fh=f(x.y);
```

then for vertices q^{k_i}, $i = 1,2,3$ in Fig. 4.1(a),

$$\texttt{fh} = f_h(x,y) = \sum_{k=1}^{n_t} \frac{f(q^{k_1}) + f(q^{k_2}) + f(q^{k_3})}{3} \phi_k$$

See Fig. 4.3 for the projection of $f(x,y) = \sin(\pi x)\cos(\pi y)$ into `Vh(Th,P0)` when the mesh `Th` with 4×4-grid of $[-1,1]^2$ as in Fig. 4.2.

4.1.2 P1-element

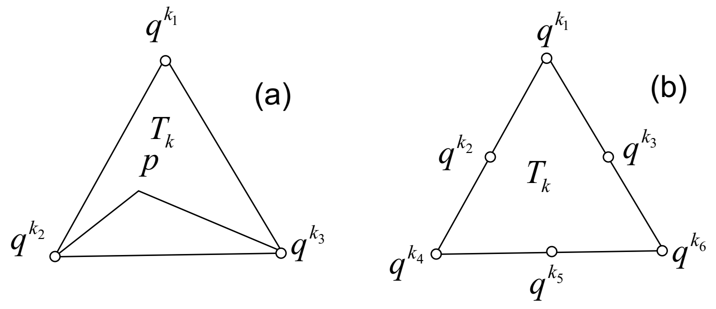


Figure 4.1: P_1 and P_2 degrees of freedom on triangle T_k

For each vertex q^i, the basis function ϕ_i in `Vh(Th,P1)` is given by

$$\begin{aligned} &\phi_i(x,y) = a_i^k + b_i^k x + c_i^k y \text{ for } (x,y) \in T_k, \\ &\phi_i(q^i) = 1, \quad \phi_i(q^j) = 0 \text{ if } i \neq j \end{aligned}$$

The basis function $\phi_{k_1}(x,y)$ with the vertex q^{k_1} in Fig. 4.1(a) at point $p=(x,y)$ in triangle T_k simply coincide with the *barycentric coordinates* λ_1^k *(area coordinates)* :

$$\phi_{k_1}(x,y)=\lambda_1^k(x,y)=\frac{\text{area of triangle}(p,q^{k_2},q^{k_3})}{\text{area of triangle}(q^{k_1},q^{k_2},q^{k_3})}$$

If we write

`Vh(Th,P1); Vh fh=`$g(x.y)$`;`

then

$$\texttt{fh}=f_h(x,y)=\sum_{i=1}^{n_v}f(q^i)\phi_i(x,y)$$

See Fig. 4.4 for the projection of $f(x,y)=\sin(\pi x)\cos(\pi y)$ into `Vh(Th,P1)`.

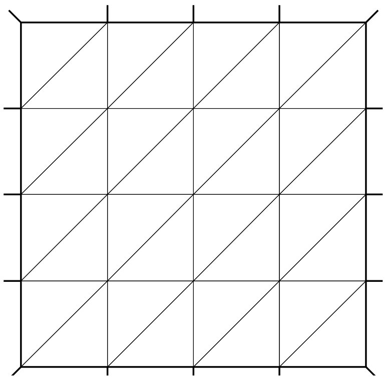


Figure 4.2: Test mesh `Th` for projection

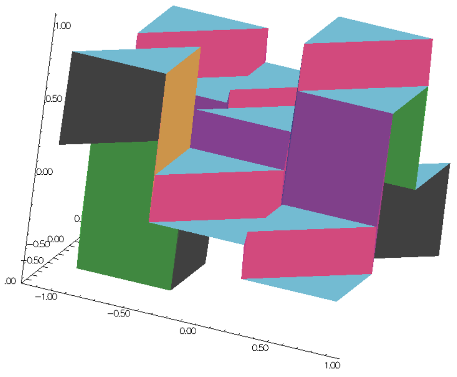


Figure 4.3: projection to `Vh(Th,P0)`

4.1.3 P2-element

For each vertex or midpoint q^i. the basis function ϕ_i in `Vh(Th,P2)` is given by

$$\phi_i(x,y)=a_i^k+b_i^kx+c_i^ky+d_i^kx^2+e_i^kxy+f_j^fy^2 \text{ for } (x,y)\in T_k,$$
$$\phi_i(q^i)=1,\quad \phi_i(q^j)=0 \text{ if } i\neq j$$

The basis function $\phi_{k_1}(x,y)$ with the vertex q^{k_1} in Fig. 4.1(b) is defined by the *barycentric coordinates*:

$$\phi_{k_1}(x,y)=\lambda_1^k(x,y)(2\lambda_1^k(x,y)-1)$$

and for the midpoint q^{k_2}

$$\phi_{k_2}(x,y)=4\lambda_1^k(x,y)\lambda_4^k(x,y)$$

If we write

`Vh(Th,P2); Vh fh=`$f(x.y)$`;`

then

$$\texttt{fh}=f_h(x,y)=\sum_{i=1}^{M}f(q^i)\phi_i(x,y)\quad\text{(summation over all vetex or midpoint)}$$

See Fig. 4.5 for the projection of $f(x,y)=\sin(\pi x)\cos(\pi y)$ into `Vh(Th,P2)`.

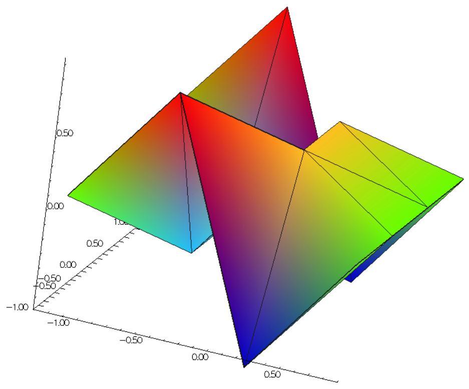


Figure 4.4: projection to `Vh(Th,P1)`

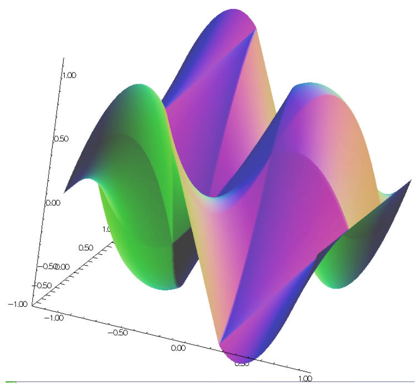


Figure 4.5: projection to `Vh(Th,P2)`

4.2 P1 Nonconforming Element

Refer [18] for detail. In Section 4.1, the approximation are a continuous function all over the domain, and

$$w_h \in V_h \subset H^1(\Omega)$$

However, we allow the continuity requirement to be relaxed. If we write

```
Vh(Th,P1nc); Vh fh=f(x.y);
```

then

$$\texttt{fh} = f_h(x,y) = \sum_{i=1}^{n_v} f(m^i)\phi_i(x,y) \quad \text{(summation over all midpoint)}$$

Here the basis function ϕ_i associat with the midpoint $m^i = (q^{k_i} + q^{k_{i+1}})/2$ where q^{k_i} is the i-th point in T_k, and we assume that $j+1 = 0$ if $j = 3$:

$$\begin{aligned} &\phi_i(x,y) = a_i^k + b_i^k x + c_i^k y \text{ for } (x,y) \in T_k, \\ &\phi_i(m^i) = 1, \quad \phi_i(m^j) = 0 \text{ if } i \neq j \end{aligned}$$

Strictly speaking $\partial\phi_i/\partial x$, $\partial\phi_i/\partial y$ contain Dirac distribution $\rho\delta_{\partial T_k}$. The numerical calculations will automatically *ignore* them. In [18], there is a proof of the estimation

$$\left(\sum_{k=1}^{n_v} \int_{T_k} |\nabla w - \nabla w_h|^2 dxdy \right)^{1/2} = O(h)$$

The basis functions ϕ_k have the following properties.

1. For the bilinear form a defined in (1.5) satisfy

$$\begin{aligned} &a(\phi_i,\phi_i) > 0, \qquad a(\phi_i,\phi_j) \leq 0 \quad \text{if } i \neq j \\ &\sum_{k=1}^{n_v} a(\phi_i,\phi_k) \geq 0 \end{aligned}$$

2. $f \geq 0 \Rightarrow u_h \geq 0$

3. If $i \neq j$, the basis function ϕ_i and ϕ_j are L^2-orthogonal:

$$\int_\Omega \phi_i \phi_j \, dxdy = 0 \qquad \text{if } i \neq j$$

which is false for P_1-element.

See Fig. 4.6 for the projection of $f(x,y) = \sin(\pi x)\cos(\pi y)$ into `Vh(Th,P1nc)`. See Fig. 4.6 for the projection of $f(x,y) = \sin(\pi x)\cos(\pi y)$ into `Vh(Th,P1nc)`.

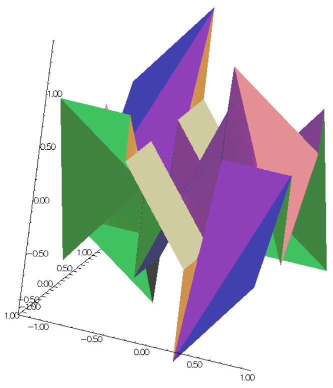


Figure 4.6: projection to `Vh(Th,P1nc)`

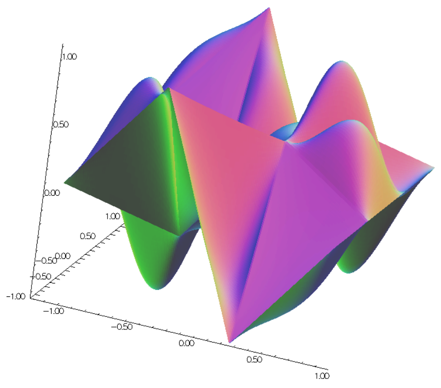


Figure 4.7: projection to `Vh(Th,P1b)`

4.3 Other FE-space

For each triangle $T_k \in \mathcal{T}_h$, let $\lambda_{k_1}(x,y)$, $\lambda_{k_2}(x,y)$, $\lambda_{k_3}(x,y)$ be the area cordinate of the triangle (see Fig. 4.1), and put

$$\beta_k(x,y) = 27\lambda_{k_1}(x,y)\lambda_{k_2}(x,y)\lambda_{k_3}(x,y) \tag{4.9}$$

called *bubble* function on T_k. The bubble function has the feature:

1. $\beta_k(x,y) = 0 \quad \text{if } (x,y) \in \partial T_k$.

2. $\beta_k(q^{k_b}) = 1$ where q^{k_b} is the barycentre $\frac{q^{k_1}+q^{k_2}+q^{k_3}}{3}$.

If we write

```
Vh(Th,P1b); Vh fh=f(x.y);
```

then

$$\mathtt{fh} = f_h(x,y) = \sum_{i=1}^{n_v} f(q^i)\phi_i(x,y) + \sum_{k=1}^{n_t} f(q^{k_b})\beta_k(x,y)$$

See Fig. 4.7 for the projection of $f(x,y) = \sin(\pi x)\cos(\pi y)$ into `Vh(Th,P1b)`.

4.4 Vector valued FE-function

Functions from $\mathbb{R}^2$ to $\mathbb{R}^N$ with $N = 1$ is called scalar function and called *vector valued* when $N > 1$. When $N = 2$

```
        fespace Vh(Th,[P0,P1]) ;
```

make the space

$$V_h = \{\mathbf{w} = (w_1, w_2) |\ w_1 \in V_h(\mathcal{T}_h, P_0),\ w_2 \in V_h(\mathcal{T}_h, P_1)\}$$

4.4.1 Raviart-Thomas element

In the Raviart-Thomas finite element $RT0_h$, the degree of freedom are the flux throw an edge e of the mesh, where the flux of the function $\mathbf{f} : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$ is $\int_e \mathbf{f}.n_e$, n_e is the unit normal of edge e.
This implies a orientation of all the edges of the mesh, for exemple we can use the global numbering of the edge vertices and we just go to small to large number.
To compute the flux, we use an quadrature formulation with one point, the middle point of the edge. Consider a triangle T_k with three vertices $(\mathbf{a}, \mathbf{b}, \mathbf{c})$. Let denote the vertices numbers by i_a, i_b, i_c, and define the three edge vectors $\mathbf{e}^1, \mathbf{e}^2, \mathbf{e}^3$ by $sgn(i_b - i_c)(\mathbf{b} - \mathbf{c})$, $sgn(i_c - i_a)(\mathbf{c} - \mathbf{a})$, $sgn(i_a - i_b)(\mathbf{a} - \mathbf{b})$,
We get three basis functions,

$$\phi_1^k = \frac{sgn(i_b - i_c)}{2|T_k|}(\mathbf{x} - \mathbf{a}), \quad \phi_2^k = \frac{sgn(i_c - i_a)}{2|T_k|}(\mathbf{x} - \mathbf{b}), \quad \phi_3^k = \frac{sgn(i_a - i_b)}{2|T_k|}(\mathbf{x} - \mathbf{c}), \quad (4.10)$$

where $|T_k|$ is the area of the triangle T_k. If we write

```
   Vh(Th,RT0); Vh [f1h,f2h]=[f1(x.y),f2(x,y)];
```

then

$$\texttt{fh} = \boldsymbol{f}_h(x,y) = \sum_{k=1}^{n_t} \sum_{l=1}^{6} n_{i_l j_l} |\mathbf{e}^{\mathbf{i_l}}| f_{j_l}(m^{i_l}) \phi_{i_l j_l}$$

where $n_{i_l j_l}$ is the j_l-th component of the normal vector $\boldsymbol{n}_{i_l}$,

$$\{m_1, m_2, m_3\} = \left\{ \frac{\mathbf{b}+\mathbf{c}}{2}, \frac{\mathbf{a}+\mathbf{c}}{2}, \frac{\mathbf{b}+\mathbf{a}}{2} \right\}$$

and $i_l = \{1, 1, 2, 2, 3, 3\}$, $j_l = \{1, 2, 1, 2, 1, 2\}$ with the order of l.

Example 26
```
mesh Th=square(2,2);
fespace Xh(Th,P1);
fespace Vh(Th,RT0);
Xh uh,vh;
Vh [Uxh,Uyh];
[Uxh,Uyh] = [sin(x),cos(y)];                          //   ok vectorial FE function
vh= x^2+y^2;                                                               //   vh
Th = square(5,5);                                               //   change the mesh
                                                                 //   Xh is unchange
uh = x^2+y^2;                                               //   compute on the new Xh
```

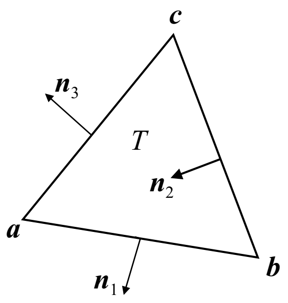


Figure 4.8: normal vectors of each edge

```
Uxh = x;                      //    error:  impossible to set only 1 component
                                                    //    of a vector FE function.
vh = Uxh;                                                                 //    ok
                                                   //    and now uh use the 5x5 mesh
                             //    but the fespace of vh is alway the 2x2 mesh
plot(uh,ps="onoldmesh.eps");                                     //    figure 4.9
uh = uh;                      //    do a interpolation of vh (old) of 5x5 mesh
                                               //    to get the new vh on 10x10 mesh.
plot(uh,ps="onnewmesh.eps");                                     //    figure 4.10
vh([x-1/2,y])= x^2 + y^2;                      //    interpole vh = ((x − 1/2)^2 + y^2)
```

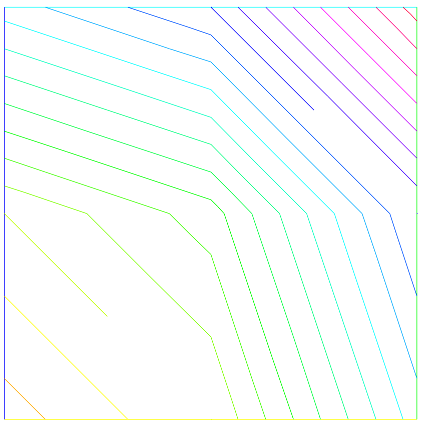


Figure 4.9: vh Iso on mesh 2×2

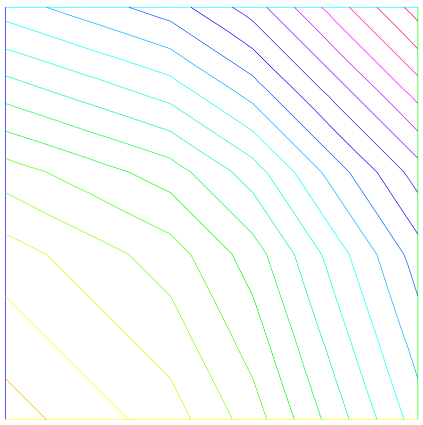


Figure 4.10: vh Iso on mesh 5×5

To get the value at a point $x = 1, y = 2$ of the FE function `uh`, or `[Uxh,Uyh]`,one writes

```
   real value;
   value = uh(2,4);                                  //    get value= uh(2,4)
   value = Uxh(2,4);                                //    get value= Uxh(2,4)
                                                      //    ------ or ------
   x=1;y=2;
   value = uh;                                       //    get value= uh(1,2)
   value = Uxh;                                     //    get value= Uxh(1,2)
```

```
value = Uyh;                                    //    get value= Uyh(1,2).
```

To get the value of the array associated to the FE function `uh`, one writes

```
real value = uh[][0] ;            //    get the value of degree of freedom 0
real maxdf = uh[].max;              //    maximum value of degree of freedom
int size = uh.n;                       //    the number of degree of freedom
real[int] array(uh.n)= uh[];         //    copy the array of the function uh
```

Warning for no scalar finite element function `[Uxh,Uyh]` the two array `Uxh[]` and `Uyh[]` are the same array, because the degre of freedom can touch more than one componant.
The other way to set a FE function is to solve a 'problem' (see below).

4.5 Problem and solve

For `freefem++` a problem must be given in variational form, so we need a bilinear form $a(u, v)$, a linear form $\ell(f, v)$, and possibly a boundary condition form must be added.

```
problem P(u,v) =
     a(u,v) - ℓ(f,v)
     + (boundary condition);
```

For example, see (1.4).

Note 11 *When you want to formulate the problem and to solve it in the same time, you can use the keywork* `solve`*,*

4.6 Parameter Description for `solve` and `problem`

The parameters are FE function real or complex, the number n of parameters is even ($n = 2 * k$), the k first function parameters are unknown, and the k last are test functions.

Note 12 *If the functions are a part of vectoriel FE then you must give all the functions of the vectorial FE in the same order (see laplaceMixte problem for example).*

Note 13 *Don't mixte complex and real parameters FE function.*

Bug: 1 *The mixing of* `fespace` *with differents periodic boundary condition is not implemented. So all the finite element space use for test or unknow functions in a problem, must have the same type of periodic boundary condition or no periodic boundary condition. No clean message is given and the result is impredictible, Sorry.*

The named parameters are:

solver= `LU`, `CG`, `Crout`,`Cholesky`,`GMRES`,`UMFPACK` ...

The default solver is `LU`. The storage mode of the matrix of the underlying linear system depends on the type of solver chosen; for `LU` the matrix is sky-line non symmetric, for `Crout` the matrix is sky-line symmetric, for `Cholesky` the matrix is sky-line symmetric positive definite, for `CG` the matrix is sparse symmetric positive, and for `GMRES` or `UMFPACK` the matrix is just sparse.

eps= a real expression. ε sets the stopping test for the iterative methods like `CG`. Note that if ε is negative then the stopping test is:

$$||Ax - b|| < |\varepsilon|$$

if it is positive then the stopping test is

$$||Ax - b|| < \frac{|\varepsilon|}{||Ax_0 - b||}$$

init= boolean expression, if it is false or 0 the matrix is reconstructed. Note that if the mesh changes the matrix is reconstructed too.

precon= name of a function (for example `P`) to set the precondioner. The prototype for the function `P` must be

```
func real[int]  P(real[int] & xx) ;
```

tgv= Huge value (10^{30}) used to lock boundary conditions (see (1.8))

4.7 Problem definition

Below `v` is the unknown function and `w` is the test function.
After the "=" sign, one may find sums of:

- a name; this is the name given to the variational form (type `varf`) for possible reuse.
- the bilinear form term: for given functions K and unknown function v, test tunctions w,

  -) `int2d(Th)( K*v*w)` $= \sum_{T\in \mathtt{Th}} \int_T K\,v\,w$

  -) `int2d(Th,1)( K*v*w)` $= \sum_{T\in \mathtt{Th}, T\subset\Omega_1} \int_T K\,v\,w$

  -) `int1d(Th,2,5)( K*v*w)` $= \sum_{T\in \mathtt{Th}} \int_{(\partial T\cup\Gamma)\cap(\Gamma_2\cup\Gamma_5)} K\,v\,w$

  -) `intalledges(Th)( K*v*w)` $= \sum_{T\in \mathtt{Th}} \int_{\partial T} K\,v\,w$

  -) `intalledges(Th,1)( K*v*w)` $= \sum_{T\in \mathtt{Th}, T\subset\Omega_1} \int_{\partial T} K\,v\,w$

  -) they become a sparse matrix of type `matrix`

- the linear form term: for given functions K, f and test functions w,

-) `int1d(Th)( K*w)` $= \sum_{T\in \mathtt{Th}} \int_T K\,w$

-) `int1d(Th,2,5)( K*w)` $= \sum_{T\in \mathtt{Th}} \int_{(\partial T\cup\Gamma)\cap(\Gamma_2\cup\Gamma_5)} K\,w$

-) `intalledges(Th)( f*w)` $= \sum_{T\in \mathtt{Th}} \int_{\partial T} f\,w$

-) a vector of type `real[int]`

- The boundary condition form term :
  - An "on" form (for Dirichlet) : `on(1, u = g )`
  - a linear form on Γ (for Neumann) `-int1d(Th))( f*w)` or `-int1d(Th,3))( f*w)`
  - a bilinear form on Γ or Γ_2 (for Robin) `int1d(Th))( K*v*w)` or `int1d(Th,2))( K*v*w)`.

If needed, the different kind of terms in the sum can appear more than once.
Remark: the integral mesh and the mesh associated to test function or unkwon function can be different in the case of linear form.

Note 14 *`N.x` and `N.y` are the normal's components.*

Important: it is not possible to write in the same integral the linear part and the bilinear part such as in `int1d(Th)( K*v*w - f*w)` .

4.8 Numerical Integration

Let D be a N-dimensional bounded domain. For an arbitrary polynomials f of degree r, if we can find particular points $\boldsymbol{\xi}_j$, $j = 1, \cdots, J$ in D and constants ω_j such that

$$\int_\Omega f(\boldsymbol{x}) = \sum_{\ell=1}^{L} c_\ell f(\boldsymbol{\xi}_\ell) \tag{4.11}$$

then we have the error estimation (see Crouzeix-Mignot (1984)), then there exists a constant $C >$ such that,

$$\left| \int_\Omega f(\boldsymbol{x}) - \sum_{\ell=1}^{L} \omega_\ell f(\boldsymbol{\xi}_\ell) \right| \le C|D|h^{r+1} \tag{4.12}$$

for any function $r+1$ times continuously differentiable f in D, where h is the diameter of D and $|D|$ its measure.
a point in the segment $[q^iq^j]$ is given as

$$\{(x,y)|\; x = (1-t)q^i_x + tq^j_x,\; y = (1-t)q^i_y + tq^j_y,\; 0 \le t \le 1\}$$

For a domain $\Omega_h = \sum_{k=1}^{n_t} T_k$, $\mathcal{T}_h = \{T_k\}$, we can calculate the integral over $\Gamma_h = \partial\Omega_h$ by

$$\begin{aligned}\int_{\Gamma_h} f(\boldsymbol{x})ds &= \texttt{int1d(Th)(f)}\\ &= \texttt{int1d(Th,qfe=*)(f)}\\ &= \texttt{int1d(Th,qforder=*)(f)}\end{aligned}$$

where * stands for the name of quadrature formulas or the order of the Gauss formula. where

L	name (`qfe=`)	order `qforder=`	point in $[q^iq^j](=t)$	ω_ℓ	degree of exact
1	qf1pE	2	0	$\lvert q^iq^j\rvert$	1
2	qf2pE	3	$(1-\sqrt{1/3})/2$	$\lvert q^iq^j\rvert/2$	3
			$(1+\sqrt{1/3})/2$	$\lvert q^iq^j\rvert/2$	
3	**qf3pE**	6	$(1-\sqrt{3/5})/2$	$(5/18)\lvert q^iq^j\rvert$	5
			$1/2$	$(8/18)\lvert q^iq^j\rvert$	
			$(1+\sqrt{3/5})/2$	$(5/18)\lvert q^iq^j\rvert$	
2	qf1pElump E	2	-1	$\lvert q^iq^j\rvert/2$	1
			$+1$	$\lvert q^iq^j\rvert/2$	

$|q^iq^j|$ is the length of segment $\overline{q^iq^j}$. For a part Γ_1 of Γ_h with the label "1", we can calculate the integral over Γ_1 by

$$\begin{aligned}\int_{\Gamma_1} f(x,y)ds &= \texttt{int1d(Th,1)(f)}\\ &= \texttt{int1d(Th,1,qfe=qf2pE)(f)}\end{aligned}$$

The integral over Γ_1, Γ_3 are given by

$$\int_{\Gamma_1\cup\Gamma_3} f(x,y)ds = \texttt{int1d(Th,1,3)(f)}$$

For each triangule $T_k = [q^{k_1}q^{k_2}q^{k_3}]$, the point $P(x,y)$ in T_k is expressed by the *area coordinate* as $P(\xi,\eta)$:

$$|T_k| = \begin{vmatrix} 1 & q_x^{k_1} & q_y^{k_1}\\ 1 & q_x^{k_2} & q_y^{k_2}\\ 1 & q_x^{k_3} & q_y^{k_3}\end{vmatrix} \quad D_1 = \begin{vmatrix} 1 & x & y\\ 1 & q_x^{k_2} & q_y^{k_2}\\ 1 & q_x^{k_3} & q_y^{k_3}\end{vmatrix} \quad D_2 = \begin{vmatrix} 1 & q_x^{k_1} & q_y^{k_1}\\ 1 & x & y\\ 1 & q_x^{k_3} & q_y^{k_3}\end{vmatrix} \quad D_3 = \begin{vmatrix} 1 & q_x^{k_1} & q_y^{k_1}\\ 1 & q_x^{k_2} & q_y^{k_2}\\ 1 & x & y\end{vmatrix}$$

$$\xi = D_1/|T_k| \qquad \eta = D_2/|T_k| \qquad \text{then } 1-\xi-\eta = D_3/|T_k|$$

For a domain $\Omega_h = \sum_{k=1}^{n_t} T_k$, $\mathcal{T}_h = \{T_k\}$, we can calculate the integral over Ω_h by

$$\begin{aligned}\int_{\Omega_h} f(x,y) &= \texttt{int2d(Th)(f)}\\ &= \texttt{int2d(Th,qft=*)(f)}\\ &= \texttt{int2d(Th,qforder=*)(f)}\end{aligned}$$

where * stands for the name of quadrature formulas or the order of the Gauss formula.

Note 15 *By default, we use the formular exact for polynomes of degrees* 5 *on triangles or edges (in bold in two tables).*

L	`qfe=`	`qforder=`	point in T_k	ω_ℓ	degree of exact
1	qf1pT	2	$\left(\frac{1}{3},\frac{1}{3}\right)$	$\|T_k\|$	1
3	qf2pT	3	$\left(\frac{1}{2},\frac{1}{2}\right)$ $\left(\frac{1}{2},0\right)$ $\left(0,\frac{1}{2}\right)$	$\|T_k\|/3$ $\|T_k\|/3$ $\|T_k\|/3$	2
7	**qf5pT**	6	$\left(\frac{1}{3},\frac{1}{3}\right)$ $\left(\frac{6-\sqrt{15}}{21},\frac{6-\sqrt{15}}{21}\right)$ $\left(\frac{6-\sqrt{15}}{21},\frac{9+2\sqrt{15}}{21}\right)$ $\left(\frac{9+2\sqrt{15}}{21},\frac{6-\sqrt{15}}{21}\right)$ $\left(\frac{6+\sqrt{15}}{21},\frac{6+\sqrt{15}}{21}\right)$ $\left(\frac{6+\sqrt{15}}{21},\frac{9-2\sqrt{15}}{21}\right)$ $\left(\frac{9-2\sqrt{15}}{21},\frac{6+\sqrt{15}}{21}\right)$	$0.225\|T_k\|$ $\frac{(155-\sqrt{15})\|T_k\|}{1200}$ $\frac{(155-\sqrt{15})\|T_k\|}{1200}$ $\frac{(155-\sqrt{15})\|T_k\|}{1200}$ $\frac{(155+\sqrt{15})\|T_k\|}{1200}$ $\frac{(155+\sqrt{15})\|T_k\|}{1200}$ $\frac{(155+\sqrt{15})\|T_k\|}{1200}$	5
3	qf1pTlump		$(0,0)$ $(1,0)$ $(0,1)$	$\|T_k\|/3$ $\|T_k\|/3$ $\|T_k\|/3$	1
9	qf2pT4P1		$\left(\frac{1}{4},\frac{3}{4}\right)$ $\left(\frac{3}{4},\frac{1}{4}\right)$ $\left(0,\frac{1}{4}\right)$ $\left(0,\frac{3}{4}\right)$ $\left(\frac{1}{4},0\right)$ $\left(\frac{3}{4},0\right)$ $\left(\frac{1}{4},\frac{1}{4}\right)$ $\left(\frac{1}{4},\frac{1}{2}\right)$ $\left(\frac{1}{2},\frac{1}{4}\right)$	$\|T_k\|/12$ $\|T_k\|/12$ $\|T_k\|/12$ $\|T_k\|/12$ $\|T_k\|/12$ $\|T_k\|/12$ $\|T_k\|/6$ $\|T_k\|/6$ $\|T_k\|/6$	1
15	qf7pT	8	see [25] for detail		7
21	qf9pT	10	see [25] for detail		9

4.9 Variational Form, Sparse Matrix, Right Hand Side Vector

It is possible to define variational forms:

```
mesh Th=square(10,10);
fespace Xh(Th,P2),Mh(Th,P1);

varf bx(u1,q) = int2d(Th)( (dx(u1)*q));
```

$$bx(u_1,q) = \int_{\Omega_h} \frac{\partial u_1}{\partial x} q$$

```
varf by(u1,q) = int2d(Th)( (dy(u1)*q));
```

$$by(u_1,q) = \int_{\Omega_h} \frac{\partial u_1}{\partial y} q$$

```
varf a(u1,u2)= int2d(Th)(  dx(u1)*dx(u2) + dy(u1)*dy(u2) )
                       +  on(1,2,4,u1=0)  +  on(3,u1=1) ;
```

Note 16 *the parameters of the variationnal form are completely formal, but is not the case in* `problem` *and* `solve` *functionality.*

$$a(u_1, v_2) = \int_{\Omega_h} \nabla u_1 . \nabla u_2; \qquad u_1 = 1 * g \text{ on } \Gamma_3, u_1 = 0 \text{ on } \Gamma_1 \cup \Gamma_2 \cup \Gamma_4$$

where f is defined later.
Later variational forms can be used to construct right hand side vectors, matrices associated to them, or to define a new problem;

```
Xh u1,u2,v1,v2;
Mh p,q,ppp;

Xh bc1; bc1[] = a(0,Xh);           //    right hand side for boundary condition
Xh b;

matrix A= a(Xh,Xh,solver=CG);                         //    the Laplace matrix
matrix<complex> CA= a(Xh,Xh,solver=CG);      //    the complex Laplace matrix

matrix Bx= bx(Xh,Mh);                        //     $Bx = (Bx_{ij})$ and $Bx_{ij} = bx(b^x_j, b^m_j)$
                                  //    where $b^x_j$ is a basis of Xh, and $b^m_j$ is a basis of Mh.
matrix By= by(Xh,Mh);                        //     $By = (By_{ij})$ and $By_{ij} = by(b^x_j, b^m_j)$
```

Note 17 *The line of the matrix corresponding to test function on the bilinear form.*

Note 18 *The vector bc1[] contains the contribution of the boundary condition $u_1 = 1$.*

Here we have three matrices A, Bx, By, and we can solve the problem:
find $u_1 \in X_h$ such that

$$a(v_1, u_1) = by(v_1, f), \forall v_1 \in X_{0h},$$

$$u_1 = g, \quad \text{on } \Gamma_1, \text{and} \quad u_1 = 0 \quad \text{on } \Gamma_1 \cup \Gamma_2 \cup \Gamma_4$$

with the following line (where $f = x$, and $g = sin(x)$)

```
Mh f=x;
Xh g=sin(x);
b[]  = Bx'*f[];                                                           //
b[] += bc1[] .*bcx[];           //    u1= g on $\Gamma_3$ boundary see following remark
u1[] = A^-1*b[];                                  //    solve the linear system
```

Note 19 *The boundary condition is implemented by penalization and the vector* `bc1[]` *contains the contribution of the boundary condition $u_1 = 1$, so to change the boundary condition, we have just to multiply the vector bc1[] by the value f of the new boundary condition term by term with the operator* `.*`. *The Section 7.6.2* `StokesUzawa.edp` *gives a real example of using all this features.*

We add automatic expression optimization by default, if this optimization trap you can remove the use of this optimization by writing for example :

```
varf a(u1,u2)= int2d(Th,optimize=false)(  dx(u1)*dx(u2) + dy(u1)*dy(u2) )
                    +  on(1,2,4,u1=0)  +  on(3,u1=1) ;
```

Remark, it is all possible to build interpolation matrix, like in the following exemple:

```
mesh  TH = square(3,4);
mesh  th = square(2,3);
mesh  Th = square(4,4);


fespace VH(TH,P1);
fespace Vh(th,P1);
fespace Wh(Th,P1);

matrix B= interpolate(VH,Vh);          //    build interpolation matrix Vh->VH
matrix BB= interpolate(Wh,Vh);         //    build interpolation matrix Vh->Wh
```

and after some operations on sparce matrices are avialable for example

```
  int N=10;
  real [int,int] A(N,N);                                    //    a full matrix
  real [int] a(N),b(N);
  A =0;
  for (int i=0;i<N;i++)
    {
      A(i,i)=1+i;
      if(i+1 < N)    A(i,i+1)=-i;
      a[i]=i;
    }
  b=A*b;
  cout << "xxxx\n";
  matrix sparseA=A;
  cout << sparseA << endl;
  sparseA = 2*sparseA+sparseA';
  sparseA = 4*sparseA+sparseA*5;                                    //
  matrix sparseB=sparseA+sparseA+sparseA; ;
  cout << "sparseB = " << sparseB(0,0) << endl;
```

4.10 Interpolation matrix

This possible to store the matrix of a linear interpolation operator from a finite element space V_h to W_h with `interpolate` function. Note by default the continuous finite function are extended. by continuity on ouside domain part.
The named parameter of function `interpolate` are:

`inside=` set to true value to set the outside extention to zero.

`t=` set to true value to get the transposed matrix

`op=` set a int value i to get

0 the default value and interpolate of the function

1 interpolate the ∂_x

2 interpolate the ∂_y

. .

A sample example `mat_interpol.edp`:

