Freefem++ Manual

version 1.31 (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

April 28, 2003

[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 numerically these equations. As its name tells, it is public domain software based on the Finite Element Method.

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 require also the interpolation of data on different meshes within one program. Thus `freefem++` can handle these difficulties: *arbitrary finite element spaces on arbitrary unstructured and adaptated meshes*

1.1 History

The project has evolved from `MacFem, PCfem`, written in Pascal. The first version written in C was `freefem 3.4`; it had mesh adaptation but on a single mesh. Later `freefem+ 1.2.10` was the result of a thorough rewriting in C++. It had multiple meshes with interpolation (functions defined on one mesh can be used on any other mesh). Interpolation from one unstructured mesh to another is not easy because it has to be fast and non-diffusive; one has to find for each point the triangle which contains it and that is one of the basic problems of computational geometry (see Preparata & Shamos[8] for example). To do it in the minimum number of operations is the challenge. We use an implementation which is $O(n \log n)$, based on a quadtree.

We are now introducing `freefem++`, an entirely new program written in C++ and based on `bison` for an easy modification of the freefem language.

The freefem language allows for a quick specification of any partial differential system of equations. The language syntax of `freefem++` is redesigned and makes use of STL [13], 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 depend on the version number. So far we have linear and quadratic Lagrangian elements, discontinuous P1 and Raviart-Thomas elements.

1.2 The language

Basically `freefem++` is a compiler, the language is typed, polymorphic and reentrant with macro generation6.5. Every variable must be typed, declared in a statement; each statement separated from the next by a semicolon '`;`'. The following is a simple example whereby the Dirichlet problem with right hand side xy for the Laplace operator is solved inside the unit circle. The finite element method of degree 2 on triangles

is used with a mesh which has 50 points on the unit circle. The linear system is solved by a Gauss LU factorization.

```
border a(t=0,2*pi){x=cos(t); y=sin(t);label=5;};
mesh Th = buildmesh (a(50));
fespace Vh(Th,P2);
Vh u,v;
func f= x*y;
problem laplace(u,v,solver=LU) =
    int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v))                    //    bilinear part
    - int2d(Th)( f*v)                                       //    right hand size part
    + on(5,u=0)  ;                                          //    Dirichlet boundary condition

real cpu=clock();
laplace;                                                    //    SOLVE THE PDE
plot(u);
cout << " CPU = " << clock()-cpu << endl;
```

We have written in blue the reserved words of the language. Later we will not write `pi,x,y,label, solver` in blue because these are reserved variables, the redefinition of which is dangerous, yet allowed.

This example shows some of the characteristics of `freefem++`

- Analytic description of boundaries (by opposition to CSG), therefore the user must specify the intersection points in case two boundaries intersect. By convention the domain is on the left side of the oriented boundary.
- Automatic mesh generator, based on the Delaunay-Voronoi algorithm with inner points generated with a density proportional to the density of points on the boundary (hence refinement is obtained by augmenting the number of boundary points).
- Arbitrary degree of the element, from the pre-programmed library.
- Every variable is typed. For instance `f` is a function, specified by the keyword `func`.
- Online graphics (see the documentation below for zoom, postscript and other commands).
- C++ like syntax.

Chapter 2

Syntax

2.1 Data Types

Every variable must be declared together with its type. 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`) , finite element 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 ∈ ℤ
real pi=4*atan(1);                                             //    pi ∈ ℝ
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); }
```

where `int`, `real`, , `complex` correspond to the C-types `long`, `double`, `complex<double>`.

The scope of a variable is the current block {...} like in `C++` , except the `fespace` variable, and the in variables local to a block are destroyed at the end of the block .

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 ubderscore character _ is not allowed, because it will be used as an operator in the future.

2.1.1 Another Example

```
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=1);
{                                                                  //    new block
  real r= 2;                                                  //    not the same r
  fespace Vh(Th,P1);                                //    error because Vh is a global name
}
                                                               //    here r==0.01
```

Remark 1 *Notice the use of* `wait` *to monitor the time a graph stays on screen (i.e. the time freefem stops before going to the next instruction).*

Generally like in `C++` the variable is destroyed at the the end of block (in {...}) except for the fespace name

2.2 List of major types

bool a C++ true, false value;

int long integer;

string a C++ string;

real double ;

complex complex<double>;

ofstream ofstream to output to a file

ifstream ifstream to input from a file

real[int] array of real (integer index) ;

real[string] array of real (string index) ;

func define a line function without argument `func f=cos(x)+sin(y) ;`. Remark the function's type is given by the expression's type.

mesh to define a mesh `mesh Th=buildmesh(circle(10));`

fespace to define a new type of finite element space. Ex: `fespace Vh(Th,P1);` so far the known finite elements are

- **P0** constant discontinuous finite element
- **P1** linear piecewise continuous finite element
- **P2** P_2 piecewise continuous finite element, where P_2 is the set of polynom of $\mathbb{R}^2$ de degrees two,
- **RT0** the Raviat-Thomas finite element,
- **P1nc** nonconforming P_1 finite element,
- **P1dc** linear piecewise discontinuous finite element
- **P2dc** P_2 piecewise discontinuous finite element indexP2dc
- **P1b** linear piecewise continuous finite element + bublle

To define a function p in this fespace Vh or a array a of 10 functions `Vh p; Vh a[10];`

problem to define a pde problem without solving it.

solve to define a pde problem and solve it.

varf to define a full variational form.

matrix to define a sparce matrix.

2.3 Globals

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

x the x coordinate of current point (real value)

y the y coordinate of current point (real value)

z the z coordinate of current point (real value) , (reserved for future use).

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

region a function which returns an `int`, the region number of the current point (x,y).

P the current point (R^2 value. `P.x`,`P.y`,`P.z`)

N the normal at the current point ($\mathbb{R}^3$) value, `N.x`,`N.y`,`N.z`) .

cout console `ostream`

cin console `istream`

true true `bool`value

false false bool value

pi real approximation 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 MacOs)

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

2.4 Arithmetic

The operators known to the language are the usual C-operators:

```
+ - * / \% ~ ^ | || &   && ! == != < > <= >= = += -= *= /= << >> ++ -- ,
```

with the same meaning as in C++ except for

- ^ which is the power operator (with right precedence (as in math)),
- | & ! are the three boolean operators 'or' , 'and' , 'not',
- and we have added ' .* ./ two array operator (like in matlab or scilab), where
  - ' is unary right transposition of array, matrix
  - .* is the term to term multiply operator.
  - ./ is the term to term divide operator.

  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)

There are automatic casts from a type to an other like in C++ . So in some sense we have

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

and *string* is also a subset of the 4 sets $bool, int, real, complex$ in the sense "make the string of the value".

Example: basics

```
real  x=3.14,y;
int  i,j;
complex c;
cout << " x = " << x << "\n";                                                    //
x = 1;y=2;
x=y;
i=0;j=1;
cout << 1 + 3 << " " <<  1/3 << "\n";
cout << 10 ^10   << "\n";
cout << 10 ^-10   << "\n";
cout << -10^-2+5   << "== 4.99 \n";
cout << 10^-2+5    << "==  5.01 \n";
cout << "------------------ complex ---- \n" ;
cout << 10-10i   << " \n";
cout << " -1^(1/3) = " << (-1+0i)^(1./3.)   << " \n";
cout << " 8^(1/3)= " << (8)^(1./3.)   << " \n";
```

```
cout << " sqrt(-1) = " << sqrt(-1+0i) << " \n";

cout << " ++i =" <<  ++i ;
cout <<  " i=" << i << "\n";
cout << " i++ = "<< i++ << "\n";
cout << " i   = " << i << "\n";
                                                    //    ---- string concatenation ----------
string str,str1;
 str="abc+";
 str1="+abcddddd+";
 str=str + str1;                                                    //    string concatenation
 str = str + 2 ;
 cout << "str=  " << str << "== abc++abcddddd+2;\n";

R3 P;
P.x=1;
x=P.x;
cout <<"P.x = "<< P.x << endl;
```

output displayed on the console:

```
 x = 3.14
4 0
1e+10
1e-10
4.99== 4.99
5.01==  5.01
------------------ complex ----
(10,-10)
 -1^(1/3) = (0.5,0.866025)
 8^(1/3)= 2
 sqrt(-1) = (6.12323e-17,1)
 ++i =1 i=1
 i++ = 1
 i   = 2
str=  abc++abcddddd+2== abc++abcddddd+2;
P.x = 1
```

The usual mathematical functions are implemented:

sqrt, pow, exp, log, sin, cos, atan, cosh, sinh, tanh, min, max , etc...

And usual C++ functions are implemented: **exit, assert**

2.5 Array

There are 2 kinds of arrays: 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. It is also possible to make an array of FE function, with the same syntax
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 texttt' 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;
```

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

```
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)
  {
    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;
```

2.6 Loops

The `for` and `while` loops are implemented, and the semantic is the same as in `C++` with `break` and `continue` keywords.

```
int i;
for (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;}
```

2.7 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, indexofstream!append 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.

Remark 2 *The file is closed at the exit of the enclosing block,*

```
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

The following keywords are discussed in this section:
square, border, buildmesh, movemesh , adaptmesh, readmesh, trunc, triangulate

3.1 Square

For easy and simple testing we have included a constructor for rectangles. All other shapes should be handled with `border+buildmesh`. The following

```
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]);
mesh th=square(4,5);
plot(Th,th,ps="twosquare.eps");
```

constructs a grid $n \times m$ in the rectangle $[1.2, 1.8] \times [0, 1]$ and a grid 4×5 in the unit square $[0, 1]^2$.

Remark 3 *The label of boundaries are* $1, 2, 3, 4$ *for bottom, right, top, left side (before the mapping* $[x0 + (x1 - x0) * x, y0 + (y1 - y0) * y]$ *(the mapping can be non linear).*

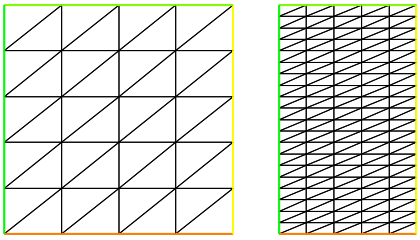

Figure 3.1: The 2 square meshes

3.2 Border

A domain is defined as being on the left (resp right) of its oriented boundary with the parametrization if the sign of the expression that returns the number of vertices on the boundary is positive (resp negative).
remark: the label must be none zero if you you want take boundary condition on this border
For instance the unit circle with a small circular hole inside would be

