

FreeFem++ Manual

version 1.42 (*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

September 6, 2004

¹<mailto:hecht@ann.jussieu.fr>

²<mailto:pironneau@ann.jussieu.fr>

³<mailto:ohtsuka@barnard.cs.hkg.ac.jp>

Contents

1	Introduction	7
1.1	History	8
1.2	The language	8
1.3	Installation	9
2	Syntax	11
2.1	Data Types	11
2.1.1	Another Example	11
2.2	List of major types	12
2.3	Globals	13
2.4	Arithmetic	14
2.5	Array	16
2.6	Loops	17
2.7	Input/Output	17
3	Mesh Generation	19
3.1	Square	19
3.2	Border	19
3.3	Movemesh	21
3.4	Reading and writing a mesh	22
3.5	Triangulate	23
3.6	Adaptmesh	23
3.7	Trunc	26
3.8	splitmesh	27
3.9	build empty mesh	27
3.10	Get Mesh numbering	28
3.11	Meshing examples	29
4	Finite Elements	31
4.1	Problem and solve	34
4.2	Parameter Description for <code>solve</code> and <code>problem</code>	37
4.3	Problem definition	37
4.4	Integrals	39
4.5	Variational Form, Sparse Matrix, Right Hand Side Vector	40
4.6	Eigen value and eigen vector	41
4.7	Plot	45
4.8	link with gnuplot	46

4.9	link with medit	48
4.10	Convect	49
5	algorithm	51
5.1	conjugate Gradient	51
5.2	Optimization	52
6	More examples	53
6.1	A_ tutorial.edp	53
6.2	Periodic	55
6.3	Adapt.edp	56
6.4	adaptindicatorP2.edp	58
6.5	Algo.edp	60
6.5.1	Non linear conjugate gradient algorithm	60
6.5.2	Newton Raphson algorithm	62
6.6	Stokes and Navier-Stokes	63
6.6.1	Cavity.edp	63
6.6.2	StokesUzawa.edp	66
6.6.3	NSUzawaCahouetChabart.edp	67
6.7	Readmesh.edp	68
6.8	Domain decomposition	69
6.8.1	Schwarz-overlap.edp	69
6.8.2	Schwarz-no-overlap.edp	71
6.8.3	Schwarz-gc.edp	72
6.9	Beam.edp	74
6.10	Fluidstruct.edp	75
6.11	Region.edp	78
6.12	FreeBoundary.edp	80
6.13	nolinear-elas.edp	83
7	Parallel version experimental	89
7.1	Schwarz in parallel	89
8	The Graphic User Interface (FFedit)	91
8.1	Unix version	91
8.1.1	Installation of Tcl and Tk	91
8.1.2	Description	92
8.1.3	Cygwin version	94
8.1.4	Installation of Cygwin	95
8.1.5	Compilation and Installation of FreeFem++ under Cygwin	95
8.1.6	Compilation and Installation of tcl8.4.0 and tk8.4.0 under Cygwin	96
8.1.7	Use	99
9	Mesh Files	101
9.1	File mesh data structure	101
9.2	bb File type for Store Solutions	102
9.3	BB File Type for Store Solutions	102
9.4	Metric File	103

9.5	List of AM_FMT, AMDBA Meshes	103
10	Add new finite element	107
10.1	Some notation	107
10.2	Which class of add	108
10.3	How to add	111
11	Grammar	113
11.1	Keywords	113
11.2	The bison grammar	113
11.3	The Types of the languages, and cast	117
11.4	All the operators	122
11.5	History of the software	127

Chapter 1

Introduction

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

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

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

The characteristics of `freefem++` are:

- A large variety of finites elements : linear and quadratic Lagrangian elements, discontinuous P1 and Raviart-Thomas elements, elements of a non-scalar type, mini-element, ...
- Automatic interpolation of data on different meshes to an over mesh.
- Linear problems description thanks to a formal variational form, with access to the vectors and the matrix if needed.
- Includes tools to define discontinuous Galerkin formulations (please refer to the following keywords: “jump”, “average”, “intalledges”)
- Analytic description of boundaries. When two boundaries intersect, the user must specify the intersection points.
- Automatic mesh generator, based on the Delaunay-Voronoi algorithm. Inner points density is proportional to the density of points on the boundary [5].