```
mesh Th=square(4,4);
mesh Th4=square(2,2,[x*0.5,y*0.5]);
plot(Th,Th4,ps="ThTh4.eps",wait=1);
fespace Vh(Th,P1);     fespace Vh4(Th4,P1);
fespace Wh(Th,P0);     fespace Wh4(Th4,P0);

matrix IV= interpolate(Vh,Vh4);                          //    here the function is
                                                        //    exended by continuity
cout << " IV Vh<-Vh4 " << IV << endl;

matrix IV0= interpolate(Vh,Vh4,inside=1);                //    here the fonction is
                                                              //    exended by zero
cout << " IV Vh<-Vh4 (inside=1)  " << IV0 << endl;

matrix IVt0= interpolate(Vh,Vh4,inside=1,t=1);
cout << " IV Vh<-Vh4^t (inside=1)  " << IVt0 << endl;

matrix IV4t0= interpolate(Vh4,Vh);
cout << " IV Vh4<-Vh^t  " << IV4t0 << endl;

matrix IW4= interpolate(Wh4,Wh);
cout << " IV Wh4<-Wh  " << IW4  << endl;

matrix IW4V= interpolate(Wh4,Vh);
cout << " IV Wh4<-Vh  " << IW4  << endl;
```

4.11 Finite elements connectivity

With the following expression we can get the connectivity information of a finite element space W_h

- `Wh.nt` gives the number of element of W_h
- `Wh.ndof` gives the number of degree of freedom or unknows
- `Wh.ndofK` gives the number of degree of freedom on one element
- `Wh(k,i)` gives the number of ith degree of freedom of element k.

See the following exemple:

```
mesh Th=square(5,5);
fespace Wh(Th,P2);
cout << " nb of degre of freedom           : " << Wh.ndof << endl;
cout << " nb of degre of freedom / ELEMENT : " << Wh.ndofK << endl;
 int k= 2;                                                 //    element 2
 int kdf= Wh.ndofK ;
 cout << " df of element " << k << ":" ;
 for (int i=0;i<kdf;i++)
    cout << Wh(k,i) << " ";
 cout << endl;
```

and the output is:

```
 Nb Of Nodes = 121
 Nb of DF = 121
 FESpace:Gibbs: old skyline = 5841  new skyline = 1377
 nb of degre of freedom           : 121
 nb of degre of freedom / ELEMENT : 6
 df of element 2:78 95 83 87 79 92
```

Chapter 5

Visualization

Numerical results in FEM create huge data, so it is very important to make obtained results visible. There are two ways of visualization in `freefem++` : One is default view supporting the draw of meshes, isovalue of real FE-functions and vector fields by the command **`plot`** (see Section 5.1). For documentation, `freefem++` make the plotting stored as postscript files.
Another method is to use the external tools, for example, gnuplot (see Section 5.2), medit (see Section 5.3) using the command **`system`**.

5.1 Plot

With the command plot, meshes, isovalues and vector fields can be displayed.
The parameters of the plot command can be , meshes,real FE functions , arrays of 2 real FE functions, arrays of two arrays of double, to plot respectively mesh, isovalue, vector field, or curve defined by the two arrays of double.
The named parameter are

wait= boolean expression to wait or not (by default no wait). If true we wait for a keyboard up event or mouse event, they respond to an event by the following characters

- **+** to zoom in around the mouse cursor,
- **-** to zoom out around the mouse cursor,
- **=** to restore de initial graphics state,
- **c** to decrease the vector arrow coef,
- **C** to increase the vector arrow coef,
- **r** to refresh the graphic window,
- **f** to toggle the filling between isovalues,
- **b** to toggle the black and white,
- **g** to toggle to grey or color ,
- **v** to toggle the plotting of value,
- **p** to save to a postscript file,
- **?** to show all actives keyboard char,

to redraw, otherwise we continue.

ps= string expression to save the plot on postscript file

coef= the vector arrow coef between arrow unit and domain unit.

fill= to fill between isovalues.

cmm= string expression to write in the graphic window

value= to plot the value of isoline and the value of vector arrow.

aspectratio= boolean to be sure that the aspect ratio of plot is preserved or not.

bb= array of 2 array (like `[[0.1,0.2],[0.5,0.6]]`), to set the bounding box and specify a partial view where the box defined by the two corner points [0.1,0.2] and [0.5,0.6].

nbiso= (int) sets the number of isovalues (20 by default)

nbarrow= (int) sets the number of colors of arrow values (20 by default)

viso= sets the array value of isovalues (an array real[int])

varrow= sets the array value of color arrows (an array real[int])

bw= (bool) sets or not the plot in black and white color.

grey= (bool) sets or not the plot in grey color.

For example:

```
real[int] xx(10),yy(10);
mesh Th=square(5,5);
fespace Vh(Th,P1);
Vh uh=x*x+y*y,vh=-y^2+x^2;
int i;
                                                              //    compute a cut
for (i=0;i<10;i++)
 {
   x=i/10.; y=i/10.;
   xx[i]=i;
   yy[i]=uh;                        //    value of uh at point (i/10.  , i/10.)
 }
 plot(Th,uh,[uh,vh],value=true,ps="three.eps",wait=true);      //    figure 5.1
    //    zoom on box defined by the two corner points [0.1,0.2] and [0.5,0.6]
 plot(uh,[uh,vh],bb=[[0.1,0.2],[0.5,0.6]],
        wait=true,grey=1,fill=1,value=1,ps="threeg.eps");      //    figure 5.2
 plot([xx,yy],ps="likegnu.eps",wait=true);                     //    figure 5.3
```

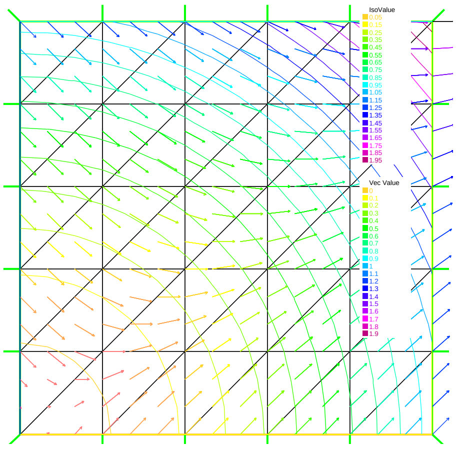


Figure 5.1: mesh, isovalue, and vector

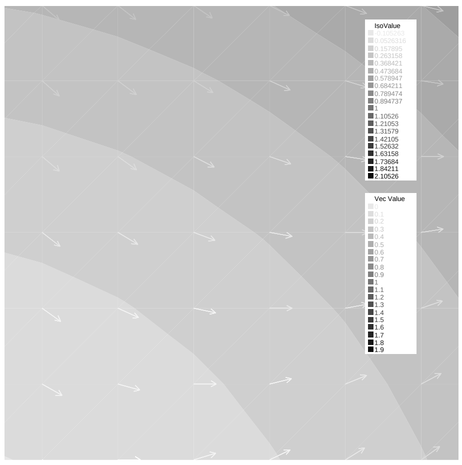


Figure 5.2: inlargement in grey of isovalue, and vector

5.2 link with gnuplot

First this work only if gnuplot[1] is installed , and only on unix computer.
You just and to the previous example:

```
                                                              //    file for gnuplot
{
  ofstream gnu("plot.gp");
  for (int i=0;i<=n;i++)
   {
     gnu <<  xx[i] << " " << yy[i] << endl;
    }
}            //    the file plot.gp is close because the variable gnu is delete

       //    to call gnuplot command and wait 5 second (tanks to unix command)
                                                       //    and make postscipt plot
exec("echo 'plot \"plot.gp\" w l \
pause 5 \
set term postscript \
set output \"gnuplot.eps\" \
replot \
quit' | gnuplot");
```

5.3 link with medit

First this work only if medit [2] software is installed.

```
                                                           //    build square ]-1,1[^2
mesh Th=square(10,10,[2*x-1,2*y-1]);
```

[1]http://www.gnuplot.info/
[2]http://www-rocq.inria.fr/gamma/medit/medit.html

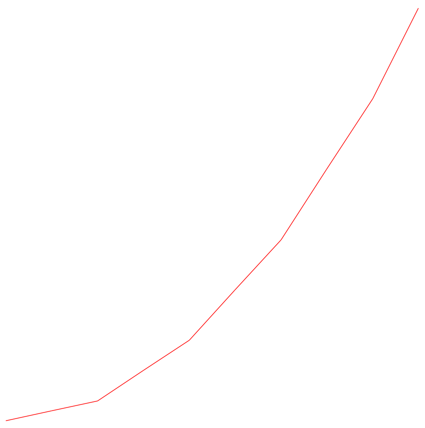


Figure 5.3: Plots a cut of uh. Note that a refinement of the same can be obtained in combination with gnuplot

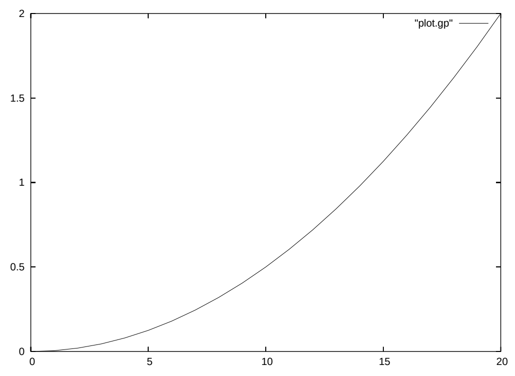


Figure 5.4: Plots a cut of uh with gnuplot

```
fespace Vh(Th,P1);
Vh u=2-x*x-y*y;

   savemesh(Th,"mm",[x,y,u*.5]);         //   save mm.points and mm.faces file
                                                              //    for medit
                                                     //    build a mm.bb file
  { ofstream file("mm.bb");
  file << "2 1 1 "<< u[].n << " 2 \n";
  int j;
  for (j=0;j<u[].n ; j++)
    file << u[][j] << endl;
    }
                                                     //    call medit command
    exec("medit mm");
                                                 //    clean files on unix OS
    exec("rm mm.bb      mm.faces   mm.points");
```

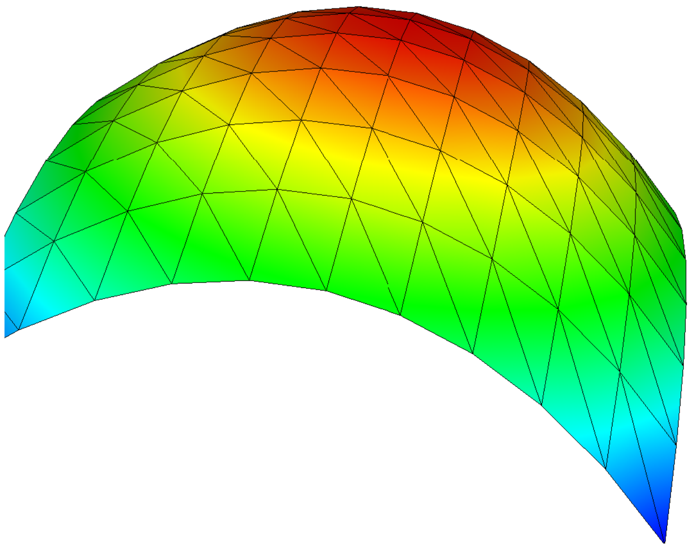


Figure 5.5: medit plot

Chapter 6

Algorithms

The associated example is fully defined in `algo.edp` file.

6.1 conjugate Gradient/GMRES

If we want to solve the Euler problem: find $x \in \mathbb{R}^n$ such that

$$\frac{\partial J}{\partial x_i}(x) = 0$$

where J is a functional (to minimize for example) from $\mathbb{R}^n$ to $\mathbb{R}$.
if the function is convex we can use the conjugate gradient to solve the problem, and we just need the function (named `dJ` for example) which compute $\frac{\partial J}{\partial x_i}$, so the two parameters are the name of the function with prototype `func real[int] dJ(real[int] & xx)` which compute $\frac{\partial J}{\partial x_i}$, a vector `x` of type `real[int]` to initialize the process and get the result.

Note 20 *You can use the* `macro` *tools (see 7.11) to build easily the differential, see example* `Newtow.edp`*.*

Three versions are available:

`LinearCG` linear case , the functional J is quadratic $J = 1/2(x, Ax) - (b, x)$ where A is a symetric positive matrix .

`LinearCMRES` linear case ,where the functional $J = 1/2(x, Ax) - (b, x)$ where A is a matrix.

`NLCG` non linear case (the functional is just convex).

The named parameter of these three functions are:

`nbiter=` set the number of iteration (by default 100)

`precon=` set the preconditionner function (`P` for example) by default it is the identity, remark the prototype is `func real[int] P(real[int] &x)`.

`eps=` set the value of the stop test ε ($= 10^{-6}$ by default) if positive then relative test $||dJ(x)||_P \leq \varepsilon * ||dJ(x_0)||_P$, otherwise the absolute test is $||dJ(x)||_P^2 \leq |\varepsilon|$.

veps= set and return the value of the stop test, if positive then relative test $||dJ(x)||_P \leq \varepsilon * ||dJ(x_0)||_P$, otherwise the absolute test is $||dJ(x)||_P^2 \leq |\varepsilon|$. The return value is minus the real stop test (remark: it is useful in loop).

Example of use:

```
  real[int] matx(10),b(10),x(10);

  func real J(real[int] & x)
    {
      real s=0;
      for (int i=0;i<x.n;i++)
        s +=(i+1)*x[i]*x[i]*0.5 - b[i]*x[i];
      return s;
    }

//    the grad of J (this is a affine version (the RHS is in )
  func real[int] dJ(real[int] &x)
    { for (int i=0;i<x.n;i++)
        matx[i]=(i+1)*x[i];
      matx -= b;                                     //    sub the right hand side
      return matx;                              //    return of global variable ok
    };
//    the grad of the bilinear part of J (the RHS in remove)
  func real[int] dJ0(real[int] &x)
    { for (int i=0;i<x.n;i++)
        matx[i]=(i+1)*x[i];
      return matx;                              //    return of global variable ok
    };
//
  func real error(real[int] & x,real[int] & b)
   {  real s=0;
     for (int i=0;i<x.n;i++)
        s += abs((i+1)*x[i] - b[i]);
   return s; }


  func real[int] matId(real[int] &x) { return x;};

  b=1; x=0;                                     //    here not rhs the CG (in dJ)
  LinearCG(dJ,x,eps=1.e-6,nbiter=20,precon=matId);
  cout << "LinearCG (Affine)  : J(x) = " << J(x) << " err=" << error(x,b) << endl;

  b=1; x=0;                                    //    here rhs the CG (in not dJ0)
  LinearCG(dJ0,x,b,eps=1.e-6,nbiter=20,precon=matId);
  cout << "LinearCG (Linear) : J(x) = " << J(x) << " err=" << error(x,b) << endl;

  b=1; x=0;                                    //    here rhs the CG (in not dJ0)
  LinearGMRES(dJ0,x,b,eps=1.e-6,nbiter=20,precon=matId);
  cout << "LinearGMRES: J(x) = " << J(x) << " err=" << error(x,b) << endl;

  b=1; x=0;                            //    set right hand side and initial gest
  NLCG(dJ,x,eps=1.e-6,nbiter=20,precon=matId);
  cout << "NLCG: J(x) = " << J(x) << " err=" << error(x,b) << endl;
```

6.2 Optimization

Two algorithms of COOOL a package [22] are interfaced with the Newton Raphson method (call `Newton`) and the `BFGS` method. Be careful these algorithms, because the implementation use full matrix.
Example of utilization of `algo.edp`

```
func real J(real[int] & x)
  {
    real s=0;
    for (int i=0;i<x.n;i++)
       s +=(i+1)*x[i]*x[i]*0.5 - b[i]*x[i];
       cout << "J ="<< s << " x =" <<  x[0] << " " << x[1] << "...\n" ;
    return s;
  }
b=1; x=2;                          //    set right hand side and initial gest
BFGS(J,dJ,x,eps=1.e-6,nbiter=20,nbiterline=20);
cout << "BFGS: J(x) = " << J(x) << " err=" << error(x,b) << endl;
```

Chapter 7

Mathematical Models

7.1 Static Problems

7.1.1 Soap Film

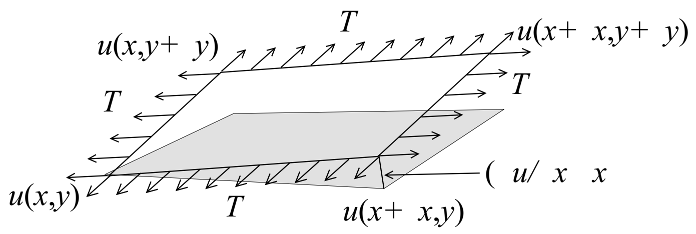
Our starting point here will be the mathematical model to find the shape of **soap film** which is glued to the ring on the $xy-$plane

$$C = \{(x,y);\ x = \cos t,\ y = \sin t,\ 0 \le t \le 2\pi\}.$$

We assume the shape of the film is descrined as the graph $(x, y, u(x, y))$ of the vertical displacement $u(x,y)\,(x^2+y^2<1)$ under a vertical pressure p in terms of force per unit area and an initial tension μ in terms of force per unit length. Consider "small plane" ABCD, A:$(x, y, u(x, y))$, B:$(x, y, u(x+\delta x, y))$, C:$(x, y, u(x+\delta x, y+\delta y))$ and D:$(x, y, u(x, y+\delta y))$. Let us denote by $\boldsymbol{n}(x,y) = (n_x(x,y), n_y(x,y), n_z(x,y))$ the normal vector of the surface $z = u(x,y)$. We see that the vertical force due to the tension μ acting along the edge AD is $-\mu n_x(x,y)\delta y$ and the the vertical force acting along the edge AD is

$$\mu n_x(x+\delta x, y)\delta y \simeq \mu\left(n_x(x,y) + \frac{\partial n_x}{\partial x}\delta x\right)(x,y)\delta y.$$

Similarly, for the edges AB and DC we have



$$-\mu n_y(x,y)\delta x, \qquad \mu\,(n_y(x,y) + \partial n_y/\partial y)\,(x,y)\delta x.$$

The force in the vertical direction on the surface ABCD due to the tension μ is given by the summension

$$\mu\,(\partial n_x/\partial x)\,\delta x\delta y + T\,(\partial n_y/\partial y)\,\delta y\delta x.$$

Assuming small displacements, we have

$$\begin{aligned}\nu_x &= (\partial u/\partial x)/\sqrt{1+(\partial u/\partial x)^2+(\partial u/\partial y)^2} \simeq \partial u/\partial x,\\ \nu_y &= (\partial u/\partial y)/\sqrt{1+(\partial u/\partial x)^2+(\partial u/\partial y)^2} \simeq \partial u/\partial y.\end{aligned}$$

Letting $\delta x \to dx$, $\delta y \to dy$, we have the equilibrium of the virtical displacement of soap film on ABCD by p

$$\mu dx dy \partial^2 u/\partial x^2 + \mu dx dy \partial^2 u/\partial y^2 + p dx dy = 0.$$

Using the Laplace operator $\Delta = \partial^2/\partial x^2 + \partial^2/\partial y^2$, we can find the virtual displacement write the following

$$-\Delta u = f \quad \text{in } \Omega \tag{7.1}$$

where $f = p/\mu$, $\Omega = \{(x,y);\ x^2+y^2<1\}$. Poisson's equation (1.1) appear also in **electrostatics** taking the form of $f = \rho/\epsilon$ where ρ is the charge density, ϵ the dielectric constant and u is named as electrostatic potential. The soap film is glued to the ring $\partial\Omega = C$, then we have the boundary condition

$$u = 0 \quad \text{on } \partial\Omega \tag{7.2}$$

If the force is gravity, for simplify, we assume that $f = -1$.

Example 27 (a_tutorial.edp)

```
border a(t=0,2*pi){ x = cos(t); y = sin(t);label=1;};

mesh disk = buildmesh(a(50));
plot(disk);
fespace femp1(disk,P1);
femp1 u,v;
func f = -1;
problem laplace(u,v) =
    int2d(disk)( dx(u)*dx(v) + dy(u)*dy(v) )          //    bilinear form
  - int2d(disk)( f*v )                                  //    linear form
  + on(1,u=0) ;                                   //    boundary condition
func ue = (x^2+y^2-1)/4;                         //    ue:  exact solution
laplace;
femp1 err = u - ue;

plot (u,ps="aTutorial.eps",value=true,wait=true);
plot(err,value=true,wait=true);

cout << "error L2=" << sqrt(int2d(disk)( err^2) )<< endl;
cout << "error H10=" << sqrt( int2d(disk)((dx(u)-x/2)^2)
                           + int2d(disk)((dy(u)-y/2)^2))<< endl;

disk = adaptmesh(disk,u,err=0.01);
plot(disk,wait=1);

laplace;

plot (u,value=true,wait=true);
err = u - ue;                       //    become FE-function on adapted mesh
plot(err,value=true,wait=true);
cout << "error L2=" << sqrt(int2d(disk)( err^2) )<< endl;
```

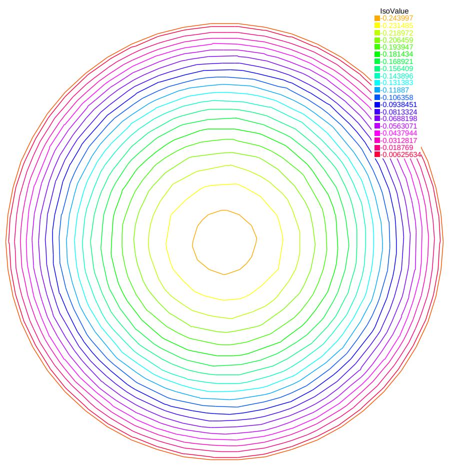


Figure 7.1: isovalue of u

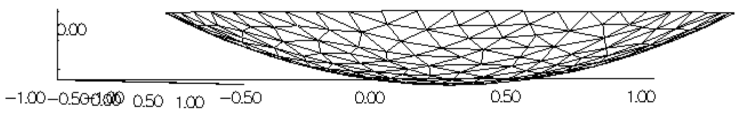


Figure 7.2: a side view of u

```
32 : cout << "error H10=" << sqrt(int2d(disk)((dx(u)-x/2)^2)
33 :                                + int2d(disk)((dy(u)-y/2)^2))<< endl;
```

In 19th line, the L^2-error estimation between the exact solution u_e,

$$\|u_h - u_e\|_{0,\Omega} = \left(\int_\Omega |u_h - u_e|^2\, dxdy\right)^{1/2}$$

and from 20th line to 21th line, the H^1-error seminorm estimation

$$|u_h - u_e|_{1,\Omega} = \left(\int_\Omega |\nabla u_h - \nabla u_e|^2\, dxdy\right)^{1/2}$$

are done on the initial mesh. The results are $\|u_h - u_e\|_{0,\Omega} = 0.000384045$, $|u_h - u_e|_{1,\Omega} = 0.0375506$.
After the adaptation, we hava $\|u_h - u_e\|_{0,\Omega} = 0.000109043$, $|u_h - u_e|_{1,\Omega} = 0.0188411$. So the numerical solution is improved by adaptation of mesh.

7.1.2 Electrostatics

We assume that there is no current and a time independent charge distribution. Then the electric field $\boldsymbol{E}$ satisfy

$$\mathrm{div}\boldsymbol{E} = \rho/\epsilon, \quad \mathrm{curl}\boldsymbol{E} = 0 \tag{7.3}$$

where ρ is the charge density and ϵ is called the permittivity of free space. From the second equation in (7.3), we can introduce the electrostatic potential such that $\boldsymbol{E} = -\nabla\phi$. Then we have Poisson equation $-\Delta\phi = f$, $f = -\rho/\epsilon$. We now obtain the equipotential line which is the level curve of ϕ, when there are no charges except conductors $\{C_i\}_{1,\cdots,K}$. Let us assume K conductors $C_1, \cdots, C_K$ within an enclosure C_0. Each one is held at an electrostatic potential φ_i. We assume that the enclosure $C0$ is held at potential 0. In order to know $\varphi(x)$ at any point x of the domain Ω, we must solve

$$-\Delta\varphi = 0 \quad \text{in } \Omega, \tag{7.4}$$

where Ω is the interior of C_0 minus the conductors C_i, and Γ is the boundary of Ω, that is $\sum_{i=0}^N C_i$. Here g is any function of x equal to φ_i on C_i and to 0 on C_0. The second equation is a reduced form for:

$$\varphi = \varphi_i \text{ on } C_i,\ i = 1...N, \varphi = 0 \text{ on } C_0. \tag{7.5}$$

Example 28 *First we give the geometical informations;* $C_0 = \{(x,y);\ x^2+y^2 = 5^2\}$, $C_1 = \{(x,y) : \frac{1}{0.3^2}(x-2)^2 + \frac{1}{3^2}y^2 = 1\}$, $C_2 = \{(x,y) : \frac{1}{0.3^2}(x+2)^2 + \frac{1}{3^2}y^2 = 1\}$. *Let* Ω *be the disk enclosed by* C_0 *with the elliptical holes enclosed by* C_1 *and* C_2*. Note that* C_0 *is described counterclockwise, whereas the elliptical holes are described clockwise, because the boundary must be oriented so that the computational domain is to its left.*

```
                              //    a circle with center at (0 ,0) and radius 5
border C0(t=0,2*pi) { x = 5 * cos(t); y = 5 * sin(t); }
border C1(t=0,2*pi) { x = 2+0.3 * cos(t); y = 3*sin(t); }
border C2(t=0,2*pi) { x = -2+0.3 * cos(t); y = 3*sin(t); }

mesh Th = buildmesh(C0(60)+C1(-50)+C2(-50));
plot(Th,ps="electroMesh");                                  //    figure 7.3
fespace Vh(Th,P1);                                         //    P1 FE-space
Vh uh,vh;                                     //    unkown and test function.
problem Electro(uh,vh) =                       //    definion of the problem
    int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) )                //    bilinear
    + on(C0,uh=0)                              //    boundary condition on C0
    + on(C1,uh=1)                                         //    +1 volt on C1
    + on(C2,uh=-1) ;                                      //    -1 volt on C2

Electro;                //    solve the problem, see figure 7.4 for the solution
plot(uh,ps="electro.eps",wait=true);                        //    figure 7.4
```

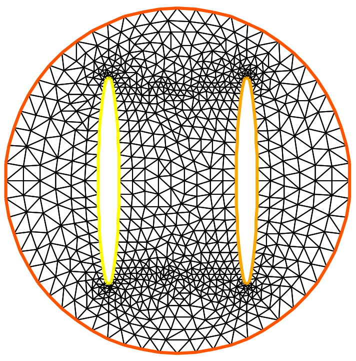


Figure 7.3: Disk with two elliptical holes

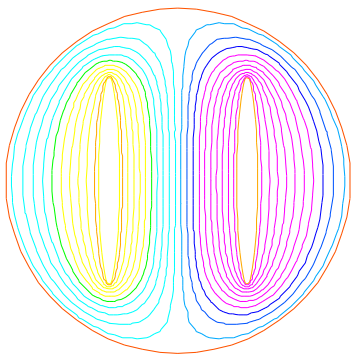


Figure 7.4: Equipotential lines, where C_1 is located in right hand side

7.1.3 Aerodynamics

Let us consider a wing profile S in a uniform flow. Infinity will be represented by a large circle Γ_∞. As previously, we must solve

$$\Delta\varphi = 0 \quad \text{in } \Omega, \quad \varphi|_S = c, \quad \varphi|_{\Gamma_\infty} = u_{\infty 1x} - u_{\infty 2x} \tag{7.6}$$

where Ω is the area occupied by the fluid, u_∞ is the air speed at infinity, c is a constant to be determined so that $\partial_n\varphi$ is continuous at the trailing edge P of S (so-called Kutta-Jukowski condition). Lift is proportional to c. To find c we use a superposition method. As all equations in (7.6) are linear, the solution φ_c is a linear function of c

$$\varphi_c = \varphi_0 + c\varphi_1, \tag{7.7}$$

where φ_0 is a solution of (7.6) with $c = 0$ and φ_1 is a solution with $c = 1$ and zero speed at infinity. With these two fields computed, we shall determine c by requiring the continuity of $\partial\varphi/\partial n$ at the trailing edge. An equation for the upper surface of a NACA0012 (this is a classical wing profile in aerodynamics; the rear of the wing is called the trailing edge) is:

$$y := 0.17735\sqrt{x} - 0.075597x - 0.212836x^2 + 0.17363x^3 - 0.06254x^4. \tag{7.8}$$

Taking an incidence angle α such that $\tan\alpha = 0.1$, we must solve

$$-\Delta\varphi = 0 \qquad \text{in } \Omega, \qquad \varphi|_{\Gamma_1} = y - 0.1x, \quad \varphi|_{\Gamma_2} = c, \tag{7.9}$$

where Γ_2 is the wing profile and Γ_1 is an approximation of infinity. One finds c by solving:

$$-\Delta\varphi_0 = 0 \ \text{ in } \Omega, \qquad \varphi_0|_{\Gamma_1} = y - 0.1x, \quad \varphi_0|_{\Gamma_2} = 0, \tag{7.10}$$

$$-\Delta\varphi_1 = 0 \ \text{ in } \Omega, \qquad \varphi_1|_{\Gamma_1} = 0, \quad \varphi_1|_{\Gamma_2} = 1. \tag{7.11}$$

The solution $\varphi = \varphi_0 + c\varphi_1$ allows us to find c by writing that $\partial_n\varphi$ has no jump at the trailing edge $P = (1,0)$. We have $\partial n\varphi - (\varphi(P^+) - \varphi(P))/\delta$ where P^+ is the point just above P in the direction normal to the profile at a distance δ. Thus the jump of $\partial_n\varphi$ is $(\varphi_0|_{P^+} + c(\varphi_1|_{P^+} - 1)) + (\varphi_0|_{P^-} + c(\varphi_1|_{P^-} - 1))$ divided by δ because the normal changes sign between the lower and upper surfaces. Thus

$$c = -\frac{\varphi_0|_{P^+} + \varphi_0|_{P^-}}{(\varphi_1|_{P^+} + \varphi_1|_{P^-} - 2)}, \tag{7.12}$$

which can be programmed as:

$$c = -\frac{\varphi_0(0.99, 0.01) + \varphi_0(0.99, -0.01)}{(\varphi_1(0.99, 0.01) + \varphi_1(0.99, -0.01) - 2)}. \tag{7.13}$$

Example 29
```
//    Computation of the potential flow around a NACA0012 airfoil.
//    The method of decomposition is used to apply the Joukowski condition
//    The solution is seeked in the form psi0 + beta psi1 and beta is
//    adjusted so that the pressure is continuous at the trailing edge

border a(t=0,2*pi) { x=5*cos(t);  y=5*sin(t); };    //    approximates infinity
border upper(t=0,1) { x = t;
     y = 0.17735*sqrt(t)-0.075597*t
  - 0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4); }
border lower(t=1,0) { x = t;
     y= -(0.17735*sqrt(t)-0.075597*t
  -0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4)); }
border c(t=0,2*pi) { x=0.8*cos(t)+0.5;  y=0.8*sin(t); }

wait = true;
mesh  Zoom = buildmesh(c(30)+upper(35)+lower(35));
mesh Th = buildmesh(a(30)+upper(35)+lower(35));
fespace Vh(Th,P2);                                     //    P1 FE space
Vh psi0,psi1,vh;                                 //    unkown and test function.
fespace ZVh(Zoom,P2);

solve Joukowski0(psi0,vh) =                       //    definion of the problem
    int2d(Th)( dx(psi0)*dx(vh) + dy(psi0)*dy(vh) )          //    bilinear form
  + on(a,psi0=y-0.1*x)                            //    boundary condition form
  + on(upper,lower,psi0=0);
plot(psi0);

solve Joukowski1(psi1,vh) =                       //    definion of the problem
    int2d(Th)( dx(psi1)*dx(vh) + dy(psi1)*dy(vh) )          //    bilinear form
  + on(a,psi1=0)                                  //    boundary condition form
  + on(upper,lower,psi1=1);

plot(psi1);

                                      //    continuity of pressure at trailing edge
real beta = psi0(0.99,0.01)+psi0(0.99,-0.01);
beta = -beta / (psi1(0.99,0.01)+ psi1(0.99,-0.01)-2);


Vh psi = beta*psi1+psi0;
plot(psi);
ZVh Zpsi=psi;
plot(Zpsi,bw=true);
Vh cp = -dx(psi)^2 - dy(psi)^2;
plot(cp);
ZVh Zcp=cp;
plot(Zcp,nbiso=40);
```

7.1.4 Error estimation

There are famous estimation between the numerical result u_h and the exact solution u of the problem 1.1 and 1.2: If triangulations $\{\mathcal{T}_h\}_{h\downarrow 0}$ is regular (see Section 3.4), then we have the estimates

$$|\nabla u - \nabla u_h|_{0,\Omega} \leq C_1 h \qquad (7.14)$$

$$\|u - u_h\|_{0,\Omega} \leq C_2 h^2 \qquad (7.15)$$

with constants C_1, C_2 independent of h, if u is in $H^2(\Omega)$. It is known that $u \in H^2(\Omega)$ if Ω is convex.

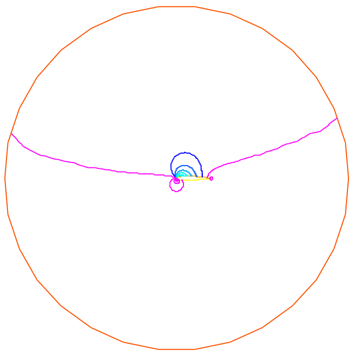


Figure 7.5: isovalue of $cp = -(\partial_x\psi)^2 - (\partial_y\psi)^2$

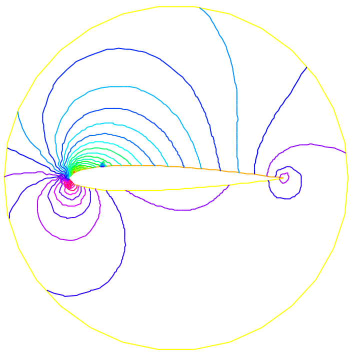


Figure 7.6: Zooming of cp

In this section we check (7.14) and (7.15). We will pick up numericall error if we use the numerical derivative, so we will use the following for (7.14).

$$\begin{aligned}\int_\Omega |\nabla u - \nabla u_h|^2\, dxdy &= \int_\Omega \nabla u\cdot\nabla(u-2u_h)\, dxdy + \int_\Omega \nabla u_h\cdot\nabla u_h\, dxdy\\ &= \int_\Omega f(u-2u_h)\, dxdy + \int_\Omega f u_h\, dxdy\end{aligned}$$

The constants C_1, C_2 are depend on $\mathcal{T}_h$ and f, so we will find them by `freefem++` . In general, we cannot get the solution u as a elementary functions (see Section 2.6) even if spetical functions are added. Instead of the exact solution, here we use the approximate solution u_0 in $V_h(\mathcal{T}_h, P_2)$, $h \sim 0$.

Example 30

```
mesh Th0 = square(100,100);
fespace V0h(Th0,P2);
V0h u0,v0;
func f = x*y;                                  //    sin(pi*x)*cos(pi*y);

solve Poisson0(u0,v0) =
    int2d(Th0)( dx(u0)*dx(v0) + dy(u0)*dy(v0) )       //    bilinear form
  - int2d(Th0)( f*v0 )                                  //    linear form
  + on(1,2,3,4,u0=0) ;                             //    boundary condition

plot(u0);

real[int] errL2(10), errH1(10);

for (int i=1; i<=10; i++) {
   mesh Th = square(5+i*3,5+i*3);
   fespace Vh(Th,P1);
   fespace Ph(Th,P0);
   Ph h = hTriangle;                        //    get the size of all triangles
   Vh u,v;
   solve Poisson(u,v) =
         int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v) )          //    bilinear form
         - int2d(Th)( f*v )                                //    linear form
         + on(1,2,3,4,u=0) ;                          //    boundary condition
    V0h uu = u;
    errL2[i-1] = sqrt( int2d(Th0)((uu - u0)^2) )/h[].max^2;
    errH1[i-1] = sqrt( int2d(Th0)( f*(u0-2*uu+uu) ) )/h[].max;
}
cout << "C1 = " << errL2.max <<"("<<errL2.min<<")"<< endl;
cout << "C2 = " << errH1.max <<"("<<errH1.min<<")"<< endl;
```

We can guess that $C_1 = 0.0179253(0.0173266)$ and $C_2 = 0.0729566(0.0707543)$, where the numbers inside the parentheses are minimum in calculation.

7.1.5 Periodic

We now solve the Poisson equation

$$-\Delta u = sin(x + \pi/4.) * cos(y + \pi/4.)$$

on the square $]0, 2\pi[^2$ under bi-periodic boundary condition $u(0, y) = u(2\pi, y)$ for all y and $u(x, 0) = u(x, 2\pi)$ for all x. These boundary conditions are achieved from the definition of the periodic finite element space.

Example 31 (periodic.edp)