```
real pi=4*atan(1);
border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
border b(t=0,2*pi){ x=0.3+0.3*cos(t); y=0.3*sin(t);label=2;}
mesh Thwithouthole= buildmesh(a(50)+b(+30));
mesh Thwithhole   = buildmesh(a(50)+b(-30));
plot(Thwithouthole,wait=1,ps="Thwithouthole.eps");                    //   figure 3.2
plot(Thwithhole,wait=1,ps="Thwithhole.eps");                          //    figure 3.3
```

Remark 4 *Notice the use of* `ps="fileName"` *to generate a postscript file identical to the plot shown on screen.*

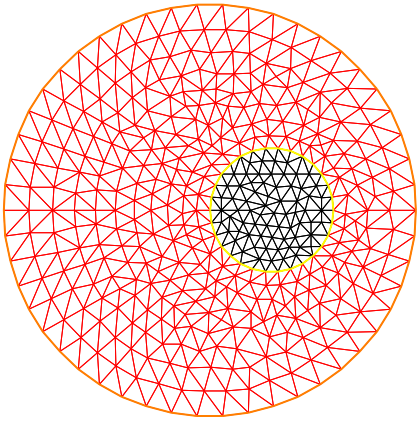

Figure 3.2: mesh without hole

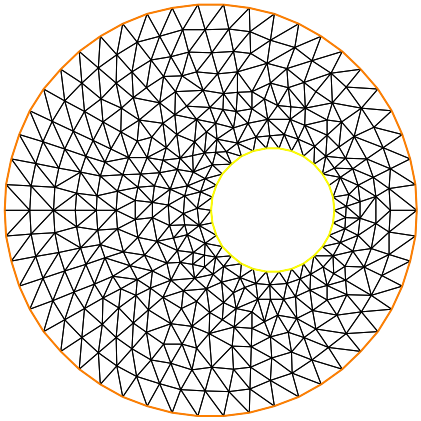

Figure 3.3: mesh with hole

Polygons are best described as unions of straight segments such as below for the $(0,2) \times (0,1)$ rectangle:

```
border a(t=0,2){x=t; y=0;label=1;};                          //    the label can be depend of t
border b(t=0,1){x=2; y=t;label=1;};
border c(t=2,0){x=t; y=1;label=1;};
border d(t=1,0){x=0; y=t;label=1;};
border sq = [ label=1 ,[0,0],[ 0,2],  [1,2,label=2]  [1,0],  [0,0,label=0] ];
int n = 20;
mesh th= buildmesh(a(2*n)+b(n)+c(2*n)+d(n));
mesh Th= buildmesh(sq(8*n));
```

Remark 5 *On the mesh* `Th` *the label of the polygon line with vertices* $|\begin{smallmatrix}0\\0\end{smallmatrix}$, $|\begin{smallmatrix}0\\2\end{smallmatrix}$, $|\begin{smallmatrix}1\\2\end{smallmatrix}$ *is* 1 *, the label of the polygon line of vertices* $|\begin{smallmatrix}1\\2\end{smallmatrix}$, $|\begin{smallmatrix}1\\0\end{smallmatrix}$, $|\begin{smallmatrix}0\\0\end{smallmatrix}$ *, to close the polygon the two labels corresponding are the same.*

Note that boundaries must cross at end points only.

3.3 Movemesh

Mesh motion is allowed as in the next two last statement of this example.

```
real Pi=atan(1)*4;
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");                                    //    figure 3.4
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");                                  //    figure 3.5
```

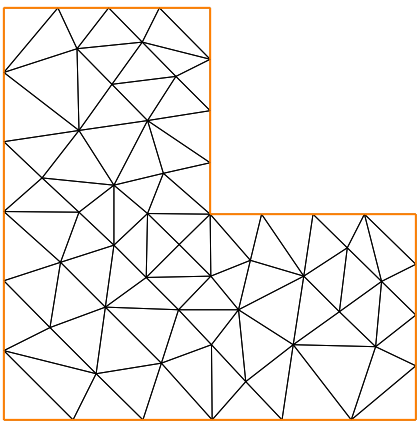

Figure 3.4: L-shape

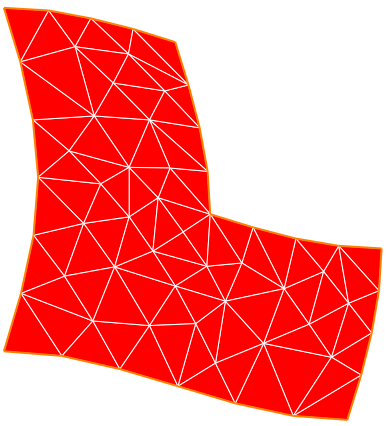

Figure 3.5: moved L-shape

Remark 6 *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. Therefore a statement like u = u will redefine u on the new Th and therefore destroy the old Th if u was the only function using it.*

3.4 Read and writre mesh

Freefem can read and write files which can be reused once read but the names of the borders are lost and they have to be replaced by the number which corresponds to their order of appearance in the program, unless the 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");                                        //    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 [?]
mesh th2 = readmesh("toto.msh");                                             //    read the mesh
```

There are many formats of mesh files available for communication with other tools such as emc2, modulef..., the suffix 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 page.

3.5 Triangulate

Build a triangulation from a set point, this triangulation is a mesh Delaunay of the convex hull of the set of point. This can be usefull to build table function.
The coordinate of the set of point are set in a file like :

```
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
```

here the third number by line is use in the command `triangulate`.
But you can use this to defined a table fonction with rows of the form: `x y f(x,y))`.

```
mesh Thxy=triangulate("xyf");

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

                                                          //    reading the third 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 only read third row
}
plot(fxu);                                                                 //    plot the function
```

3.6 Adaptmesh

Mesh adaptation is a very powerful tool which should be used whenever possible for best results. `freefem++` uses a variable metric/Delaunay automatic mesh algorithm which takes for input one or more functions (see in the following example) and builds a mesh adapted to the second differentiated field of the prescribed function.

```
verbosity=2;
mesh Th=square(10,10,[10*x,5*y]);
fespace Vh(Th,P1);
Vh u,v,zero;

u=0;
u=0;
zero=0;
func f= 1;
func g= 0;
int i=0;
real error=0.1, coef= 0.1^(1./5.);

problem Problem1(u,v,solver=CG,init=i,eps=-1.0e-6) =
    int2d(Th)(  dx(u)*dx(v) + dy(u)*dy(v))
  + int2d(Th) ( v*f )
  + on(1,2,3,4,u=g)  ;

real cpu=clock();

for (i=0;i< 10;i++)
{
  real d = clock();
  Problem1;
  plot(u,zero,wait=1);
  Th=adaptmesh(Th,u,inquire=1,err=error);
  cout << " CPU = " << clock()-d << endl;
  error = error * coef;
} ;

 cout << " CPU = " << clock()-cpu << endl;
```

The method is described in detail in [5]. It has a number of default parameters which can be overwritten for better results.

`hmin=` Sets the value of the minimal edge size. (`val` is of type double precision and default value is related to the size of the domain to be meshed and the precision of the mesh generator).

`hmax=` Sets the value of the maximal edge size. (`val` is of type double precision and the default value is the diameter of the domain to be meshed)

`err=` Sets the level of the P^1 interpolation error (0.01 is the default's value).

`errg=` Sets the value of the relative error on geometry. By default this error is 0.01, and in any case this value must be greater than $1/\sqrt{2}$. Remark that mesh size created by this option can be smaller than the `-hmin` argument due to geometrical constraint.

`nbvx=` Sets the maximal number of vertices generated by the mesh generator (9000 is the default's value).

`nbsmooth=` Set the number of iterations of the smoothing procedure (5 is the default's value).

`nbjacoby=` Set the number of iterations in a smoothing procedure during the metric construction, 0 imply no smoothing (6 is the default's value).

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

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

omega= Set the relaxation parameter of the smoothing procedure (1.0 is the default's value).

iso= Forces the metric to be isotropic or not (false is the default's value).

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

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

otherwire, the metric is evaluated using the criterium of equidistribution of errors. In this case the metric is define by

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

cutoff= Sets the limit value of the relative error evaluation (1.0e-6 is the default's value).

verbosity= Sets the level of printing (verbosity , which can be chosen between 0 and ∞) verbosity, and change the value of the global variable verbosity (obsolete).

inquire= To inquire or not the mesh (false is the default's value).

splitpbedge= If is true then split in two all internal edges with two boundary vertices (true is the default's value).

maxsubdiv= Change the metric such that the maximal subdivision of a background's edge is bound by the `val` number (always limited by 10,and 10 is also the default's value).

rescaling= the function with respect to which the mesh is adapted is re-scaled to be between 0 and 1 (true is the default's value).

keepbackvertices= if true will try to keep as many vertices of the previous mesh as possible (true is the default's value).

isMetric= if it is true the metric is given by hand (false is the default's value). So 1 or 3 functions given defining directly a symmetric matrix field (if one function is given then isotropic mesh size is given directly at every point through a function) , otherwise for all given functions, its Hessian is computed to define a metric.

power= exponent power of the Hessian to compute the metric (1 is the default's value).

3.7 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` exemple 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 the function's support
  u[][i]=0;                                                                  //    reset
 }
```

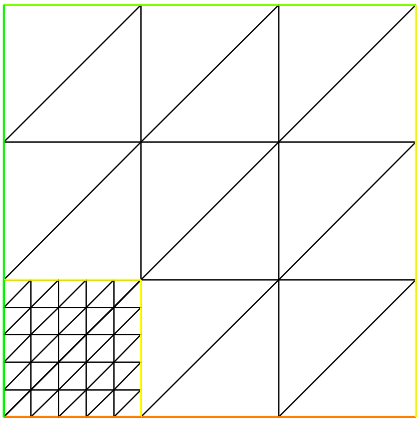

Figure 3.6: mesh of support the function P1 number 0, splitted in 5×5

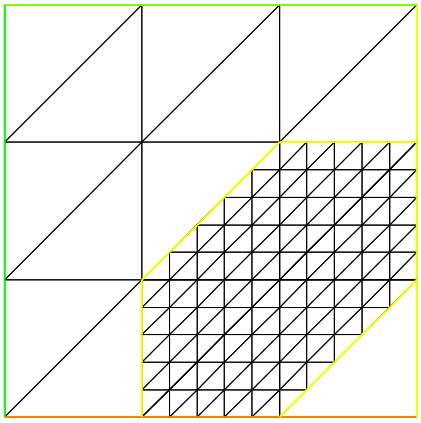

Figure 3.7: mesh of support the function P1 number 6, splitted in 5×5

3.8 Meshing examples

Example: 1: The mesh for a corner singularity The domain is an L-shape:

```
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;};

mesh rh = buildmesh (a(6) + b(4) + c(4) +d(4) + e(4) + f(6));
```

Example: 2: Meshes for domain decompositions To test the domain decomposition algorithms described below we will need 2 overlapping meshes of a single domain.