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

1.1 History

The project has evolved from `MacFem`, `PCfem`, written in Pascal. The first C version was `freefem 3.4`; it offered mesh adaptation on a single mesh. A thorough rewriting in C++ led to `freefem+ 1.2.10`, which also included interpolation over multiple meshes (functions defined on one mesh can be used on any other mesh). Implementing the interpolation from one unstructured mesh to another was not easy because it had to be fast and non-diffusive; for each point, one had to find the containing triangle. This is one of the basic problems of computational geometry (see Preparata & Shamos[10] for example). Doing it in a minimum number of operations was a challenge. Our implementation was $O(n \log n)$ and based on a quadtree.

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

The `freefem` language allows for a quick specification of any partial differential system of equations. The language syntax of `freefem++` is the result of a new design which makes use of the STL [15], templates and `bison` for its implementation. The outcome is a versatile software in which any new finite element can be included in a few hours; but a recompilation is then necessary. The library of finite elements available in `freefem++` will therefore grow with the version number. So far we have linear and quadratic Lagrangian elements, discontinuous P1 and Raviart-Thomas elements.

1.2 The language

Basically `freefem++` is a compiler. The language it defines is typed, polymorphic and reentrant with macro generation (see 6.13). Every variable must be typed and declared in a statement; each statement separated from the next by a semicolon `;`.

Here is a simple example solving the Dirichlet problem inside the unit circle. We have chosen xy as the right hand side for the Laplace operator, a second degree finite element method on triangles, and a mesh with 50 points on the boundary. The linear system is solved by a Gauss LU factorisation.

```

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 side
+ on(5,u=0) ; //      Dirichlet boundary condition

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

```

Reserved words are shown in blue. `pi`, `x`, `y`, `label`, `solver` are reserved variables. It is allowed (although not advisable) to redefine these variables, so they will not be highlighted again in the following example programs.

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

- Boundaries are described analytically (by opposition to CSG). In the case where two boundaries intersect, the user must specify the intersection points in case two boundaries intersect. The domain will be on the left side of the oriented boundary.
- Automatic mesh generation, based on the Delaunay-Voronoi algorithm. Inner points are generated with a density proportional to the density of points on the boundary (hence a refined mesh is obtained by increasing the number of boundary points).
- Preprogrammed library of arbitrary degree elements.
- 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.

1.3 Installation

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

```

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

```

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

1. `FreeFem++`, standard version, with a graphical interface based on X11, Win32 or MacOS
2. `FreeFem++-nw`, postscript output only (batch version, no windows)
3. `FreeFem++-mpi`, parallel version, postscript output only
4. `FreeFem++-glx`, graphics using OpenGL and X11
5. `/Applications/FreeFem++.app`, Drag and Drop CoCoa MacOS Application
6. `FreeFem++-CoCoa`, MacOS Shell script for MacOS OpenGL version (MacOS 10.2 or better) (note: it uses `/Applications/FreeFem++.app`)

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

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

Chapter 2

Syntax

2.1 Data Types

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 (`fespace variable`), functions (`func`), arrays of finite element functions (`fespace variable[basictype]`), linear and bilinear operators, sparse matrices, vectors, etc. For instance :

```
int i,n=20; // i,n ∈ ℤ
real pi=4*atan(1.); // pi ∈ ℝ(redefine)
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 this
{ 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 (delimited by “{...}”) as in C++ , except the `fespace` variable. The variables which are local to a block are destroyed at the end of the block.

Type declarations are compulsory in `freefem++` . This is necessary to prevent many potential programming errors related to type mismatch. A variable name is just an alphanumeric string. The underscore character `_` is not allowed because it will be used as an operator in the future.

2.1.1 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);
{
  real r= 2;
  fespace Vh(Th,P1);
}

```

// new block
// not the same r
// error because Vh is a global name

// here r==0.01

Note 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).