```
mesh Th=square(10,10,[2*x*pi,2*y*pi]);
                                           //    defined the fespacewith periodic condition
       //    label :  2 and 4 are left and right side with y the curve abcissa
              //    1 and 2 are bottom and upper side with x the curve abcissa
fespace Vh(Th,P2,periodic=[[2,y],[4,y],[1,x],[3,x]]);
 Vh uh,vh;                                                 //    unkown and test function.
 func f=sin(x+pi/4.)*cos(y+pi/4.);                      //    right hand side function

 problem laplace(uh,vh) =                                   //    definion of the problem
    int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) )                      //    bilinear form
  + int2d(Th)( -f*vh )                                                //    linear form
;

  laplace;                    //    solve the problem plot(uh); // to see the result
  plot(uh,ps="period.eps",value=true);
```

The periodic condition does not necessarily require parallel to the axis. Example 32 give such example.

Example 32 (periodic4.edp)

```
real r=0.25;
                                                              //    a diamond with a hole
border a(t=0,1){x=-t+1; y=t;label=1;};
border b(t=0,1){ x=-t; y=1-t;label=2;};
border c(t=0,1){ x=t-1; y=-t;label=3;};
border d(t=0,1){ x=t; y=-1+t;label=4;};
```

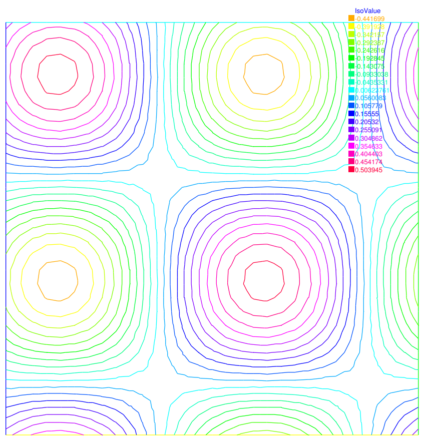


Figure 7.7: The isovalue of solution u with periodic boundary condition

```
border e(t=0,2*pi){ x=r*cos(t); y=-r*sin(t);label=0;};
int n = 10;
mesh Th= buildmesh(a(n)+b(n)+c(n)+d(n)+e(n));
plot(Th,wait=1);
real r2=1.732;
func abs=sqrt(x^2+y^2);
//    warning for periodic condition:
//    side a and c
//    on side a (label 1) $x \in [0,1]$ or $x-y \in [-1,1]$
//    on side c (label 3) $x \in [-1,0]$ or $x-y \in [-1,1]$
                     //    so the common abcissa can be repectively $x$ and $x+1$
                      //    or you can can try curviline abcissa $x-y$ and $x-y$
//    1 first way
//    fespace Vh(Th,P2,periodic=[[2,1+x],[4,x],[1,x],[3,1+x]]);
//    2 second way
 fespace Vh(Th,P2,periodic=[[2,x+y],[4,x+y],[1,x-y],[3,x-y]]);

 Vh uh,vh;

 func f=(y+x+1)*(y+x-1)*(y-x+1)*(y-x-1);
 real intf = int2d(Th)(f);
 real mTh = int2d(Th)(1);
 real k =  intf/mTh;
 cout << k << endl;
 problem laplace(uh,vh) =
    int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) ) + int2d(Th)( (k-f)*vh ) ;
 laplace;
 plot(uh,wait=1,ps="perio4.eps");
```

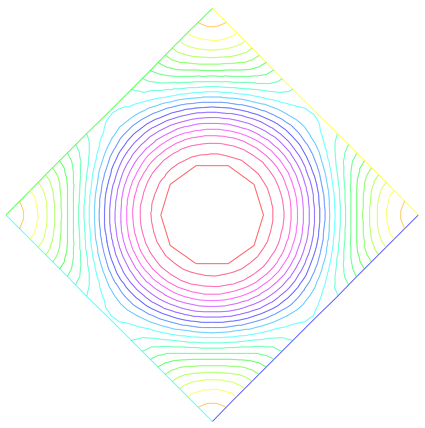


Figure 7.8: The isovalue of solution u for $\Delta u = ((y+x)^2+1)((y-x)^2+1) - k$, in Ω and $\partial_n u = 0$ on hole,and with two periodic boundary condition on external border

7.1.6 Poisson with mixed boundary condition

Here we consider the Poisson equation with mixed boundary value problems: For given functions f and g, find u such that

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= g \quad \text{on } \Gamma_D, \quad \partial u/\partial n = 0 \quad \text{on } \Gamma_N \end{aligned} \tag{7.16}$$

where Γ_D is a part of the boundary Γ and $\Gamma_N = \Gamma \setminus \overline{\Gamma_D}$. The solution u has the singularity at the points $\{\gamma_1, \gamma_2\} = \overline{\Gamma_D} \cap \overline{\Gamma_N}$. When $\Omega = \{(x,y);\ -1 < x < 1, 0 < y < 1\}$, $\Gamma_N = \{(x,y);\ -1 \le x < 0, y = 0\}$, $\Gamma_D = \partial\Omega \setminus \Gamma_N$, the singularity will appear at $\gamma_1 = (0,0)$, $\gamma_2(-1,0)$, and u has the expression

$$u = K_i u_S + u_R,\ u_R \in H^2(\text{near } \gamma_i),\ i = 1,2$$

with a constants K_i. Here $u_S = r_j^{1/2}\sin(\theta_j/2)$ by the local polar coordinate $(r_j, \theta_j$ at γ_j such that $(r_1, \theta_1) = (r, \theta)$. Instead of poler coordinate system (r, θ), we use that $r =$ `sqrt( x^2+y^2 )` and $\theta =$ `atan2(y,x)` in `freefem++` .

Example 33 *Assume that* $f = -2\times 30(x^2+y^2)$ *and* $g = u_e = 10(x^2+y^2)^{1/4}\sin\left([\tan^{-1}(y/x)]/2\right) + 30(x^2y^2)$, *where* u_eS *is the exact solution.*

```
border N(t=0,1) { x=-1+t; y=0; label=1; };
border D1(t=0,1){ x=t;  y=0; label=2;};
border D2(t=0,1){ x=1; y=t; label=2; };
border D3(t=0,2){ x=1-t; y=1; label=2;};
border D4(t=0,1) { x=-1; y=1-t; label=2; };

mesh T0h = buildmesh(N(10)+D1(10)+D2(10)+D3(20)+D4(10));
plot(T0h,wait=true);
fespace V0h(T0h,P1);
V0h u0, v0;

```

```
12 : func f=-2*30*(x^2+y^2);                                   //    given function
13 :               //    the singular term of the solution is K*us (K: constant)
14 : func us = sin(atan2(y,x)/2)*sqrt( sqrt(x^2+y^2) );
15 : real K=10.;
16 : func ue = K*us + 30*(x^2*y^2);
17 :
18 : solve Poisson0(u0,v0) =
19 :     int2d(T0h)( dx(u0)*dx(v0) + dy(u0)*dy(v0) )              //    bilinear form
20 :   - int2d(T0h)( f*v0 )                                         //    linear form
21 :   + on(2,u0=ue) ;                                      //    boundary condition
22 :
23 :                                         //    adaptation by the singular term
24 : mesh Th = adaptmesh(T0h,us);
25 : for (int i=0;i< 5;i++)
26 : {
27 :   mesh Th=adaptmesh(Th,us);
28 : } ;
29 :
30 : fespace Vh(Th, P1);
31 : Vh u, v;
32 : solve Poisson(u,v) =
33 :     int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v) )                   //    bilinear form
34 :   - int2d(Th)( f*v )                                           //    linear form
35 :   + on(2,u=ue) ;                                       //    boundary condition
36 :
37 : /* plot the solution */
38 : plot(Th,ps="adaptDNmix.ps");
39 : plot(u,wait=true);
40 :
41 : Vh uue = ue;
42 : real  H1e = sqrt( int2d(Th)( dx(uue)^2 + dy(uue)^2 + uue^2 ) );
43 :
44 : /* calculate the H1 Sobolev norm */
45 : Vh err0 = u0 - ue;
46 : Vh  err = u - ue;
47 : Vh  H1err0 = int2d(Th)( dx(err0)^2+dy(err0)^2+err0^2 );
48 : Vh  H1err = int2d(Th)( dx(err)^2+dy(err)^2+err^2 );
49 : cout <<"Relative error in first mesh "<< int2d(Th)(H1err0)/H1e<<endl;
50 : cout <<"Relative error in adaptive mesh "<< int2d(Th)(H1err)/H1e<<endl;
```

From 24th line to 28th, adaptation of meshes are done using the base of singular term. In 42th line, `H1e`$=\|u_e\|_{1,\Omega}$ *is calculated. In last 2 lines, the relative errors are calculated, that is,*

$$\begin{aligned}\|u_h^0-u_e\|_{1,\Omega}/\texttt{H1e} &= 0.120421\\ \|u_h^a-u_e\|_{1,\Omega}/\texttt{H1e} &= 0.0150581\end{aligned}$$

where u_h^0 *is the numerical solution in* `T0h` *and* u_h^a *is* `u` *in this program.*

7.1.7 Adaptation with residual error indicator

We do metric mesh adaption and compute the classical residual error indicator η_T on the element T for the Poisson problem.

Example 34 (adaptindicatorP2.edp) *First, we solve the same problem as in a previous example.*

```
border ba(t=0,1.0){x=t;   y=0;  label=1;};                //    see Fig,3.13
border bb(t=0,0.5){x=1;   y=t;  label=2;};
border bc(t=0,0.5){x=1-t; y=0.5;label=3;};
border bd(t=0.5,1){x=0.5; y=t;  label=4;};
border be(t=0.5,1){x=1-t; y=1;  label=5;};
border bf(t=0.0,1){x=0;   y=1-t;label=6;};
mesh Th = buildmesh (ba(6) + bb(4) + bc(4) +bd(4) + be(4) + bf(6));
savemesh(Th,"th.msh");
fespace Vh(Th,P2);
fespace Nh(Th,P0);
Vh u,v;
Nh rho;
real[int] viso(21);
for (int i=0;i<viso.n;i++)
  viso[i]=10.^(+(i-16.)/2.);
real error=0.01;
func f=(x-y);
problem Probem1(u,v,solver=CG,eps=1.0e-6) =
    int2d(Th,qforder=5)( u*v*1.0e-10+  dx(u)*dx(v) + dy(u)*dy(v))
  + int2d(Th,qforder=5)( -f*v);
/*************
```

Now, the local error indicator η_T is:

$$\eta_T = \left(h_T^2 || f + \Delta u_h ||^2_{L^2(T)} + \sum_{e \in \mathcal{E}_K} h_e \, || \, [\frac{\partial u_h}{\partial n_k}] \, ||^2_{L^2(e)} \right)^{\frac{1}{2}}$$

where h_T is the longest's edge of T, $\mathcal{E}_K$ is the set of T edge not on $\Gamma = \partial\Omega$, n_T is the outside unit normal to K, h_e is the length of edge e, $[g]$ is the jump of the function g across edge (left value minus rigth value).
Of coarse, we can use a variational form to compute η_T^2, with test function constant function in each triangle.

```
*************/

varf indicator2(uu,chiK) =
     intalledges(Th)(chiK*lenEdge*square(jump(N.x*dx(u)+N.y*dy(u))))
    +int2d(Th)(chiK*square(hTriangle*(f+dxx(u)+dyy(u))) );
for (int i=0;i< 4;i++)
{
  Probem1;
   cout << u[].min << " " << u[].max << endl;
   plot(u,wait=1);
   cout << " indicator2 " << endl;

   rho[] = indicator2(0,Nh);
```

```
    rho=sqrt(rho);
    cout << "rho =   min " << rho[].min << " max=" << rho[].max << endl;
    plot(rho,fill=1,wait=1,cmm="indicator density ",ps="rhoP2.eps",
                               value=1,viso=viso,nbiso=viso.n);
    plot(Th,wait=1,cmm="Mesh ",ps="ThrhoP2.eps");
    Th=adaptmesh(Th,[dx(u),dy(u)],err=error,anisomax=1);
    plot(Th,wait=1);
    u=u;
    rho=rho;
   error = error/2;
 } ;
```

If the method is correct, we expect to look the graphics by an almost constant function η on your computer as in Fig. 34.

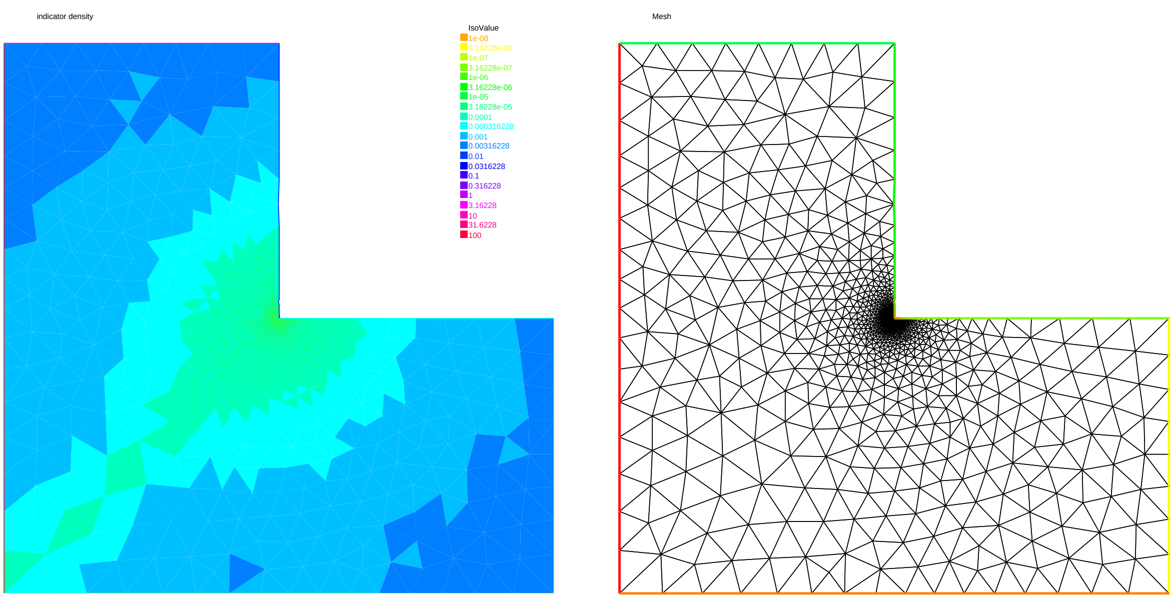


Figure 7.9: Density of the error indicator with isotropic P^2 metric

7.2 Elasticity

Consider an elastic plate with undeformed shape $\Omega\times]-h,h[$ in $\mathbb{R}^3$, $\Omega\subset\mathbb{R}^2$. By the deformation of the plate, we assume that a point $P(x_1,x_2,x_3)$ moves to $\mathcal{P}(\xi_1,\xi_2,\xi_3)$. The vector $\boldsymbol{u}=(u_1,u_2,u_3)=(\xi_1-x_1,\xi_2-x_2,\xi_3-x_3)$ is called *displacement vector*. By the deformation, the line segment $\overline{\mathbf{x},\mathbf{x}+\tau\Delta\mathbf{x}}$ moves approximately to $\overline{\mathbf{x}+u(\mathbf{x}),\mathbf{x}+\tau\Delta\mathbf{x}+u(\mathbf{x}+\tau\Delta\mathbf{x})}$ for small τ, where $\mathbf{x}=(x_1,x_2,x_3)$, $\Delta\mathbf{x}=(\Delta x_1,\Delta x_2,\Delta x_3)$. We now calculate the ratio between two segments

$$\eta(\tau)=\tau^{-1}|\Delta\mathbf{x}|^{-1}\left(|u(\mathbf{x}+\tau\Delta\mathbf{x})-u(\mathbf{x})+\tau\Delta\mathbf{x}|-\tau|\Delta\mathbf{x}|\right)$$

then we have (see e.g. [11, p.32])

$$\lim_{\tau\to 0}\eta(\tau)=(1+2e_{ij}\nu_i\nu_j)^{1/2}-1,\quad 2e_{ij}=\frac{\partial u_k}{\partial x_i}\frac{\partial u_k}{\partial x_j}+\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)$$

where $\nu_i = \Delta x_i |\Delta \mathbf{x}|^{-1}$. If the deformation is *small*, then we may consider that

$$(\partial u_k/\partial x_i)(\partial u_k/\partial x_i) \approx 0$$

and the following is called *small* strain tensor

$$\varepsilon_{ij}(u) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$$

The tensor e_{ij} is called *finite strain tensor.*
Consider the small plane $\Delta\Pi(\mathbf{x})$ centered at $\mathbf{x}$ with the unit normal direction $\boldsymbol{n} = (n_1, n_2, n_3)$, then the surface on $\Delta\Pi(\mathbf{x})$ at $\mathbf{x}$ is

$$(\sigma_{1j}(\mathbf{x})n_j, \sigma_{2j}(\mathbf{x})n_j, \sigma_{3j}(\mathbf{x})n_j)$$

where $\sigma_{ij}(\mathbf{x})$ is called *stress tensor* at $\mathbf{x}$. Hooke's law is the assumption of a linear relation between σ_{ij} and ε_{ij} such as

$$\sigma_{ij}(\mathbf{x}) = c_{ijkl}(\mathbf{x})\varepsilon_{ij}(\mathbf{x})$$

with the symmetry $c_{ijkl} = c_{jikl}, c_{ijkl} = c_{ijlk}, c_{ijkl} = c_{klij}$.
If Hooke's tensor $c_{ijkl}(\mathbf{x})$ do not depend on the choice of coordinate system, the material is called *isotropic* at $\mathbf{x}$. If c_{ijkl} is constant, the material is called *homogeneous*. In homogeneous isotropic case, there is *Lamé constants* λ, μ (see e.g. [11, p.43]) satisfying

$$\sigma_{ij} = \lambda\delta_{ij}\mathrm{div} u + 2\mu\varepsilon_{ij} \tag{7.17}$$

where δ_{ij} is Kronecker's delta. We assume that the elastic plate is fixed on $\Gamma_D\times]-h,h[$, $\Gamma_D \subset \partial\Omega$. If the body force $f = (f_1, f_2, f_3)$ is given in $\Omega\times]-h,h[$ and surface force g is given in $\Gamma_N\times]-h,h[, \Gamma_N = \partial\Omega \setminus \overline{\Gamma_D}$, then the equation of equilibrium is given as follows:

$$-\partial_j\sigma_{ij} = f_i \text{ in } \Omega\times]-h,h[, \quad i = 1,2,3 \tag{7.18}$$

$$\sigma_{ij}n_j = g_i \text{ on } \Gamma_N\times]-h,h[, \quad u_i = 0 \text{ on } \Gamma_D\times]-h,h[, \quad i = 1,2,3 \tag{7.19}$$

We now explain the plain elasticity.

Plain strain: On the end of plate, the contact condition $u_3 = 0$, $g_3 =$ is satisfied. In this case, we can suppose that $f_3 = g_3 = u_3 = 0$ and $\boldsymbol{u}(x_1, x_2, x_3) = \overline{u}(x_1, x_2)$ for all $-h < x_3 < h$.

Plain stress: The cylinder is assumed to be very thin and subjected to no load on the ends $x_3 = \pm h$, that is,

$$\sigma_{3i} = 0, \quad x_3 = \pm h, \quad i\ 1,2,3$$

The assumption leads that $\sigma_{3i} = 0$ in $\Omega\times]-h,h[$ and $\boldsymbol{u}(x_1, x_2, x_3) = \overline{u}(x_1, x_2)$ for all $-h < x_3 < h$.

Generalized plain stress: The cylinder is subjected to no load on the ends $x_3 = \pm h$. Introducing the mean values with respect to thickness,

$$\overline{u}_i(x_1, x_2) = \frac{1}{2h}\int_{-h}^{h} u(x_1, x_2, x_3)dx_3$$

and we derinde $\overline{u}_3 \equiv 0$. Similary we define the mean values $\overline{f}, \overline{g}$ of the body force and surface force as well as the mean values $\overline{\varepsilon}_{ij}$ and $\overline{\sigma}_{ij}$ of the components of stress and strain, respectively.

In what follows we omit the overlines of $\overline{u}, \overline{f}, \overline{g}, \overline{\varepsilon}_{ij}$ and $\overline{\varepsilon}_{ij}$. Then we obtain similar equation of equilibrium given in (7.18) replacing $\Omega\times]-h,h[$ with Ω and changing $i=1,2$. In the case of plane stress, $\sigma_{ij}=\lambda^*\delta_{ij}\mathrm{div}u+2\mu\varepsilon_{ij}, \lambda^*=(2\lambda\mu)/(\lambda+\mu)$.
The equations of elasticity are naturally written in variational form for the displacement vector $u(x)\in V$ as

$$\int_\Omega [2\mu\epsilon_{ij}(\boldsymbol{u})\epsilon_{ij}(\boldsymbol{v})+\lambda\epsilon_{ii}(u)\epsilon_{jj}(\boldsymbol{v})]=\int_\Omega \boldsymbol{f}\cdot\boldsymbol{v}+\int_\Gamma \boldsymbol{g}\cdot\boldsymbol{v}, \forall\boldsymbol{v}\in V$$

where V is the linear closed subspace of $H^1(\Omega)^2$.

Example 35 (Beam.edp) *Consider elastic plate with the undeformed rectangle shape* $[0,10]\times[0,2]$. *The body force is the gravity force* $\boldsymbol{f}$ *and the boundary force* $\boldsymbol{g}$ *is zero on lower and upper side. On the two vertical sides of the beam are fixed.*

```
                                                  //    a weighting beam sitting on a

int bottombeam = 2;
border a(t=2,0)  { x=0; y=t ;label=1;};                        //    left beam
border b(t=0,10) { x=t; y=0 ;label=bottombeam;};           //    bottom of beam
border c(t=0,2)  { x=10; y=t ;label=1;};                      //    rigth beam
border d(t=0,10) { x=10-t; y=2; label=3;};                      //    top beam
real E = 21.5;
real sigma = 0.29;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -0.05;
mesh th = buildmesh( b(20)+c(5)+d(20)+a(5));
fespace Vh(th,[P1,P1]);
Vh [uu,vv], [w,s];
cout << "lambda,mu,gravity ="<<lambda<< " " << mu << " " << gravity << endl;
                                      //    deformation of a beam under its own weight
solve  bb([uu,vv],[w,s])  =
    int2d(th)(
          2*mu*(dx(uu)*dx(w)+dy(vv)*dy(s)+ ((dx(vv)+dy(uu))*(dx(s)+dy(w)))/2 )
          + lambda*(dx(uu)+dy(vv))*(dx(w)+dy(s))
    )
  + int2d(th) (-gravity*s)
  + on(1,uu=0,vv=0)
 ;

plot([uu,vv],wait=1);
plot([uu,vv],wait=1,bb=[[-0.5,2.5],[2.5,-0.5]]);
mesh th1 = movemesh(th, [x+uu, y+vv]);
plot(th1,wait=1);
```

7.2.1 Fracture Mechanics

Consider the plate with the crack whose undeformed shape is a curve Σ with the two edges γ_1, γ_2. We assume the stress tensor σ_{ij} is the state of plate stress regarding $(x,y) \in \Omega_\Sigma = \Omega \setminus \Sigma$. Here Ω stands for the undeformed shape of elastic plate without crack. If the part Γ_N of the boundary $\partial\Omega$ is fixed and a load $\boldsymbol{\mathcal{L}} = (\boldsymbol{f}, \boldsymbol{g}) \in L^2(\Omega)^2 \times L^2(\Gamma_N)^2$ is given, then the displacement $\boldsymbol{u}$ is the minimizer of the potential energy functional

$$\mathcal{E}(\boldsymbol{v}; \mathcal{L}, \Omega_\Sigma) = \int_{\Omega_\Sigma} \{w(x,\boldsymbol{v}) - \boldsymbol{f}\cdot\boldsymbol{v}\} - \int_{\Gamma_N} \boldsymbol{g}\cdot\boldsymbol{v}$$

over the functional space $V(\Omega_\Sigma)$,

$$V(\Omega_\Sigma) = \left\{\boldsymbol{v} \in H^1(\Omega_\Sigma)^2;\ \boldsymbol{v} = 0 \quad \text{on } \Gamma_D = \partial\Omega \setminus \overline{\Gamma_N}\right\},$$

where $w(x,\boldsymbol{v}) = \sigma_{ij}(\boldsymbol{v})\varepsilon_{ij}(\boldsymbol{v})/2$,

$$\sigma_{ij}(\boldsymbol{v}) = C_{ijkl}(x)\varepsilon_{kl}(\boldsymbol{v}), \quad \varepsilon_{ij}(\boldsymbol{v}) = (\partial v_i/\partial x_j + \partial v_j/\partial x_i)/2, \qquad (C_{ijkl}: \quad \text{Hooke's tensor}).$$

If the elasticity is homogeneous isotropic, then the displacement $\boldsymbol{u}(x)$ is decomposed in an open neighborhood U_k of γ_k as in (see e.g. [12])

$$\boldsymbol{u}(x) = \sum_{l=1}^{2} K_l(\gamma_k) r_k^{1/2} S_{kl}^C(\theta_k) + \boldsymbol{u}_{k,R}(x) \quad \text{for } x \in \Omega_\Sigma \cap U_k,\ k = 1,2 \tag{7.20}$$

with $\boldsymbol{u}_{k,R} \in H^2(\Omega_\Sigma \cap U_k)^2$, where U_k, $k = 1,2$ are open neighborhoods of γ_k such that $\partial L_1 \cap U_1 = \gamma_1$, $\partial L_m \cap U_2 = \gamma_2$, and

$$\begin{aligned} S_{k1}^C(\theta_k) &= \frac{1}{4\mu}\frac{1}{(2\pi)^{1/2}} \begin{bmatrix} [2\kappa - 1]\cos(\theta_k/2) - \cos(3\theta_k/2) \\ -[2\kappa + 1]\sin(\theta_k/2) + \sin(3\theta_k/2) \end{bmatrix}, \\ S_{k2}^C(\theta_k) &= \frac{1}{4\mu}\frac{1}{(2\pi)^{1/2}} \begin{bmatrix} -[2\kappa - 1]\sin(\theta_k/2) + 3\sin(3\theta_k/2) \\ -[2\kappa + 1]\cos(\theta_k/2) + \cos(3\theta_k/2) \end{bmatrix}. \end{aligned} \tag{7.21}$$

where μ is the shear modulus of elasticity, $\kappa = 3 - 4\nu$ (ν is the Poisson's ratio) for plane strain and $\kappa = \frac{3-\nu}{1+\nu}$ for plane stress.
The coefficients $K_1(\gamma_i)$ and $K_2(\gamma_i)$, which are important parameters in fracture mechanics, are called stress intensity factors of the opening mode (mode I) and the sliding mode (mode II), respectively.
For simlicity, we consider the following simple crack

$$\Omega = \{(x,y):\ -1 < x < 1, -1 < y < 1\}, \qquad \Sigma = \{(x,y):\ -1 \le x \le 0, y = 0\}$$

with only one crack tip $\gamma = (0,0)$. Unfortunately, `freefem++` cannot treat crack, so we use the modification of the domain with U-shape channel (see Fig. 3.26) with $d = 0.0001$. The undeformed crack Σ is approximated by

$$\begin{aligned} \Sigma_d &= \{(x,y):\ -1 \le x \le -10 * d, -d \le y \le d\} \\ &\quad \cup\{(x,y):\ -10 * d \le x \le 0, -d + 0.1 * x \le y \le d - 0.1 * x\} \end{aligned}$$

and $\Gamma_D =$ `R` in Fig. 3.26. In this example, we use three technique:

- Fast Finite Element Interpolator from the mesh `Th` to `Zoom` for the scale-up of near γ.
- After obtaining the displacement vector $\boldsymbol{u} = (u, v)$, we shall watch the deformation of the crack near γ as follows,

```
mesh Plate = movemesh(Zoom,[x+u,y+v]);
plot(Plate);
```

- Important technique is adaptive mesh, because the large singularity occur at γ as shown in (7.20).

First example create mode I deformation by the opposed surface force on `B` and `T` in the vertical direction of Σ, and the displacement is fixed on `R`.
In a laboratory, fracture engineer use photoelasticity to make stress field visible, which shows the principal stress difference

$$\sigma_1 - \sigma_2 = \sqrt{(\sigma_{11} - \sigma_{22})^2 + 4\sigma_{12}^2} \tag{7.22}$$

where σ_1 and σ_2 are the principal stresses. In opening mode, the photoelasticity make symmetric pattern concentrated at γ.

Example 36 (Crack Opening, $K_2(\gamma) = 0$) `{CrackOpen.edp}`

```
real d = 0.0001;
int n = 5;
real cb=1, ca=1, tip=0.0;
border L1(t=0,ca-d) { x=-cb; y=-d-t; }
border L2(t=0,ca-d) { x=-cb; y=ca-t; }
border B(t=0,2) { x=cb*(t-1); y=-ca; }
border C1(t=0,1) { x=-ca*(1-t)+(tip-10*d)*t; y=d; }
border C21(t=0,1) { x=(tip-10*d)*(1-t)+tip*t; y=d*(1-t); }
border C22(t=0,1) { x=(tip-10*d)*t+tip*(1-t); y=-d*t; }
border C3(t=0,1) { x=(tip-10*d)*(1-t)-ca*t; y=-d; }
border C4(t=0,2*d) { x=-ca; y=-d+t; }
border R(t=0,2) { x=cb; y=cb*(t-1); }
border T(t=0,2) { x=cb*(1-t); y=ca; }
mesh Th = buildmesh (L1(n/2)+L2(n/2)+B(n)
                    +C1(n)+C21(3)+C22(3)+C3(n)+R(n)+T(n));
cb=0.1; ca=0.1;
plot(Th,wait=1);
mesh Zoom = buildmesh (L1(n/2)+L2(n/2)+B(n)+C1(n)
                      +C21(3)+C22(3)+C3(n)+R(n)+T(n));
plot(Zoom,wait=1);
real E = 21.5;
real sigma = 0.29;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
fespace Vh(Th,[P2,P2]);
fespace zVh(Zoom,P2);
Vh [u,v], [w,s];
solve  Problem([u,v],[w,s])  =
    int2d(Th)(
             2*mu*(dx(u)*dx(w)+ ((dx(v)+dy(u))*(dx(s)+dy(w)))/4 )
             + lambda*(dx(u)+dy(v))*(dx(w)+dy(s))/2
             )
    -int1d(Th,T)(0.1*(4-x)*s)+int1d(Th,B)(0.1*(4-x)*s)
    +on(R,u=0)+on(R,v=0);                                        //   fixed
;

zVh Sx, Sy, Sxy, N;
for (int i=1; i<=5; i++)
{
  mesh Plate = movemesh(Zoom,[x+u,y+v]);              //   deformation near γ
  Sx = lambda*(dx(u)+dy(v)) + 2*mu*dx(u);
  Sy = lambda*(dx(u)+dy(v)) + 2*mu*dy(v);
  Sxy = mu*(dy(u) + dx(v));
  N = 0.1*1*sqrt((Sx-Sy)ˆ2+4*Sxyˆ2);            //   principal stress difference
  if (i==1) {
     plot(Plate,ps="1stCOD.eps",bw=1);                         //   Fig.  7.10
     plot(N,ps="1stPhoto.eps",bw=1);                           //   Fig.  7.10
  } else if (i==5) {
     plot(Plate,ps="LastCOD.eps",bw=1);                        //   Fig.  7.11
     plot(N,ps="LastPhoto.eps",bw=1);                          //   Fig.  7.11
     break;
  }
  Th=adaptmesh(Th,[u,v]);
  Problem;
}
```

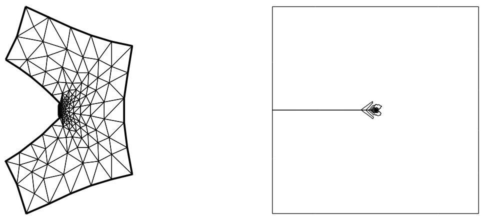


Figure 7.10: Crack open displacement (COD) and Principal stress difference in the first mesh

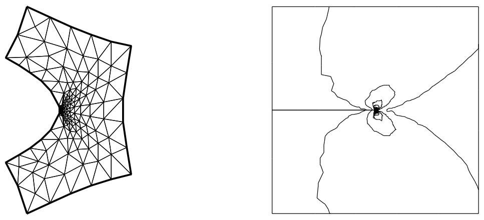


Figure 7.11: COD and Principal stress difference in the last adaptive mesh

It is difficult to create mode II deformation by the opposed shear force on `B` and `T` that is observed in a laboratory. So we use the body shear force along Σ, that is, the x-component f_1 of the body force $\boldsymbol{f}$ is given by

$$f_1(x,y) = H(y-0.001)*H(0.1-y) - H(-y-0.001)*H(y+0.1)$$

where $H(t) = 1$ if $t > 0$; $= 0$ if $t < 0$.

Example 37 (Crack Sliding, $K_2(\gamma) = 0$) `(use the same mesh Th)`

```
cb=0.01; ca=0.01;
mesh Zoom = buildmesh (L1(n/2)+L2(n/2)+B(n)+C1(n)
                          +C21(3)+C22(3)+C3(n)+R(n)+T(n));
(use same FE-space Vh and elastic modulus)
fespace Vh1(Th,P1);
```

```
Vh1 fx = ((y>0.001)*(y<0.1))-((y<-0.001)*(y>-0.1)) ;

solve  Problem([u,v],[w,s])  =
    int2d(Th)(
             2*mu*(dx(u)*dx(w)+ ((dx(v)+dy(u))*(dx(s)+dy(w)))/4 )
             + lambda*(dx(u)+dy(v))*(dx(w)+dy(s))/2
             )
    -int2d(Th)(fx*w)
    +on(R,u=0)+on(R,v=0);                                       //    fixed
;

for (int i=1; i<=3; i++)
{
  mesh Plate = movemesh(Zoom,[x+u,y+v]);                 //    deformation near γ
  Sx = lambda*(dx(u)+dy(v)) + 2*mu*dx(u);
  Sy = lambda*(dx(u)+dy(v)) + 2*mu*dy(v);
  Sxy = mu*(dy(u) + dx(v));
  N = 0.1*1*sqrt((Sx-Sy)^2+4*Sxy^2);              //   principal stress difference
  if (i==1) {
     plot(Plate,ps="1stCOD2.eps",bw=1);                      //    Fig.  7.13
     plot(N,ps="1stPhoto2.eps",bw=1);                        //    Fig.  7.12
  } else if (i==3) {
     plot(Plate,ps="LastCOD2.eps",bw=1);                     //    Fig.  7.13
     plot(N,ps="LastPhoto2.eps",bw=1);                       //    Fig.  7.13
     break;
  }
  Th=adaptmesh(Th,[u,v]);
  Problem;
}
```

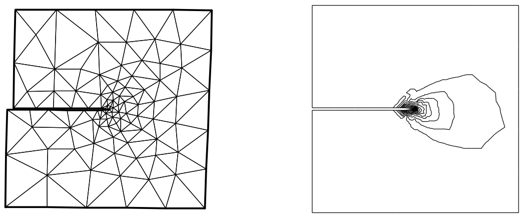


Figure 7.12: (COD) and Principal stress difference in the first mesh

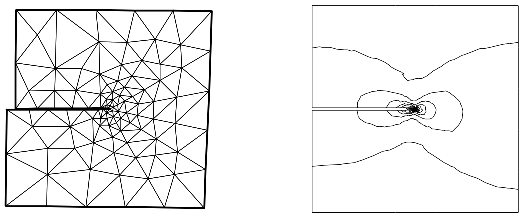


Figure 7.13: COD and Principal stress difference in the last adaptive mesh

7.3 Nonlinear Static Problems

We propose how to solve the following non-linear academic problem of minimization of a functional

$$J(u) = \int_\Omega f(|\nabla u|^2) - u * b$$

where u is function of $H^1_0(\Omega)$ and f defined by

$$f(x) = a*x + x - ln(1+x), \quad f'(x) = a + \frac{x}{1+x}, \quad f''(x) = \frac{1}{(1+x)^2}$$

7.3.1 Non linear conjugate gradient algorithm

```
mesh Th=square(10,10);                                   //   mesh definition of Ω
fespace Vh(Th,P1);                                       //   finite element space
fespace Ph(Th,P0);                                          //   make optimization
```

A small hack to construct a function

$$Cl = \begin{cases} 1 & \text{on interior degree of freedom} \\ 0 & \text{on boundary degree of freedom} \end{cases}$$

```
                                                //   Hack to construct an array :
                                    //   1 on interior nodes and 0 on boundary nodes
varf vCl(u,v) = on(1,2,3,4,u=1);
Vh Cl;
Cl[]= vCl(0,Vh,tgv=1);                                             //   0 and tgv
real tgv=Cl[].max;                                                          //
Cl[] = -Cl[];  Cl[] += tgv; Cl[] /=tgv;
```

the definition of f, f', f'' and b

```
//   $J(u) = \int_\Omega f(|\nabla u|^2) - \int \Omega ub$
//   $f(x) = a*x + x - ln(1+x), \quad f'(x) = a + \frac{x}{1+x}, \quad f''(x) = \frac{1}{(1+x)^2}$
real a=0.001;

func real f(real u) { return u*a+u-log(1+u); }
func real df(real u) { return a+u/(1+u);}
func real ddf(real u) { return 1/((1+u)*(1+u));}
Vh b=1;                                                        //   to defined b
//   the routine to compute the functional J
func real J(real[int] & x)
  {
    Vh u;u[]=x;
    real r=int2d(Th)(f( dx(u)*dx(u) + dy(u)*dy(u) ) - b*u) ;
    cout << "J(x) =" << r << " " << x.min <<  " " << x.max << endl;
    return r;
  }
```

The function to compute DJ, where u is the current solution.