```
border a(t=0,1){x=t;y=0;};
border a1(t=1,2){x=t;y=0;};
```

```
border b(t=0,1){x=2;y=t;};
border c(t=2,0){x=t ;y=1;};
border d(t=1,0){x = 0; y = t;};
border e(t=0, pi/2){ x= cos(t); y = sin(t);};
border e1(t=pi/2, 2*pi){ x= cos(t); y = sin(t);};
n:=4;
mesh sh = buildmesh(a(5*n)+a1(5*n)+b(5*n)+c(10*n)+d(5*n));
mesh SH = buildmesh ( e(5*n) + e1(25*n) );
plot(sh,SH);
```

Example: 3: Meshes for fluid-structure interactions 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);
```

Chapter 4

Finite Elements

To use a finite element, one needs to define a finite element space with the keyword `fespace` (short of finite element space) like

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

or with k pair of periodic boundary condition

```
@fespace IDspace(IDmesh,<IDFE>,periodic=[[la1,sa1],[lb1,sb1],...,[lak,sak],[lbk,sbk]])
```

where `IDspace` is the name of the space for example `Vh`, `IDmesh` is the name of the associated mesh and `<IDFE>` is a identifier of finite element type, where a pair of periodic boundary condition is defined by `[lai,sai],[lbi,sbi]`. The `int` expressions `lai` and `lbi` are defined the 2 labels of the piece of the boundary to be equivalence, and the `real` expressions `sai` and `sbi` a give two common abcissa on the two boundary curve.
As of today, the known types of finite element are

P0 piecewise constante discontinuous finite element

$$P0_h = \{v \in L^2(\Omega)/\forall K \in \mathcal{T}_h \quad v_{|K} = \alpha_K \text{constant of } \mathbb{R}\} \tag{4.1}$$

P1 piecewise linear continuous finite element

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

P1dc piecewise linear continuous finite element

$$P1_h = \{v \in L^2(\Omega)/\forall K \in \mathcal{T}_h \quad v_{|K} \in P_1\} \tag{4.3}$$

P1b piecewise linear continuous finite element plus bubble

$$P1_h = \left\{v \in L^2(\Omega)/\forall K \in \mathcal{T}_h \quad v_{|K} \in P_1 \bigoplus Span\{\Pi_{i=0}^2 \lambda_i^K\}\right\} \tag{4.4}$$

where λ_i^K is the barycentric coordinate fonction of the traingle K

P2 piecewise P_2 continuous finite element,

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

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

P2dc piecewise P_2 discontinuous finite element,

$$P2_h = \{v \in L^2(\Omega)/\forall K \in \mathcal{T}_h \quad v_{|K} \in P_2\} \tag{4.6}$$

RT0 Raviart-Thomas finite element

$$RT0_h = \{\mathbf{v} \in H(div)/\forall K \in \mathcal{T}_h \quad \mathbf{v}_{|K}(x,y) = \left|\begin{smallmatrix}\alpha_K\\ \beta_K\end{smallmatrix}\right. + \gamma_K \left|\begin{smallmatrix}x\\ y\end{smallmatrix}\right. \} \tag{4.7}$$

where $H(div)$ is the set of function of $L^2(\Omega)$ with divergence in $L^2(\Omega)$, and where $\alpha_K, \beta_K, \gamma_K$ are real numbers.

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

To define the finite element spaces

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

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

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

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

where $\mathcal{T}$ is a mesh 10×10 of the unit square $]0,1[^2$,

the corresponding `freefem++` definitions are:

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

so `Xh,Mh,Rh` are finite element spaces (called FE spaces). Now to use functions $u_h, v_h \in X_h$ and $p_h, q_h \in M_h$ and $U_h, V_h \in \mathbb{R}_h$ one can define the FE function like this

```
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

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

.
Like in the previous version, `freefem+`, the finite element functions (type `FE` functions) are both functions from $\mathbb{R}^2$ to $\mathbb{R}^N$ and arrays of real.
To interpolate a function, one writes

```
 uh = x^2 + y^2;                                                    //    ok uh is scalar FE function
[Uxh,Uyh] = [sin(x),cos(y)];                                  //    ok vectorial FE function
Uxh = x;              //    error:  impossible to set only 1 component of a vector FE function.
vh = Uxh;                                                                                  //    ok
Th=square(5,5);
vh=vh;                                    //    re-interpolates vh on the new mesh square(5,5);
vh([x-1/2,y])= x^2 + y^2;                                //    interpole vh = ((x − 1/2)^2 + Y^2)
```

To get the value at a point $x = 1, y = 2$ of the FE function `uh`, or `uh,[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;                          //    maximal 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
```

The other way to set a FE function is to solve a 'problem' (see below).

Remark 7 *It is possible to change a mesh to do a convergence test for example, but see what happens in this trivial example. In fact a FE function is three pointers, one pointer to the values, second a pointer to the definition of* `fespace`*, third a pointer to the* `fespace`*. This* `fespace` *is defined when the FE function is set with operator* `=` *or when a* `problem` *is solved.*

```
mesh Th=square(2,2);
fespace Xh(Th,P1);
 Xh uh,vh;
 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
                                                                       //    and now uh use the 5x5 mesh
                                                                       //    but the fespace of vh is alway the 2x2 mesh
 plot(vh,ps="onoldmesh.eps");                                          //    figure 4.1
 vh = vh;                                                              //    do a interpolation of vh (old) of 5x5 mesh
                                                                       //    to get the new vh on 10x10 mesh.
 plot(vh,ps="onnewmesh.eps");                                          //    figure 4.2
```

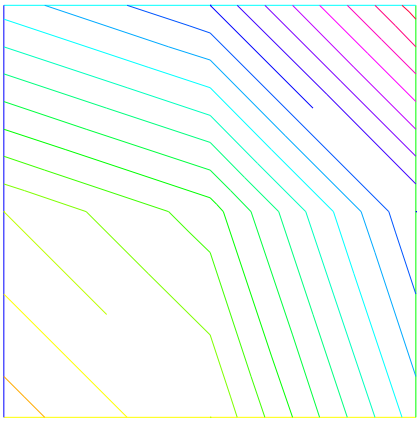

Figure 4.1: vh Iso on mesh 2×2

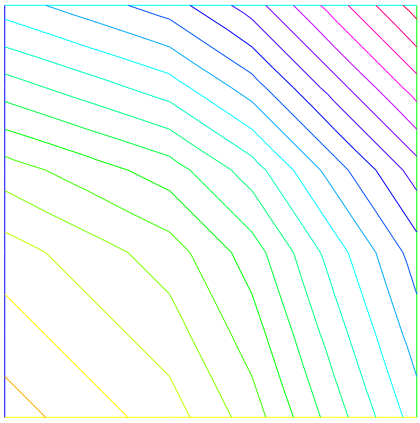

Figure 4.2: vh Iso on mesh 5×5

4.1 Problem and solve

For `freefem++` a problem must be given in variational form, so a bilinear form, a linear form, and possibly a boundary condition form must be input. For example consider the Dirichlet problem:

$$-\Delta v = 1 \text{ in } \Omega = \{(x,y) \in]0,1[^2\}, \quad v = 0 \text{ on } \Gamma = \partial\Omega.$$

The problem can be solved by the finite element method, namely:
Find $u_h \in V_{0h}$ the space of continuous piecewise linear functions on a triangulation of Ω which are zero on the boundary $\partial\Omega$ such that

$$\int_\Omega \nabla u_h \cdot \nabla w_h = \int_\Omega w_h \quad \forall w_h \in V_{0h}$$

The `freefem++` version of the same is

```
mesh Th=square(10,10);
fespace Vh(Th,P1);                                          //    P1 FE space
Vh uh,vh;                                                 //    unknown and test function.
func f=1;                                                   //    right hand side function
func g=0;                                                 //    boundary condition function

solve laplace(uh,vh) =                                  //    definion of the problem and solve
   int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) )                        //    bilinear form
 + int2d(Th)( -f*vh )                                                //    linear form
 + on(1,2,3,4,uh=g) ;                                     //    a lock boundary condition form

f=x+y;
laplace;                                             //    solve again the problem with this new f
plot(uh,ps="Laplace.eps",value=true);                             //    to see the result
```

Remark 8 *Using the keywork* `problem` *in place of* `solve` *would define the problem only and not solve it.*

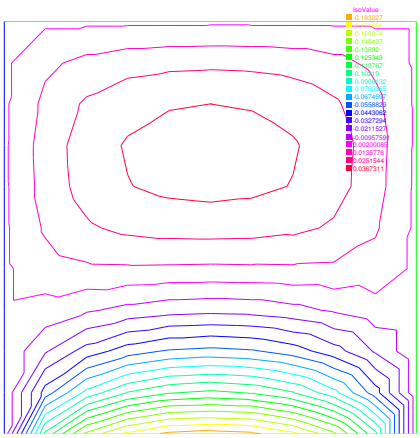

Figure 4.3: Isovalues of the solution

A laplacian in mixed finite formulation .

```
mesh Th=square(10,10);
fespace Vh(Th,RT0);
fespace Ph(Th,P0);

Vh [u1,u2],[v1,v2];
Ph p,q;

problem laplaceMixte([u1,u2,p],[v1,v2,q],solver=LU,eps=1.0e-30) =
   int2d(Th)( p*q*1e-10+ u1*v1 + u2*v2 + p*(dx(v1)+dy(v2)) + (dx(u1)+dy(u2))*q )
 + int2d(Th) ( q)
 + int1d(Th)( (v1*N.x +v2*N.y)/-2);                                    //    int on gamma

 laplaceMixte;                                                 //    the problem is now solved
 plot([u1,u2],coef=0.1,wait=1,ps="lapRTuv.eps",value=true);                  //    figure 4.4
 plot(p,fill=1,wait=1,ps="laRTp.eps",value=true);                            //    figure 4.5
```

Remark 9 *To make programs more readable we stop now using blue color on dx,dy.*

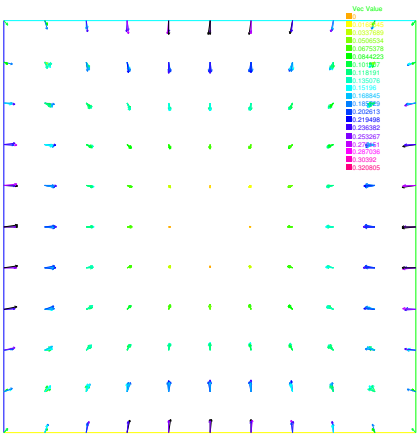

Figure 4.4: Flux (u_1, u_2)

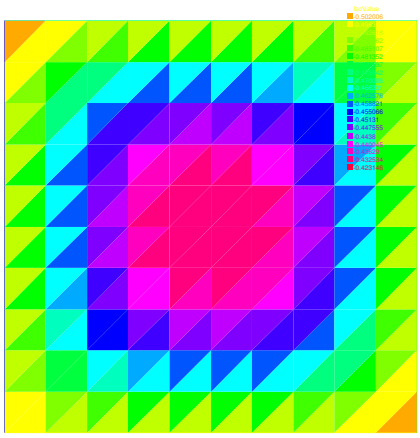

Figure 4.5: Isovalue of p

4.2 Parameter Description for `solve` and `problem`

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

Remark 10 *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).*

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` ...

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` 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, for lock boundary conditions

dimKrylov= dimension of the Krylov space.

4.3 Problem definition

Below `v` is the unknown function and `vv` 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.
- A bilinear form `int1d(Th)( K*v*vv)` , `int1d(Th,2,5)( K*v*vv)` , a sparce matrix of type `matrix`
- A linear form `int1d(Th)( K*v)` , `int1d(Th,2,5)( K*v)` , a vector of type `real[int]`
- The boundary condition form
  - An "on" form (for Dirichlet) : `on(1, u = g )`
  - a linear form on Γ (for Neumann) `int1d(Th))( -f*vv)` or `int1d(Th,3))( -f*vv)`
  - a bilinear form on Γ or Γ_2 (for Robin) `int1d(Th))( K*v*vv)` or `int1d(Th,2))( K*v*vv)`.