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 reals (integer index);

real[int,int] matrix of reals (2 integers indices);

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

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

mesh[int] array of meshes (integer index);

func define a function (on one line, without arguments) `func f=cos(x)+sin(y);`. Please note that the function type is given by the type of the expression.

mesh defines a mesh. e.g. `mesh Th=buildmesh(circle(10));`

fespace to define a new type of finite element space. e.g. `fespace Vh(Th,P1);`. At present, the available 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 polynoms of \mathbb{R}^2 of degree two,

RT0 the Raviart-Thomas finite element,

P1nc non-conforming P_1 finite element,

P1dc linear piecewise discontinuous finite element

P2dc P_2 piecewise discontinuous finite element

P1b linear piecewise continuous finite element + bubble

To define p as a function belonging to $\text{fespace } \forall h : \forall h \ p ;$. To define a as an array of 10 functions from $\forall h : \forall h \ a[10];$

problem defines a pde problem without solving it.

solve defines a pde problem and solves it.

varf defines a full variational form.

matrix defines a sparse matrix.

Constant values for the above type can be :

true is a constant of type bool,

false is a constant of type bool,

123 is an integer,

123. is of type real,

123.e10 is a constant of type real and value $123 \cdot 10^{10}$. Please note that the syntax of a real number is the same as in C++ or fortran.

"abcd" is a constant of type string. There are two escape characters allowed in string constants : `\n` to insert a newline and `\"` to insert a " .

2.3 Globals

The following names are related to the underlying finite element abstractions :

x the x coordinate of the current point (real value)

y the y coordinate of the current point (real value)

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

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

region a function returning the region number of the current point (x,y,z). Integer value.

P current point (\mathbb{R}^3 value. `P.x`, `P.y`, `P.z`)

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

lenEdge length of the current edge

hTriangle size of the current triangle

nuTriangle integer index of the current triangle

nuEdge integer index of the current edge in the triangle

nTonEdge integer: number of triangle containing the current edge (1 or 2).

area area of the current triangle

cout console ostream

cin console istream

true bool value

false bool value

pi real approximation of π

endl the end of line string

Here is how to show all types, operators and functions from inside a `freefem++` program :

```
dumptable(cout);
```

To run a system command stored in a string (not implemented on Carbon MacOS) :

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

2.4 Arithmetic

The available operators 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 or in maple),
- `|` `&` `!` are the three boolean operators ‘or’ , ‘and’ , ‘not’
- `'` `.*` `./` are array operators (as in matlab or scilab), where
 - `'` is the unary right transposition of an array or a matrix
 - `.*` is the term by term multiply operator.
 - `./` is the term by term divide operator.

some compound operators:

- `^-1` means solving a linear system (example: `b = A^-1 x`)
- `' *'` is a transposition followed by a matrix product, i.e. the dot product (example `real DotProduct=a'*b`)

As in C++ , there are automatic casts from a type to another. So in some sense we have

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

and if any of these four types is copied into a *string*, it will be converted into a human-readable 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="+abcdddd+";
str=str + str1; // string concatenation
str = str + 2 ;
cout << "str= " << str << " == abc++abcdddd+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++abcdddd+2== abc++abcdddd+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 known at execution time, and the implementation is done with the `KN<>` class so all the vector operations of `KN<>` are implemented. It is also possible to build an array of FE functions with the same syntax.

The second case is just a map of the STL¹[15] so no vector operation is allowed, except for the selection of an item.

The transpose operator is `'` (as in MathLab or SciLab), so the way to compute the dot product of two arrays `a,b` is **real** `ab= a'*b`.

```

int i;
real [int] tab(10), tab1(10);           // 2 arrays of 10 reals
    real [int] tab2;                   // bug! missing array size
tab = 1;                               // set all array elements to 1
tab[1]=2;
cout << tab[1] << " " << tab[9] << " size of tab = "
    << tab.n << " " << tab.min << " " << tab.max << " " << endl;
tab1=tab;
tab=tab+tab1;
tab=2*tab+tab1*5;
tab1=2*tab-tab1*5;
tab+=tab;
cout << " dot product " << tab'*tab << endl;           // 'tabtab
cout << tab << endl;
cout << tab[1] << " " << tab[9] << endl;
real[string] map;                       // a dynamic 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"

```

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

```

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

string[string] tt; // a array of string

```

2.6 Loops

for and while loops are implemented, with the same semantics as in C++ , including keywords break and continue.

```

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, 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 opening a file in append mode.

Note 2 *The file is closed at the end of the enclosing block,*