```
Vh u=0;                                        //   the current value of the solution
Vh alpha;                                                   //   of store $f(|\nabla u|^2)$
int iter=0;
```

```
alpha=df( dx(u)*dx(u) + dy(u)*dy(u) );                        //    optimization

func real[int] dJ(real[int] & x)
  {
    int verb=verbosity; verbosity=0;
    Vh u;u[]=x;
    alpha=df( dx(u)*dx(u) + dy(u)*dy(u) );                    //    optimization
    varf au(uh,vh) = int2d(Th)( alpha*( dx(u)*dx(vh) + dy(u)*dy(vh) ) - b*vh);
    x= au(0,Vh);
    x = x.* Cl[];                                 //    the grad in 0 on boundary
    verbosity=verb;
    return x;                               //    warning no return of local array
  }
```

We want to construct also a preconditionner function C with solving the problem: find $u_h \in V_{0h}$ such that

$$\forall v_h \in V_{0h}, \quad \int_\Omega \alpha \nabla u_h . \nabla v_h = \int_\Omega b v_h$$

where $\alpha = f(|\nabla u|^2)$.

```
varf alap(uh,vh,solver=Cholesky,init=iter)=
   int2d(Th)( alpha *( dx(uh)*dx(vh) + dy(uh)*dy(vh) ))   + on(1,2,3,4,uh=0);

varf amass(uh,vh,solver=Cholesky,init=iter)=  int2d(Th)( uh*vh)  + on(1,2,3,4,uh=0);

matrix Amass = alap(Vh,Vh,solver=CG);                                       //

matrix Alap=  alap(Vh,Vh,solver=Cholesky,factorize=1);                      //

                                             //    the preconditionner function
func real[int] C(real[int] & x)
{
   real[int] u(x.n);
   u=Amass*x;
   x = Alap^-1*u;
   x = x .* Cl[];
   return x;                              //    no return of local array variable
}
```

A good idea to solve the problem is make 10 iteration of the conjugate gradient, recompute the preconditioning and restart the conjugate gradient:

```
   verbosity=5;
   int conv=0;
   real eps=1e-6;
   for (int i=0;i<20;i++)
   {
     conv=NLCG(dJ,u[],nbiter=10,precon=C,veps=eps);                         //
     if (conv) break;                                  //    if converge break loop

     alpha=df( dx(u)*dx(u) + dy(u)*dy(u) ); //    recompute alpha optimization
     Alap = alap(Vh,Vh,solver=Cholesky,factorize=1);
     cout << " restart with new preconditionner " << conv << " eps =" << eps <<
endl;
   }
```

```
  plot (u,wait=1,cmm="solution with NLCG");
```

Note 21 *The keycode* `veps=eps` *change the value of the current* `eps`*, this is usefule in this case, because at the first iteration the value of* `eps` *is change to − the absolute stop test and we save this initial stop test of for the all process. We remove the problem of the relative stop test in iterative procedure, because we start close to the solution and the relative stop test become very hard to reach.*

7.3.2 Newton Ralphson algorithm

Now, we solve the Euler problem $\nabla J(u) = 0$ with Newton Ralphson algorithm, that is,

$$u^{n+1} = u^n - (\nabla^2 J(u^n))^{-1} * dJ(u^n)$$

First we introduice the two variational form `vdJ` and `vhJ` to compute respectively ∇J and $\nabla^2 J$

```
//    methode of Newton Ralphson to solve dJ(u)=0;
                                                                              //
```

$$u^{n+1} = u^n - (\frac{\partial dJ}{\partial u_i})^{-1} * dJ(u^n)$$

```
//    ---------------------------------------------
  Ph dalpha ;                               //   to store = f''(|∇u|²) optimisation


  //    the variational form of evaluate dJ = ∇J
  //    --------------------------------------
  //    dJ = f'()*( dx(u)*dx(vh) + dy(u)*dy(vh)
  varf vdJ(uh,vh) =  int2d(Th)( alpha*( dx(u)*dx(vh) + dy(u)*dy(vh) ) - b*vh)
  + on(1,2,3,4, uh=0);


  //    the variational form of evaluate ddJ  = ∇²J
  //    hJ(uh,vh) = f'()*( dx(uh)*dx(vh) + dy(uh)*dy(vh)
   //    + f''()( dx(u)*dx(uh) + dy(u)*dy(uh) ) * (dx(u)*dx(vh) + dy(u)*dy(vh))

  varf vhJ(uh,vh) = int2d(Th)( alpha*( dx(uh)*dx(vh) + dy(uh)*dy(vh) )
   +  dalpha*( dx(u)*dx(vh) + dy(u)*dy(vh)  )*( dx(u)*dx(uh) + dy(u)*dy(uh) ) )
   + on(1,2,3,4, uh=0);

 //    the Newton algorithm
  Vh v,w;
  u=0;
  for (int i=0;i<100;i++)
   {
    alpha = df( dx(u)*dx(u) + dy(u)*dy(u) ) ;              //    optimization
    dalpha = ddf( dx(u)*dx(u) + dy(u)*dy(u) ) ;            //    optimization
    v[]= vdJ(0,Vh);                                            //     v = ∇J(u)
    real res= v[]'*v[];                                 //    the dot product
    cout << i <<  " residuˆ2 = " <<  res  << endl;
    if( res< 1e-12) break;
    matrix H= vhJ(Vh,Vh,factorize=1,solver=LU);                          //
    w[]=H^-1*v[];
    u[] -= w[];
  }
  plot (u,wait=1,cmm="solution with Newton Ralphson");
```

7.4 Eigenvalue Problems

This section depend on your FreeFem++ compilation process (see `README_arpack`), of this tools. This tools is available in `FreeFem++` if the word "eigenvalue" appear in line "Load:", like:

```
-- FreeFem++ v1.28 (date Thu Dec 26 10:56:34 CET 2002)
 file : LapEigenValue.edp
 Load: lg_fem lg_mesh eigenvalue
```

This tools is based on the `arpack++` [1] the object-oriented version of ARPACK eigenvalue package [1].
The function EigenValue compute the generalized eigenvalue of $Au = \lambda Bu$ where sigma $=\sigma$ is the shift of the method. The matrix OP is defined with $A - \sigma B$. The return value is the number of converged eigenvalue (can be greater than the number of eigen value nev=)

```
int k=EigenValue(OP,B,nev= , sigma= );
```

where the matrix $OP = A - \sigma B$ with a solver and boundary condition, and the matrix B.

sym= the problem is symmetric (all the eigen value are real)

nev= the number desired eigenvalues (nev) close to the shift.

value= the array to store the real part of the eigenvalues

ivalue= the array to store the imag. part of the eigenvalues

vector= the array to store the eigenvectors. For real nonsymmetric problems, complex eigenvectors are given as two consecutive vectors, so if eigenvalue k and $k+1$ are complex conjugate eigenvalues, the kth vector will contain the real part and the $k+1$th vector the imaginary part of the corresponding complex conjugate eigenvectors.

tol= the relative accuracy to which eigenvalues are to be determined;

sigma= the shift value;

maxit= the maximum number of iterations allowed;

ncv= the number of Arnoldi vectors generated at each iteration of ARPACK.

[1] `http://www.caam.rice.edu/software/ARPACK/`

Example 38 (lapEignenValue.edp) *In the first example, we compute the eigenvalue and the eigenvector of the Dirichlet problem on square $\Omega =]0,\pi[^2$.*
The problem is find: λ, and ∇u_λ in $\mathbb{R} \times H^1_0(\Omega)$

$$\int_\Omega \nabla u_\lambda \nabla v = \lambda \int_\Omega uv \quad \forall v \in H^1_0(\Omega)$$

The exact eigenvalues are $\lambda_{n,m} = (n^2 + m^2), (n,m) \in {\mathbb{N}_*}^2$ with the associated eigenvectors are $u_{m,n} = sin(nx) * sin(my)$.
We use the generalized inverse shift mode of the `arpack++` *library, to find 20 eigenvalue and eigenvector close to the shift value $\sigma = 20$.*

```
//    Computation of the eigen value and eigen vector of the
//    Dirichlet problem on square ]0,π[^2
//    ----------------------------------------
//    we use the inverse shift mode
//    the shift is given with the real sigma
//    -------------------------------------
//    find λ and u_λ ∈ H^1_0(Ω) such that:
//                              ∫_Ω ∇u_λ∇v = λ ∫_Ω u_λ v, ∀v ∈ H^1_0(Ω)
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 of the lock 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);
```

```
//    tol= the tolerace
//    maxit= the maximum iteration see arpack doc.
//    ncv see arpack doc.  http://www.caam.rice.edu/software/ARPACK/
//    the return value is number of converged eigen value.

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);
}
```

The output of this example is:

```
   Nb of edges on Mortars  = 0
   Nb of edges on Boundary = 80, neb = 80
 Nb Of Nodes = 1681
 Nb of DF = 1681
Real symmetric eigenvalue problem: A*x - B*x*lambda


Thanks to ARPACK++ class ARrcSymGenEig
Real symmetric eigenvalue problem: A*x - B*x*lambda
Shift and invert mode  sigma=20

Dimension of the system            : 1681
Number of 'requested' eigenvalues  : 20
Number of 'converged' eigenvalues  : 20
Number of Arnoldi vectors generated: 41
Number of iterations taken         : 2

Eigenvalues:
  lambda[1]: 5.0002
  lambda[2]: 8.00074
  lambda[3]: 10.0011
  lambda[4]: 10.0011
  lambda[5]: 13.002
  lambda[6]: 13.0039
  lambda[7]: 17.0046
  lambda[8]: 17.0048
  lambda[9]: 18.0083
  lambda[10]: 20.0096
  lambda[11]: 20.0096
  lambda[12]: 25.014
  lambda[13]: 25.0283
  lambda[14]: 26.0159
  lambda[15]: 26.0159
  lambda[16]: 29.0258
  lambda[17]: 29.0273
  lambda[18]: 32.0449
  lambda[19]: 34.049
  lambda[20]: 34.0492

 ---- 0 5.0002 err= -0.000225891 ---
 ---- 1 8.00074 err= -0.000787446 ---
 ---- 2 10.0011 err= -0.00134596 ---
 ---- 3 10.0011 err= -0.00134619 ---
 ---- 4 13.002 err= -0.00227747 ---
 ---- 5 13.0039 err= -0.004179 ---
 ---- 6 17.0046 err= -0.00623649 ---
 ---- 7 17.0048 err= -0.00639952 ---
 ---- 8 18.0083 err= -0.00862954 ---
 ---- 9 20.0096 err= -0.0110483 ---
 ---- 10 20.0096 err= -0.0110696 ---
 ---- 11 25.014 err= -0.0154412 ---
 ---- 12 25.0283 err= -0.0291014 ---
 ---- 13 26.0159 err= -0.0218532 ---
 ---- 14 26.0159 err= -0.0218544 ---
 ---- 15 29.0258 err= -0.0311961 ---
 ---- 16 29.0273 err= -0.0326472 ---
 ---- 17 32.0449 err= -0.0457328 ---
 ---- 18 34.049 err= -0.0530978 ---
 ---- 19 34.0492 err= -0.0536275 ---
```

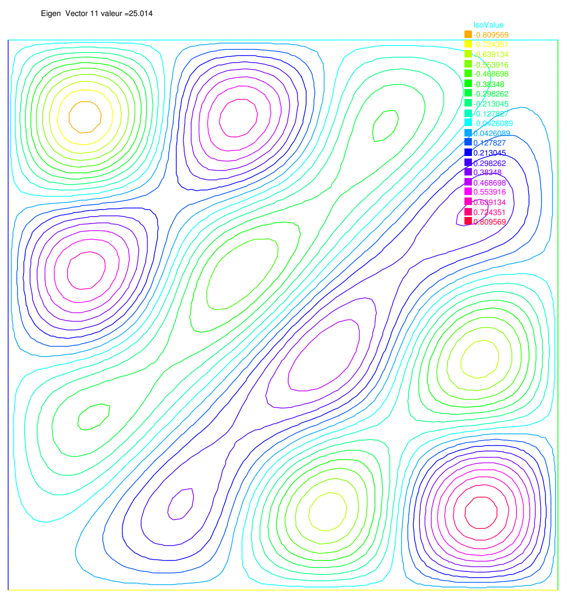


Figure 7.14: Isovalue of 11th eigenvector $u_{4,3} - u_{3,4}$

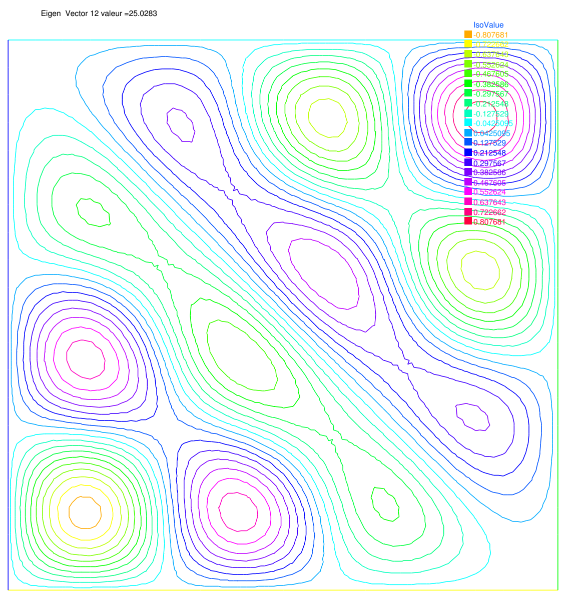


Figure 7.15: Isovalue of 12th eigenvector $u_{4,3} + u_{3,4}$

7.5 Evolution Problems

`freefem++` also solve evolution problems such as the heat problem

$$\begin{aligned}&\frac{\partial u}{\partial t} - \mu\Delta u = f \quad \text{in } \Omega\times]0,T[, &(7.23)\\ &u(\boldsymbol{x},0) = u_0(\boldsymbol{x}) \quad \text{in } \Omega; \qquad (\partial u/\partial n)(\boldsymbol{x},t) = 0 \quad \text{on } \partial\Omega\times]0,T[.\end{aligned}$$

with a positive viscosity coefficient μ and homogeneous Neumann boundary conditions. We solve (7.23) by FEM in space and finite differences in time. We use the definition of the partial derivative of the solution in the time derivative,

$$\frac{\partial u}{\partial t}(x,y,t) = \lim_{\tau\to 0}\frac{u(x,y,t)-u(x,y,t-\tau)}{\tau}$$

which indicate that $u^m(x,y) = u(x,y,m\tau)$ imply

$$\frac{\partial u}{\partial t}(x,y,m\tau) \simeq \frac{u^m(x,y)-u^{m-1}(x,y)}{\tau}$$

The time descrezation of heat equation (7.24) is as follows:

$$\begin{aligned}&\frac{u^{m+1}-u^m}{\tau} - \mu\Delta u^{m+1} = f^{m+1} \quad \text{in } \Omega &(7.24)\\ &u^0(\boldsymbol{x}) = u_0(\boldsymbol{x}) \quad \text{in } \Omega; \qquad \partial u^{m+1}/\partial n(\boldsymbol{x}) = 0 \quad \text{on } \partial\Omega, \quad \text{for all } m = 0,\cdots,[T/\tau],\end{aligned}$$

which is so-called *backward Euler method* for (7.24). Multiplying the test function v both sides of the formula just above, we have

$$\int_\Omega \{u^{m+1}v - \tau\Delta u^{m+1}v\} = \int_\Omega \{u^m + \tau f^{m+1}\}v\,.$$

By the divergence theorem, we have

$$\int_\Omega \{u^{m+1}v + \tau\nabla u^{m+1}\cdot\nabla v\} - \int_{\partial\Omega}\tau\left(\partial u^{m+1}/\partial n\right)v = \int_\Omega \{u^m v + \tau f^{m+1}v\}.$$

By the boundary condition $\partial u^{m+1}/\partial n = 0$, it follows that

$$\int_\Omega \{u^{m+1}v + \tau\nabla u^{m+1}\cdot\nabla v\} - \int_\Omega \{u^m v + \tau f^{m+1}v\} = 0. \qquad (7.25)$$

Using the identity just above, we can calculate the finite element approximation u_h^m of u^m in a step-by-step manner with respect to t.

Example 39 *We now solve the following example with the exact solution $u(x,y,t) = tx^4$.*

$$\begin{aligned}&\frac{\partial u}{\partial t} - \mu\Delta u = x^4 - \mu 12tx^2 \ \ \textit{in } \Omega\times]0,3[,\ \Omega =]0,1[^2\\ &u(x,y,0) = 0 \quad \textit{on } \Omega, \qquad u|_{\partial\Omega} = t*x^4\end{aligned}$$

```
                                        //     heat equation ∂_t u = −μΔu = x^4 − μ12tx^2
mesh Th=square(16,16);
fespace Vh(Th,P1);

Vh u,v,uu,f,g;
real dt = 0.1, mu = 0.01;
problem dHeat(u,v) =
    int2d(Th)( u*v + dt*mu*(dx(u)*dx(v) + dy(u)*dy(v)))
    + int2d(Th) (- uu*v - dt*f*v )
    + on(1,2,3,4,u=g);

real t = 0;                                              //     start from t=0
uu = 0;                                                      //     u(x,y,0)=0
for (int m=0;m<=3/dt;m++)
{
   t=t+dt;
   f = x^4-mu*t*12*x^2;
   g = t*x^4;
   dHeat;
   plot(u,wait=true);
   uu = u;
   cout <<"t="<<t<<"L^2-Error="<<sqrt( int2d(Th)((u-t*x^4)^2) ) << endl;
}
```

In the last statement, the L^2*-error* $\left(\int_\Omega |u-tx^4|^2\right)^{1/2}$ *is calculated at* $t=m\tau, \tau=0.1$*. At* $t=0.1$*, the error is 0.000213269. The errors increase with* m *and 0.00628589 at* $t=3$*. The iteration of the backward Euler (7.25) is made by* **for loop** *(see Section 2.8).*

Note 22 *The stiffness matrix in loop is used over and over again.* `freefem++` *support reuses of stiffness matrix.*

7.5.1 Mathematical Theory on time difference

In this section, we show the advantage of the backward Euler. Let V,H be separable Hilbert space and V is dense in H. Let a be a continuous bilinear form over $V\times V$ with coercivity and symmetry. Then $\sqrt{a(v,v)}$ become equivalent to the norm $\|v\|$ of V.
problem Ev(f,Ω): For a given $f\in L^2(0,T;V')$, $u^0\in H$

$$\begin{aligned} \frac{d}{dt}(u(t),v)+a(u(t),v) &= (f(t),v) \qquad \forall v\in V,, \quad a.e.\, t\in[0,T] \\ u(0) &= u^0 \end{aligned} \tag{7.26}$$

where V' is the dual space of V. Then, there is an unique solution $u\in L^\infty(0,T;H)\cap L^2(0,T;V)$.
Let us denote the time step by $\tau>0$, $N_T=[T/\tau]$. For the discretization, we put $u^n=u(n\tau)$ and consider the time difference for each $\theta\in[0,1]$

$$\begin{gathered} \frac{1}{\tau}\left(u_h^{n+1}-u_h^n,\phi_i\right)+a\left(u_h^{n+\theta},\phi_i\right)=\langle f^{n+\theta},\phi_i\rangle \\ i=1,\cdots,m, \quad n=0,\cdots,N_T \\ u_h^{n+\theta}=\theta u_h^{n+1}+(1-\theta)u_h^n, \quad f^{n+\theta}=\theta f^{n+1}+(1-\theta)f^n \end{gathered} \tag{7.27}$$

Formula (7.27) is the *forward Euler scheme* if $\theta = 0$, *Crank-Nicolson scheme* if $\theta = 1/2$, the *backward Euler scheme* if $\theta = 1$.
Unknown vectors $u^n = (u_h^1, \cdots, u_h^M)^T$ in

$$u_h^n(x) = u_1^n \phi_1(x) + \cdots + u_m^n \phi_m(x), \quad u_1^n, \cdots, u_m^n \in \mathbb{R}$$

are obtained from solving the matrix

$$\begin{aligned} (M + \theta\tau A)u^{n+1} &= \{M - (1-\theta)\tau A\}u^n + \tau\left\{\theta f^{n+1} + (1-\theta)f^n\right\} \\ M = (m_{ij}), \quad m_{ij} &= (\phi_j, \phi_i), \qquad A = (a_{ij}), \quad a_{ij} = a(\phi_j, \phi_i) \end{aligned} \tag{7.28}$$

Refer [17, pp.70–75] for solvability of (7.28). The stability of (7.28) is in [17, Theorem 2.13]:

> Let $\{\mathcal{T}_h\}_{h\downarrow 0}$ be regular triangulations (see Section 3.4). Then there is a number $c_0 > 0$ independent of h such that,
>
> $$|u_h^n|^2 \le \begin{cases} \frac{1}{\delta}\left\{|u_h^0|^2 + \tau\sum_{k=0}^{n-1}\|f^{k+\theta}\|_{V_h'}^2\right\} & \theta \in [0, 1/2) \\ |u_h^0|^2 + \tau\sum_{k=0}^{n-1}\|f^{k+\theta}\|_{V_h'}^2 & \theta \in [1/2, 1] \end{cases} \tag{7.29}$$
>
> if the following are satisfied:
>
> 1. When $\theta \in [0, 1/2)$, then we can take a time step τ in such a way that
>
> $$\tau < \frac{2(1-\delta)}{(1-2\theta)c_0^2}h \tag{7.30}$$
>
>   for arbitrary $\delta \in (0, 1)$.
> 2. When $1/2 \le \theta \le 1$, we can take τ arbitrary.

Example 40

```
mesh Th=square(12,12);
fespace Vh(Th,P1);
fespace Ph(Th,P0);

Ph h = hTriangle;                                    //    mesh sizes for each triangle
real tau = 0.1, theta=0.;
func real f(real t) {
   return x^2*(x-1)^2 + t*(-2 + 12*x - 11*x^2 - 2*x^3 + x^4);
}
ofstream out("err02.csv");                             //    file to store calculations
out << "mesh size = "<<h[].max<<", time step = "<<tau<<endl;
for (int n=0;n<5/tau;n++) \\
   out<<n*tau<<",";
out << endl;
Vh u,v,oldU;
Vh f1, f0;
problem aTau(u,v) =
  int2d(Th)( u*v + theta*tau*(dx(u)*dx(v) + dy(u)*dy(v) + u*v))
  - int2d(Th)(oldU*v - (1-theta)*tau*(dx(oldU)*dx(v)+dy(oldU)*dy(v)+oldU*v))
  - int2d(Th)(tau*( theta*f1+(1-theta)*f0 )*v );
```

```
while (theta <= 1.0) {
  real t = 0, T=3;                                            //     from t=0 to T
  oldU = 0;                                                      //     u(x,y,0)=0
  out <<theta<<",";
  for (int n=0;n<T/tau;n++) {
      t = t+tau;
      f0 = f(n*tau); f1 = f((n+1)*tau);
      aTau;
      oldU = u;
      plot(u);
      Vh uex = t*x^2*(1-x)^2;                       //     exact sol.= tx^2(1 - x)^2
      Vh err = u - uex;                                  //     err =FE-sol - exact
      out<< abs(err[].max)/abs(uex[].max) <<",";       //     ||err||_{L^∞(Ω)}/||u_ex||_{L^∞(Ω)}
  }
  out << endl;
  theta = theta + 0.1;
}
```

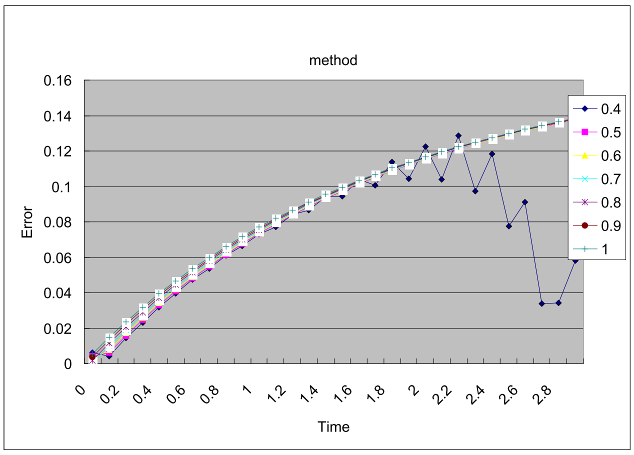


Figure 7.16: $\max_{x\in\Omega}|u_h^n(\theta) - u_{ex}(n\tau)|/\max_{x\in\Omega}|u_{ex}(n\tau)|$ at $n = 0, 1, \cdots, 29$

We can see in Fig. 7.16 that $u_h^n(\theta)$ become unstable at $\theta = 0.4$, and figures are omitted in the case $\theta < 0.4$.

7.5.2 Convection

The hyperbolic equation

$$\partial_t u - \boldsymbol{\alpha}\cdot\nabla u = f; \quad \text{for a vector-valued function } \boldsymbol{\alpha},\ \partial_t = \frac{\partial}{\partial t}, \tag{7.31}$$

appear frequently in scientific problems, for example, Navier-Stokes equation, Convection-Diffusion equation, etc.
In the case of 1-dimensional space, we can easily find the general solution $(x,t)\mapsto u(x,t) = u^0(x - \alpha t)$ of the following equation, if α is constant,

$$\partial_t u + \alpha\partial_x u = 0, \qquad u(x,0) = u^0(x), \tag{7.32}$$

because $\partial_t u + \alpha\partial_x u = -\alpha\dot{u}^0 + a\dot{u}^0 = 0$, where $\dot{u}^0 = du^0(x)/dx$. Even if α is not constant construction, the principle is similar. One begins the ordinary differentielle equation (with convention which α is prolonged by zero apart from $(0,L)\times(0,T)$):

$$\dot{X}(\tau) = -\alpha(X(\tau),\tau), \quad \tau \in (0,t) \quad X(t) = x$$

In this equation τ is the variable and x,t is parameters, and we denote the solution by $X_{x,t}(\tau)$. Then it is noticed that $(x,t) \to v(X(\tau),\tau)$ in $\tau = t$ satisfy the equation

$$\partial_t v + \alpha\partial_x v = \partial_t X\dot{v} + a\partial_x X\dot{v} = 0$$

and by the definition $\partial_t X = \dot{X} = -\alpha$ and $\partial_x X = \partial_x x$ in $\tau = t$, because if $\tau = t$ we have $X(\tau) = x$. The general solution of (7.32) is thus the value of the boundary condition in $X_{x,t}(0)$, it is has to say $u(x,t) = u^0(X_{x,t}(0))$ if $X_{x,t}(0)$ is on the x axis, $u(x,t) = u^0(X_{x,t}(0))$ if $X_{x,t}(0)$ is on the axis of t.
In higher dimension $\Omega \subset R^d,\ d = 2,3$, the equation of the convection is written

$$\partial_t u + \boldsymbol{\alpha}\cdot\nabla u = 0 \text{ in } \Omega\times(0,T)$$

where $\boldsymbol{a}(x,t) \in R^d$. `freefem++` implements the Characteristic-Galerkin method for convection operators. Recall that the equation (7.31) can be discretized as

$$\frac{Du}{Dt} = f \text{ i.e. } \frac{du}{dt}(X(t),t) = f(X(t),t) \text{ where } \frac{dX}{dt}(t) = \boldsymbol{\alpha}(X(t),t)$$

where D is the total derivative operator. So a good scheme is one step of backward convection by the method of Characteristics-Galerkin

$$\frac{1}{\tau}\left(u^{m+1}(x) - u^m(X^m(x))\right) = f^m(x) \tag{7.33}$$

where $X^m(x)$ is an approximation of the solution at $t = m\tau$ of the ordinary differential equation

$$\frac{d\boldsymbol{X}}{dt}(t) = \boldsymbol{\alpha}^m(\boldsymbol{X}(t)),\ \boldsymbol{X}((m+1)\tau) = x.$$

where $\boldsymbol{\alpha}^m(x) = (\alpha_1(x,m\tau),\alpha_2(x,m\tau))$. Because, by Taylor's expansion, we have

$$\begin{aligned} u^m(\boldsymbol{X}(m\tau)) &= u^m(\boldsymbol{X}((m+1)\tau)) - \tau\sum_{i=1}^{d}\frac{\partial u^m}{\partial x_i}(\boldsymbol{X}((m+1)\tau))\frac{\partial X_i}{\partial t}((m+1)\tau) + o(\tau) \\ &= u^m(x) - \tau\boldsymbol{\alpha}^m(x)\cdot\nabla u^m(x) + o(\tau) \end{aligned} \tag{7.34}$$

where $X_i(t)$ are the i-th component of $\boldsymbol{X}(t)$, $u^m(x) = u(x,m\tau)$ and we used the chain rule and $x = \boldsymbol{X}((m+1)\tau)$. From (7.34), it follows that

$$u^m(X^m(x)) = u^m(x) - \tau\boldsymbol{\alpha}^m(x)\cdot\nabla u^m(x) + o(\tau). \tag{7.35}$$

Also we apply Taylor's expansion for $t \mapsto u^m(x - \boldsymbol{\alpha}^m(x)t)$, $0 \le t \le \tau$, then

$$u^m(x - \boldsymbol{\alpha}\tau) = u^m(x) - \tau\boldsymbol{\alpha}^m(x)\cdot\nabla u^m(x) + o(\tau).$$

Putting

$$\texttt{convect}\left(\boldsymbol{\alpha},\tau,u^m\right)=u^m\left(x-\boldsymbol{\alpha}^m\tau\right),$$

we can get the approximation

$$u^m\left(X^m(x)\right)\approx\texttt{convect}\left([a_1^m,a_2^m],\tau,u^m\right)\ \text{ by } x=X((m+1)\tau).$$

A classical convection problem is that of the "rotating bell" (quoted from [10][p.16]). Let Ω be the unit disk centered at 0, with its center rotating with speed $\alpha_1=y$, $\alpha_2=-x$ We consider the problem (7.31) with $f=0$ and the initial condition $u(x,0)=u^0(x)$, that is, from (7.33)

$$u^{m+1}(x)=u^m(X^m(x))\approx\texttt{convect}(\boldsymbol{\alpha},\tau,u^m).$$

The exact solution is $u(x,t)=u(\boldsymbol{X}(t))$ where $\boldsymbol{X}$ equals x rotated around the origin by an angle $\theta=-t$ (rotate in clockwise). So, if u^0 in a 3D perspective looks like a bell, then u will have exactly the same shape, but rotated by the same amount. The program consists in solving the equation until $T=2\pi$, that is for a full revolution and to compare the final solution with the initial one; they should be equal.

Example 41 (convect.edp)
```
border C(t=0, 2*pi) { x=cos(t);  y=sin(t); };        // the unit circle
mesh Th = buildmesh(C(70));                          //    triangulates the disk
fespace Vh(Th,P1);
Vh u0 = exp(-10*((x-0.3)^2 +(y-0.3)^2));                          //    give u0

real dt = 0.17,t=0;                                             //    time step
Vh a1 = -y, a2 = x;                                     //    rotation velocity
Vh u;                                                             //     um+1
for (int m=0; m<2*pi/dt ; m++) {
    t += dt;
    u=convect([a1,a2],dt,u0);                          //     um+1 = um(Xm(x))
    u0=u;                                                           //    m++
    plot(u,cmm="convection: t="+t + ", min=" + u[].min + ", max=" +  u[].max,wait=0);
};
```

Note 23 *The scheme* `convect` *is unconditionally stable, then the bell become lower and lower (the maximum of u^{37} is 0.406 as shown in Fig. 7.18).*

7.5.3 Two-dimensional Black-Scholes equation

In mathematical finance, an option on two assets is modeled by a Black-Scholes equations in two space variables, (see for example Wilmott's book : a student introduction to mathematical finance, Cambridge University Press).

$$\begin{aligned}\partial_t u+\frac{(\sigma_1 x)^2}{2}\frac{\partial^2 u}{\partial x^2}+\frac{(\sigma_2 y)^2}{2}\frac{\partial^2 u}{\partial y^2}\\ +\rho xy\frac{\partial^2 u}{\partial x\partial y}+rS_1\frac{\partial u}{\partial x}+rS_2\frac{\partial u}{\partial y}-rP=0\end{aligned}\tag{7.36}$$

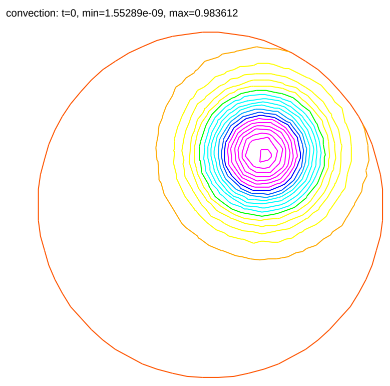


Figure 7.17: $u^0 = e^{-10((x-0.3)^2+(y-0.3)^2)}$

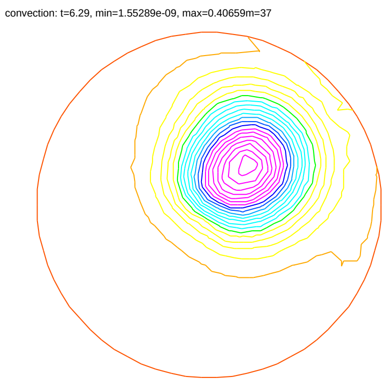


Figure 7.18: The bell at $t = 6.29$

which is to be integrated in $(0, T) \times \mathbb{R}^+ \times \mathbb{R}^+$ subject to, in the case of a put

$$u(x, y, T) = (K - \max(x, y))^+ . \tag{7.37}$$

Boundary conditions for this problem may not be so easy to device. As in the one dimensional case the PDE contains boundary conditions on the axis $x_1 = 0$ and on the axis $x_2 = 0$, namely two one dimensional Black-Scholes equations driven respectively by the data $u(0, +\infty, T)$ and $u(+\infty, 0, T)$. These will be automatically accounted for because they are embedded in the PDE. So if we do nothing in the variational form (i.e. if we take a Neuman boundary condition at these two axis in the strong form) there will be no disturbance to these. At infinity in one of the variable, as in 1D, it makes sense to match the final condition:

$$u(x, y, t) \approx (K - \max(x, y))^+ e^{r(T-t)} \text{when } |\text{x}| \to \infty \tag{7.38}$$

For an American put we will also have the constraint

$$u(x, y, t) \geq (K - \max(x, y))^+ e^{r(T-t)}. \tag{7.39}$$

We take

$$\sigma_1 = 0.3, \quad \sigma_2 = 0.3, \quad \rho = 0.3, \quad r = 0.05, \quad K = 40, \quad T = 0.5 \tag{7.40}$$

An implicit Euler scheme with projection is used and a mesh adaptation is done every 10 time steps. The first order terms are treated by the Characteristic Galerkin method, which, roughly, approximates

$$\frac{\partial u}{\partial t} + a_1 \frac{\partial u}{\partial x} + a_2 \frac{\partial u}{\partial y} \approx \frac{1}{\tau} \left(u^{n+1}(x) - u^n(x - \boldsymbol{\alpha}\tau) \right) \tag{7.41}$$

Example 42 *[blakschol.edp]*