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 fonction can be different in the case of linear form.

Remark 11 *`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*vv - f*vv` .

4.4 Integrals

There are two kinds of integrals:

- surface integral defined with the keyword `int2d`
- integrals on curves `int1d`.

Integrals can be used to define the variational form, or to compute integrals proper. It is possible to choose the order of the integration formula by adding a parameter `qforder=` to define the order of the Gauss formula, or directly the name of the formula with `qft=name` in 2d integrals and `qfe=name` in 1d integrals.
The integration formulae on triangles are:

name (`qft=`)	on	order `qforder=`	exact	number of quadrature points
`qf1pT`	triangle	2	1	1
`qf2pT`	triangle	3	2	3
`qf3pT`	triangle	6	4	7

The integration formulae on edges are:

name (`qfe=`)	on	order `qforder=`	exact	number of quadrature points
`qf1pE`	segment	2	1	1
`qf2pE`	segment	3	2	2
`qf3pE`	segment	6	5	3

4.5 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);
Xh u1,u2,v1,v2;
Mh p,q,ppp;

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) ;
```

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

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

matrix A= a(Xh,Xh,solver=CG);                                    //    the Laplace matrix
matrix Bx= bx(Xh,Mh);                                  //    Bx = (Bx_ij) and Bx_ij = bx(b_j^x, b_j^m)
                                          //    where b_j^x is a basis of Xh, and b_j^m is a basis of Mh.
matrix By= by(Xh,Mh);                                  //    By = (By_ij) and By_ij = by(b_j^x, b_j^m)
```

Remark 12 *The line of the matrix corresponding to test function on the bilinear form.*

Remark 13 *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 Γ3 boundary see following remark
u1[] = A^-1*b[];                                                //    solve the linear system
```

Remark 14 *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* `StokesUzawa.edp` *6.6.2 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) ;
```

4.6 Eigen value and eigen vector

This section depend of your FreeFem++ compilation process (see `README_arpack`), to compile 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 base on the `arpack++` [1] the object-oriented version of ARPACK eigenvalue package [**?**, arpack] 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.

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 $\mathbb{R} \times H_0^1(\Omega)$

$$\int_\Omega \nabla u_\lambda \nabla v = \lambda int_\Omega uv \quad \forall v \in H_0^1(\Omega)$$

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
//    ---------------------------------------
```

[1] `http://www.caam.rice.edu/software/ARPACK/`

```
//    we use the inverse shift mode
//    the shift is given with the real sigma
//    -------------------------------------
//    find λ and u_λ ∈ H_0^1(Ω) such that:
//                                  ∫_Ω ∇u_λ∇v = λ ∫_Ω u_λ v, ∀v ∈ H_0^1(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 maximal 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 exemple 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 ---
```

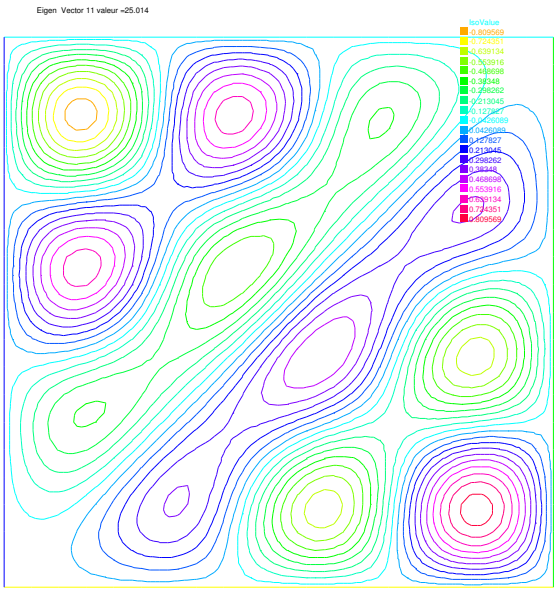

Figure 4.6: Isovalue of 11th eigenvector $u_{4,3} - u_{3,4}$

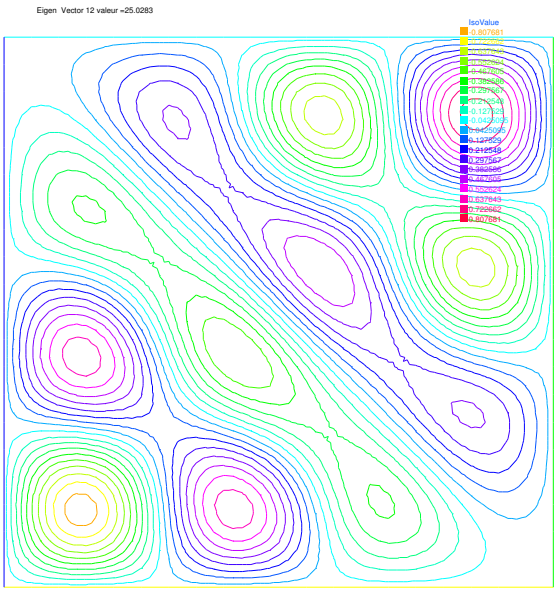

Figure 4.7: Isovalue of 12th eigenvector $u_{4,3} + u_{3,4}$

4.7 Plot

With the command plot, meshes, isovalues and vector fields can be displayed.
The parameters of the plot command can be , meshes, FE functions , arrays of 2 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, the character event can be

- **+** 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,
- **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.

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.

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 4.8
 plot(Th,uh,[uh,vh],bb=[[0.1,0.2],[0.5,0.6]],wait=true);                      //    a zoom
 plot([xx,yy],ps="likegnu.eps",wait=true);                                 //    figure 4.9
```

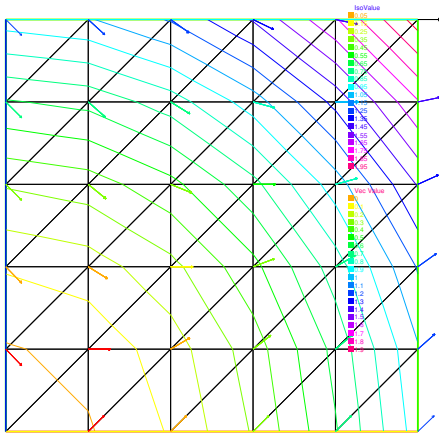

Figure 4.8: mesh, isovalue, and vector

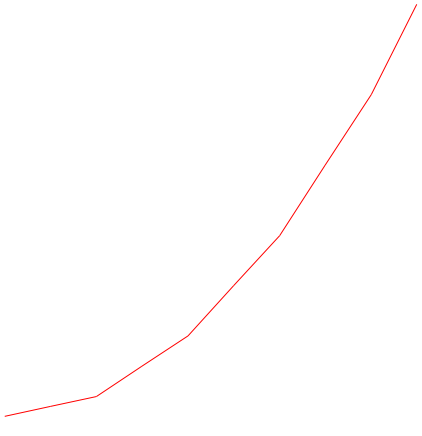

Figure 4.9: Plots a cut of uh. Note that a refinement of the same can be obtained in combination with gnuplot

4.8 Convect

This operator performs one step of backward convection by the method of Characteristics-Galerkin. An equation like

$$\partial_t \phi + u \nabla \phi = 0, \quad \phi(x,0) = \phi^0(x)$$

is approximated by

$$\frac{1}{\delta t}(\phi^{n+1}(x) - \phi^n(X^n(x))) = 0$$

Roughly the term,$\phi^n o X^n$ is approximated by $\phi^n(x + u^n(x)\delta t)$. Up to quadrature errors the scheme is unconditionnally stable. The syntax is

```
<FE> = convect([<exp1>,<exp2>],<exp3>,<exp4>)
```

<FE> is a name of finite element function to store the result $u \circ \chi$;

<exp1> is real expression of the x-velocity,

<exp2> is real expression of the y-velocity,

<exp3> is **minus**[2] the time step,

<exp4> is the name of the finite element function which is convected (u in the exemple above)

Warning `convect` is a non-local operator; in the instruction "phi=convec([u1,u2],-dt,phi0)" every values of phi0 are used to compute phi So `phi=convec([u1,u2],-dt,phi0)` won't work.

[2]The minus is due to the backwark schema

Chapter 5

algorithm

5.1 conjugate Gradient

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 from $\mathbb{R}^n$ to $\mathbb{R}$.
if the function is convexe we can use the conjugate gradient to solve the problem, an 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.
Two version are available:

`linearCG` linear case , the functional J is quadratic.

`NLCG` non linear case (the function is just convexe).

The named parameter of this two function are:

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

`precon=` set the preconditionner function (`P` for exemple) 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 et 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 usefull in loop).

Example of use:

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

func real[int] mat(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
};

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

b=1; x=0;                                          //    set right hand side and initial gest
```

```
LinearCG(mat,x,eps=1.e-6,nbiter=20,precon=matId);
cout << x;
                                                                    //    verification
for (int i=0;i<x.n;i++)
    assert(abs(x[i]*(i+1) - b[i]) < 1e-5);
                                                                                  //
b=1; x=0;                                                  //    set right hand side and initial gest
NLCG(mat,x,eps=1.e-6,nbiter=20,precon=matId);
```

5.2 Optimization

Two algorithms of COOOL a package [14] are interfaced. the Newton Raphson method (call `Newton`) and the `BFGS` method. Be careful this algorithm 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,mat,x,eps=1.e-6,nbiter=20,nbiterline=20);
  cout << "BFGS: J(x) = " << J(x) << " err=" << error(x,b) << endl;
```

Chapter 6

More examples

6.1 A_tutorial.edp

Consider the problem

$$-\Delta v = 1 \text{ in } \Omega = \{(x,y) \in R^2 : x^2 + y^2 \leq 1\}, \quad v = 0 \text{ on } \Gamma = \partial\Omega.$$

The problem is solved by the finite element method, namely:
Find $u \in V$ the space of continuous piecewise linear functions on a triangulation of Ω which are zero on the boundary $\partial\Omega$ such that

$$\int_\Omega \nabla u \cdot \nabla w = \int_\Omega w \quad \forall w \in V$$

The first thing to do is to prepare the mesh (i.e. the triangulation) ; that is done by first defining the border (the unit circle) with label one and then call the mesh generator (buildmesh) with the right orientation of the border (by definition Ω is on the left side of the oriented Γ, the).

```
border a(t=0,2*pi){ x = cos(t); y = sin(t);label=1;};
mesh disk = buildmesh(a(50));
plot(disk);                                                    //    to see the mesh
```

The second thing is to define the continuous piecewise linear functions spaces.

```
fespace femp1(disk,P1);                                        //    define the femp1 space
femp1 u,v;                                          //    introduice the function and test function
```

Next, `freefem++` will define the PDE discretized by FEM in variational form with the following instruction, and solve the problem

```
  problem laplace(u,v) =                          //    u is the unknown and v is the test function
    int2d(disk)( dx(u)*dx(v) + dy(u)*dy(v) )                    //    bilinear form
  + int2d(disk)( -1*v )                                          //    linear form
  + on(1,u=0) ;                                                //    boundary condition

laplace;                                                       //    solve the problem
```