```

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

{
  ifstream f("toto.txt");
  f >> i;
}

{
  ofstream f("toto.txt",append); // appending data to file "toto.txt"
  f << i << "coucou'\n";
}; // file f closed here because variable f is deleted

cout << i << endl;

```


Chapter 3

Mesh Generation

The following keywords are discussed in this section:

square, border, buildmesh, movemesh, adaptmesh, readmesh, trunc, triangulate, splitmesh, emptymesh

All the examples in this section come from the files `mesh.edp` and `tablefunction.edp`.

3.1 Square

For easy and simple testing we have included a constructor for rectangles. All other shapes should be handled with `border` and `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 $n \times m$ grid in the rectangle $[1.2, 1.8] \times [0, 1]$ and a 4×5 grid in the unit square $[0, 1]^2$.

Note 3 *Boundary labels are numbered 1, 2, 3, 4 for bottom, right, top, left side (before the mapping $[x_0 + (x_1 - x_0) * x, y_0 + (y_1 - y_0) * y]$ (the mapping can be non linear).*

3.2 Border

A domain is defined as being on the left (resp right) of its parameterized boundary (when looking towards increasing values of the parameter) if the sign of the expression that returns the number of vertices on the boundary is positive (resp negative).

note: the boundary label must be non-zero if you you want to specify a boundary condition on this border

For instance the domain delimited by the unit circle, with a small circular hole inside would be described as:

```
real pi=4*atan(1);
```

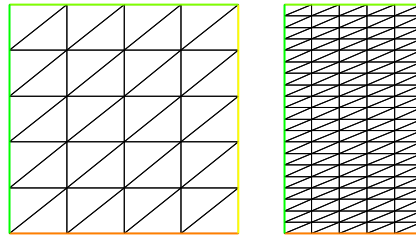


Figure 3.1: The 2 square meshes

```

border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
border b(t=0,2*pi){ x=0.3+0.3*cos(t); y=0.3*sin(t);label=2;}
plot(a(50)+b(+30)) ; // to see a plot of the border mesh
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

```

Note 4 *The use of `ps="fileName"` generates a postscript file identical to the plot shown on screen.*

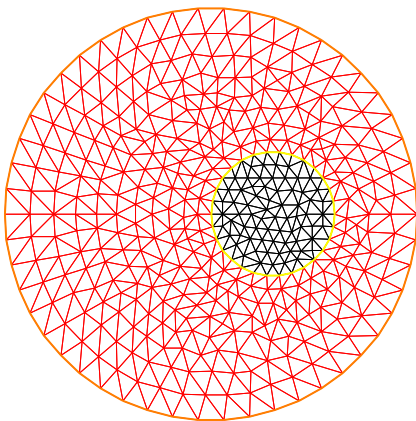


Figure 3.2: mesh without hole

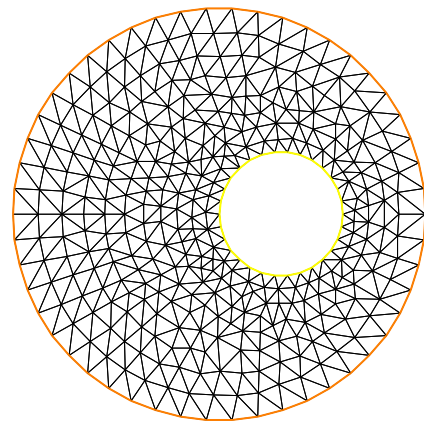


Figure 3.3: mesh with hole

Note that boundaries can only intersect at their end points.

3.3 Movemesh

As the two last statements in the example below show, mesh motion is possible. Sometimes the moved mesh is invalid because some triangle becomes reversed (with a negative area). This is why we check the minimum triangle area in the transformed mesh with `checkmovemesh` before any real transformation.