```
                                                      //    verbosity=1;
int s=10;                                                   //    y-scale
int m=30;
int L=80;
int LL=80;
border aa(t=0,L){x=t;y=0;};
border bb(t=0,LL){x=L;y=t;};
border cc(t=L,0){x=t ;y=LL;};
border dd(t=LL,0){x = 0; y = t;};

mesh th = buildmesh(aa(m)+bb(m)+cc(m)+dd(m));
fespace Vh(th,P1);

real sigmax=0.3;
real sigmay=0.3;
real rho=0.3;
real r=0.05;
real K=40;
real dt=0.01;

real eps=0.3;

func f = max(K-max(x,y),0.);

Vh u=f,v,w;

func beta = 1;                                  //   (w<=f-eps)*eps + (w>=f) +
(w<f)*(w>f-eps)*(eps+(w-f+eps)/eps)*(1-eps);

plot(u,wait=1);

 th = adaptmesh(th,u,abserror=1,nbjacoby=2,
        err=0.004, nbvx=5000, omega=1.8,ratio=1.8, nbsmooth=3,
            splitpbedge=1, maxsubdiv=5,rescaling=1 );
 u=u;

Vh xveloc = -x*r+x*sigmax^2+x*rho*sigmax*sigmay/2;
Vh yveloc = -y*r+y*sigmay^2+y*rho*sigmax*sigmay/2;

int j=0;
int n;
problem eq1(u,v,init=j,solver=LU) = int2d(th)(
                          u*v*(r+1/dt/beta)
                        + dx(u)*dx(v)*(x*sigmax)^2/2.
                        + dy(u)*dy(v)*(y*sigmay)^2/2.
      + dy(u)*dx(v)*rho*sigmax*sigmay*x*y/2.
                  + dx(u)*dy(v)*rho*sigmax*sigmay*x*y/2.   )
            + int2d(th)( -v*convect([xveloc,yveloc],dt,w)/dt/beta)
                  + on(bb,cc,u=f)                          //   *exp(-r*t);
;
int ww=1;
for ( n=0; n*dt <= 1.0; n++)
{
  cout <<" iteration " << n   <<  " j=" << j << endl;
  w=u;
  eq1;
  v = max(u-f,0.);
  plot(v,wait=ww);
  u = max(u,f);
  ww=0;

  if(j>10)  { cout << " adaptmesh " << endl;
    th = adaptmesh(th,u,verbosity=1,abserror=1,nbjacoby=2,
      err=0.001, nbvx=5000, omega=1.8, ratio=1.8, nbsmooth=3,
            splitpbedge=1, maxsubdiv=5,rescaling=1) ;
        j=-1;
     xveloc = -x*r+x*sigmax^2+x*rho*sigmax*sigmay/2;
     yveloc = -y*r+y*sigmay^2+y*rho*sigmax*sigmay/2;
     u=u;
     ww=1;
    };
  j=j+1;
  cout << " j = " <<  j << endl;
};
v = max(u-f,0.);
plot(v,wait=1,value=1);
plot(u,wait=1,value=1);
```

7.6 Navier-Stokes Equation

7.6.1 Stokes and Navier-Stokes

The Stokes equations are: for a given $\boldsymbol{f} \in L^2(\Omega)^2$,

$$\left.\begin{array}{ll} -\Delta \boldsymbol{u} + \nabla p & = \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} & = 0 \end{array}\right\} \quad \text{in } \Omega \tag{7.42}$$

where $\boldsymbol{u} = (u_1, u_2)$ is the velocity vector and p the pressure. For simplicity, let us choose Dirichlet boundary conditions on the velocity, $\boldsymbol{u} = \boldsymbol{u}_\Gamma$ on Γ.
In Teman[Theorem 2.2], there ia a weak form of (7.42): Find $\boldsymbol{v} = (v_1, v_2) \in \boldsymbol{V}(\Omega)$

$$\boldsymbol{V}(\Omega) = \{\boldsymbol{w} \in H_0^1(\Omega)^2 |\ \mathrm{div}\boldsymbol{w} = 0\}$$

which satisfy

$$\sum_{i=1}^{2} \int_\Omega \nabla u_i \cdot \nabla v_i = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{w} \quad \text{for all } v \in V$$

Here it is used the existence $p \in H^1(\Omega)$ such that $\boldsymbol{u} = \nabla p$, if

$$\int_\Omega \boldsymbol{u} \cdot \boldsymbol{v} = 0 \quad \text{for all } \boldsymbol{v} \in V$$

Another weak form is derived as follows: We put

$$\boldsymbol{V} = H_0^1(\Omega)^2; \quad W = \left\{ q \in L^2(\Omega) \middle| \int_\Omega q = 0 \right\}$$

By multiplying the first equation in (7.42) with $v \in V$ and the second with $q \in W$, subsequent integration over Ω, and an application of Green's formula, we have

$$\begin{aligned} \int_\Omega \nabla \boldsymbol{u} \cdot \nabla \boldsymbol{v} - \int_\Omega \mathrm{div} \boldsymbol{v}\, p &= \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} \\ \int_\Omega \mathrm{div} \boldsymbol{u}\, q &= 0 \end{aligned}$$

This yields the weak form of (7.42): Find $(\boldsymbol{u}, p) \in \boldsymbol{V} \times W$ such that

$$a(\boldsymbol{u}, \boldsymbol{v}) + b(\boldsymbol{v}, p) = (\boldsymbol{f}, \boldsymbol{v}) \tag{7.43}$$

$$b(\boldsymbol{u}, q) = 0 \tag{7.44}$$

for all $(\boldsymbol{v}, q) \in V \times W$, where

$$a(\boldsymbol{u}, \boldsymbol{v}) = \int_\Omega \nabla \boldsymbol{u} \cdot \nabla \boldsymbol{v} = \sum_{i=1}^{2} \int_\Omega \nabla u_i \cdot \nabla v_i \tag{7.45}$$

$$b(\boldsymbol{u}, q) = -\int_\Omega \mathrm{div} \boldsymbol{u}\, q \tag{7.46}$$

Now, we consider finite element spaces $\boldsymbol{V}_h \subset \boldsymbol{V}$ and $W_h \subset W$, and we assume the following basis functions

$$\begin{aligned} &\boldsymbol{V}_h = V_h \times V_h, \quad V_h = \{v_h |\; v_h = v_1\phi_1 + \cdots + v_{M_V}\phi_{M_V}\}, \\ &W_h = \{q_h |\; q_h = q_1\varphi_1 + \cdots + q_{M_W}\varphi_{M_W}\} \end{aligned}$$

The discrete weak form is: Find $(\boldsymbol{u}_h, p_h) \in \boldsymbol{V}_h \times W_h$ such that

$$\begin{aligned} a(\boldsymbol{u}_h, \boldsymbol{v}_h) + b(\boldsymbol{v}_h, p) &= (\boldsymbol{f}, \boldsymbol{v}_h), & \forall \boldsymbol{v}_h \in \boldsymbol{V}_h \\ b(\boldsymbol{u}_h, q_h) &= 0, & \forall q_h \in W_h \end{aligned} \tag{7.47}$$

Note 24 *Assume that:*

1. *There is a constant $\alpha_h > 0$ such that*

$$a(\boldsymbol{v}_h, \boldsymbol{v}_h) \geq \alpha \|\boldsymbol{v}_h\|_{1,\Omega}^2 \quad \textit{for all } \boldsymbol{v}_h \in Z_h$$

where

$$Z_h = \{\boldsymbol{v}_h \in \boldsymbol{V}_h |\; b(\boldsymbol{w}_h, q_h) = 0 \quad \textit{for all } q_h \in W_h\}$$

2. *There is a constant $\beta_h > 0$ such that*

$$\sup_{\boldsymbol{v}_h \in \boldsymbol{V}_h} \frac{b(\boldsymbol{v}_h, q_h)}{\|\boldsymbol{v}_h\|_{1,\Omega}} \geq \beta_h \|q_h\|_{0,\Omega} \quad \textit{for all } q_h \in W_h$$

Then we have an unique solution $(\boldsymbol{u}_h, p_h)$ of (7.47) satisfying

$$\|\boldsymbol{u} - \boldsymbol{u}_h\|_{1,\Omega} + \|p - p_h\|_{0,\Omega} \leq C \left(\inf_{\boldsymbol{v}_h \in \boldsymbol{V}_h} \|u - v_h\|_{1,\Omega} + \inf_{q_h \in W_h} \|p - q_h\|_{0,\Omega} \right)$$

with a constant $C > 0$ (see e.g. [15, Theorem 10.4]).

Let us denote that

$$\begin{aligned} A &= (A_{ij}),\ A_{ij} = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \qquad i,j = 1,\cdots,M_{\boldsymbol{V}} \\ \boldsymbol{B} &= (Bx_{ij}, By_{ij}),\ Bx_{ij} = -\int_\Omega \partial\phi_j/\partial x\, \varphi_i \qquad By_{ij} = -\int_\Omega \partial\phi_j/\partial y\, \varphi_i \\ & \qquad i = 1,\cdots,M_W; j = 1,\cdots,M_V \end{aligned} \tag{7.48}$$

then (7.47) is written by

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B}^* \\ \boldsymbol{B} & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{U}_h \\ \{p_h\} \end{pmatrix} = \begin{pmatrix} \boldsymbol{F}_h \\ 0 \end{pmatrix} \tag{7.49}$$

where

$$\boldsymbol{A} = \begin{pmatrix} A & 0 \\ 0 & A \end{pmatrix} \qquad \boldsymbol{B}^* = \begin{Bmatrix} Bx^T \\ By^T \end{Bmatrix} \qquad \boldsymbol{U}_h = \begin{Bmatrix} \{u_{1,h}\} \\ \{u_{2,h}\} \end{Bmatrix} \qquad \boldsymbol{F}_h = \begin{Bmatrix} \{\int_\Omega f_1\phi_i\} \\ \{\int_\Omega f_2\phi_i\} \end{Bmatrix}$$

Penalty method: This method consists of replacing (7.47) by a more regular problem: Find $(\boldsymbol{v}_h^\epsilon, p_h^\epsilon) \in \boldsymbol{V}_h \times \tilde{W}_h$ satisfying

$$\begin{aligned} a(\boldsymbol{u}_h^\epsilon, \boldsymbol{v}_h) + b(\boldsymbol{v}_h, p_h^\epsilon) &= (\boldsymbol{f}, \boldsymbol{v}_h), & \forall \boldsymbol{v}_h \in \boldsymbol{V}_h \\ b(\boldsymbol{u}_h^\epsilon, q_h) - \epsilon(p_h^\epsilon, q_h) &= 0, & \forall q_h \in \tilde{W}_h \end{aligned} \tag{7.50}$$

where $\tilde{W}_h \subset L^2(\Omega)$. Formally, we have

$$\mathrm{div}\boldsymbol{u}_h^\epsilon = \epsilon p_h^\epsilon$$

and the corresponding algebraic problem

$$\begin{pmatrix} \boldsymbol{A} & B^* \\ B & \epsilon I \end{pmatrix} \begin{pmatrix} \boldsymbol{U}_h^\epsilon \\ \{p_h^\epsilon\} \end{pmatrix} = \begin{pmatrix} \boldsymbol{F}_h \\ 0 \end{pmatrix}$$

Note 25 *We can eliminate $p_h^\epsilon = (1/\epsilon)BU_h^\epsilon$ to obtain*

$$(A + (1/\epsilon)B^*B)\boldsymbol{U}_h^\epsilon = \boldsymbol{F}_h^\epsilon \tag{7.51}$$

Since the matrix $A + (1/\epsilon)B^*B$ is symmetric, positive-definite, and sparse, (7.51) can be solved by known technique. There is a constant $C > 0$ independent of ϵ such that

$$\|\boldsymbol{u}_h - \boldsymbol{u}_h^\epsilon\|_{1,\Omega} + \|p_h - p_h^\epsilon\|_{0,\Omega} \le C\epsilon$$

(see e.g. [15, 17.2])

Example 43 (Cavity.edp) *The driven cavity flow problem is solved first at zero Reynolds number (Stokes flow) and then at Reynolds 100. The velocity pressure formulation is used first and then the calculation is repeated with the stream function vorticity formulation.*
We solve the driven cavity problem by the penalty method (7.50) where $\boldsymbol{u}_\Gamma \cdot \boldsymbol{n} = 0$ and $\boldsymbol{u}_\Gamma \cdot \boldsymbol{s} = 1$ on the top boundary and zero elsewhere ($\boldsymbol{n}$ is the unit normal to Γ, and $\boldsymbol{s}$ the

unit tangent to Γ).
The mesh is constructed by

```
mesh Th=square(8,8);
```

We use a classical Taylor-Hood element technic to solve the problem:

The velocity is approximated with the P_2 FE (X_h space), and the the pressure is approximated with the P_1 FE (M_h space),

where

$$X_h = \left\{ \boldsymbol{v} \in H^1(]0,1[^2) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_2 \right\}$$

and

$$M_h = \left\{ v \in H^1(]0,1[^2) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_1 \right\}$$

The FE spaces and functions are constructed by

```
fespace Xh(Th,P2);            //    definition of the velocity component space
fespace Mh(Th,P1);                      //    definition of the pressure space
Xh u2,v2;
Xh u1,v1;
Xh p,q;
```

The Stokes operator is implemented as a system-solve for the velocity $(u1, u2)$ and the pressure p. The test function for the velocity is $(v1, v2)$ and q for the pressure, so the variational form (7.47) in freefem language is:

```
solve Stokes (u1,u2,p,v1,v2,q,solver=Crout) =
    int2d(Th)( ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
            +  dx(u2)*dx(v2) + dy(u2)*dy(v2) )
            - p*q*(0.000001)
            - p*dx(v1) - p*dy(v2)
            - dx(u1)*q - dy(u2)*q
           )
  + on(3,u1=1,u2=0)
  + on(1,2,4,u1=0,u2=0);              //    see Section 3.1.1 for labels 1,2,3,4
```

Each unknown has its own boundary conditions.

If the streamlines are required, they can be computed by finding ψ such that $\mathrm{rot}\psi = u$ or better,

$$-\Delta\psi = \nabla \times u$$

```
Xh psi,phi;

solve streamlines(psi,phi) =
      int2d(Th)( dx(psi)*dx(phi) + dy(psi)*dy(phi))
   +  int2d(Th)( -phi*(dy(u1)-dx(u2)))
   +  on(1,2,3,4,psi=0);
```

Now the Navier-Stokes equations are solved

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \nu \Delta u + \nabla p = 0, \quad \nabla \cdot u = 0$$

with the same boundary conditions and with initial conditions $u = 0$.
This is implemented by using the convection operator `convect` *for the term* $\frac{\partial u}{\partial t} + u \cdot \nabla u$, *giving a discretization in time*

$$\begin{array}{ll} \frac{1}{\tau}(u^{n+1} - u^n \circ X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} & = 0, \\ \nabla \cdot u^{n+1} & = 0 \end{array} \tag{7.52}$$

The term $u^n \circ X^n(x) \approx u^n(x - u^n(x)\tau)$ *will be computed by the operator "convect" , so we obtain*

```
int i=0;
real  nu=1./100.;
real dt=0.1;
real alpha=1/dt;

Xh up1,up2;

problem  NS (u1,u2,p,v1,v2,q,solver=Crout,init=i) =
    int2d(Th)(
             alpha*( u1*v1 + u2*v2)
            + nu * ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
            +  dx(u2)*dx(v2) + dy(u2)*dy(v2) )
            - p*q*(0.000001)
            - p*dx(v1) - p*dy(v2)
            - dx(u1)*q - dy(u2)*q
           )
  + int2d(Th) ( -alpha*
       convect([up1,up2],-dt,up1)*v1 -alpha*convect([up1,up2],-dt,up2)*v2 )
  + on(3,u1=1,u2=0)
  + on(1,2,4,u1=0,u2=0)
;

for (i=0;i<=10;i++)
 {
   up1=u1;
   up2=u2;
   NS;
   if ( !(i % 10))                                //    plot every 10 iteration
    plot(coef=0.2,cmm=" [u1,u2] and p  ",p,[u1,u2]);
 } ;
```

Notice that the stiffness matrices are reused (keyword `init=i`*)*

7.6.2 Uzawa Conjugate Gradient

We solve Stokes problem without penalty. The classical iterative method of Uzawa is described by the algorithm (see e.g.[15, 17.3]):

Initialize: Let p_h^0 be an arbitrary chosen element of $L^2(\Omega)$.

Calculate $\boldsymbol{u}_h$: Once p_h^n is known, $\boldsymbol{v}_h^n$ is the solution of

$$\boldsymbol{u}_h^n = A^{-1}(\boldsymbol{f}_h - \boldsymbol{B}^* p_h^n)$$

Advance p_h: Let p_h^{n+1} be defined by

$$p_h^{n+1} = p_h^n + \rho_n \boldsymbol{B}\boldsymbol{u}_h^n$$

There is a constant $\alpha > 0$ such that $\alpha \le \rho_n \le 2$ for each n, then $\boldsymbol{u}_h^n$ converges to the solution $\boldsymbol{u}_h$, and then $B\boldsymbol{v}_h^n \to 0$ as $n \to \infty$ from the *Advance* p_h. This method in general converges quite slowly.
First we define mesh, and the Taylor-Hood approximation. So X_h is the velocity space, and M_h is the pressure space.

Example 44 (StokesUzawa.edp)

```
mesh Th=square(10,10);
fespace Xh(Th,P2),Mh(Th,P1);
Xh u1,u2,v1,v2;
Mh p,q,ppp;                                          //    ppp is a working pressure


varf bx(u1,q) = int2d(Th)( -(dx(u1)*q));
varf by(u1,q) = int2d(Th)( -(dy(u1)*q));
varf a(u1,u2)= int2d(Th)(  dx(u1)*dx(u2) + dy(u1)*dy(u2) )
                    +  on(3,u1=1)  +  on(1,2,4,u1=0) ;
                       //    remark:  put the on(3,u1=1) before on(1,2,4,u1=0)
                                   //    because we want zero on intersection

matrix A= a(Xh,Xh,solver=CG);
matrix Bx= bx(Xh,Mh);                                              //     B = (Bx   By)
matrix By= by(Xh,Mh);

Xh bc1; bc1[] = a(0,Xh);                 //    boundary condition contribution on u1
Xh bc2; bc2   = 0 ;                   //    no boundary condition contribution on u2
Xh b;
```

$p_h^n \to \boldsymbol{B}A^{-1}(-\boldsymbol{B}^* p_h^n) = -div\boldsymbol{u}_h$ *is realized as the function* divup.

```
func real[int] divup(real[int] & pp)
{
                                                                  //    compute u1(pp)
   b[]  = Bx'*pp; b[] *=-1; b[] += bc1[] ;    u1[] = A^-1*b[];
                                                                  //    compute u2(pp)
   b[]  = By'*pp; b[] *=-1; b[] += bc2[] ;    u2[] = A^-1*b[];
   //     u^n = A^-1(Bx^T p^n   By^T p^n)^T
   ppp[] =   Bx*u1[];                                                  //     ppp = Bxu_1
   ppp[] +=  By*u2[];                                          //      +Byu_2
   return ppp[] ;
};
```

Call now the conjugate gradient algorithm:

```
p=0;q=0;                                                          //     p_h^0 = 0
LinearCG(divup,p[],eps=1.e-6,nbiter=50);                  //     p_h^{n+1} = p_h^n + Bu_h^n
//    if n > 50 or |p_h^{n+1} - p_h^n| <= 10^{-6}, then the loop end.
divup(p[]);                                       //    compute the final solution

plot([u1,u2],p,wait=1,value=true,coef=0.1);
```

7.6.3 NSUzawaCahouetChabart.edp

In this example we solve the Navier-Stokes equation, in the driven-cavity, with the Uzawa algorithm preconditioned by the Cahouet-Chabart method.
The idea of the preconditionner is that in a periodic domain, all differential operators commute and the Uzawa algorithm comes to solving the linear operator $\nabla.((\alpha Id + \nu\Delta)^{-1}\nabla$, where Id is the identity operator. So the preconditioner suggested is $\alpha\Delta^{-1} + \nu Id$.

To implement this, we reuse the previous example, by including a file. Then we define the time step Δt, viscosity, and new variational form and matrix.

Example 45 (NSUzawaCahouetChabart.edp)

```
include "StokesUzawa.edp"                          //    include the Stokes part
real dt=0.05, alpha=1/dt;                                            //    Δt

cout << " alpha = " << alpha;
real xnu=1./400;                            //    viscosity ν = Reynolds number^{-1}

                                  //    the new variational form with mass term
varf at(u1,u2)= int2d(Th)( xnu*dx(u1)*dx(u2)
                         + xnu*dy(u1)*dy(u2) + u1*u2*alpha  )
                         +  on(1,2,4,u1=0)  + on(3,u1=1) ;

A = at(Xh,Xh,solver=CG);                                //    change the matrix

                                      //    set the 2 convect variational form
varf  vfconv1(uu,vv)  = int2d(Th,qforder=5) (convect([u1,u2],-dt,u1)*vv*alpha);
varf  vfconv2(v2,v1)  = int2d(Th,qforder=5) (convect([u1,u2],-dt,u2)*v1*alpha);

int idt;                                                //    index of of time set
real temps=0;                                                 //    current time

Mh pprec,prhs;
varf vfMass(p,q) = int2d(Th)(p*q);
matrix MassMh=vfMass(Mh,Mh,solver=CG);

varf vfLap(p,q)  = int2d(Th)(dx(pprec)*dx(q)+dy(pprec)*dy(q) + pprec*q*1e-10);
matrix LapMh= vfLap(Mh,Mh,solver=Cholesky);
```

The function to define the preconditioner

```
func real[int]  CahouetChabart(real[int] & xx)
{                                                        //    xx = ∫(divu)w_i
                                                  //    αLapMh^-1 + νMassMh^-1
   pprec[]= LapMh^-1* xx;
   prhs[] =  MassMh^-1*xx;
   pprec[] = alpha*pprec[]+xnu* prhs[];
   return pprec[];
};
```

The loop in time. Warning with the stop test of the conjugate gradient, because we start from the previous solution and the end the previous solution is close to the final solution, don't take a relative stop test to the first residual, take an absolue stop test (negative here)

```
for (idt = 1; idt < 50; idt++)
 {
   temps += dt;
   cout << " --------- temps " << temps << " \n ";
   b1[] =  vfconv1(0,Xh);
   b2[] =  vfconv2(0,Xh);
   cout << "  min b1 b2  " << b1[].min << " " << b2[].min << endl;
   cout << "  max b1 b2  " << b1[].max << " " << b2[].max << endl;
                                 //    call Conjugued Gradient with preconditioner '
                                         //    warning eps < 0 => absolue stop test
   LinearCG(divup,p[],eps=-1.e-6,nbiter=50,precon=CahouetChabart);
   divup(p[]);                                        //    computed the velocity

   plot([u1,u2],p,wait=!(idt%10),value= 1,coef=0.1);
 }
```

7.7 Domain decomposition

We present, three classique examples, of domain decomposition technique: first, Schwarz algorithm with overlapping, second Schwarz algorithm without overlapping (also call Shur complement), and last we show to use the conjugate gradient to solve the boundary problem of the Shur complement.

7.7.1 Schwarz Overlap Scheme

To solve

$$-\Delta u = f, \quad \text{in } \Omega = \Omega_1 \cup \Omega_2 \quad u|_\Gamma = 0$$

the Schwarz algorithm runs like this

$$\begin{aligned} -\Delta u_1^{n+1} &= f \text{ in } \Omega_1 \quad u_1^{n+1}|_{\Gamma_1} = u_2^n \\ -\Delta u_2^{n+1} &= f \text{ in } \Omega_2 \quad u_2^{n+1}|_{\Gamma_2} = u_1^n \end{aligned}$$

where Γ_i is the boundary of Ω_i and on the condition that $\Omega_1 \cap \Omega_2 \neq \emptyset$ and that u_i are zero at iteration 1.

Here we take Ω_1 to be a quadrangle, Ω_2 a disk and we apply the algorithm starting from zero.

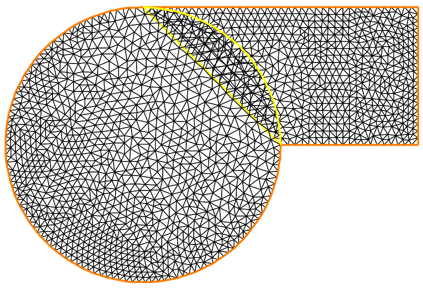


Figure 7.19: The 2 overlapping mesh `TH` and `th`

Example 46 (Schwarz-overlap.edp)

```
int inside = 2;                                              //    inside boundary
int outside = 1;                                             //    outside boundary
border a(t=1,2){x=t;y=0;label=outside;};
border b(t=0,1){x=2;y=t;label=outside;};
border c(t=2,0){x=t ;y=1;label=outside;};
border d(t=1,0){x = 1-t; y = t;label=inside;};
border e(t=0, pi/2){ x= cos(t); y = sin(t);label=inside;};
border e1(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;
mesh th = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
mesh TH = buildmesh( e(5*n) + e1(25*n) );
plot(th,TH,wait=1);                                          //    to see the 2 meshes
```

The space and problem definition is :

```
fespace vh(th,P1);
fespace VH(TH,P1);
vh u=0,v; VH U,V;
int i=0;

problem PB(U,V,init=i,solver=Cholesky) =
    int2d(TH)( dx(U)*dx(V)+dy(U)*dy(V) )
  + int2d(TH)( -V) + on(inside,U = u)  + on(outside,U= 0 ) ;
problem pb(u,v,init=i,solver=Cholesky) =
    int2d(th)( dx(u)*dx(v)+dy(u)*dy(v) )
  + int2d(th)( -v) + on(inside ,u = U) + on(outside,u = 0 ) ;
```

The calculation loop:

```
for ( i=0 ;i< 10; i++)
{
   PB;
   pb;
   plot(U,u,wait=true);
};
```

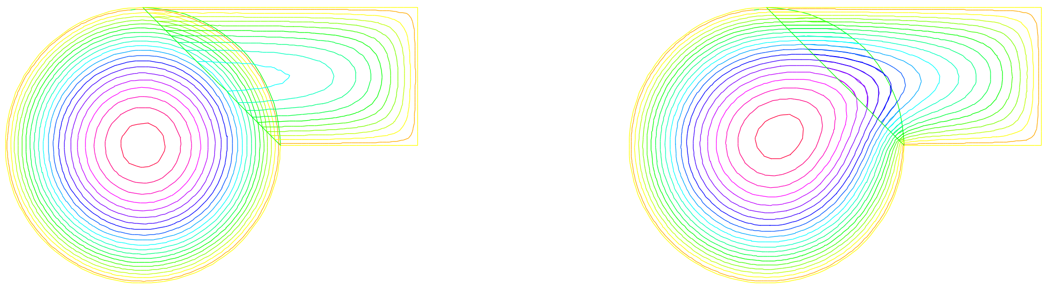


Figure 7.20: Isovalues of the solution at iteration 0 and iteration 9

7.7.2 Schwarz non Overlap Scheme

To solve

$$-\Delta u = f \text{ in } \Omega = \Omega_1 \cup \Omega_2 \quad u|_\Gamma = 0,$$

the Schwarz algorithm for domain decomposition without overlapping runs like this

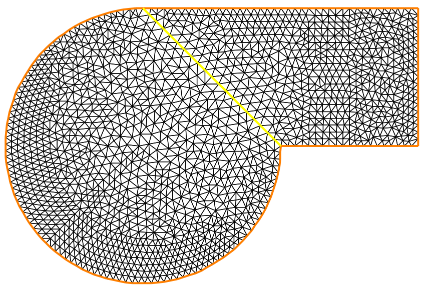


Figure 7.21: The two none overlapping mesh `TH` and `th`

Let introduce Γ_i is common the boundary of Ω_1 and Ω_2 and $\Gamma_e^i = \partial\Omega_i \setminus \Gamma_i$.
The probem find λ such that $(u_1|_{\Gamma_i} = u_2|_{\Gamma_i})$ where u_i is solution of the following Laplace problem:

$$-\Delta u_i = f \text{ in } \Omega_i \quad u_i|_{\Gamma_i} = \lambda \quad u_i|_{\Gamma_e^i} = 0$$

To solve this problem we just make a loop with upgradingλ with

$$\lambda = \lambda \pm \frac{(u_1 - u_2)}{2}$$

where the sign $+$ or $-$ of $\pm$ is choose to have convergence.

Example 47 (Schwarz-no-overlap.edp)

```
                                                    //    schwarz1 without overlapping
int inside = 2;
int outside = 1;
border a(t=1,2){x=t;y=0;label=outside;};
border b(t=0,1){x=2;y=t;label=outside;};
border c(t=2,0){x=t ;y=1;label=outside;};
border d(t=1,0){x = 1-t; y = t;label=inside;};
border e(t=0, 1){ x= 1-t; y = t;label=inside;};
border e1(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;
mesh th = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
mesh TH = buildmesh ( e(5*n) + e1(25*n) );
plot(th,TH,wait=1,ps="schwarz-no-u.eps");
fespace vh(th,P1);
fespace VH(TH,P1);
vh u=0,v; VH U,V;
vh lambda=0;
int i=0;

problem PB(U,V,init=i,solver=Cholesky) =
    int2d(TH)( dx(U)*dx(V)+dy(U)*dy(V) )
  + int2d(TH)( -V)
  + int1d(TH,inside)(-lambda*V) +    on(outside,U= 0 ) ;
problem pb(u,v,init=i,solver=Cholesky) =
    int2d(th)( dx(u)*dx(v)+dy(u)*dy(v) )
  + int2d(th)( -v)
  + int1d(th,inside)(+lambda*v) +    on(outside,u = 0 ) ;


for ( i=0 ;i< 10; i++)
{
   PB;
   pb;
   lambda = lambda - (u-U)/2;
   plot(U,u,wait=true);
};

plot(U,u,ps="schwarz-no-u.eps");
```

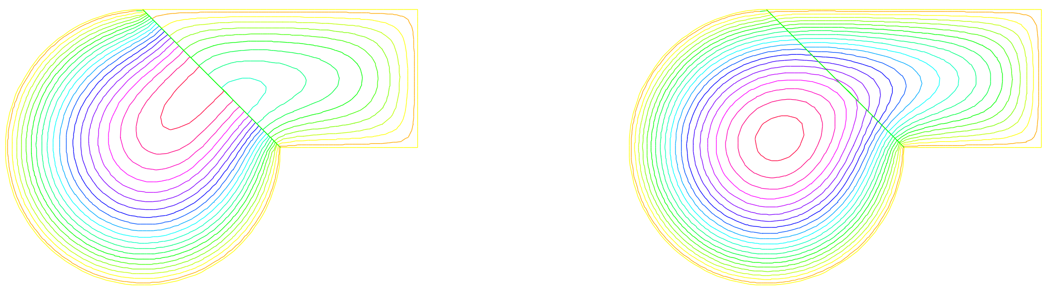


Figure 7.22: Isovalues of the solution at iteration 0 and iteration 9 without overlapping

7.7.3 Schwarz-gc.edp

To solve

$$-\Delta u = f \text{ in } \Omega = \Omega_1 \cup \Omega_2 \quad u|_\Gamma = 0,$$

the Schwarz algorithm for domain decomposition without overlapping runs like this
Let introduce Γ_i is common the boundary of Ω_1 and Ω_2 and $\Gamma_e^i = \partial\Omega_i \setminus \Gamma_i$.
The probem find λ such that $(u_1|_{\Gamma_i} = u_2|_{\Gamma_i})$ where u_i is solution of the following Laplace problem:

$$-\Delta u_i = f \text{ in } \Omega_i \quad u_i|_{\Gamma_i} = \lambda \quad u_i|_{\Gamma_e^i} = 0$$

The version of this example for Shur componant. The border problem is solve with conjugate gradient.
First, we construct the two domain

Example 48 (Schwarz-gc.edp)

```
     //    Schwarz without overlapping (Shur complenement Neumann -> Dirichet)
real cpu=clock();
int inside = 2;
int outside = 1;

border Gamma1(t=1,2){x=t;y=0;label=outside;};
border Gamma2(t=0,1){x=2;y=t;label=outside;};
border Gamma3(t=2,0){x=t ;y=1;label=outside;};

border GammaInside(t=1,0){x = 1-t; y = t;label=inside;};

border GammaArc(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;
                                                        //    build the mesh of Ω1 and Ω2
mesh Th1 = buildmesh( Gamma1(5*n) + Gamma2(5*n) + GammaInside(5*n) + Gamma3(5*n));
mesh Th2 = buildmesh ( GammaInside(-5*n) + GammaArc(25*n) );
plot(Th1,Th2);

                                                             //    defined the 2 FE space
```

```
fespace Vh1(Th1,P1),      Vh2(Th2,P1);
```

Note 26 *It is impossible to define a function just on a part of boundary, so the lambda function must be defined on the all domain Ω_1 such as*

```
Vh1 lambda=0;                                              //    take λ ∈ V_h1
```

The two Poisson problem:

```
Vh1 u1,v1;              Vh2 u2,v2;
int i=0;                                    //    for factorization optimization
problem Pb2(u2,v2,init=i,solver=Cholesky) =
    int2d(Th2)( dx(u2)*dx(v2)+dy(u2)*dy(v2) )
  + int2d(Th2)( -v2)
  + int1d(Th2,inside)(-lambda*v2) +    on(outside,u2= 0 ) ;
problem Pb1(u1,v1,init=i,solver=Cholesky) =
    int2d(Th1)( dx(u1)*dx(v1)+dy(u1)*dy(v1) )
  + int2d(Th1)( -v1)
  + int1d(Th1,inside)(+lambda*v1) +    on(outside,u1 = 0 ) ;
```

or, we define a border matrix , because the lambda function is none zero inside the domain Ω_1:

```
varf b(u2,v2,solver=CG) =int1d(Th1,inside)(u2*v2);
matrix B= b(Vh1,Vh1,solver=CG);
```

The boundary problem function,

$$\lambda \longrightarrow \int_{\Gamma_i} (u_1 - u_2) v_1$$

```
func real[int] BoundaryProblem(real[int] &l)
{
   lambda[]=l;                                 //    make FE function form l
   Pb1;     Pb2;
   i++;                                        //    no refactorization i !=0
   v1=-(u1-u2);
   lambda[]=B*v1[];
   return lambda[] ;
};
```

Note 27 *The difference between the two notations* `v1` *and* `v1[]` *is:* `v1` *is the finite element function and* `v1[]` *is the vector in the canonical basis of the finite element function* `v1` *.*

```
Vh1 p=0,q=0;
                                //    solve the problem with Conjugue Gradient
LinearCG(BoundaryProblem,p[],eps=1.e-6,nbiter=100);
                //    compute the final solution, because CG works with increment
BoundaryProblem(p[]);                       //    solve again to have right u1,u2

cout << " -- CPU time  schwarz-gc:" <<  clock()-cpu << endl;
plot(u1,u2);                                                           //    plot
```

7.8 Fluid/Structures Coupled Problem

This problem involves the Lamé system of elasticity and the Stokes system for viscous fluids with velocity $\boldsymbol{u}$ and pressure p:

$$-\Delta \boldsymbol{u} + \boldsymbol{\nabla} p = 0,\ \nabla \cdot \boldsymbol{u} = 0, \quad \text{in} \quad \Omega,\ \boldsymbol{u} = \boldsymbol{u}_\Gamma \quad \text{on} \quad \Gamma = \partial\Omega$$

where u_Γ is the velocity of the boundaries. The force that the fluid applies to the boundaries is the normal stress

$$\boldsymbol{h} = (\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T)\boldsymbol{n} - p\boldsymbol{n}$$

Elastic solids subject to forces deform: a point in the solid, originaly at (x,y) goes to (X,Y) after. When the displacement vector $\boldsymbol{v} = (v_1, v_2) = (X - x, Y - y)$ is small, Hooke's law relates the stress tensor σ inside the solid to the deformation tensor ϵ:

$$\sigma_{ij} = \lambda \delta_{ij} \nabla.\boldsymbol{v} + 2\mu\epsilon_{ij},\ \epsilon_{ij} = \frac{1}{2}(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i})$$

where δ is the Kronecker symbol and where λ, μ are two constants describing the material mechanical properties in terms of the modulus of elasticity, and Young's modulus.
The equations of elasticity are naturally written in variational form for the displacement vector $v(x) \in V$ as

$$\int_\Omega [2\mu\epsilon_{ij}(\boldsymbol{v})\epsilon_{ij}(\boldsymbol{w}) + \lambda\epsilon_{ii}(v)\epsilon_{jj}(\boldsymbol{w})] = \int_\Omega \boldsymbol{g} \cdot \boldsymbol{w} + \int_\Gamma \boldsymbol{h} \cdot \boldsymbol{w}, \forall \boldsymbol{w} \in V$$

The data are the gravity force $\boldsymbol{g}$ and the boundary stress $\boldsymbol{h}$.

Example 49 (Fluidstruct.edp) *In our example the Lamé system and the Stokes system are coupled by a common boundary on which the fluid stress creates a displacement of the boundary and hence changes the shape of the domain where the Stokes problem is integrated. The geometry is that of a vertical driven cavity with an elastic lid. The lid is a beam with weight so it will be deformed by its own weight and by the normal stress due to the fluid reaction. The cavity is the* 10×10 *square and the lid is a rectangle of height* $l = 2$.

A beam sits on a box full of fluid rotating because the left vertical side has velocity one. The beam is bent by its own weight, but the pressure of the fluid modifies the bending.
The bending displacement of the beam is given by (uu,vv) whose solution is given as follows.