Next we can check that the result is correct. Here we display the result first and then display the error field and compute the L^2 error and the H^1 error

```
plot (u,value=true,wait=true);                          //    to see the value of isoline and wait
femp1 error=u-(1-x^2-y^2)/4;                  //    you only plot FE function, so do interpolation
plot(error,value=true,wait=true);                                //    plot the error

cout << "error L2=" << sqrt(int2d(disk)( (u-(1-x^2-y^2)/4) ^2) )<< endl;
cout << "error H10=" << sqrt(   int2d(disk)((dx(u)+x/2)^2)
                             + int2d(disk)((dy(u)+y/2)^2))<< endl;
```

For better results we can use mesh adaptation. This module constructs a mesh which fits best a function of V, so u is the main argument of `adaptmesh`. Note that adaptmesh "improves" a mesh, so it requires also the name of a mesh for argument. Therefore mesh adaptation is done in `freefem++` by

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

where `disk` is now a new mesh adapted to u.
To check that this mesh is better, we solve the problem again and compute the errors. Notice the improvement!

```
laplace;
plot (u,value=true,wait=true);
err =u-(1-x^2-y^2)/4;
plot(err,value=true,wait=true);
cout << "error L2=" << sqrt(int2d(disk)( (u-(1-x^2-y^2)/4) ^2) )<< endl;
cout << "error H10=" << sqrt(  int2d(disk)((dx(u)+x/2)^2)
                            + int2d(disk)((dy(u)+y/2)^2))<< endl;
```

Output seen on the console:

```
  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 Mortars 0
    Number of Edges                = 676
    Number of Boundary Edges       = 50
    Number of Mortars  Edges       = 0
    Nb Of Mortars with Paper Def   = 0 Nb Of Mortars = 0
    Euler Number nt- NbOfEdges + nv = 1= Nb of Connected Componant - Nb Of Hole
 min xy -1 -0.998027 max xy1 0.998027
 Nb Of Nodes = 243
 Nb of DF = 243
 -- Solve :           min 5.343e-32  max 0.249999
 -- borne de la function  (DF)-0.000890628 0.000858928
 min xy -1 -0.998027 max xy1 0.998027
 min xy -1 -0.998027 max xy1 0.998027
error L2=0.00211901
error H10=0.0383498
  --  mesh:  Nb of Triangles =   1535, Nb of Vertices 813
   Nb of edges on Mortars  = 0
   Nb of edges on Boundary = 89, neb = 89
    Nb Mortars 0
    Number of Edges                = 2347
    Number of Boundary Edges       = 89
    Number of Mortars  Edges       = 0
    Nb Of Mortars with Paper Def   = 0 Nb Of Mortars = 0
    Euler Number nt- NbOfEdges + nv = 1= Nb of Connected Componant - Nb Of Hole
 min xy -0.999441 -0.999752 max xy1 0.999828
 Nb Of Nodes = 813
 Nb of DF = 813
 -- Solve :           min 3.18031e-32  max 0.249946
 min xy -0.999441 -0.999752 max xy1 0.999828
 -- function's bound   -0.000303323 0.000402198
 min xy -0.999441 -0.999752 max xy1 0.999828
error L2=0.000585005
error H10=0.0189227
```

6.2 Periodic

Solve of the Laplace equation

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

on a square $]0, 2\pi[^2$ with bi-periodic boundary condition.

```
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 abcissa
                                           //    1 and 2 are bottom and upper side with x 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);
```

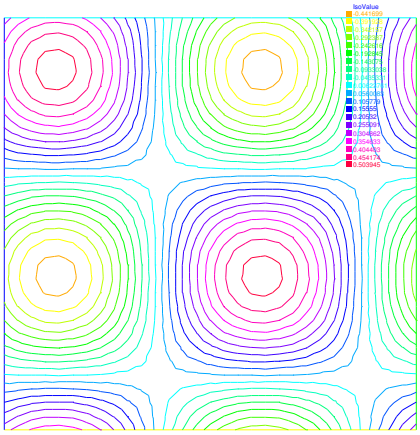

Figure 6.1: The isovalue of solution u with periodic boundary condition

6.3 Adapt.edp

Here we use more systematically the mesh adaptation to track the singularity at an obtuse angle of the domain.
The domain is L-shaped and defined by a set of connecting segments a, b, c, d, e, f labeled $1, 2, 3, 4, 5, 6$.

```
border a(t=0,1.0){x=t;   y=0;  label=1;};
border b(t=0,0.5){x=1;   y=t;  label=2;};
border c(t=0,0.5){x=1-t; y=0.5;label=3;};
border d(t=0.5,1){x=0.5; y=t;  label=4;};
border e(t=0.5,1){x=1-t; y=1;  label=5;};
border f(t=0.0,1){x=0;   y=1-t;label=6;};
mesh Th = buildmesh (a(6) + b(4) + c(4) +d(4) + e(4) + f(6));
fespace Vh(Th,P1);
plot(Th,ps="th.eps");
```

Here `plot` has an extra parameter `ps="th.eps"`. Its effect is to create a postscript file named "th.eps" containing the triangulation `th` displayed during the execution of the program.

Then we write the triangulation data on disk with `savemesh` and `th` for argument and a file name, here `th.msh`

```
savemesh(th,"th.msh");                                    //    saves mesh th in freefem format
```

There are several formats available to store the mesh.

Now we are going to solve the Laplace equation with Dirichlet boundary conditions. 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`).
Next we solve he same problem on an adapted (and finer) mesh 4 times:

```
fespace Vh(Th,P1);                                                    //    set FE space
Vh u,v;                                                  //    set unknown and test function
real error=0.1;                                                       //    level of error
problem Probem1(u,v,solver=CG,eps=1.0e-6) =
    int2d(Th)(  dx(u)*dx(v) + dy(u)*dy(v))
  + int2d(Th) ( -v*1 )
  + on(1,2,3,4,5,6,u=0)  ;
int i;                                                          //    declare loop index
for (i=0;i< 4;i++)
{
  Probem1;
  Th=adaptmesh(Th,u,err=error);
  error = error/2;
} ;
```

after each solve a new mesh adapted to u is computed. To speed up the adaptation we change by hand a default parameter of `adaptmesh:err`, which specifies the required precision, so as to make the new mesh finer.

In practice the program is more complex for two reasons

- We must use a dynamic name for files if we want to keep track of all iterations. This is done with the concatenation operator $+$. for instance

```
for(i = 0; i< 4;i++)
    savemesh("th"+i+".msh",th);
```

  saves mesh th four times in files `th1.msh`,`th2.msh`,`th3.msh`, `th3.msh`.

- There are many default parameters which can be redefined either throughout the rest of the program or locally within `adaptmesh`. The list with their default value is in section 3.6.

6.4 Algo.edp

We propose the 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 $H_0^1(\Omega)$. and where f is defined by

$$f(x) = a * x + x - ln(1 + x), \quad f'(x) = a + \frac{x}{1+x}, \quad f''(x) = \frac{1}{(1+x)^2}$$

6.4.1 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) = ∫_Ω f(|∇u|^2) − ∫ Ωub
                                        //     f(x) = a ∗ x + x − ln(1 + x),   f'(x) = a + x/(1+x),   f''(x) = 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(|∇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 the solve the problem, is make 10 iteration of the conjugate gradient, recompute de preconditonner 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");
```

Remark: 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.

6.4.2 Newton Ralphson algorithm

Now, we solve the problem with Newton Ralphson algorithm, to solve the Euler problem $\nabla J(u) = 0$ the algorithme 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 <<  " residu^2 = " <<  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");
```

6.5 Nonlinear elasticity

The nonlinear elasticity problem is find the de-placement (u, v) minimizing J

$$minJ(u,v) = \int f(F2) - \int g * vn$$

where $F2(u,v) = A([u,v],[u,v])$ and $A([u,v],[w,s])$ is bilinear in $[u,v],[w,s]$ sym. positive. and where f is a given $\mathcal{C}^2$ function.
The differential of J is

$$DJ(u,v)(w,s) = \int 2A(u,v,w,s)df(F2(u,v))) - \int gs$$

$$D^2J(u,v)((w,s),(uu,vv)) = \int 2A([uu,vv],[w,s])f'(F2(u,v)))+4A([u,v],[w,s])A([u,v],[uu,vv])f''(F2(u,v))$$

The Newton Method is
choose $n = 0$,u_n, v_n initial displacement

- loop:
- find (du, dv) : solution of

$$D^2J(u_n,v_n)((w,s),(du,dv)) = DJ(u_n,v_n)(w,s), \quad \forall w,s$$

- $un- = du, vn = -dv$
- 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 the begin of the comment //.

```
real cc=6.4337*10^6;

                                                        //    the function f and derivative --
macro f(u) (cc*(exp(u)-1)) //    end of macro f
macro df(u) (cc*(exp(u))) //    end of macro df = f'
macro ddf(u) (cc*(exp(u))) //    end of macro ddf = f''f

                                                                  //    the bilineare form
real aqq=0.7472,azz=1.6372,aqz=-0.2631;
macro A(u,v,w,s)  (2*aqq*u*w/x/x+2*azz*dy(v)*dy(s)+aqz*u*dy(s)/x) //    end of macro
macro F2(u,v)  A(u,v,u,v) //    end of macro
```

Now, the Newton algorithme is:

```
Wh f2;                                                              //    to store F2
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^2 J(un) ) part
             2*A(uu,vv,w,s)*df(f2)
          +  4*A(un,vn,w,s)*A(un,vn,uu,vv)*ddf(f2)
             )
       -int2d(th,qforder=1)(                                    //    (DJ(un)) part
             2*A(un,vn,w,s)*df(f2) )
             + int1d(th,4) (-Pa*r1*w)
+ on(1,3,vv=0);

                                                                  //    Newton's method
                                                                  //    ---------------
for (int i=0;i<10;i++)
{
  cout << "Loop " << i << endl;
  f2 = F2(un,vn) ;                                                //    optimization
  NonLin;                                          //    compute w = (D^2 J(un))^-1 (DJ(un))
  w[]   = M*uu[];
  real res = w[]' * uu[];
  cout << " residu = " << res << endl;
  if (res<1e-5) break;
}
```

6.6 Stokes and Navier-Stokes

The Stokes equations are:

$$\left.\begin{array}{rl} -\Delta u + \nabla p & = 0 \\ \nabla \cdot u & = 0 \end{array}\right\} \quad \text{in } \Omega \tag{6.1}$$

where u is the velocity vector and p the pressure. For simplicity, let us choose Dirichlet boundary conditions on the velocity, $u = u_\Gamma$ on Γ.
A classical way to discretize the Stokes equation with a mixed formulation, is to solve the variational problem and then discretize it:

Find $(u_h, p_h) \in X_h^2 \times M_h$ such that $u_h = u_{\Gamma h}$, and such that

$$\begin{aligned} \int_{\Omega_h} \nabla u_h \cdot \nabla v_h + \int \nabla p_h \cdot v_h &= 0, \quad \forall v_h \in X_{0h} \\ \int_{\Omega_h} \nabla . u_h q_h &= 0, \quad \forall q_h \in M_h \end{aligned} \tag{6.2}$$

where X_{0h} is the space of functions of X_h which are zero on Γ. The velocity space is approximated by X_h space, and the pressure space is approximated by M_h space.

6.6.1 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.
The driven cavity problem is the problem (6.1) where $u_\Gamma \cdot n = 0$ and $u_\Gamma \cdot s = 1$ on the top boundary and zero elsewhere (n is the Γ normal, and s is the Γ tangent).
The mesh is constructed by

```
mesh Th=square(8,8);
```

The labels assigned by `square` to the bottom,right,up and left edges are respectively $1, 2, 3, 4$.
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

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

and

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

The FE spaces and functions are constructed by

```
fespace Xh(Th,P2);
fespace Mh(Th,P1);
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 (6.2) 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);
```

Each unknown has its own boundary conditions.

Technical Remark There is some arbitrary decision here as to where to affect the boundary condition within the linear system. Basically the Dirichlet operator (`on`) should be associated with the unknown which contains it so that the penalization appears on the diagonal of the matrix of the underlying discrete linear system, otherwise it will be ill conditioned.

Remark 15 *Notice the term* `p*q*(0.000001)` *is added, because the solver Crout needs it: all the sub-matrices must be invertible.*

If the streamlines are required, they can be computed by finding ψ such that 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

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \Delta u + \nabla p = 0, \quad \nabla \cdot u = 0$$

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

$$\begin{array}{ll} \frac{1}{\delta t}(u^{n+1} - u^n \circ X^n) - \nu\Delta u^{n+1} + \nabla p^{n+1} & = 0, \\ \nabla \cdot u^{n+1} & = 0 \end{array} \tag{6.3}$$

The term,$u^n \circ X^n(x) \approx u^n(x - u^n(x)\delta t)$ 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] et p  ",p,[u1,u2]);
 } ;
```

Notice that the matrices are reused (keyword `init=i`)

6.6.2 StokesUzawa.edp

In this example we have a full Stokes problem, solve also the cavity problem, with the classical Uzawa conjugate gradient.
The idea of the algorithm is very simple, in the first equation of the Stokes problem, if we know the pressure, when we can compute the velocity $u(p)$, and to solve the problem is to find p, such that $\nabla.u(p) = 0$. The last problem is linear, symmetric negative, so we can use the conjugate gradient algoritm .
First we define mesh, and the Taylor-Hood approximation. So X_h is the velocity space, and M_h is the pressure space.

```
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);
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;
```

Construct the function `divup` $p \longrightarrow \nabla.u(p)$.

```
func real[int] divup(real[int] & pp)
{
                                                                                    //    compute u1(pp)
   b[]  = Bx'*pp; b[] += bc1[] ;    u1[] = A^-1*b[];
                                                                                    //    compute u2(pp)
   b[]  = By'*pp; b[] += bc2[] ;    u2[] = A^-1*b[];
                                                          //    div(u1,u2) = Bx'*u1[] + By'*u2[];
   ppp[] =   Bx*u1[];                                                                //    ppp = ^tBx u1
   ppp[] +=  By*u2[];                                                                //      +^tBy u2
   return ppp[] ;
};
```

Call now the conjugate gradient algorithm:

```
p=0;q=0;
LinearCG(divup,p[],eps=1.e-6,nbiter=50);
divup(p[]);                                                    //    compute the final solution

plot([u1,u2],p,wait=1,value=true,coef=0.1);
```

6.6.3 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 precondioner 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.

```
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);
 }
```

6.7 Readmesh.edp

Freefem can read and write files which can be reused once read but the names of the borders are lost and they have to be replaced by the number which corresponds to their order of appearance in the program, unless the 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");                                    //    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 [?]
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);
```

There are many formats of mesh files available for communication with other tools such as emc2, modulef..., the suffix 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 page.
Note also the wrong sign in the Laplace equation, but freefem can handle it as long as it is not a resonance mode (i.e. the matrix of the linear system should be non-singular).

6.8 Domain decomposition

We present, three classique exemples, 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.

6.8.1 Schwarz-overlap.edp

To solve

$$-\Delta u = f, \quad \text{in } \Omega = \Omega_1 \cup \Omega_2 \quad u|_\Gamma = 0$$

the Schwarz algorithm runs like this

$$-\Delta u_1^{m+1} = f \text{ in } \Omega_1 \quad u_1^{m+1}|_{\Gamma_1} = u_2^m$$

$$-\Delta u_2^{m+1} = f \text{ in } \Omega_2 \quad u_2^{m+1}|_{\Gamma_2} = u_1^m$$

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.

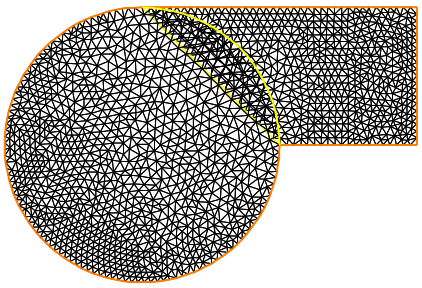

Figure 6.2: The 2 overlapping mesh `TH` and `th`

```
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);
};
```

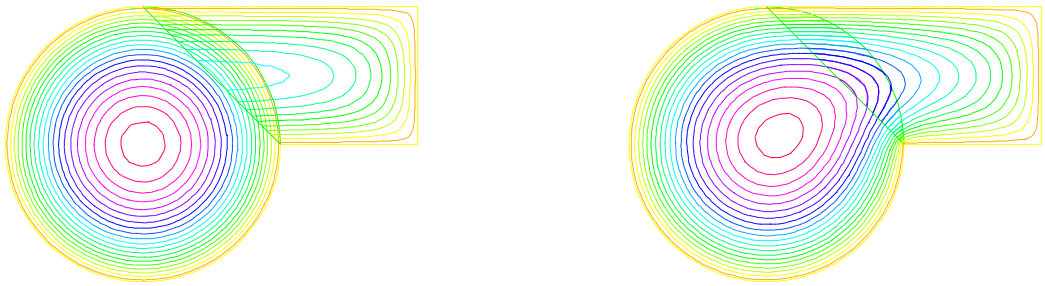

Figure 6.3: Isovalues of the solution at iteration 0 and iteration 9

6.8.2 Schwarz-no-overlap.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

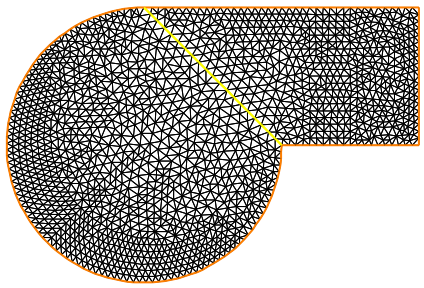

Figure 6.4: The two none overlapping mesh `TH` and `th`

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.

```
//    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");
```

6.8.3 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

```
//    Schwarz without overlapping (Shur complenement Neumann -> Dirichet)
```

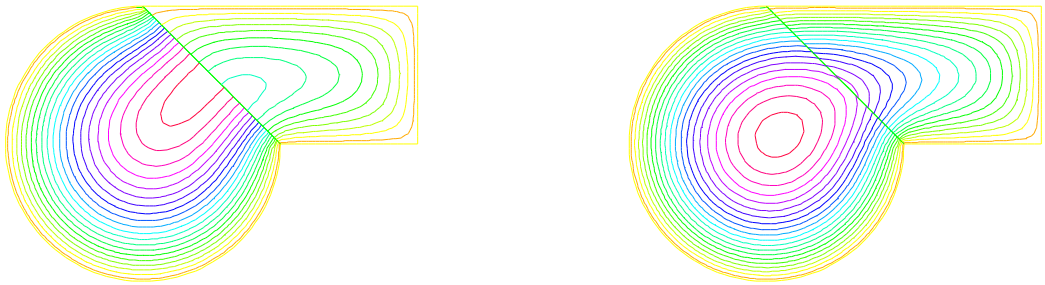

Figure 6.5: Isovalues of the solution at iteration 0 and iteration 9 without overlapping

```
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);
```

Remark, to day is not possible to defined a function just on a border, so the *lambda* function is defined on the all domain Ω_1 by:

```
Vh1 lambda=0;                                                    //    take λ ∈ Vh1
```

The two Laplace 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 ) ;
```

Now, 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[] ;
};
```

Remark, 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
```

6.9 Beam.edp

Elastic solids subject to forces deform: a point in the solid, originaly at (x,y) goes to (X,Y) after. When the displacement vector $\vec{v} = (v_1, v_2) = (X - x, Y - y)$ is small, Hooke's law relates the stress tensor σ inside the solid to the deformation tensor ϵ:

$$\sigma_{ij} = \lambda\delta_{ij}\nabla.\vec{v} + \mu\epsilon_{ij},\ \epsilon_{ij} = \frac{1}{2}(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i})$$

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

$$\int_\Omega [\mu\epsilon_{ij}(\vec{v})\epsilon_{ij}(\vec{w}) + \lambda\epsilon_{ii}(v)\epsilon_{jj}(\vec{w})] = \int_\Omega \vec{g}\cdot\vec{w} + \int_\Gamma \vec{h}\cdot\vec{w}, \forall\vec{w} \in V$$

The elastic solids is a rectangle beam $[0, 10] \times [0, 2]$. The data are the gravity force $\vec{g}$ and the boundary stress i$\vec{h}$ is zero on lower and upper side, and on the two vertical sides of the beam are locked.

```
                                                    //    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)+ ((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)
 ;


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);
```

6.10 Fluidstruct.edp

This problem involves the Lamé system of elasticity and the Stokes system for viscous fluids with velocity $\vec{u}$ and pressure p:

$$-\Delta\vec{u} + \vec{\nabla}p = 0,\ \nabla\cdot\vec{u} = 0,\quad \text{in}\ \ \Omega,\ \vec{u} = \vec{u}_\Gamma\ \ \text{on}\ \ \Gamma = \partial\Omega$$

where u_Γ is the velocity of the boundaries. The force that the fluid applies to the boundaries is the normal stress

$$\vec{h} = (\nabla\vec{u} + \nabla\vec{u}^T)\vec{n} - p\vec{n}$$

Elastic solids subject to forces deform: a point in the solid, originaly at (x,y) goes to (X,Y) after. When the displacement vector $\vec{v} = (v_1, v_2) = (X - x, Y - y)$ is small, Hooke's law relates the stress tensor σ inside the solid to the deformation tensor ϵ:

$$\sigma_{ij} = \lambda\delta_{ij}\nabla.\vec{v} + \mu\epsilon_{ij},\ \epsilon_{ij} = \frac{1}{2}(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i})$$

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

$$\int_\Omega [\mu\epsilon_{ij}(\vec{v})\epsilon_{ij}(\vec{w}) + \lambda\epsilon_{ii}(v)\epsilon_{jj}(\vec{w})] = \int_\Omega \vec{g}\cdot\vec{w} + \int_\Gamma \vec{h}\cdot\vec{w}, \forall\vec{w} \in V$$

The data are the gravity force $\vec{g}$ and the boundary stress $\vec{h}$.
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) solution of