```
           //    Fluid-structure interaction for a weighting beam sitting on a
                                     //    square cavity filled with a fluid.

int bottombeam = 2;                                 //    label of bottombeam
border a(t=2,0)  { x=0; y=t ;label=1;};                     //    left beam
border b(t=0,10) { x=t; y=0 ;label=bottombeam;};         //    bottom of beam
border c(t=0,2)  { x=10; y=t ;label=1;};                   //    rigth beam
border d(t=0,10) { x=10-t; y=2; label=3;};                   //    top beam
real E = 21.5;
real sigma = 0.29;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -0.05;
mesh th = buildmesh( b(20)+c(5)+d(20)+a(5));
fespace Vh(th,P1);
Vh uu,w,vv,s,fluidforce=0;
cout << "lambda,mu,gravity ="<<lambda<< " " << mu << " " << gravity << endl;
                                                   //    deformation of a beam under its own weight
solve  bb([uu,vv],[w,s])  =
    int2d(th)(
                  2*mu*(dx(uu)*dx(w)+ ((dx(vv)+dy(uu))*(dx(s)+dy(w)))/4 )
                + lambda*(dx(uu)+dy(vv))*(dx(w)+dy(s))/2
              )
  + int2d(th) (-gravity*s)
  + on(1,uu=0,vv=0)
  + fluidforce[];
 ;

 plot([uu,vv],wait=1);
 mesh th1 = movemesh(th, [x+uu, y+vv]);
 plot(th1,wait=1);
```

Then Stokes equation for fluids ast low speed are solved in the box below the beam, but the beam has deformed the box (see border h):

```
                    //   Stokes on square b,e,f,g driven cavite on left side g
border e(t=0,10) { x=t; y=-10; label= 1; };                       //    bottom
border f(t=0,10) { x=10; y=-10+t ; label= 1; };                    //    right
border g(t=0,10) { x=0; y=-t ;label= 2;};                           //    left
border h(t=0,10) { x=t; y=vv(t,0)*( t>=0.001 )*(t <= 9.999);
                    label=3;};                          //    top of cavity deforme

mesh sh = buildmesh(h(-20)+f(10)+e(10)+g(10));
plot(sh,wait=1);
```

We use the Uzawa conjugate gradient to solve the Stokes problem like in example Section 7.6.2

```
fespace Xh(sh,P2),Mh(sh,P1);
Xh u1,u2,v1,v2;
Mh p,q,ppp;


varf bx(u1,q) = int2d(sh)( -(dx(u1)*q));

varf by(u1,q) = int2d(sh)( -(dy(u1)*q));

varf Lap(u1,u2)= int2d(sh)(  dx(u1)*dx(u2) + dy(u1)*dy(u2) )
                    +  on(2,u1=1) +  on(1,3,u1=0)  ;

Xh bc1; bc1[] = Lap(0,Xh);
Xh brhs;

matrix A= Lap(Xh,Xh,solver=CG);
matrix Bx= bx(Xh,Mh);
matrix By= by(Xh,Mh);
Xh bcx=0,bcy=1;
```

```
func real[int] divup(real[int] & pp)
{
  int verb=verbosity;
   verbosity=0;
   brhs[]  = Bx'*pp; brhs[] += bc1[] .*bcx[];
   u1[] = A^-1*brhs[];
   brhs[]  = By'*pp; brhs[] += bc1[] .*bcy[];
   u2[] = A^-1*brhs[];
   ppp[] =   Bx*u1[];
   ppp[] +=  By*u2[];
   verbosity=verb;
   return ppp[] ;
};

 p=0;q=0;u1=0;v1=0;

 LinearCG(divup,p[],eps=1.e-3,nbiter=50);
 divup(p[]);
```

Now the beam will feel the stress constraint from the fluid:

```
  Vh sigma11,sigma22,sigma12;
  Vh uu1=uu,vv1=vv;

  sigma11([x+uu,y+vv]) = (2*dx(u1)-p);
  sigma22([x+uu,y+vv]) = (2*dy(u2)-p);
  sigma12([x+uu,y+vv]) = (dx(u1)+dy(u2));
```

which comes as a boundary condition to the PDE of the beam:

```
varf fluidf([uu,vv],[w,s]) fluidforce =
solve  bbst([uu,vv],[w,s],init=i)  =
    int2d(th)(
                  2*mu*(dx(uu)*dx(w)+ ((dx(vv)+dy(uu))*(dx(s)+dy(w)))/4 )
                + lambda*(dx(uu)+dy(vv))*(dx(w)+dy(s))/2
             )
  + int2d(th) (-gravity*s)
  + int1d(th,bottombeam)( -coef*(   sigma11*N.x*w + sigma22*N.y*s
                                  + sigma12*(N.y*w+N.x*s) )  )
  + on(1,uu=0,vv=0);
 plot([uu,vv],wait=1);
 real  err = sqrt(int2d(th)( (uu-uu1)^2 + (vv-vv1)^2 ));
 cout <<  " Erreur L2 = " << err << "----------\n";
```

Notice that the matrix generated by bbst is reused (see `init=i`*). Finally we deform the beam*

```
 th1 = movemesh(th, [x+0.2*uu, y+0.2*vv]);
 plot(th1,wait=1);
```

7.9 Transmission Problem

Consider an elastic plate whose displacement change vertically, which is made up of three plates of different materials, welded on each other. Let Ω_i, $i = 1, 2, 3$ be the domain occupied by i-th material with tension μ_i (see Section 7.1.1). The computational domain Ω is the interior of $\overline{\Omega_1} \cup \overline{\Omega_2} \cup \overline{\Omega_3}$. The vertical displacement $u(x, y)$ is obtained from

$$-\mu_i \Delta u = f \text{ in } \Omega_i \tag{7.53}$$
$$\mu_i \partial_n u|_{\Gamma_i} = -\mu_j \partial_n u|_{\Gamma_j} \quad \text{on } \overline{\Omega_i} \cap \overline{\Omega_j} \qquad \text{if } 1 \le i < j \le 3 \tag{7.54}$$

where $\partial_n u|_{\Gamma_i}$ denotes the value of the normal derivative $\partial_n u$ on the boundary Γ_i of the domain Ω_i.
By introducing the characteristic function χ_i of Ω_i, that is,

$$\chi_i(x) = 1 \quad \text{if } x \in \Omega_i; \qquad \chi_i(x) = 0 \quad \text{if } x \notin \Omega_i \tag{7.55}$$

we can easily rewrite (7.53) and (7.54) to the weak form. Here we assume that $u = 0$ on $\Gamma = \partial\Omega$.
problem Transmission: For a given function f, find u such that

$$a(u, v) = \ell(f, v) \quad \text{for all } v \in H_0^1(\Omega) \tag{7.56}$$
$$a(u, v) = \int_\Omega \mu \nabla u \cdot \nabla v, \quad \ell(f, v) = \int_\Omega f v$$

where $\mu = \mu_1\chi_1 + \mu_2\chi_2 + \mu_3\chi_3$. Here we notice that μ become the discontinuous function.
With dissipation, and at the thermal equilibrium, the temperature equation is:
This example explains the definition and manipulation of *region*, i.e. subdomains of the whole domain.
Consider this L-shaped domain with 3 diagonals as internal boundaries, defining 4 subdomains:

```
                                            //    example using region keyword
                                 //    construct a mesh with 4 regions (sub-domains)
border a(t=0,1){x=t;y=0;};
border b(t=0,0.5){x=1;y=t;};
border c(t=0,0.5){x=1-t;y=0.5;};
border d(t=0.5,1){x=0.5;y=t;};
border e(t=0.5,1){x=1-t;y=1;};
border f(t=0,1){x=0;y=1-t;};
                                                         //    internal boundary
border i1(t=0,0.5){x=t;y=1-t;};
border i2(t=0,0.5){x=t;y=t;};
border i3(t=0,0.5){x=1-t;y=t;};

mesh th = buildmesh (a(6) + b(4) + c(4) +d(4) + e(4) +
    f(6)+i1(6)+i2(6)+i3(6));
fespace Ph(th,P0);             //    constant discontinuous functions / element
fespace Vh(th,P1);                       //    P1 ontinuous functions / element

Ph reg=region;           //    defined the P0 function associed to region number
plot(reg,fill=1,wait=1,value=1);
```

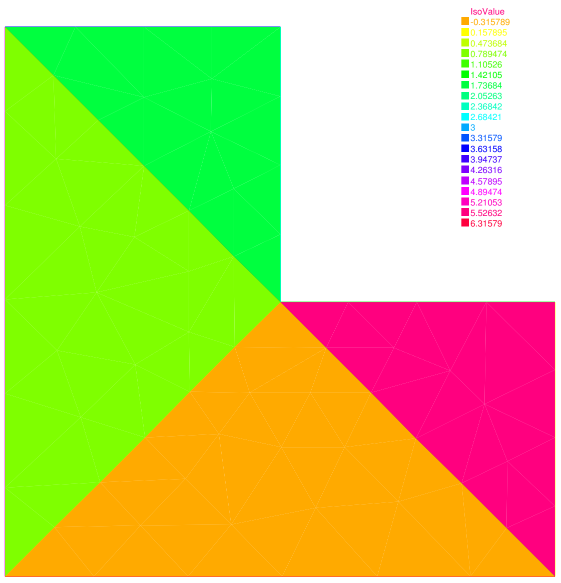


Figure 7.23: the function `reg`

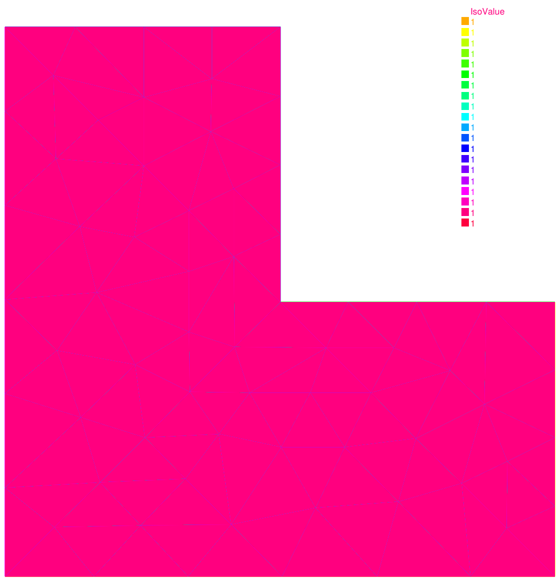


Figure 7.24: the function `nu`

`region` is a keyword of freefem++ which is in fact a variable depending of the current position (is not a function today, use `Ph reg=region;` to set a function). This variable value returned is the number of the subdomain of the current position. This number is defined by "buildmesh" which scans while building the mesh all its connected component. So to get the number of a region containing a particular point one does:

```
int nupper=reg(0.4,0.9);         //    get the region number of point (0.4,0.9)
int nlower=reg(0.9,0.1);         //    get the region number of point (0.4,0.1)
cout << " nlower " <<  nlower << ", nupper = " << nupper<< endl;
         //    defined the characteristics functions of upper and lower region
Ph nu=1+5*(region==nlower) + 10*(region==nupper);
plot(nu,fill=1,wait=1);
```

This is particularly useful to define discontinuous functions such as might occur when one part of the domain is copper and the other one is iron, for example.
We this in mind we proceed to solve a Laplace equation with discontinuous coefficients (ν is 1, 6 and 11 below).

```
Ph nu=1+5*(region==nlower) + 10*(region==nupper);
plot(nu,fill=1,wait=1);
problem lap(u,v) =   int2d(th)( nu*( dx(u)*dx(v)*dy(u)*dy(v) )) + int2d(-1*v) +
on(a,b,c,d,e,f,u=0);
plot(u);
```

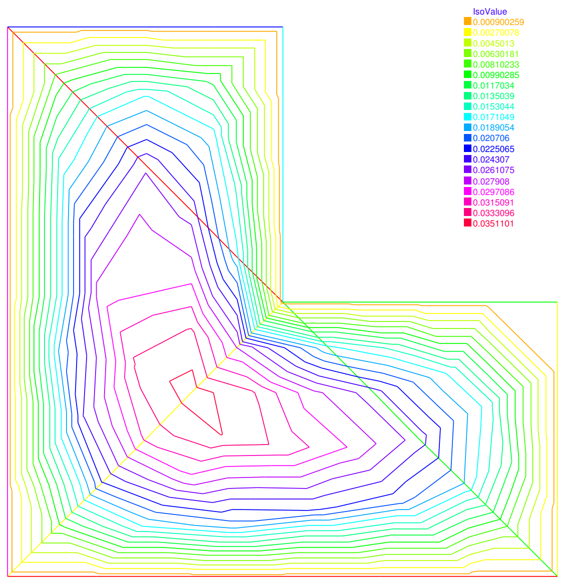


Figure 7.25: the isovalue of the solution u

7.10 Free Boundary Problem

The domain Ω is defined with:

```
real L=10;                                        //    longueur du domaine
real h=2.1;                                    //    hauteur du bord gauche
real h1=0.35;                                  //    hauteur du bord droite

                                                 //    maillage d'un tapeze
border a(t=0,L){x=t;y=0;};                  //    bottom:  Γ_a
border b(t=0,h1){x=L;y=t;};                  //    right:  Γ_b
border f(t=L,0){x=t;y=t*(h1-h)/L+h;};       //    free surface:  Γ_f
border d(t=h,0){x=0;y=t;};                   //    left:  Γ_d

int n=4;
mesh Th=buildmesh (a(10*n)+b(6*n)+f(8*n)+d(3*n));
plot(Th,ps="dTh.eps");
```

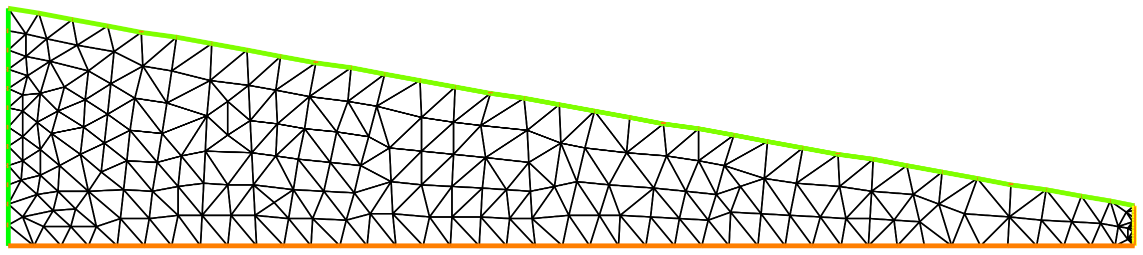


Figure 7.26: The mesh of the domain Ω

The free boundary problem is:
Find u and Ω such that:

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega \\ u = y & \text{on } \Gamma_b \\ \dfrac{\partial u}{\partial n} = 0 & \text{on } \Gamma_d \cup \Gamma_a \\ \dfrac{\partial u}{\partial n} = \dfrac{q}{K} n_x \text{ and } \; u = y & \text{on } \Gamma_f \end{cases}$$

We use a fixed point method; $\Omega^0 = \Omega$
in two step, fist we solve the classical following problem:

$$\begin{cases} -\Delta u & = 0 & \text{in } \Omega^n \\ u & = y & \text{on } \Gamma_b^n \\ \dfrac{\partial u}{\partial n} & = 0 & \text{on } \Gamma_d^n \cup \Gamma_a^n \\ u & = y & \text{on } \Gamma_f^n \end{cases}$$

The varitional formulation is:
find u on $V = H^1(\Omega^n)$, such than $u = y$ on Γ_b^n and Γ_f^n

$$\int_{\Omega^n} \nabla u \nabla u' = 0, \quad \forall u' \in V \text{ with } u' = 0 \text{ on } \Gamma_b^n \cup \Gamma_f^n$$

and secondly to construte a domain deformation $\mathcal{F}(x, y) = [x, y - v(x, y)]$
where v is solution of the following problem:

$$\begin{cases} -\Delta v & = 0 & \text{in } \Omega^n \\ v & = 0 & \text{on } \Gamma_a^n \\ \dfrac{\partial v}{\partial n} & = 0 & \text{on } \Gamma_b^n \cup \Gamma_d^n \\ \dfrac{\partial v}{\partial n} & = \dfrac{\partial u}{\partial n} - \dfrac{q}{K} n_x & \text{on } \Gamma_f^n \end{cases}$$

The varitional formulation is:
find v on V, such than $v = 0$ on Γ_a^n

$$\int_{\Omega^n} \nabla v \nabla v' = \int_{\Gamma_f^n} (\frac{\partial u}{\partial n} - \frac{q}{K} n_x) v', \quad \forall v' \in V \text{ with } v' = 0 \text{ on } \Gamma_a^n$$

finaly the new domain $\Omega^{n+1} = \mathcal{F}(\Omega^n)$

Example 50 (freeboundary.edp) *The* `FreeFem++` *:implementation is:*

```
real q=0.02;                                                    //   flux entrant
real K=0.5;                                                     //   permeabilité

fespace Vh(Th,P1);
int j=0;

Vh u,v,uu,vv;

problem Pu(u,uu,solver=CG) = int2d(Th)( dx(u)*dx(uu)+dy(u)*dy(uu))
  + on(b,f,u=y) ;
```

```
problem Pv(v,vv,solver=CG) = int2d(Th)( dx(v)*dx(vv)+dy(v)*dy(vv))
  +  on (a, v=0) + int1d(Th,f)(vv*((q/K)*N.y- (dx(u)*N.x+dy(u)*N.y)));


real errv=1;
real erradap=0.001;
verbosity=1;
while(errv>1e-6)
{
  j++;
  Pu;
  Pv;
  plot(Th,u,v ,wait=0);
  errv=int1d(Th,f)(v*v);
   real coef=1;

                                                                    //
  real mintcc = checkmovemesh(Th,[x,y])/5.;
  real mint = checkmovemesh(Th,[x,y-v*coef]);

  if (mint<mintcc ||  j%10==0) {                       //    mesh to bad => remeshing
    Th=adaptmesh(Th,u,err=erradap ) ;
    mintcc = checkmovemesh(Th,[x,y])/5.;
  }

  while (1)
  {
    real mint = checkmovemesh(Th,[x,y-v*coef]);

    if (mint>mintcc) break;

    cout << " min |T]  " << mint << endl;
    coef /= 1.5;
  }

  Th=movemesh(Th,[x,y-coef*v]);                        //    calcul de la deformation
  cout << "\n\n"<<j <<"------------ errv = " << errv << "\n\n";

}
plot(Th,ps="d_Thf.eps");
plot(u,wait=1,ps="d_u.eps");
```

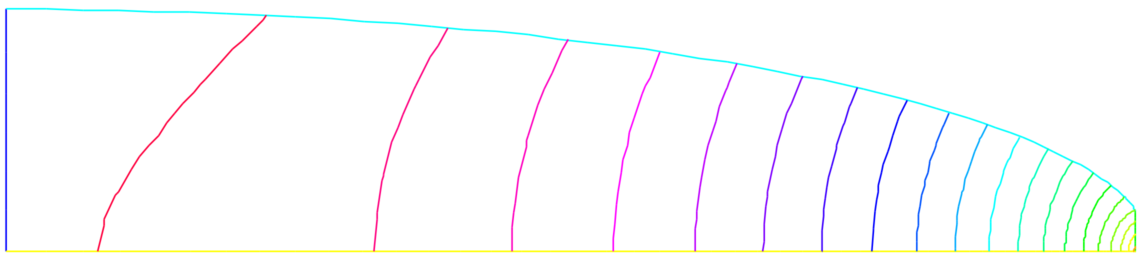


Figure 7.27: The final solution on the new domain Ω^{72}

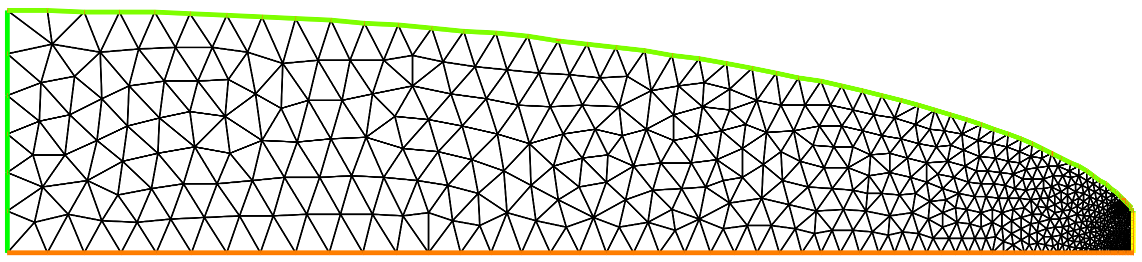


Figure 7.28: The adapted mesh of the domain Ω^{72}

7.11 nolinear-elas.edp

The nonlinear elasticity problem is find the deplacement (u_1, u_2) minimizing J

$$\min J(u_1, u_2) = \int_\Omega f(F2) - \int_{\Gamma_p} P_a\, u_2$$

where $F2(u_1, u_2) = A(E[u_1, u_2], E[u_1, u_2])$ and $A(X, Y)$ is bilinear sym. positive form with respect two matrix X, Y. where f is a given $\mathcal{C}^2$ function, and $E[u_1, u_2] = (E_{ij})_{i=1,2,\, j=1,2}$ is the Green-Saint Venant deformation tensor defined with:

$$E_{ij} = 0.5(\partial_i u_j + \partial_j u_i) + \sum_k \partial_i u_k \times \partial_j u_k$$

The differential of J is

$$DJ(u_1, u_2)(v_1, v_2) = \int 2A(E[u_1, u_2], DE[u_1, u_2](v_1, v_2))f'(F2(u_1, u_2))) - \int_{\Gamma_p} P_a u_2$$

denote $\mathbf{u} = u_1, u_2$, $\mathbf{v} = v_1, v_2$, $\mathbf{w} = (w_1, w_2)$ and the second order differential is

$$\begin{aligned} D^2J(\mathbf{u})((\mathbf{v}), (\mathbf{w})) &= A(E[\mathbf{u}], DE[\mathbf{u}](\mathbf{v}))A(E[\mathbf{u}], DE[\mathbf{u}](\mathbf{w}))f''(F2(\mathbf{u}))) \\ &+ A(DE[\mathbf{u}](\mathbf{v}), DE[\mathbf{u}](\mathbf{w}))f'(F2(\mathbf{u}))) \\ &+ A(DE[\mathbf{u}], D^2E[\mathbf{u}]((\mathbf{v}), (\mathbf{w})))f'(F2(\mathbf{u}))) \end{aligned}$$

where DE and D^2E are the first and second differential of E.

The Newton Method is
choose $n = 0$,and u_O, v_O the initial displacement

- loop:
- find (du, dv) : solution of

$$D^2J(u_n, v_n)((w, s), (du, dv)) = DJ(u_n, v_n)(w, s), \quad \forall w, s$$

- $un = un - du, \quad vn = vn - dv$
- until (du, dv) small is enough

The way to implement this algorithme in `freefem++` is use a macro tool to implement A and $F2$, f, f',f''.
A macro is like is `ccp` preprocessor of `C++` , but this begin by `macro` and the end of the macro definition is the begin of the comment //. In this case the macro is very useful because the type of parameter can be change. And it is easy to make automatic differentiation.

```
//    non linear elasticity model
//
//    ------------------------------
//    with huge utilisation of macro
//    --------------------------
//    optimize version
//    ------------
//    problem is find $(uu, vn)$ minimizing $J$
//    $minJ(un, vn) = intf(F2) - intPa * un$
//    $dJ(u, u, uu, vv) = intdF2(u, v, uu, vv)df(F2(u, v))$
//    where $F2 = ({}^tEAE)$ ,
//    $E(U) = 1/2(\nabla U + \nabla U^t + \nabla U^t\nabla U)$
//    ($u_1$)
//    with U=( )
//    ($u_2$)
//    so:
                                                                           //
```

(1) $$E_{ij} = 0.5(d_iu_j + d_ju_i) + \sum_k d_iu_k * d_j * u_k$$

```
//    the 3 componantes of the Green Saint Venant deformation tensor:
//    $E1(u1, u2) = E_{11}$
//    $E2(u1, u2) = E_{12} = E_21$
//    $E3(u1, u2) = E_{22}$



//    remark :  we can parametrize E1,E2,E3 with:
//    EE(da,db,a,b,u1,u2)
//    where $da, db$ correspond to $d_i, d_j$ in (1)
//    where $a, b$ correspond to $u_i, u_j$ in (1)
//    where $u1, u2$ correspond to $u_1, u_2$ in (1)
                          //    ---------------------------------------------

//    first the linear part of EE linear elasticite
//    remark a macro end with a // comment
macro EEL(di,dj,ui,uj) ( (di(uj)+dj(ui))*0.5 )                    //    11

//    non linear par of EE (bilinear) simple to differential
macro bEENL(di,dj,u1,u2,v1,v2) (di(u1)*dj(v1)*.5+di(u2)*dj(v2)*0.5)
                                                                      //
macro EENL(di,dj,u1,u2) bEENL(di,dj,u1,u2,u1,u2)                      //
macro dEENL(di,dj,u1,u2,du1,du2) ( bEENL(di,dj,du1,du2,u1,u2)
                                 + bEENL(di,dj,u1,u2,du1,du2) )
//    ------------
macro EE(di,dj,ui,uj,u1,u2) (EEL(di,dj,u1,uj) + EENL(di,dj,u1,u2))    //
macro dEE(di,dj,dui,duj,u1,u2,du1,du2) (EEL(di,dj,du1,duj)
                                    + dEENL(di,dj,u1,u2,du1,du2))    //
macro ddEE(di,dj,du1,du2,ddu1,ddu2) ( dEENL(di,dj,du1,du2,ddu1,ddu2))
```

```
                                                                          //
//    remark :
//    dEE(di,dj,dui,duj,u1,u2,du1,du2) is "the formal differential of EE"
//    where du1 = δu1 ,du2 = δu2
//    ddEE(di,dj,dui,duj,u1,u2,du1,du2) is "the formal differential of dEE"
//    where ddu1 = δ^2u1 ,ddu2 = δ^2u2
//    ---

//    the macro corresponding to the 3 componante of E
macro E1(u,v)  /*E11*/EE(dx,dx,u,u,u,v)                                  //
macro E2(u,v)  /*E12*/EE(dx,dy,u,v,u,v)                                  //
macro E3(u,v)  /*E22*/EE(dy,dy,v,v,u,v)                                  //

macro dE1(u,v,uu,vv) /*dE11*/dEE(dx,dx,uu,uu,u,v,uu,vv)                  //
macro dE2(u,v,uu,vv) /*dE12*/dEE(dx,dy,uu,vv,u,v,uu,vv)                  //
macro dE3(u,v,uu,vv) /*dE22*/dEE(dy,dy,vv,vv,u,v,uu,vv)                  //
macro ddE1(u,v,uu,vv,uuu,vvv) /*ddE11*/ddEE(dx,dx,uu,vv,uuu,vvv)         //
macro ddE2(u,v,uu,vv,uuu,vvv) /*ddE12*/ddEE(dx,dy,uu,vv,uuu,vvv)         //
macro ddE3(u,v,uu,vv,uuu,vvv) /*ddE22*/ddEE(dy,dy,uu,vv,uuu,vvv)
                                                                          //
//    a formal bilinear term
macro PP(A,B,u,v) (A(u,v)*B(u,v))
                                                                          //
//    a formal diff bilinear term
macro dPP(A,B,dA,dB,u,v,uu,vv) (dA(u,v,uu,vv)*B(u,v) + A(u,v)*dB(u,v,uu,vv))
                                                                          //
//    a formal diff^2 bilinear term
macro ddPP(A,B,dA,dB,ddA,ddB,u,v,uu,vv,uuu,vvv) (
  dA(u,v,uu,vv)*dB(u,v,uuu,vvv) + dA(u,v,uuu,vvv)*dB(u,v,uu,vv)
  +  ddA(u,v,uu,vv,uuu,vvv)*B(u,v) + A(u,v)*ddB(u,v,uu,vv,uuu,vvv)
  )                                                                       //
//    so the matrix A is 6 coef
                                                                          //
//    a11 a12 a13
//    a12 a22 a23
//    a13 a23 a33
macro F2(u,v)  /* F2 */  (
     a11*PP(E1,E1,u,v)
  +  a22*PP(E2,E2,u,v)
  +  a33*PP(E3,E3,u,v)
  +  a13*PP(E1,E3,u,v)
  +  a13*PP(E3,E1,u,v)
  +  a12*PP(E1,E2,u,v)
  +  a12*PP(E2,E1,u,v)
  +  a23*PP(E2,E3,u,v)
  +  a23*PP(E3,E2,u,v)
)                                                             //    end macro F2

macro dF2(u,v,uu,vv)  /* dF2 */  (
       a11*dPP(E1,E1,dE1,dE1,u,v,uu,vv)
     + a12*dPP(E1,E2,dE1,dE2,u,v,uu,vv)
     + a13*dPP(E1,E3,dE1,dE3,u,v,uu,vv)
     + a21*dPP(E2,E1,dE2,dE1,u,v,uu,vv)
     + a22*dPP(E2,E2,dE2,dE2,u,v,uu,vv)
     + a23*dPP(E2,E3,dE2,dE3,u,v,uu,vv)
     + a31*dPP(E3,E1,dE3,dE1,u,v,uu,vv)
     + a32*dPP(E3,E2,dE3,dE2,u,v,uu,vv)
     + a33*dPP(E3,E3,dE3,dE3,u,v,uu,vv)
)                                                      //    end macro dF2 (DF2)

macro ddF2(u,v,uu,vv,uuu,vvv)  /* ddF2 */  (
       a11*ddPP(E1,E1,dE1,dE1,ddE1,ddE1,u,v,uu,vv,uuu,vvv)
     + a12*ddPP(E1,E2,dE1,dE2,ddE1,ddE2,u,v,uu,vv,uuu,vvv)
     + a13*ddPP(E1,E3,dE1,dE3,ddE1,ddE3,u,v,uu,vv,uuu,vvv)
     + a21*ddPP(E2,E1,dE2,dE1,ddE2,ddE1,u,v,uu,vv,uuu,vvv)
     + a22*ddPP(E2,E2,dE2,dE2,ddE2,ddE2,u,v,uu,vv,uuu,vvv)
     + a23*ddPP(E2,E3,dE2,dE3,ddE2,ddE3,u,v,uu,vv,uuu,vvv)
     + a31*ddPP(E3,E1,dE3,dE1,ddE3,ddE1,u,v,uu,vv,uuu,vvv)
     + a32*ddPP(E3,E2,dE3,dE2,ddE3,ddE2,u,v,uu,vv,uuu,vvv)
     + a33*ddPP(E3,E3,dE3,dE3,ddE3,ddE3,u,v,uu,vv,uuu,vvv)
)                                                    //    end macro ddF2 (D^2F2)

//    differential of J:

//    for hyper elasticity problem
//    ------------------------------

macro f(u) (u)                                                //    end of macro
macro df(u) (1)                                         //    end of macro df = f'
macro ddf(u) (0)                                      //    end of macro ddf = f''