```
                                //    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 6.6.2

```
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);
```

6.11 Region.edp

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 keywork
                                              //    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);
```

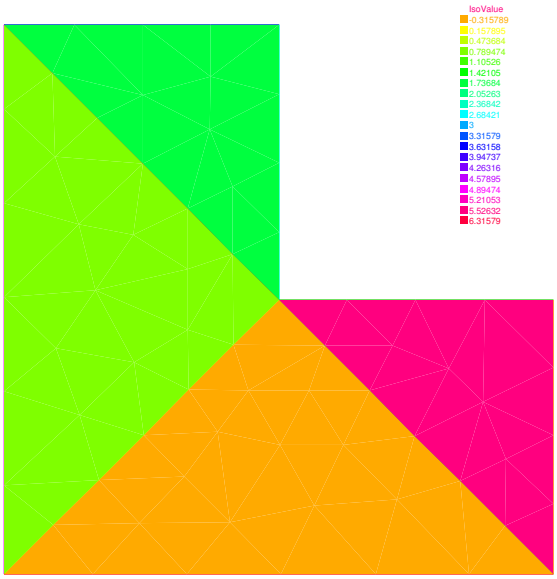

Figure 6.6: the function `reg`

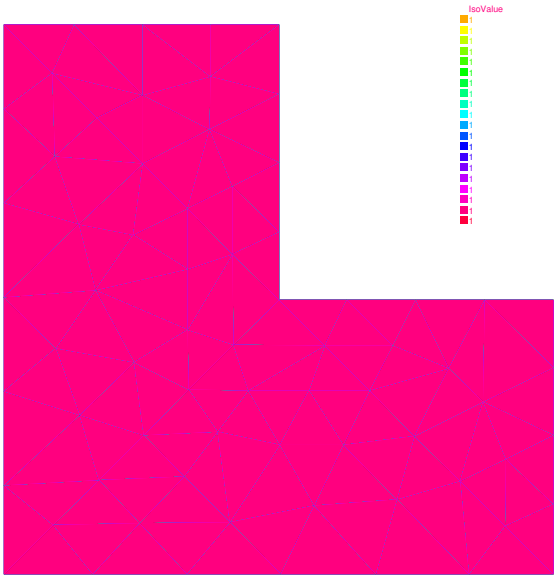

Figure 6.7: the function `nu`

`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 fonctions 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)( dx(u)*dx(v)*dy(u)*dy(v)) + int2d(-1*v) + on(a,b,c,d,e,f,u=0);
plot(u);
```

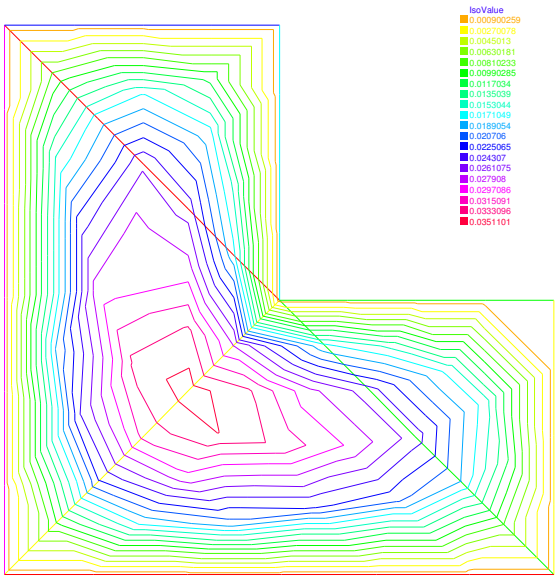

Figure 6.8: the isovalue of the solution u

6.12 FreeBoundary.edp

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");
```

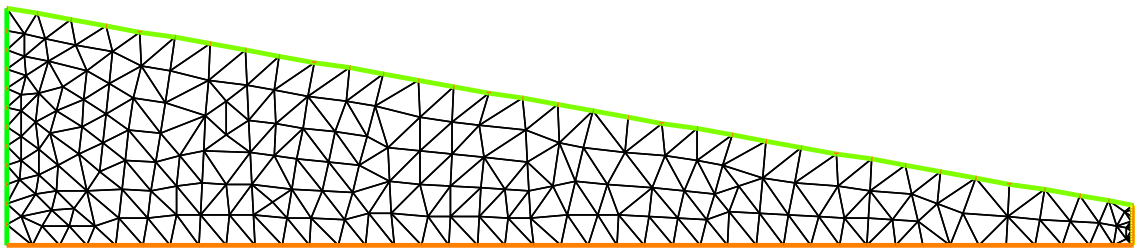

Figure 6.9: The mesh of the domain Ω

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 fixe 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 $\Omega^{n+1} = \mathcal{F}(\Omega^n)$
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");
```

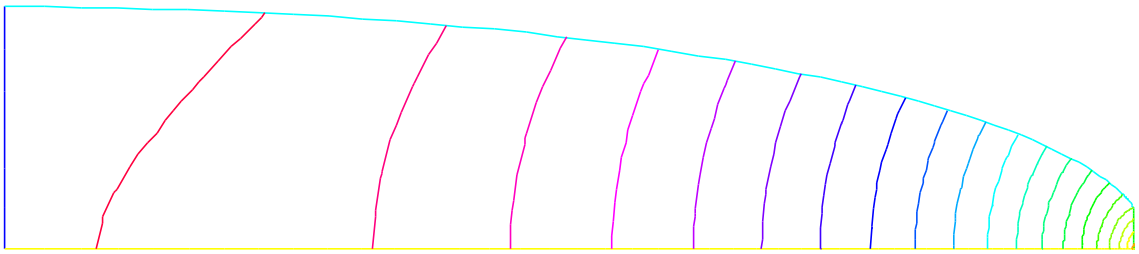

Figure 6.10: The final solution on the new domain Ω^{72}

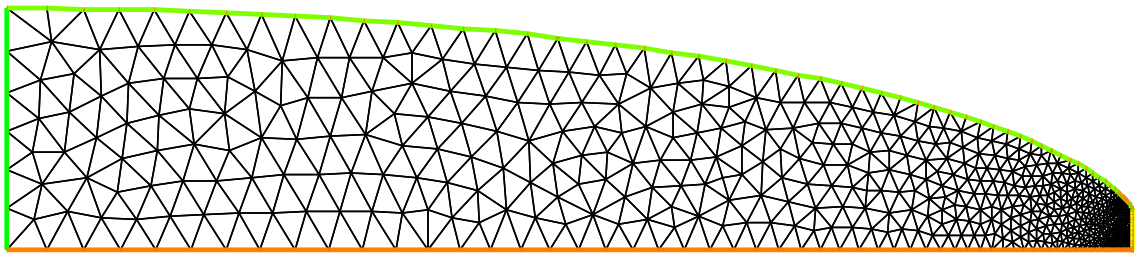

Figure 6.11: The adapted mesh of the domain Ω^{72}

Chapter 7

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 curent process in $\{0, ..., mpisize - 1\}$.

processor a function to set the possessor to send or receive data

```
processor(10) << a ;                          //    send to the process 10 the data a;
processor(10) >> a ;                        //    receive from the process 10 the data a;
```

Chapter 8

Schwarz in parallel

If exemple is just the rewritting of exemple `schwarz-overlap` page **??**

```
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 th,TH;

if (mpirank == 0)
 {
   th = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
   processor(1) << th ;
   processor(1) >> TH;
}
else
 {
   TH = buildmesh ( e(5*n) + e1(25*n) );
   cout << " end TH  " << endl;
   processor(0) << TH ;
   processor(0) >> th;
 }



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 ) ;


for ( i=0 ;i< 10; i++)
```

```
{
  if (mpirank == 0)
  {
   PB;                                                    //    compute U
   processor(1) << U[];                                   //    send U to 1
   processor(1) >> u[];                                   //    receive u from 1
  }
  else
  {
   pb;                                                    //    compute u
   processor(0) << u[];                                   //    send u to 0
   processor(0) >> U[];                                   //    recieve U from 0
  }
};
 if (mpirank==0)
    plot(U,u,ps="Uu.eps");
```

Chapter 9

Mesh Files

9.1 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
  $\Big(\big($ `(R)` x_i^j **,** j=1,dim $\big)$, `(I)` $Ref\phi_i^v$ **,** i=1 , NbOfVertices $\Big)$
- `Edges (I)` NbOfEdges
  $\Big($ `@@Vertex`$_i^1$, `@@Vertex`$_i^2$, `(I)` $Ref\phi_i^e$ **,** i=1 , NbOfEdges $\Big)$
- `Triangles (I)` NbOfTriangles
  $\Big(\big($ `@@Vertex`$_i^j$ **,** j=1,3 $\big)$, `(I)` $Ref\phi_i^t$ **,** i=1 , NbOfTriangles $\Big)$
- `Quadrilaterals (I)` NbOfQuadrilaterals
  $\Big(\big($ `@@Vertex`$_i^j$ **,** j=1,4 $\big)$, `(I)` $Ref\phi_i^t$ **,** i=1 , NbOfQuadrilaterals $\Big)$
- `Geometry`
  `(C*)` FileNameOfGeometricSupport

  - `VertexOnGeometricVertex`
    `(I)` NbOfVertexOnGeometricVertex
    $\Big($ `@@Vertex`$_i$, `@@Vertex`$_i^{geo}$ **,** i=1,NbOfVertexOnGeometricVertex $\Big)$
  - `EdgeOnGeometricEdge`
    `(I)` NbOfEdgeOnGeometricEdge
    $\Big($ `@@Edge`$_i$, `@@Edge`$_i^{geo}$ **,** i=1,NbOfEdgeOnGeometricEdge $\Big)$
- `CrackedEdges (I)` NbOfCrackedEdges
  $\Big($ `@@Edge`$_i^1$, `@@Edge`$_i^2$ **,** i=1 , 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 $\Big)$

9.2 bb File type for Store Solutions

The file is formatted such that:
`2 nbsol nbv 2`
$((\mathtt{U}_{ij}, \quad \forall i \in \{1, ..., \mathtt{nbsol}\}), \quad \forall j \in \{1, ..., \mathtt{nbv}\})$
where

- `nbsol` is a integer equal to the number of solutions.
- `nbv` is a integer equal to the number of vertex .
- $\mathtt{U}_{ij}$ 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.

9.3 BB File Type for Store Solutions

The file is formatted such that:
`2 n typesol`^{1} ... `typesol`^{n} `nbv 2`
$\Big(\big((\mathtt{U}_{ij}^k, \quad \forall i \in \{1, ..., \mathtt{typesol}^k\}\big), \quad \forall k \in \{1, ...\mathtt{n}\}\Big) \quad \forall j \in \{1, ..., \mathtt{nbv}\}\Big)$
where

- `n` is a integer equal to the number of solutions
- `typesol`$^{\mathtt{k}}$, type of the solution number k, is
  - `typesol`$^{\mathtt{k}} = 1$ the solution `k` is scalare (1 value per vertex)
  - `typesol`$^{\mathtt{k}} = 2$ the solution `k` is vectorial (2 values per unknown)
  - `typesol`$^{\mathtt{k}} = 3$ the solution `k` is a 2×2 symmetric matrix (3 values per vertex)
  - `typesol`$^{\mathtt{k}} = 4$ 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.