//    -- du caouchouc --- CF cours de Herve Le Dret.
//    -------------------------------
real mu = 0.012e5;                                                  //    kg/cm^2
real lambda =  0.4e5;                                               //    kg/cm^2
//
//    σ = 2μE + λtr(E)Id
//
//    ( a b )
//    ( b c )
//
//    tr*Id :  (a,b,c) -> (a+c,0,a+c)
//    so the associed matrix is:
//    ( 1 0 1 )
//    ( 0 0 0 )
//    ( 1 0 1 )
//    ------------------ the coef
real a11= 2*mu +  lambda  ;
real a22= 2*mu ;
real a33= 2*mu +   lambda ;
real a12= 0 ;
real a13= lambda ;
real a23= 0 ;
                                                            //    symetric part
real a21= a12 ;
real a31= a13 ;
real a32= a23 ;
real Pa=1e2;                                          //    a pressure of 100 Pa
                                                          //    ----------------

int n=30,m=10;
mesh Th= square(n,m,[x,.3*y]);  //    label:  1 bottom, 2 right, 3 up, 4 left;
```

```
int bottom=1, right=2,upper=3,left=4;

plot(Th);

fespace Wh(Th,P1dc);
fespace Vh(Th,[P1,P1]);
fespace Sh(Th,P1);

Wh e2,fe2,dfe2,ddfe2;                                                //    optimisation
Wh ett,ezz,err,erz;                                                  //    optimisation

Vh [uu,vv], [w,s],[un,vn];
[un,vn]=[0,0];                                                      //    intialisation
[uu,vv]=[0,0];

varf vmass([uu,vv],[w,s],solver=CG) =  int2d(Th)( uu*w + vv*s );
matrix M=vmass(Vh,Vh);

problem NonLin([uu,vv],[w,s],solver=LU)=
 int2d(Th,qforder=1)(                                               //    (D^2J(un)) part
              ddF2(un,vn,uu,vv,w,s)* dfe2
      + dF2(un,vn,uu,vv)*dF2(un,vn,w,s)*ddfe2
       )
   -int2d(Th,<1)(                                                    //    (DJ(un)) part
          dF2(un,vn,w,s) * dfe2  )
   - int1d(Th,3)(Pa*s)
   + on(right,left,uu=0,vv=0);
;
                                                                  //    Newton's method
                                                                  //    ---------------
Sh u1,v1;
for (int i=0;i<10;i++)
{
  cout << "Loop " << i << endl;
  e2 = F2(un,vn);
  dfe2 = df(e2) ;
  ddfe2 = ddf(e2);
  cout << "  e2 max " <<e2[].max << " , min" << e2[].min << endl;
  cout << " de2 max "<< dfe2[].max << " , min" << dfe2[].min << endl;
  cout << "dde2 max "<< ddfe2[].max << " , min" << ddfe2[].min << endl;
  NonLin;                                    //    compute [uu,vv] = (D^2J(un))^{-1}(DJ(un))

  w[]   = M*uu[];
  real res = sqrt(w[]' * uu[]);                                   //    norme L^2of[uu,vv]
  u1 = uu;
  v1 = vv;
  cout << " L^2 residual = " << res << endl;
  cout << " u1 min =" <<u1[].min << ", u1 max= " << u1[].max << endl;
  cout << " v1 min =" <<v1[].min << ", v2 max= " << v1[].max << endl;
  plot([uu,vv],wait=1,cmm=" uu, vv " );
  un[] -= uu[];
  plot([un,vn],wait=1,cmm=" deplacement " );
  if (res<1e-5) break;
}

plot([un,vn],wait=1);
```

```
mesh th1 = movemesh(Th, [x+un, y+vn]);
plot(th1,wait=1);                                                    //    see figure 7.29
```

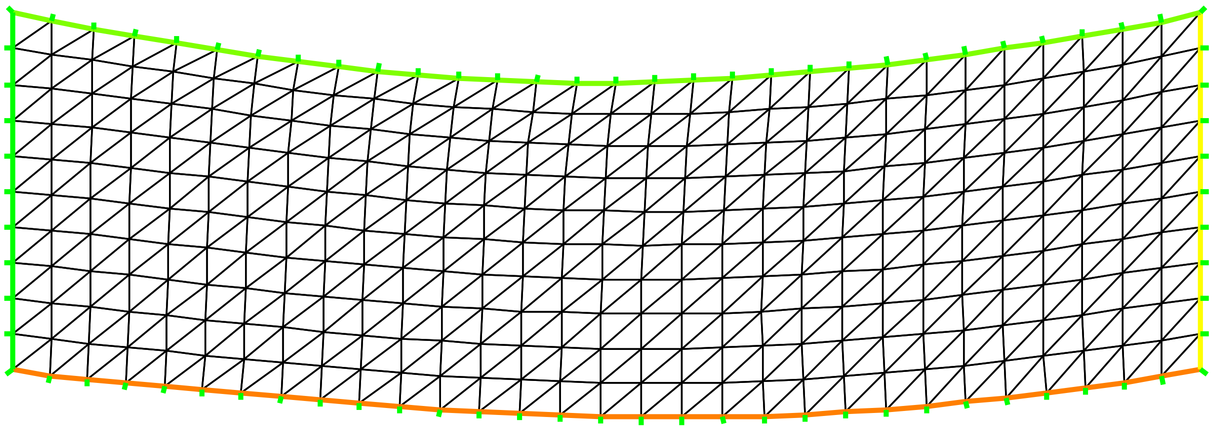


Figure 7.29: The deformated domain

Chapter 8

Parallel version experimental

A first test of parallisation of `FreeFem++` is make under mpi. We add three word in the language:

`mpisize` The total number of processes

`mpirank` the number of my current process in $\{0, ..., mpisize - 1\}$.

`processor` a function to set the possessor to send or receive data

`broadcast` a function to broadcast from a processor to all other a data

```
    processor(10) << a ;              //    send to the process 10 the data a;
    processor(10) >> a ;         //    receive from the process 10 the data a;
```

8.1 Schwarz in parallel

If example is just the rewritting of example `schwarz-overlap` in section 7.7.1.
How to use

```
[examples++-mpi] hecht%lamboot

LAM 6.5.9/MPI 2 C++/ROMIO - Indiana University


[examples++-mpi] hecht% mpirun -np 2 FreeFem++-mpi schwarz-c.edp
```

```
//    a new coding verion c, methode de schwarz in parallele
//    with 2 proc.
//    -------------------------------
//    F.Hecht december 2003
//    ----------------------------------
//    to test the broadcast instruction
//    and array of mesh
//    add add the stop test
//    ---------------------------------
```

```
if ( mpisize != 2 ) {
  cout << " sorry number of processeur !=2 " << endl;
  exit(1);}
verbosity=3;
real pi=4*atan(1);
int inside = 2;
int outside = 1;
border a(t=1,2){x=t;y=0;label=outside;};
border b(t=0,1){x=2;y=t;label=outside;};
border c(t=2,0){x=t ;y=1;label=outside;};
border d(t=1,0){x = 1-t; y = t;label=inside;};
border e(t=0, pi/2){ x= cos(t); y = sin(t);label=inside;};
border e1(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;
mesh[int]  Th(mpisize);
if (mpirank == 0)
 Th[0] = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
else
 Th[1] = buildmesh ( e(5*n) + e1(25*n) );

broadcast(processor(0),Th[0]);
broadcast(processor(1),Th[1]);

fespace Vh(Th[mpirank],P1);
fespace Vhother(Th[1-mpirank],P1);

Vh u=0,v;
Vhother U=0;
int i=0;

problem pb(u,v,init=i,solver=Cholesky) =
    int2d(Th[mpirank])( dx(u)*dx(v)+dy(u)*dy(v) )
  - int2d(Th[mpirank])( v)
  + on(inside,u = U)  +  on(outside,u= U ) ;

for ( i=0 ;i< 20; i++)
{
  cout << mpirank << " looP " << i << endl;
   pb;
                                  //    send u to the other proc, receive in U
   processor(1-mpirank) << u[];   processor(1-mpirank) >> U[];
   real err0,err1;
   err0 = int1d(Th[mpirank],inside)(square(U-u)) ;
                            //    send err0 to the other proc, receive in err1
   processor(1-mpirank)<<err0;   processor(1-mpirank)>>err1;
   real err= sqrt(err0+err1);
   cout <<" err = " << err << " err0 = " << err0 << ", err1 = " << err1 << endl;
   if(err<1e-3) break;
};
if (mpirank==0)
    plot(u,U,ps="uU.eps");
```

Chapter 9

Graphical User Interface

There are two different graphical user interfaces available for `freefem++` :

- `FreeFem++-cs` is part of the standard `freefem++` package. It runs on Linux, MacOS X and Windows.
- `FFedit` is available as a separate package. It runs on Linux and MacOS X and requires tcl/tk to be installed.

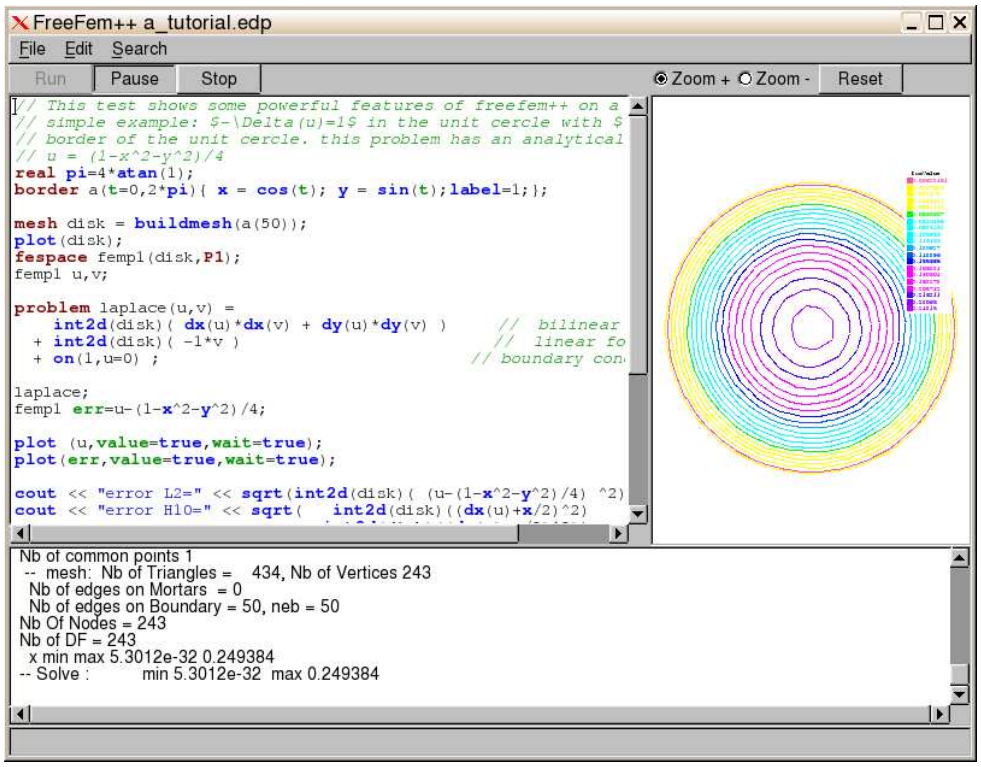
9.1 FreeFem++-cs

"-cs" stands for "client/server". The executable program named `FreeFem++-cs` is the client. It automatically starts a server program named `FreeFem++-cs-server` every time the user asks for a script to be run. To run `FreeFem++-cs`, just type its name in, optionally followed by a script name. It will open a window corresponding to fig. 9.1.

The main characteristics of `FreeFem++-cs` are :

- A main window composed of three panels : editor with syntax highlighting (right), `freefem++` messages (bottom) and graphics (left).
- Panel sizes can be changed by dragging their borders with the mouse. Any of the three panels can be made to fill the whole window.
- The edited script can be run at any time by clicking on the "Run" button.
- Dragging a `freefem++` script file icon from a file manager into the editor window makes `FreeFem++-cs` edit that script.
- Graphics can be examined (e.g. zoomed) while `freefem++` is running.
- A running `freefem++` computation can be paused or stopped at any time.

All commands should be self-explanatory. Here are just a few useful hints :



```
// This test shows some powerful features of freefem++ on a
// simple example: $-\Delta(u)=1$ in the unit cercle with $
// border of the unit cercle. this problem has an analytical
// u = (1-x^2-y^2)/4
real pi=4*atan(1);
border a(t=0,2*pi){ x = cos(t); y = sin(t);label=1;};

mesh disk = buildmesh(a(50));
plot(disk);
fespace femp1(disk,P1);
femp1 u,v;

problem laplace(u,v) =
    int2d(disk)( dx(u)*dx(v) + dy(u)*dy(v) )      //  bilinear
  + int2d(disk)( -1*v )                           //  linear fo
  + on(1,u=0) ;                                  // boundary con

laplace;
femp1 err=u-(1-x^2-y^2)/4;

plot (u,value=true,wait=true);
plot(err,value=true,wait=true);

cout << "error L2=" << sqrt(int2d(disk)( (u-(1-x^2-y^2)/4) ^2)
cout << "error H10=" << sqrt(   int2d(disk)((dx(u)+x/2)^2)
```

```
Nb of common points 1
 -- mesh:  Nb of Triangles =    434, Nb of Vertices 243
  Nb of edges on Mortars  = 0
  Nb of edges on Boundary = 50, neb = 50
Nb Of Nodes = 243
Nb of DF = 243
  x min max 5.3012e-32 0.249384
-- Solve :          min 5.3012e-32  max 0.249384
```

Figure 9.1: `FreeFem++-cs` main window

- There is no need to save a `freefem++` script to run it. It is run exactly as displayed in the editor window, and the corresponding file is not touched.
- The current directory is updated every time a script is loaded or saved. All `include` directives are therefore relative to the directory where the main script is located.
- Specifying `wait=1` in a `plot` command is exactly equivalent to clicking on the "Pause" button when the plot is displayed.
- In zoom mode, if your mouse has more than one button, a middle-click resets any zooming coefficient, and a right-click zooms in the opposite way of the left-click.

9.2 FFedit

FFedit runs on Linux and MacOSX.

9.2.1 Installation of Tcl and Tk

You have to install tcl8.4.6 and tk8.4.6 for the GUI to work.
First go to http:
//www.tcl.tk./software/tcltk/downloadnow84.tml
and download :
tcl8.4.6-src.tar.gz and tk8.4.6-src.tar.gz

Then do :
tar zxvf tcl8.4.6-src.tar.gz
tar zxvf tk8.4.6-src.tar.gz
It creates two directories : tcl8.4.6 and tk8.4.6
Now do :
cd tcl8.4.6
cd unix (if your OS is Linux or MacOSX)
./configure
make
make install

then
cd tk8.4.6
cd unix
./configure
make
make install

At the end of installation, you have to find where is your binary "wish" or "wish84" or "wish8.4" by typing :
-> which wish (or wish84 or wish8.4)
If wish84 does exist it is all. If not, you have to go in the directory where is "wish" or "wish8.4" (for example /usr/bin/)
and then create a link :
-> ln -s wish wish84
or
-> ln -s wish8.4 wish84

9.2.2 Description

The Graphic User Interface is in the directory called FFedit. You can run it by typing ./FFedit.tcl
The Graphic User Interface shows a text window with buttons on the left and right side, a horizontal menu bar above and an entry below where you can see the path of the script when it is opened or where you can type the path of a script to run it.

This is the description of the different functions of the GUI :

*New : you can access it by the button "New" on the right side or by selecting it in the menu File on the horizontal bar. It deletes the current script and enables you to type a new script.

*Open : you can access it by the button "Open" on the right side or by selecting it in the menu File or by typing simultaneously ctrl+o on the keyboard. It enables you to select a script which has been saved. This script is then opened in the text window. Then, you can modify it, save the changes, save under another name or run it.

*Save : you can access it by the button “Save” on the right side or by selecting it in the menu File or by typing simultaneously ctrl+s on the keyboard. It enables you to save the changes of a script.

*Save as : you can access it by the button “Save As” on the right side or by selecting it in the menu File.
It enables you to locate where you want to save your script and to choose your the name of your script.

*Run : you can access it by the button “Run” on the right side or by selecting it in the menu File or by typing simultaneously ctrl+r on the keyboard.
It enables you to run the current script.
Warning: when you run by typing ctrl+r, the cursor must be in the text window.

*Print : you can access it by the button “Print” on the right side or by selecting it in the menu File or by typing simultaneously ctrl+p on the keyboard.
It enables you to print your script. You have to choose the name of the printer.

*Help (under construction) : you can access it by the button “Help” on the left side. When an example is opened, it shows you a documentation about this example.
Warning : The bottom of each page is not accessible, you have to print to see the entire document.

*Example : you can access it by the button “Ex” on the left side. It runs a little example.

*Read Mesh : you can access it by the button “R.M” on the left side. It enables you to read a mesh which has been saved. It adds the command in your script so that you can use it in your script.

*Polygonal Border : you can access it by the button “P.B” on the left side.
It enables you to build a polygonal border.
When you click on this button, a new window is opened : you have to click on the button “Border” then it asks you how many borders you want. When you enter a number and click on “OK” the exact number of couple of entries enable you to enter the coordinates of the vertices. And then you have to enter the number of points on each border. The border number 1 is the segment between the vertex number 1 and the vertex number 2 ...etc...
You must turn in the opposite sens of needles of a watch.
When you have finished, you have to click on the button “Build”. It builds the domain with polygonal border and shows the result.
Then you can save the result by clicking on the button “S.Mesh”. Choose the extension .msh for the name.

*Navier Stokes : you can access it by the button “N.S” on the left side.
A new window is opened. You have to build your polygonal domain by clicking on the button “Border” it works like the Polygonal Border function.
You can save by clicking on the button “S.Mesh”. Choose the extension .msh for the name.

Then you have to choose the Limits Condition by clicking on the buton “L.C”.
First choose between “Free” or “Imposed’, then click on the button “Validate” and enter the expression of Imposed condition. You have to enter the tangential and the normal component of the velocity. Then click on “Validate” again.
The expression of the limit condition can be a number but a mathematical expression as well.

*emc2 : you can access it by the button “emc2”. It runs emc2 the mesh building software (http://www-rocq1.inria.fr/gamma/cdrom/
www/emc2/fra.htm)

*Set up : you can access it by the button “set up”. It is the first thing you have to do when you use FFedit for the first time.
When you click on this button, a new window is opened, with two entries where you have to enter the path of the binary of FreeFem++ (where you compiled or installed) and the path of the scripts (programs written with FreeFem++) for instance the examples provided in FreeFem++.

*Undo : you can access it by selecting in the menu Edit or by typing simultaneously ctrl+z. It is an unlimited undo function.

*Redo : you can access it by selecting in the menu Edit or by typing simultaneously ctrl+e. It is an unlimited redo function.

*Cut : you can access it by selecting in the menu Edit or by typing simultaneously ctrl+x on the keyboard.

*Copy : you can access it by selecting in the menu Edit or by typing simultaneously ctrl+c on the keyboard.

*Paste : you can access it by selectiog in the menu Edit or by typing simultaneously ctrl+y on the keyboard.

*Delete : you can access it by selecting in the Edit menu. It is a Delete function.

*Select all : you can access it by selecting in the Edit menu or by typing simultaneously ctrl+l. This function select all the text you have written, so you can delete, cut copy paste etc...

*Background : you can access it by selecting in the menu Color. You can then choose the color of the background.

*Foreground : you can access it by selecting in the menu Color. You can then choose the color of the foreground.

*Syntax Color : you can access it by selecting in the menu Color. It enables the syntax coloring of your FreeFem++ script.

*Family : you can access it by selecting in the menu Font. You can then choose the style of your characters.

*Size : you can access it by selecting in the menu Font. You can then choose the size of your characters.

*Find : you can access it by selecting in the menu Search. It enables you to find a word in the whole text. (This function doesn't work yet.)

*Find next : you can access it by slecting in the menu Search. It enables you to find a word from the position of the cursor.(This function doesn't work yet.)

*Replace : you can access it by selecting in the menu Search. It enables you to replace a word by another in the whole text.

9.2.3 Cygwin version

This version is for Windows.

9.2.4 Installation of Cygwin

First you have to go to the site:
http://www.cygwin.com
Click on "Install or Update now"
A window appears.
Click on "Open" then "Next" and then choose "Install from Internet"
Click on "Next" twice and choose "Direct Connection".
Then choose one of the download sites.
For instance : ftp://ftp-stud.fht-esslingen.de
Then choose in each category the options to install.
(click on the symbol + for each, you can then see the options appear)

Here are the required options for FreeFem++ and TCL TK to work :
+ all options of "Devel" (it includes gcc ...etc...)
+ all options of "Graphics" (specially "Gnuplot" to be able
to see the results of FreeFem++ on a graphic)
+ all options of X11 and XFree86

You can install "Xemacs" and others editors in the category "Editors".

Then click on "Next" and wait that Cygwin is installed.
Then an icône appears on yours Desktop.
Click on it and the Cygwin window is opened. It is like an Unix terminal.

To work on a terminal X type :
-> startX
You will have to work on a terminal X to have Gnuplot and visualize
the results of FreeFem++ scripts)

Now you have to compile and install FreeFem++.

9.2.5 Compilation and Installation of FreeFem++ under Cygwin

go to the site :
http://www.freefem.org
click on FreeFem++
and download FreeFem++ (the version when I wrote this is 1.38)
Choose to download in your cygwin/home/you directory.

Then on your terminal Cygwin type :
-> tar zxvf freefem++.tgz
to uncompress this file.

Go into FreeFem++v1.38 directory by typing:
-> cd FreeFem++v1.38

Now you have to compile FreeFem++.
type:
-> make all HOSTTYPE=i-386
to compile FreeFem++
If there is an error : "... -ldl : no such file or directory"
Then you have to modify the Makefile-i386 which is in the directory src:
-> cd src
Edit it (with xemacs for example):
-> xemacs Makefile-i386
at line 1 : replace "LIBLOCAL = -ldl" by "#LIBLOCAL = -ldl"
It will comment this line because -ldl is not on your machine.
Then return to the main directory
-> cd
and type:
-> make all HOSTTYPE=i-386

At the end of compilation, a directory called "c-i386" is created.
In this directory you can find the binary FreeFem++.

You can now run an example:
First open an X terminal:
-> startX
In this terminal go in to FreeFem++v1.38:

-> cd FreeFem++v1.38
and type:
->c-i386/FreeFem++ examples++-tutorial/adapt.edp
You can then see the gnuplot window with the graphical results.

9.2.6 Compilation and Installation of tcl8.4.0 and tk8.4.0 under Cygwin

Now if you want to use the Graphical User Interface of FreeFem++
(called FFedit)
you have to install the language TCL TK in which FFedit has been written.
The version which works under cygwin is tcl8.4.0 and tk8.4.0
The latest version when I wrote this is tcl8.4.6 and tk8.4.6

DON'T USE IT

It works under Linux and MacOsX but not under Cygwin.

You have to download tcl8.4.0 and tk8.4.0 :
For instance, go to Google.fr and type download tcl tk 8.4.0
And choose the "Sourceforge.net: Project Filelist".
Choose :
tcl8.4.0-src.tar.gz and tk8.4.0-src.tar.gz
When the download is finished you have to uncompress these directories:
-> tar zxvf tcl8.4.0-src.tar.gz
-> tar zxvf tk8.4.0-src.tar.gz

tcl8.4.0 and tk8.4.0 will work under Cygwin only if you apply
a patch on both:
These patches are on the site:
http://www.xraylith.wisc.edu/ khan/software/tcl
Choose "Tcl/Tk8.4.0 for Cygwin
Click on "very preliminary Cygwin ports of Tcl/Tk8.4.0
You are then on the site ftp
Follow the instructions of the README or follow these instructions:

1) Run :
-> xemacs tcl-8.4.0-cygwin.diff
By doing this, you create a new file called "tcl-8.4.0-cygwin.diff"
on the site ftp click on "tcl-8.4.0-cygwin.diff"
Do a Copy/Paste of the contain into your xemacs window and save it.

2) Do the same with "tk-8.4.0-cygwin.diff"

Now you have to apply the patch in tcl8.4.0 and tk8.4.0

The two previous patch files (.diff) must be respectively
in tcl8.4.0 and tk8.4.0 directories.
-> cp tcl-8.4.0-cygwin.diff tcl8.4.0
-> cp tcl-8.4.0-cygwin.diff tk8.4.0
(If the two files are one level under tcl8.4.0 and tk8.4.0)

Now apply the patches:
type:

-> cd tcl8.4.0
-> patch -p0 -s < tcl-8.4.0-cygwin.diff

-> cd
-> cd tk8.4.0
-> patch -p0 -s < tk-8.4.0-cygwin.diff

Now you can compile and install TCL TK under Cygwin:

* compilation and installation of tcl8.4.0

Go in to the directory tcl8.4.0/win
-> cd tcl8.4.0
-> cd win
Then type :
-> ./configure
The two steps remaining are make and make install
type
-> make
The compilation starts, when finished install by typing:
-> make install

When finished try to see if it works by typing:
-> tclsh84
if ok quit by typing ctrl-c

*Compilation and installation of tk8.4.0

Go in to the directory tk8.4.0/win
-> cd tk8.4.0
-> cd win
Then type :
-> ./configure
The two steps remaining are make and make install
type
-> make
The compilation starts, if you have errors like :

windres -o tk.res.o –include “C:/cygwin/home/ly/tk8.4.0/generic”
–include “C Option-I is deprecated for setting the input format,
please use -J instead”
windres : can’t open icon file ’tk.ico’ : no such file or directory

This file ’tk.ico’ is in fact in the directory win/rc
You have to copy it in the directory ’generic’:
Be in tk8.4.0
type :
-> cp win/rc/tk.ico generic/

If you compile again you will see that there is the same
errors with the files:
“buttons.bmp” “cursor00.cur” “cursor02.cur” ...etc...
“wish.exe.manifest” and “wish.ico”

Do the same for these files.
For the cursor*.cur files do once the command:
-> cp win/rc/cursor*.cur generic/

when finished install by typing:
-> make install

When finished try to see if it works by typing:
-> wish84
if ok quit by typing ctrl-c

9.2.7 Use

Now everything is ok to use FFedit and FreeFem++ under Windows by Cygwin.
WARNING: if you work on the Cygwin terminal you will not be able
to see the graphical results of FreeFem++.

You have to run an X terminal and run FFedit under this X terminal:

To run an X terminal under cygwin type on your Cygwin terminal:
-> startX

An X terminal runs:
Under this terminal:
type
-> cd FFedit
-> ./FFedit.tcl

Chapter 10

Mesh Files

10.1 File mesh data structure

The mesh data structure, output of a mesh generation algorithm, refers to the geometric data structure and in some case to another mesh data structure.
In this case, the fields are

- `MeshVersionFormatted 0`

- `Dimension (I)` dim
- `Vertices (I)` NbOfVertices
  $\Big(\big(\texttt{(R)}\ x_i^j ,\ j{=}1,\mathrm{dim} \big),\ \texttt{(I)}\ Ref\phi_i^v ,\ i{=}1,\mathrm{NbOfVertices} \Big)$
- `Edges (I)` NbOfEdges
  $\Big(\texttt{@@Vertex}_i^1, \texttt{@@Vertex}_i^2,\ \texttt{(I)}\ Ref\phi_i^e ,\ i{=}1,\mathrm{NbOfEdges} \Big)$
- `Triangles (I)` NbOfTriangles
  $\Big(\big(\texttt{@@Vertex}_i^j ,\ j{=}1,3 \big),\ \texttt{(I)}\ Ref\phi_i^t ,\ i{=}1,\mathrm{NbOfTriangles} \Big)$
- `Quadrilaterals (I)` NbOfQuadrilaterals
  $\Big(\big(\texttt{@@Vertex}_i^j ,\ j{=}1,4 \big),\ \texttt{(I)}\ Ref\phi_i^t ,\ i{=}1,\mathrm{NbOfQuadrilaterals} \Big)$
- `Geometry`
  `(C*)` FileNameOfGeometricSupport

  - `VertexOnGeometricVertex`
    `(I)` NbOfVertexOnGeometricVertex
    $\Big(\texttt{@@Vertex}_i, \texttt{@@Vertex}_i^{geo} ,\ i{=}1,\mathrm{NbOfVertexOnGeometricVertex} \Big)$
  - `EdgeOnGeometricEdge`
    `(I)` NbOfEdgeOnGeometricEdge
    $\Big(\texttt{@@Edge}_i, \texttt{@@Edge}_i^{geo} ,\ i{=}1,\mathrm{NbOfEdgeOnGeometricEdge} \Big)$

- `CrackedEdges (I)` NbOfCrackedEdges
  $\Big(\texttt{@@Edge}_i^1, \texttt{@@Edge}_i^2 ,\ i{=}1,\mathrm{NbOfCrackedEdges} \Big)$

When the current mesh refers to a previous mesh, we have in addition

- `MeshSupportOfVertices`
  `(C*)` FileNameOfMeshSupport

  - `VertexOnSupportVertex`
    `(I)` NbOfVertexOnSupportVertex
    $\Big($`@@Vertex`$_i$, `@@Vertex`$_i^{supp}$, i=1,NbOfVertexOnSupportVertex $\Big)$
  - `VertexOnSupportEdge`
    `(I)` NbOfVertexOnSupportEdge
    $\Big($`@@Vertex`$_i$, `@@Edge`$_i^{supp}$, `(R)` u_i^{supp} , i=1,NbOfVertexOnSupportEdge $\Big)$
  - `VertexOnSupportTriangle`
    `(I)` NbOfVertexOnSupportTriangle
    $\Big($`@@Vertex`$_i$, `@@Tria`$_i^{supp}$, `(R)` u_i^{supp} , `(R)` v_i^{supp} ,
    i=1 , NbOfVertexOnSupportTriangle $\Big)$
  - `VertexOnSupportQuadrilaterals`
    `(I)` NbOfVertexOnSupportQuadrilaterals
    $\Big($`@@Vertex`$_i$, `@@Quad`$_i^{supp}$, `(R)` u_i^{supp} , `(R)` v_i^{supp} ,
    i=1 , NbOfVertexOnSupportQuadrilaterals $\Big)$

10.2 bb File type for Store Solutions

The file is formatted such that:
`2 nbsol nbv 2`
$((\texttt{U}_{ij}, \quad \forall i \in \{1,...,\texttt{nbsol}\}), \quad \forall j \in \{1,...,\texttt{nbv}\})$
where

- `nbsol` is a integer equal to the number of solutions.
- `nbv` is a integer equal to the number of vertex .
- $\texttt{U}_{ij}$ is a real equal the value of the i solution at vertex j on the associated mesh background if read file, generated if write file.

10.3 BB File Type for Store Solutions

The file is formatted such that:
$\texttt{2 n typesol}^1 \; ... \; \texttt{typesol}^n \; \texttt{nbv 2}$
$\Big(\Big(\big(\texttt{U}_{ij}^k, \quad \forall i \in \{1,...,\texttt{typesol}^k\}\big), \quad \forall k \in \{1,...\texttt{n}\}\Big) \quad \forall j \in \{1,...,\texttt{nbv}\}\Big)$
where

- `n` is a integer equal to the number of solutions
- $\texttt{typesol}^k$, type of the solution number k, is

  - `typesol`$^k = 1$ the solution `k` is scalare (1 value per vertex)
  - `typesol`$^k = 2$ the solution `k` is vectorial (2 values per unknown)
  - `typesol`$^k = 3$ the solution `k` is a 2×2 symmetric matrix (3 values per vertex)
  - `typesol`$^k = 4$ the solution `k` is a 2×2 matrix (4 values per vertex)

- `nbv` is a integer equal to the number of vertices
- $\mathtt{U}_{ij}^k$ is a real equal the value of the component i of the solution k at vertex j on the associated mesh background if read file, generated if write file.

10.4 Metric File

A metric file can be of two types, isotropic or anisotropic.
the isotrope file is such that
`nbv 1`
$\mathtt{h}_i \quad \forall i \in \{1, ..., \mathtt{nbv}\}$
where

- `nbv` is a integer equal to the number of vertices.
- $\mathtt{h}_i$ is the wanted mesh size near the vertex i on background mesh, the metric is $\mathcal{M}_i = h_i^{-2} Id$, where Id is the identity matrix.

The metric anisotrope
`nbv 3`
$\mathtt{a11}_i, \mathtt{a21}_i, \mathtt{a22}_i \quad \forall i \in \{1, ..., \mathtt{nbv}\}$
where

- `nbv` is a integer equal to the number of vertices,
- $\mathrm{a11}_i$, $\mathrm{a12}_i$, $\mathrm{a22}_i$ is metric $\mathcal{M}_i = \begin{pmatrix} a11_i & a12_i \\ a12_i & a22_i \end{pmatrix}$ which define the wanted mesh size in a vicinity of the vertex i such that h in direction $u \in \mathbb{R}^2$ is equal to $|u|/\sqrt{u \cdot \mathcal{M}_i\, u}$, where $\cdot$ is the dot product in $\mathbb{R}^2$, and $| \cdot |$ is the classical norm.

10.5 List of AM_FMT, AMDBA Meshes

The mesh is only composed of triangles and can be defined with the help of the following two integers and four arrays:

`nbt`	is the number of triangles.
`nbv`	is the number of vertices.
`nu(1:3,1:nbt)`	is an integer array giving the three vertex numbers counterclockwise for each triangle.

`c(1:2,nbv)`	is a real array giving the two coordinates of each vertex.
`refs(nbv)`	is an integer array giving the reference numbers of the vertices.
`reft(nbv)`	is an integer array giving the reference numbers of the triangles.

AM_FMT Files In fortran the `am_fmt` files are read as follows:

```
open(1,file='xxx.am_fmt',form='formatted',status='old')
  read (1,*) nbv,nbt
  read (1,*)  ((nu(i,j),i=1,3),j=1,nbt)
  read (1,*)  ((c(i,j),i=1,2),j=1,nbv)
  read (1,*)  ( reft(i),i=1,nbt)
  read (1,*)  ( refs(i),i=1,nbv)
close(1)
```

AM Files In fortran the `am` files are read as follows:

```
open(1,file='xxx.am',form='unformatted',status='old')
  read (1,*) nbv,nbt
  read (1)  ((nu(i,j),i=1,3),j=1,nbt),
&   ((c(i,j),i=1,2),j=1,nbv),
&   ( reft(i),i=1,nbt),
&   ( refs(i),i=1,nbv)
close(1)
```

AMDBA Files In fortran the `amdba` files are read as follows:

```
open(1,file='xxx.amdba',form='formatted',status='old')
  read (1,*) nbv,nbt
  read (1,*) (k,(c(i,k),i=1,2),refs(k),j=1,nbv)
  read (1,*) (k,(nu(i,k),i=1,3),reft(k),j=1,nbt)
close(1)
```

msh Files First, we add the notions of boundary edges

`nbbe`	is the number of boundary edge.
`nube(1:2,1:nbbe)`	is an integer array giving the two vertex numbers
`refbe(1:nbbe)`	is an integer array giving the two vertex numbers

In fortran the `msh` files are read as follows:

```
open(1,file='xxx.msh',form='formatted',status='old')
  read (1,*) nbv,nbt,nbbe
  read (1,*) ((c(i,k),i=1,2),refs(k),j=1,nbv)
  read (1,*) ((nu(i,k),i=1,3),reft(k),j=1,nbt)
  read (1,*) ((ne(i,k),i=1,2), refbe(k),j=1,nbbe)
close(1)
```

ftq Files In fortran the `ftq` files are read as follows:

```
open(1,file='xxx.ftq',form='formatted',status='old')
 read (1,*) nbv,nbe,nbt,nbq
 read (1,*) (k(j),(nu(i,j),i=1,k(j)),reft(j),j=1,nbe)
 read (1,*) ((c(i,k),i=1,2),refs(k),j=1,nbv)
close(1)
```

where if `k(j) = 3` then the element j is a triangle and if `k = 4` the the element j is a quadrilateral.

Chapter 11

Add new finite element

11.1 Some notation

For a function $\boldsymbol{f}$ taking value in $\mathbb{R}^N$, $N = 1, 2, \cdots$, we define the finite element approximation $\Pi_h \boldsymbol{f}$ of $\boldsymbol{f}$. Let us denote the number of degrees of freedom of the finite element by $NbDoF$. Then the i-th base $\boldsymbol{\omega}_i^K$ $(i = 0, \cdots, NbDoF - 1)$ of the finite element space has the j-th componante ω_{ij}^K for $j = 0, \cdots, N - 1$.
The operator Π_h is called the interpolator of the finite element. We have the identity $\boldsymbol{\omega}_i^K = \Pi_h \boldsymbol{\omega}_i^K$.
Formally, the interpolator Π_h is constructed by the following formula:

$$\Pi_h \boldsymbol{f} = \sum_{k=0}^{\mathtt{kPi}-1} \alpha_k \boldsymbol{f}_{j_k}(P_{p_k}) \boldsymbol{\omega}_{i_k}^K \tag{11.1}$$

where P_p is a set of $npPi$ points,
In the formula (11.1), the list p_k, j_k, i_k depend just on the type of finite element (not on the element), but the coefficient α_k can be depending on the element.

Example 1: classical scalar Lagrange finite element, first we have `kPi` = `npPi` = `NbOfNode` and

- P_p is the point of the nodal points
- the $\alpha_k = 1$, because we take the value of the function at the point P_k
- $p_k = k$, $j_k = k$ because we have one node per function.
- $j_k = 0$ because $N = 1$

Example 2: The Raviart-Thomas finite element:

$$RT0_h = \{\mathbf{v} \in H(div)/\forall K \in \mathcal{T}_h \quad \mathbf{v}_{|K}(x, y) = \left| \begin{smallmatrix} \alpha_K \\ \beta_K \end{smallmatrix} + \gamma_K \right| \begin{smallmatrix} x \\ y \end{smallmatrix} \} \tag{11.2}$$

The degree of freedom are the flux throw an edge e of the mesh, where the flux of the function $\mathbf{f} : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$ is $\int_e \mathbf{f}.n_e$, n_e is the unit normal of edge e (this implies a orientation of all the edges of the mesh, for exemple we can use the global numbering of the edge vertices and we just go to small to large number).

To compute this flux, we use an quadrature formular with one point, the middle point of the edge. Consider a triangle T with three vertices $(\mathbf{a}, \mathbf{b}, \mathbf{c})$. Let denote the vertices numbers by i_a, i_b, i_c, and define the three edge vectors $\mathbf{e}^0, \mathbf{e}^1, \mathbf{e}^2$ by $sgn(i_b - i_c)(\mathbf{b} - \mathbf{c})$, $sgn(i_c - i_a)(\mathbf{c} - \mathbf{a})$, $sgn(i_a - i_b)(\mathbf{a} - \mathbf{b})$,
The three basis functions are:

$$\boldsymbol{\omega}_0^K = \frac{sgn(i_b - i_c)}{2|T|}(x - a), \quad \boldsymbol{\omega}_1^K = \frac{sgn(i_c - i_a)}{2|T|}(x - b), \quad \boldsymbol{\omega}_2^K = \frac{sgn(i_a - i_b)}{2|T|}(x - c), \quad (11.3)$$

where $|T|$ is the area of the triangle T.
So we have $N = 2$, `kPi` $= 6$; `npPi` $= 3$; and:

- $P_p = \left\{\frac{\mathbf{b}+\mathbf{c}}{2}, \frac{\mathbf{a}+\mathbf{c}}{2}, \frac{\mathbf{b}+\mathbf{a}}{2}\right\}$
- $\alpha_0 = -\mathbf{e}_2^0, \alpha_1 = \mathbf{e}_1^0$, $\alpha_2 = -\mathbf{e}_2^1, \alpha_3 = \mathbf{e}_1^1$, $\alpha_4 = -\mathbf{e}_2^2, \alpha_5 = \mathbf{e}_1^2$ (effectively, the vector $(-\mathbf{e}_2^m, \mathbf{e}_1^m)$ is orthogonal to the edge $\mathbf{e}^m = (e_1^m, e_2^m)$ with a length equal to the side of the edge or equal to $\int_{e^m} 1$).
- $i_k = \{0, 0, 1, 1, 2, 2\}$,
- $p_k = \{0, 0, 1, 1, 2, 2\}$, $j_k = \{0, 1, 0, 1, 0, 1, 0, 1\}$.

11.2 Which class of add

Add file `FE_ADD.cpp` in directory `src/femlib` for exemple first to initialize :

```
#include "error.hpp"
#include "rgraph.hpp"
using namespace std;
#include "RNM.hpp"
#include "fem.hpp"
#include "FESpace.hpp"

namespace  Fem2D {
```

Second, you are just a class which derive for `public TypeOfFE` like:

```
class TypeOfFE_RTortho : public  TypeOfFE { public:
  static int Data[]; //    some numbers
  TypeOfFE_RTortho():
    TypeOfFE( 0+3+0,   //    nb degree of freedom on element
       2,      //    dimension N of vectorial FE (1 if scalar FE)
       Data,   //    the array data
       1,      //    nb of subdivision for plotting
       1,      //    nb of sub finite element (generaly 1)
       6,      //    number kPi of coef to build the interpolator (11.1)
       3,      //    number npPi of integration point to build interpolator
       0       //    an array to store the coef αk to build interpolator
               //    here this array is no constant so we have
               //    to rebuilt for each element.
       )
  {
    const R2 Pt[] = { R2(0.5,0.5), R2(0.0,0.5), R2(0.5,0.0) };
                                                    //    the set of Point in $\hat{K}$
    for (int p=0,kk=0;p<3;p++) {
      P_Pi_h[p]=Pt[p];
      for (int j=0;j<2;j++)
        pij_alpha[kk++]= IPJ(p,p,j); }}   //    definition of $i_k, p_k, j_k$ in (11.1)

  void FB(const bool * watdd, const Mesh & Th,const Triangle & K,
          const R2 &PHat, RNMK_ & val) const;

  void Pi_h_alpha(const baseFElement & K,KN_<double> & v) const ;

} ;
```

where the array data is form with the concatenation of five array of size `NbDoF` and one array of size `N`.
This array is:

```
int TypeOfFE_RTortho::Data[]={
       //    for each df 0,1,3 :
        3,4,5,  //    the support of the node of the df
        0,0,0,  //    the number of the df on the node
        0,1,2,  //    the node of the df
        0,0,0,  //    the df come from which FE (generaly 0)
        0,1,2,  //    which are de df on sub FE
        0,0 };          //    for each compontant $j = 0, N-1$ it give the sub FE
associated
```

where the support is a number $0, 1, 2$ for vertex support, $3, 4, 5$ for edge support, and finaly 6 for element support.
The function to defined the function $\boldsymbol{\omega}_i^K$, this function return the value of all the basics function or this derivatives in array `val`, computed at point `PHat` on the reference triangle corresponding to point `R2 P=K(Phat);` on the current triangle `K`.
The index i, j, k of the array $val(i, j, k)$ corresponding to:

i is basic function number on finite element $i \in [0, NoF[$

j is the value of component $j \in [0, N[$

k is the type of computed value $f(P), dx(f)(P), dy(f)(P), ...$ $i \in [0,$ `last_operatortype`$[$. Remark for optimisation, this value is computed only if $whatd[k]$ is true, and the numbering is defined with

```
enum operatortype { op_id=0,
   op_dx=1,op_dy=2,
   op_dxx=3,op_dyy=4,
   op_dyx=5,op_dxy=5,
   op_dz=6,
   op_dzz=7,
   op_dzx=8,op_dxz=8,
   op_dzy=9,op_dyz=9
   };
const int last_operatortype=10;
```

The shape function :

```
 void TypeOfFE_RTortho::FB(const bool *whatd,const Mesh & Th,const Triangle & K,
                           const R2 & PHat,RNMK_ & val) const
{                                                                            //
  R2 P(K(PHat));
  R2 A(K[0]), B(K[1]),C(K[2]);
  R l0=1-P.x-P.y,l1=P.x,l2=P.y;
  assert(val.N() >=3);
  assert(val.M()==2 );
  val=0;
  R a=1./(2*K.area);
  R a0=   K.EdgeOrientation(0) * a ;
  R a1=   K.EdgeOrientation(1) * a  ;
  R a2=   K.EdgeOrientation(2) * a ;

                                                                 //    ------------
  if (whatd[op_id])                                      //    value of the function
   {
     assert(val.K()>op_id);
     RN_ f0(val('.',0,0));                               //    value first component
     RN_ f1(val('.',1,0));                              //    value second component
     f1[0] =  (P.x-A.x)*a0;
     f0[0] = -(P.y-A.y)*a0;

     f1[1] =  (P.x-B.x)*a1;
     f0[1] = -(P.y-B.y)*a1;

     f1[2] =  (P.x-C.x)*a2;
     f0[2] = -(P.y-C.y)*a2;
    }
                                                             //    ----------------
    if (whatd[op_dx])                               //    value of the dx of function
    {
     assert(val.K()>op_dx);
     val(0,1,op_dx) =  a0;
     val(1,1,op_dx) =  a1;
     val(2,1,op_dx) =  a2;
     }
    if (whatd[op_dy])
    {
     assert(val.K()>op_dy);
     val(0,0,op_dy) =  -a0;
     val(1,0,op_dy) =  -a1;
     val(2,0,op_dy) =  -a2;
    }

  for (int i= op_dy; i< last_operatortype ; i++)
   if (whatd[op_dx])
     assert(op_dy);

}
```

The function to defined the coefficient α_k:

```
void TypeOfFE_RT::Pi_h_alpha(const baseFElement & K,KN_<double> & v) const
```

```
{
  const Triangle & T(K.T);

   for (int i=0,k=0;i<3;i++)
     {
        R2 E(T.Edge(i));
        R signe = T.EdgeOrientation(i) ;
        v[k++]= signe*E.y;
        v[k++]=-signe*E.x;
     }
}
```

Now , we just need to add a new key work in FreeFem++, so at the end of the file, we add:

```
                                             //    let the 2 globals variables
static TypeOfFE_RTortho The_TypeOfFE_RTortho;                             //
                                          //    ----- the name in freefem ----
static  ListOfTFE typefemRTOrtho("RT0Ortho", & The_TypeOfFE_RTortho);    //

//    link with FreeFem++ do not work with static library .a
//    FH so add a extern name to call in init_static_FE
//    (see end of FESpace.cpp)
void init_FE_ADD() { };
//    --- end --
}                                                         //    FEM2d namespace
```

To inforce in loading of this new finite element, we have to add the two new lignes close to the end of files src/femlib/FESpace.cpp like:

```
              //    correct Probleme of static library link with new make file
void init_static_FE()
{                                                //    list of other FE file.o
   extern void init_FE_P2h() ;
  init_FE_P2h() ;
   extern void init_FE_ADD() ;                                //    new ligne 1
   init_FE_ADD();                                             //    new ligne 2
}
```

11.3 How to add

First, create a file FE_ADD.cpp contening all this code, like in file src/femlib/Element_P2h.cpp, after modifier the Makefile.am by adding the name of your file to the variable EXTRA_DIST like:

```
# Makefile using Automake + Autoconf
# ----------------------------------
# $Id: addfe.tex,v 1.3 2004/09/28 08:58:40 hecht Exp $

# This is not compiled as a separate library because its
# interconnections with other libraries have not been solved.
```

```
EXTRA_DIST=BamgFreeFem.cpp BamgFreeFem.hpp CGNL.hpp CheckPtr.cpp        \
ConjuguedGradrientNL.cpp DOperator.hpp Drawing.cpp Element_P2h.cpp      \
Element_P3.cpp Element_RT.cpp fem3.hpp fem.cpp fem.hpp FESpace.cpp      \
FESpace.hpp FESpace-v0.cpp FQuadTree.cpp FQuadTree.hpp gibbs.cpp        \
glutdraw.cpp gmres.hpp MatriceCreuse.hpp MatriceCreuse_tpl.hpp          \
MeshPoint.hpp mortar.cpp mshptg.cpp QuadratureFormular.cpp              \
QuadratureFormular.hpp RefCounter.hpp RNM.hpp RNM_opc.hpp RNM_op.hpp    \
RNM_tpl.hpp   FE_ADD.cpp
```

and recompile
For codewarrior compilation add the file in the project an remove the flag in panal PPC linker FreeFEm++ Setting Dead-strip Static Initializition Code Flag.

Appendix A

Table of Notations

Here mathematical expressions and corresponding `freefem++` commands are noted.

A.1 Generalities

δ_{ij} Kronecker delta (0 if $i \neq j$, 1 if $i = j$ for integers i, j)

$\forall$ for all

$\exists$ there exist

i.e. that is

PDE partial differential equation (with boundary conditions)

$\emptyset$ the empty set

$\mathbb{N}$ the set of integers ($a \in \mathbb{N} \Leftrightarrow$ `int a`); "`int`" means *long integer* inside `freefem++`

$\mathbb{R}$ the set of real numbers ($a \in \mathbb{R} \Leftrightarrow$ `real a`) ;*double* inside `freefem++`

$\mathbb{C}$ the set of complex numbers ($a \in \mathbb{C} \Leftrightarrow$ `complex a`); *complex¡double¿*

$\mathbb{R}^d$ d-dimensional Euclidean space

A.2 Sets, Mappings, Matrices, Vectors

Let E, F, G be three sets and A subset of E.

$\{x \in E |\ P\}$ the subset of E consisting of the elements possessing the property P

$E \cup F$ the set of elements belonging to E or F

$E \cap F$ the set of elements belonging to E and F

$E \setminus A$ the set $\{x \in E |\ x \notin A\}$

$E + F$ $E \cup F$ with $E \cap F = \emptyset$

$E \times F$ the cartesian product of E and F

E^n the n-th power of E ($E^2 = E \times E$, $E^n = E \times E^{n-1}$)

$f : E \to F$ the mapping form E into F, i.e., $E \ni x \mapsto f(x) \in F$

I_E **or** I the identity mapping in E,i.e., $I(x) = x \quad \forall x \in E$

$f \circ g$ for $f : F \to G$ and $g : E \to F$, $E \ni x \mapsto (f \circ g)(x) = f(g(x)) \in G$ (see Section 2.5)

$f|_A$ the restriction of $f : E \to F$ to the subset A of E

$\{a_k\}$ column vector with components a_k

(a_k) row vector with components a_k

$(a_k)^T$ denotes the transpose of a matrix (a_k), and is $\{a_k\}$

$\{a_{ij}\}$ matrix with components a_{ij}, and $(a_{ij})^T = (a_{ji})$

A.3 Numbers

For two real numbers a, b

$[a, b\ ]$ $\{x \in \mathbb{R} |\ a \le x \le b\}$

$\ a, b]]$ $\{x \in \mathbb{R} |\ a < x \le b\}$

$[a, b[$ $\{x \in \mathbb{R} |\ a \le x < b\}$

$\ a, b[]$ $\{x \in \mathbb{R} |\ a < x < b\}$

A.4 Differential Calculus

$\partial f / \partial x$ the partial derivative of $f : \mathbb{R}^d \to \mathbb{R}$ with respect to x (`dx(f)`)

∇f the gradient of $f : \Omega \to \mathbb{R}$,i.e., $\nabla f = (\partial f / \partial x, \partial f / \partial y)$

div$\boldsymbol{f}$ **or** $\nabla . \boldsymbol{f}$ the divergence of $\boldsymbol{f} : \Omega \to \mathbb{R}^d$, i.e., $\text{div}\boldsymbol{f} = \partial f_1 / \partial x + \partial f_2 / \partial y$

Δf the Laplacian of $f : \Omega \to \mathbb{R}$, i.e., $\Delta f = \partial^2 f / \partial x^2 + \partial^2 f / \partial y^2$

A.5 Meshes

Ω usually denotes a domain on which PDE is defined

Γ denotes the boundary of Ω,i.e., $\Gamma = \partial\Omega$ (keyword **border**, see Section 3.1.2)

$\mathcal{T}_h$ the triangulation of Ω, i.e., the set of triangles T_k, where h stands for mesh size (keyword **mesh**, **buildmesh**, see Section 3)

n_t the number of triangles in $\mathcal{T}_h$ (get by `Th.nt`, see "mesh.edp")

Ω_h denotes the approximated domain $\Omega_h = \cup_{k=1}^{n_t} T_k$ of Ω. If Ω is polygonal domain, then it will be $\Omega = \Omega_h$

Γ_h the boundary of Ω_h

n_v the number of vertices in $\mathcal{T}_h$ (get by `Th.nv`)

$[q^i q^j\ ]$ the segment connecting q^i and q^j

$q^{k_1}, q^{k_2}, q^{k_3}$ the vertices of a triangle T_k with anti-clock direction (get the coordinate of q^{k_j} by (`Th[k-1][j-1].x, Th[k-1][j-1].y`))

I_Ω the set $\{i \in \mathbb{N} |\ q^i \notin \Gamma_h\}$

A.6 Finite Element Spaces

$L^2(\Omega)$ the set $\left\{ w(x,y) \,\middle|\, \int_\Omega |w(x,y)|^2 dxdy < \infty \right\}$

$$\text{norm: } \|w\|_{0,\Omega} = \left(\int_\Omega |w(x,y)|^2 dxdy \right)^{1/2}$$

$$\text{scalar product: } (v,w) = \int_\Omega vw$$

$H^1(\Omega)$ the set $\left\{ w \in L^2(\Omega) \,\middle|\, \int_\Omega \left(|\partial w/\partial x|^2 + |\partial w/\partial y|^2 \right) dxdy < \infty \right\}$

$$\text{norm: } \|w\|_{1,\Omega} = \left(\|w\|_{0,\Omega}^2 + \|\nabla u\|_{0.\Omega}^2 \right)^{1/2}$$

$H^m(\Omega)$ the set $\left\{ w \in L^2(\Omega) \,\middle|\, \int_\Omega \frac{\partial^{|\alpha|} w}{\partial x^{\alpha_1} \partial y^{\alpha_2}} \in L^2(\Omega) \quad \forall \alpha = (\alpha_1, \alpha_2) \in \mathbb{N}^2,\ |\alpha| = \alpha_1 + \alpha_2 \right\}$

$$\text{scalar product: } (v,w)_{1,\Omega} = \sum_{|\alpha| \le m} \int_\Omega D^\alpha v D^\alpha w$$

$H_0^1(\Omega)$ the set $\{ w \in H^1(\Omega) \mid u = 0 \quad \text{on } \Gamma \}$

$L^2(\Omega)^2$ denotes $L^2(\Omega) \times L^2(\Omega)$, and also $H^1(\Omega)^2 = H^1(\Omega) \times H^1(\Omega)$

V_h denotes the finite element space created by "`fespace` Vh(Th,*)" in `freefem++` (see Section 4 for "*")

$\Pi_h f$ the projection of the function f into V_h ("`func f=x^2*y^3; Vh v = f;`" means `v` $=$ Π_h`f`)

$\{v\}$ for FE-function v in V_h means the column vector $(v_1, \cdots, v_M)^T$ if $v = v_1\phi_1 + \cdots + v_M\phi_M$, which is shown by "`fespace Vh(Th,P2); Vh v; cout << v[] << endl;`"

Appendix B

Grammar

B.1 Keywords

```
Cmatrix
R3
bool
border
break
complex
continue
element
else
end
fespace
for
func
if
ifstream
include
int
load
macro
matrix
mesh
ofstream
problem
real
return
solve
string
vertex
varf
while
```

B.2 The bison grammar

```
start:   input ENDOFFILE;

input:   instructions ;

instructions:  instruction
        | instructions  instruction   ;

list_of_id_args:
            | id
            | id '=' no_comma_expr
            | FESPACE id
            | type_of_dcl id
            | type_of_dcl '&' id
            | '[' list_of_id_args ']'
            | list_of_id_args ',' id
            | list_of_id_args ',' '[' list_of_id_args ']'
            | list_of_id_args ',' id '=' no_comma_expr
            | list_of_id_args ',' FESPACE id
            | list_of_id_args ',' type_of_dcl id
            | list_of_id_args ',' type_of_dcl '&' id ;

list_of_id1:  id
            | list_of_id1 ',' id   ;

id: ID | FESPACE ;

list_of_dcls:    ID
              |  ID '='   no_comma_expr
              |  ID  '(' parameters_list ')'
              |  list_of_dcls ',' list_of_dcls  ;


parameters_list:
           no_set_expr
        |  FESPACE  ID
        |  ID '=' no_set_expr
        | parameters_list ',' no_set_expr
        | parameters_list ',' id '=' no_set_expr ;

type_of_dcl:   TYPE
             | TYPE '[' TYPE ']' ;

ID_space:
    ID
 |  ID '[' no_set_expr ']'
 |  ID '=' no_set_expr
 |  '[' list_of_id1 ']'
 |  '[' list_of_id1 ']' '[' no_set_expr ']'
 |  '[' list_of_id1 ']' '=' no_set_expr ;

ID_array_space:
    ID '(' no_set_expr ')'
 |  '[' list_of_id1 ']' '(' no_set_expr ')' ;

fespace: FESPACE ;

spaceIDa  :      ID_array_space
            |    spaceIDa ',' ID_array_space  ;

spaceIDb  :      ID_space
            |    spaceIDb ',' ID_space ;

spaceIDs :    fespace                spaceIDb
           |  fespace '[' TYPE ']'   spaceIDa    ;

fespace_def: ID '(' parameters_list ')' ;

fespace_def_list:  fespace_def
                 | fespace_def_list ',' fespace_def ;


declaration:   type_of_dcl list_of_dcls ';'
             | 'fespace' fespace_def_list    ';'
             | spaceIDs ';'
             | FUNCTION ID '=' Expr ';'
             | FUNCTION type_of_dcl ID  '(' list_of_id_args ')'  '{' instructions'}'
             | FUNCTION ID '(' list_of_id_args ')'   '='   no_comma_expr  ';'     ;

begin: '{'  ;
end:   '}'  ;

for_loop:    'for'   ;
while_loop:  'while' ;

instruction:   ';'
         | 'include'  STRING
         | 'load'  STRING
         |  Expr  ';'
         |  declaration
         |  for_loop  '(' Expr ';' Expr ';' Expr ')' instruction
         |  while_loop '(' Expr ')' instruction
         |  'if' '(' Expr ')'   instruction
         |  'if' '(' Expr ')'   instruction  ELSE instruction
         |  begin  instructions end
         |  'border'  ID   border_expr
         |  'border'   ID   '['  array ']' ';'
         |  'break'  ';'
         |  'continue'  ';'
         |  'return'  Expr ';'  ;


bornes: '(' ID '=' Expr ',' Expr ')' ;

border_expr:   bornes instruction  ;

Expr:    no_comma_expr
       | Expr ',' Expr ;
```

```
unop:     '-'
        | '+'
        | '!'
        | '++'
        | '--'  ;

no_comma_expr:
          no_set_expr
        | no_set_expr '=' no_comma_expr
        | no_set_expr '+=' no_comma_expr
        | no_set_expr '-=' no_comma_expr
        | no_set_expr '*=' no_comma_expr
        | no_set_expr '/=' no_comma_expr ;

no_set_expr:
          unary_expr
        | no_set_expr '*' no_set_expr
        | no_set_expr '.*' no_set_expr
        | no_set_expr './' no_set_expr
        | no_set_expr '/' no_set_expr
        | no_set_expr '%' no_set_expr
        | no_set_expr '+' no_set_expr
        | no_set_expr '-' no_set_expr
        | no_set_expr '<<' no_set_expr
        | no_set_expr '>>' no_set_expr
        | no_set_expr '&' no_set_expr
        | no_set_expr '&&' no_set_expr
        | no_set_expr '|' no_set_expr
        | no_set_expr '||' no_set_expr
        | no_set_expr '<' no_set_expr
        | no_set_expr '<=' no_set_expr
        | no_set_expr '>' no_set_expr
        | no_set_expr '>=' no_set_expr
        | no_set_expr '==' no_set_expr
        | no_set_expr '!=' no_set_expr ;


parameters:
        |   no_set_expr
        |   FESPACE
        |   id '=' no_set_expr
        |   parameters ',' FESPACE
        |   parameters ',' no_set_expr
        |   parameters ',' id '=' no_set_expr ;

array:   no_comma_expr
       | array ',' no_comma_expr ;


unary_expr:
    pow_expr
  | unop  pow_expr %prec UNARY ;

pow_expr: primary
  |      primary  '^' unary_expr
  |     primary  '_' unary_expr
  |     primary ''  ;                                    //    transpose

primary:
           ID
  |        LNUM
  |        DNUM
  |        CNUM
  |        STRING
  |        primary '('  parameters ')'
  |        primary '[' Expr ']'
  |        primary '['  ']'
  |        primary '.'  ID
  |        primary '++'
  |        primary '--'
  |        TYPE '('  Expr ')' ;
  |        '(' Expr ')'
  |        '[' array  ']' ;
```

B.3 The Types of the languages, and cast

B.4 All the operators

```
 - CG,  type :<TypeSolveMat>
 - Cholesky,  type :<TypeSolveMat>
 - Crout,  type :<TypeSolveMat>
 - GMRES,  type :<TypeSolveMat>
 - LU,  type :<TypeSolveMat>
 - LinearCG,  type :<Polymorphic>   operator() :
  (    <long> :   <Polymorphic>, <KN<double> *>, <KN<double> *> )

 - N,  type :<Fem2D::R3>
 - NoUseOfWait,  type :<bool *>
 - P,  type :<Fem2D::R3>
 - P0,  type :<Fem2D::TypeOfFE>
 - P1,  type :<Fem2D::TypeOfFE>
 - P1nc,  type :<Fem2D::TypeOfFE>
 - P2,  type :<Fem2D::TypeOfFE>
 - RT0,  type :<Fem2D::TypeOfFE>
 - RTmodif,  type :<Fem2D::TypeOfFE>
 - abs,  type :<Polymorphic>  operator() :
  (    <double> :   <double> )

 - acos,  type :<Polymorphic>   operator() :
  (    <double> :   <double> )

 - acosh,  type :<Polymorphic>   operator() :
  (    <double> :   <double> )

 - adaptmesh,  type :<Polymorphic>   operator() :
  (    <Fem2D::Mesh> :   <Fem2D::Mesh>... )

 - append,  type :<std::ios_base::openmode>
 - asin,  type :<Polymorphic>   operator() :
  (    <double> :   <double> )

 - asinh,  type :<Polymorphic>  operator() :
  (    <double> :   <double> )

 - atan,  type :<Polymorphic>   operator() :
  (    <double> :   <double> )
  (    <double> :   <double>, <double> )

 - atan2,  type :<Polymorphic>   operator() :
  (    <double> :   <double>, <double> )

 - atanh,  type :<Polymorphic>   operator() :
  (    <double> :   <double> )

 - buildmesh,  type :<Polymorphic>   operator() :
  (    <Fem2D::Mesh> :   <E_BorderN> )

 - buildmeshborder,  type :<Polymorphic>   operator() :
  (    <Fem2D::Mesh> :   <E_BorderN> )

 - cin,  type :<istream>
 - clock,  type :<Polymorphic>
  (    <double> :   )

 - conj,  type :<Polymorphic>   operator() :
  (    <complex> :   <complex> )

 - convect,  type :<Polymorphic>   operator() :
  (    <double> :   <E_Array>, <double>, <double> )

 - cos,  type :<Polymorphic>  operator() :
  (    <double> :   <double> )
  (    <complex> :   <complex> )

 - cosh,  type :<Polymorphic>   operator() :
  (    <double> :   <double> )
  (    <complex> :   <complex> )

 - cout,  type :<ostream>
```

```
- dumptable,  type :<Polymorphic>   operator() :
 (    <ostream> :   <ostream> )

- dx,  type :<Polymorphic>   operator() :
 (    <LinearComb<MDroit, C_F0>> :   <LinearComb<MDroit, C_F0>> )
 (    <double> :   <std::pair<FEbase<double> *, int>> )
 (    <LinearComb<MGauche, C_F0>> :   <LinearComb<MGauche, C_F0>> )

- dy,  type :<Polymorphic>   operator() :
 (    <LinearComb<MDroit, C_F0>> :   <LinearComb<MDroit, C_F0>> )
 (    <double> :   <std::pair<FEbase<double> *, int>> )
 (    <LinearComb<MGauche, C_F0>> :   <LinearComb<MGauche, C_F0>> )

- endl,  type :<char>
- exec,  type :<Polymorphic>   operator() :
 (    <long> :   <string> )

- exit,  type :<Polymorphic>  operator() :
 (    <long> :   <long> )

- exp,  type :<Polymorphic>  operator() :
 (    <double> :   <double> )
 (    <complex> :   <complex> )

- false,  type :<bool>
- imag,  type :<Polymorphic>   operator() :
 (    <double> :   <complex> )

- int1d,  type :<Polymorphic>   operator() :
 (    <CDomainOfIntegration> :   <Fem2D::Mesh>... )

- int2d,  type :<Polymorphic>   operator() :
 (    <CDomainOfIntegration> :   <Fem2D::Mesh>... )

- intalledges,  type :<Polymorphic>
 operator( :
 (    <CDomainOfIntegration> :   <Fem2D::Mesh>... )

- jump,  type :<Polymorphic>
 operator( :
 (    <LinearComb<MDroit, C_F0>> :   <LinearComb<MDroit, C_F0>> )
 (    <double> :   <double> )
 (    <complex > :   <complex > )
 (    <LinearComb<MGauche, C_F0>> :   <LinearComb<MGauche, C_F0>> )

- label,  type :<long *>
- log,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )
 (    <complex> :   <complex> )

- log10,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )

- max,  type :<Polymorphic>   operator() :
 (    <double> :   <double>, <double> )
 (    <long> :   <long>, <long> )

- mean,  type :<Polymorphic>
 operator( :
 (    <double> :   <double> )
 (    <complex> :   <complex> )

- min,  type :<Polymorphic>  operator() :
 (    <double> :   <double>, <double> )
 (    <long> :   <long>, <long> )

- movemesh,  type :<Polymorphic>   operator() :
 (    <Fem2D::Mesh> :   <Fem2D::Mesh>, <E_Array>... )

- norm,  type :<Polymorphic>
 operator( :
 (    <double> :   <std::complex<double>> )

- nuTriangle,  type :<long>
- nuEdge,  type :<long>
- on,  type :<Polymorphic>   operator() :
 (    <BC_set<double>> :   <long>... )

- otherside,  type :<Polymorphic>
 operator( :
 (    <LinearComb<MDroit, C_F0>> :   <LinearComb<MDroit, C_F0>> )
 (    <LinearComb<MGauche, C_F0>> :   <LinearComb<MGauche, C_F0>> )

- pi,  type :<double>
- plot,  type :<Polymorphic>   operator() :
 (    <long> :  ... )

- pow,  type :<Polymorphic>   operator() :
 (    <double> :   <double>, <double> )
 (    <complex> :   <complex>, <complex> )

- qf1pE,  type :<Fem2D::QuadratureFormular1d>
- qf1pT,  type :<Fem2D::QuadratureFormular>
- qf1pTlump,  type :<Fem2D::QuadratureFormular>
- qf2pE,  type :<Fem2D::QuadratureFormular1d>
- qf2pT,  type :<Fem2D::QuadratureFormular>
- qf2pT4P1,  type :<Fem2D::QuadratureFormular>
- qf3pE,  type :<Fem2D::QuadratureFormular1d>
- qf5pT,  type :<Fem2D::QuadratureFormular>

- readmesh,  type :<Polymorphic>   operator() :
 (    <Fem2D::Mesh> :   <string> )

- real,  type :<Polymorphic>   operator() :
 (    <double> :   <complex> )

- region,  type :<long *>
- savemesh,  type :<Polymorphic>  operator() :
 (    <Fem2D::Mesh> :   <Fem2D::Mesh>, <string>... )

- sin,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )
 (    <complex> :   <complex> )

- sinh,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )
 (    <complex> :   <complex> )

- sqrt,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )
 (    <complex> :   <complex> )

- square,  type :<Polymorphic>    operator() :
 (    <Fem2D::Mesh> :   <long>, <long> )
 (    <Fem2D::Mesh> :   <long>, <long>, <E_Array> )

- tan,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )

- true,  type :<bool>
- trunc,  type :<Polymorphic>   operator() :
 (    <Fem2D::Mesh> :   <Fem2D::Mesh>, <bool> )

- verbosity,  type :<long *>
- wait,  type :<bool *>
- x,  type :<double *>
- y,  type :<double *>
- z,  type :<double *>
```

Bibliography

[1] R. B. Lehoucq, D. C. Sorensen, and C. Yang *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods* SIAM, ISBN 0-89871-407-9 // `http://www.caam.rice.edu/software/ARPACK/`

[2] Babbuška, I: Error bounds for finite element method, Numer. Math. 16, 322-333.

[3] D. Bernardi, F.Hecht, K. Ohtsuka, O. Pironneau: *freefem+ documentation*, on the web at ftp://www.freefem.org/freefemplus.

[4] D. Bernardi, F.Hecht, O. Pironneau, C. Prud'homme: *freefem documentation*, on the web at http://www.asci.fr

[5] Davis, T. A: Algorithm 8xx: UMFPACK V4.1, an unsymmetric-pattern multi-frontal method TOMS, 2003 (under submission) `http://www.cise.ufl.edu/research/sparse/umfpack`

[6] George, P.L: *Automatic triangulation*, Wiley 1996.

[7] Hecht, F: The mesh adapting software: bamg. INRIA report 1998.

[8] Modulef ?????????

[9] J.L. Lions, O. Pironneau: Parallel Algorithms for boundary value problems, Note CRAS. Dec 1998. Also : Superpositions for composite domains (to appear)

[10] B. Lucquin, O. Pironneau: *Scientific Computing for Engineers* Wiley 1998.

[11] J. Nečas and L/ Hlaváček, Mathematical theory of elastic and elasto-plastic bodies: An introduction, Elsevier, 1981.

[12] K. Ohtsuka, O. Pironneau and F. Hecht: Theoretical and Numerical analysis of energy release rate in 2D fracture, INFORMATION **3** (2000), 303–315.

[13] F. Preparata, M. Shamos; *Computational Geometry* Springer series in Computer sciences, 1984.

[14] R. Rannacher: On Chorin's projection method for the incompressible Navier-Stokes equations, in "Navier-Stokes Equations: Theory and Numerical Methods" (R. Rautmann, et al., eds.), Proc. Oberwolfach Conf., August 19-23, 1991, Springer, 1992

[15] Roberts, J.E. and Thomas J.-M: Mixed and Hybrid Methods, Handbook of Numerical Anaysis, Vol.II, North-Holland, 1993

[16] J.L. Steger: The Chimera method of flow simulation, Workshop on applied CFD, Univ of Tennessee Space Institute, August 1991.

[17] Tabata, M: Numerical solutions of partial differential equations II (in Japanese), Iwanami Applied Math., 1994

[18] Thomasset, F: Implementation of finite element methods of Navier-Stokes Equations, Springer-Verlag, 1981

[19] N. Wirth: *Algorthims + Data Structures = Programs*, Prentice Hall, 1976

[20] Bison documentation

[21] Bjarne Stroustrup: The `C++` , programming language, Third edition, Addison-Wesley 1997.

[22] COOOL: a package of tools for writing optimization code and solving optimization problems,

[23] B. Riviere, M. Wheeler, V. Girault, A priori error estimates for finite element methods based on discontinuous approximation spaces for elliptic problems. SIAM J. Numer. Anal. 39 (2001), no. 3, 902–931 (electronic).

[24] COOL package , `http://coool.mines.edu`

[25] M. A. Taylor, B. A. Wingate , L. P. Bos, Several new quadrature formulas for polynomial integration in the triangle , Report-no: SAND2005-0034J, `http://xyz.lanl.gov/format/math.NA/0501496`

Index