9.4 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`
$\texttt{a11}_i\texttt{,a21}_i\texttt{,a22}_i \quad \forall i \in \{1, ..., \texttt{nbv}\}$
where

- `nbv` is a integer equal to the number of vertices,
- a11_i, a12_i, 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.

9.5 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 10

Grammar

10.1 Keywords

```
R3
bool
border
break
complex
continue
else
end
fespace
for
func
if
ifstream
include
int
macro
matrix
mesh
ofstream
problem
real
return
solve
string
varf
while
```

10.2 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
         |  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  ']' ;
```

10.3 The Types of the languages, and cast

```
 --Add_KN_<double> =  <Add_KN_<double>>

 --Add_Mulc_KN_<double> * =  <Add_Mulc_KN_<double>>

 --AnyTypeWithOutCheck =  <AnyTypeWithOutCheck>

 --C_F0 =  <C_F0>

 --DotStar_KN_<double> =  <DotStar_KN_<double>>

 --E_Array =  <E_Array>

 --FEbase<double> * =  <FEbase<double>>
  <FEbase<double>> :   <FEbase<double>>
 --FEbase<double> ** =  <FEbase<double> **>

 --FEbaseArray<double> * =  <FEbaseArray<double>>
```

```
--FEbaseArray<double> ** =  <FEbaseArray<double> **>
   []  type :<Polymorphic>   operator :
(   <FEbase<double> **> :   <FEbaseArray<double> **>, <long> )


--Fem2D::Mesh * =  <Fem2D::Mesh>
 <Fem2D::Mesh> :   <Fem2D::Mesh **>
--Fem2D::Mesh ** =  <Fem2D::Mesh **>
   <-,  type :<Polymorphic>
  (   <Fem2D::Mesh> :   <string> )
(   <long> :   <Fem2D::Mesh **>, <double>, <double> )

   area,  type :<Polymorphic>   operator. :
(   <double> :   <Fem2D::Mesh **> )

   nt,  type :<Polymorphic>
  operator. :
(   <long> :   <Fem2D::Mesh **> )

   nv,  type :<Polymorphic>   operator. :
(   <long> :   <Fem2D::Mesh **> )


--Fem2D::MeshPoint * =  <Fem2D::MeshPoint>
   N,  type :<Polymorphic>   operator. :
(   <Fem2D::R3> :   <Fem2D::MeshPoint> )

   P,  type :<Polymorphic>   operator. :
(   <Fem2D::R3> :   <Fem2D::MeshPoint> )


--Fem2D::R2 * =  <Fem2D::R2>

--Fem2D::R3 * =  <Fem2D::R3>
   x,  type :<Polymorphic>   operator. :
(   <double *> :   <Fem2D::R3> )

   y,  type :<Polymorphic>   operator. :
(   <double *> :   <Fem2D::R3> )

   z,  type :<Polymorphic>   operator. :
(   <double *> :   <Fem2D::R3> )


--Fem2D::TypeOfFE * =  <Fem2D::TypeOfFE>

--KN<double> =  <KN<double>>
   []  type :<Polymorphic>   operator :
(   <double *> :   <KN<double>>, <long> )


--KN<double> * =  <KN<double> *>
   <-,  type :<Polymorphic>
(   <KN<double> *> :   <KN<double> *>, <long> )

   []  type :<Polymorphic>   operator :
```

```
(   <double *> :   <KN<double> *>, <long> )

   max,  type :<Polymorphic>   operator. :
(   <double> :   <KN<double> *> )

   min,  type :<Polymorphic>   operator. :
(   <double> :   <KN<double> *> )

   n,  type :<Polymorphic>
  operator. :
(   <long> :   <KN<double> *> )

   sum,  type :<Polymorphic>   operator. :
(   <double> :   <KN<double> *> )


--KN_<double> =  <KN_<double>>

--KN_<double> * =  <KN_<double> *>

--Matrice_Creuse<double> * =  <Matrice_Creuse<double>>
 <Matrice_Creuse<double>> :   <Problem>
--Matrice_Creuse_Transpose<double> =  <Matrice_Creuse_Transpose<double>>

--Matrice_Creuse_inv<double> =  <Matrice_Creuse_inv<double>>

--Mulc_KN_<double> =  <Mulc_KN_<double>>

--MyMap<String, double> * =  <MyMap<String, double>>
   []  type :<Polymorphic>   operator :
(   <double *> :   <MyMap<String, double>>, <string> )


--Polymorphic * =  <Polymorphic>

--Sub_KN_<double> =  <Sub_KN_<double>>

--Transpose<KN<double>> =  <Transpose<KN<double>>>

--TypeSolveMat * =  <TypeSolveMat>

--VirtualMatrice<double>::plusAtx =  <VirtualMatrice<double>::plusAtx>

--VirtualMatrice<double>::plusAx =  <VirtualMatrice<double>::plusAx>

--VirtualMatrice<double>::solveAxeqb =  <VirtualMatrice<double>::solveAxeqb>

--bool =  <bool>
 <bool> :   <bool *>
--bool * =  <bool *>

--char * =  <char>

--const BC_set<double> * =  <BC_set<double>>

--const CDomainOfIntegration * =  <CDomainOfIntegration>
```

```
   ()  type :<Polymorphic>   operator :
(   <FormBilinear> :   <CDomainOfIntegration>, <LinearComb<std::pair<MGauche, MDroit>, C_
(   <double> :   <CDomainOfIntegration>, <double> )
(   <FormLinear> :   <CDomainOfIntegration>, <LinearComb<MDroit, C_F0>> )


--const C_args * =  <C_args>
 <C_args> :   <FormBilinear>     ()  type :<Polymorphic>   operator :
(   <Call_FormLinear> :   <C_args>, <long>, <v_fes **> )
(   <Call_FormBilinear> :   <C_args>, <v_fes **>, <v_fes **> )


--const Call_FormBilinear * =  <Call_FormBilinear>

--const Call_FormLinear * =  <Call_FormLinear>

--const E_Border * =  <E_Border>

--const E_BorderN * =  <E_BorderN>

--const Fem2D::QuadratureFormular * =  <Fem2D::QuadratureFormular>

--const Fem2D::QuadratureFormular1d * =  <Fem2D::QuadratureFormular1d>

--const FormBilinear * =  <FormBilinear>
   ()  type :<Polymorphic>   operator :
(   <Call_FormBilinear> :   <FormBilinear>, <v_fes **>, <v_fes **> )
(   <Call_FormLinear> :   <FormBilinear>, <long>, <v_fes **> )


--const FormLinear * =  <FormLinear>
   ()  type :<Polymorphic>   operator :
(   <Call_FormLinear> :   <FormLinear>, <v_fes **> )


--const IntFunction * =  <IntFunction>

--const LinearComb<MDroit, C_F0> * =  <LinearComb<MDroit, C_F0>>

--const LinearComb<MGauche, C_F0> * =  <LinearComb<MGauche, C_F0>>

--const LinearComb<std::pair<MGauche, MDroit>, C_F0> * =  <LinearComb<std::pair<MGauche, M

--const Problem * =  <Problem>

--const Solve * =  <Solve>

--const char * =  <char>

--double =  <double>
 <double> :   <double *>     ()  type :<Polymorphic>   operator :
(   <double> :   <double>, <double>, <double> )


--double * =  <double *>
```

```
--interpolate_f_X_1<double>::type =  <interpolate_f_X_1<double>::type>

--long =  <long>
 <long> :   <long *>
--long * =  <long *>

--istream * =  <istream>
 <istream> :   <istream **>
--istream ** =  <istream **>

--ostream * =  <ostream>
 <ostream> :   <ostream **>
--ostream ** =  <ostream **>
   <-,  type :<Polymorphic>   operator( ):
(   <ostream> :   <string> )


--string * =  <string>
 <string> :   <string **>
--string ** =  <string **>

--std::complex<double> =  <complex>
 <complex> :   <complex *>
--std::complex<double> * =  <complex *>

--std::ios_base::openmode =  <std::ios_base::openmode>

--std::pair<FEbase<double> *, int> =  <std::pair<FEbase<double> *, int>>
   (),  type :<Polymorphic>   operator :
(   <double> :   <std::pair<FEbase<double> *, int>>, <double>, <double> )
(   <interpolate_f_X_1<double>::type> :   <std::pair<FEbase<double> *, int>>, <E_Array> )

   [],  type :<Polymorphic>  operator. :
(   <KN<double> *> :   <std::pair<FEbase<double> *, int>> )

   n,  type :<Polymorphic>   operator. :
(   <long> :   <std::pair<FEbase<double> *, int>> )
--std::pair<FEbaseArray<double> *, int> =  <std::pair<FEbaseArray<double> *, int>>
   []  type :<Polymorphic>
  operator :
(   <std::pair<FEbase<double> *, int>> :   <std::pair<FEbaseArray<double> *, int>>, <long>
--std::pair<Fem2D::Mesh **, int> * =  <std::pair<Fem2D::Mesh **, int>>
--v_fes * =  <v_fes>
 <v_fes> :   <v_fes **>
--v_fes ** =  <v_fes **>

--void =  <void>
```

10.4 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>... )

 - 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> )

 - min,  type :<Polymorphic>  operator() :
```

```
(   <double> :   <double>, <double> )
(   <long> :   <long>, <long> )

 - movemesh,  type :<Polymorphic>   operator() :
(   <Fem2D::Mesh> :   <Fem2D::Mesh>, <E_Array>... )

 - nu_triangle,  type :<long *>
 - on,  type :<Polymorphic>   operator() :
(   <BC_set<double>> :   <long>... )

 - 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>
 - qf2pE,  type :<Fem2D::QuadratureFormular1d>
 - qf2pT,  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] D. Bernardi, F.Hecht, K. Ohtsuka, O. Pironneau: *freefem+ documentation*, on the web at ftp://www.freefem.org/freefemplus.

[3] D. Bernardi, F.Hecht, O. Pironneau, C. Prud'homme: *freefem documentation*, on the web at http://www.asci.fr

[4] P.L. George: *Automatic triangulation*, Wiley 1996.

[5] F. Hecht: The mesh adapting software: bamg. INRIA report 1998.

[6] J.L. Lions, O. Pironneau: Parallel Algorithms for boundary value problems, Note CRAS. Dec 1998. Also : Superpositions for composite domains (to appear)

[7] B. Lucquin, O. Pironneau: *Scientific Computing for Engineers* Wiley 1998.

[8] F. Preparata, M. Shamos; *Computational Geometry* Springer series in Computer sciences, 1984.

[9] 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

[10] J.L. Steger: The Chimera method of flow simulation, Workshop on applied CFD, Univ of Tennessee Space Institute, August 1991.

[11] N. Wirth: *Algorthims + Data Structures = Programs*, Prentice Hall, 1976

[12] ison documentation

[13] The `C++` , programming language, Third edition, Bjarne Stroustrup, Addison-Wesley 1997.

[14] COOOL: a package of tools for writing optimization code and solving optimization problems, `http://coool.mines.edu`

Index