```

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"); // see figure ??
real coef=1;
real minT0= checkmovemesh(Th,[x,y]); // the min triangle area
while(1) // find a correct move mesh
{
  real minT=checkmovemesh(Th,[x+coef*uu,y+coef*vv]); // the min triangle area
  if (minT > minT0/5) break ; // if big enough
  coef/=1.5;
}

Th=movemesh(Th,[x+coef*uu,y+coef*vv]);
plot(Th,wait=1,fill=1,ps="movemesh.eps"); // see figure 3.5

```

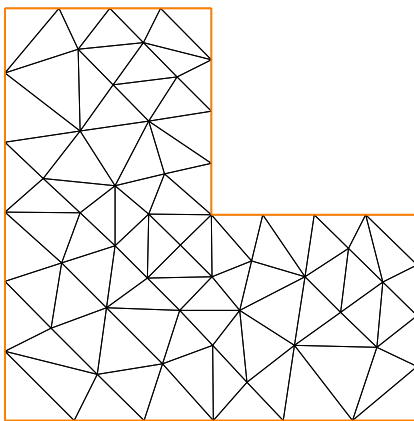


Figure 3.4: L-shape

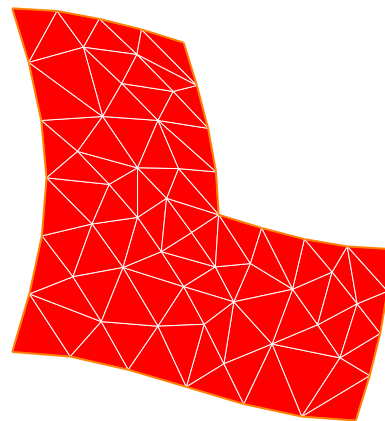


Figure 3.5: moved L-shape

Note 5 Consider a function u defined on a mesh Th . A statement like $Th=movemesh(Th\dots)$ does not change u and so the old mesh still exists. It will be destroyed when no function use it. A statement like $u = u$ redefines u on the new mesh Th with interpolation and therefore destroys the old Th if u was the only function using it.

Now, an example of moving mesh, with lagrangian function u defined on the moving mesh.

```

// simple movemesh example
mesh Th=square(10,10);
fespace Vh(Th,P1);
real t=0;

// ---
// the problem is how to build data without interpolation
// so the data u is moving with the mesh as you can see in the plot
// ---

Vh u=y;
for (int i=0;i<4;i++)
{
  t=i*0.1;
  Vh f= x*t;
  real minarea=checkmovemesh(Th,[x,y+f]);
  if (minarea >0 ) // movemesh will be ok
    Th=movemesh(Th,[x,y+f]);

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

  real[int] tmp(u[.].u);
  tmp=u[.]; // save the value
  u=0; // to change the FESpace and mesh associated with u
  u[.]=tmp; // set the value of u without any mesh update
  plot(Th,u,wait=1);
};

// In this program, since u is only defined on the last mesh, all the
// previous meshes are deleted from memory.
// -----

```

3.4 Reading and writing a mesh

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

```

border floor(t=0,1){ x=t; y=0; label=1;}; // the unit square
border right(t=0,1){ x=1; y=t; label=5;};
border ceiling(t=1,0){ x=t; y=1; label=5;};
border left(t=1,0){ x=0; y=t; label=5;};
int n=10;
mesh th= buildmesh(floor(n)+right(n)+ceiling(n)+left(n));
savemesh(th,"toto.am_fmt"); // "formatted Marrocco" format
savemesh(th,"toto.Th"); // "bang"-type mesh
savemesh(th,"toto.msh"); // freefem format

```

```
savemesh(th,"toto.nopo"); // modulef format see [7]
mesh th2 = readmesh("toto.msh"); // read the mesh
```

There are many mesh file formats 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 "bang : a bidimensional anisotropic mesh generator" available from the freefem web page.

3.5 Triangulate

freefem++ is able to build a triangulation from a set of points. This triangulation is a Delaunay mesh of the convex hull of the set of points. It can be useful to build a table function.

The coordinates of the points and the value of the table function are defined in a separate file which looks 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
```

The third column of each line is left untouched by the `triangulate` command. But you can use this third value to define a table function with rows of the form: $x \ y \ f(x,y)$.

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

fespace Vhxy(Thxy,P1); // create a 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 read third row only. xx and yy are
  just skipped
}
plot(fxu,ps="xyf.ps"); // plot the function (see figure ??)
```

3.6 Adaptmesh

Mesh adaptation is a very powerful tool which should be used whenever possible. freefem++ uses a variable metric/Delaunay automatic meshing algorithm. It takes one or more functions as input (see in the following example) and builds a mesh adapted to the second derivative of the given 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 Probleml(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();
    Probleml;
    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;

```

This method is described in detail in [6]. It has a number of default parameters which can be modified :

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

hmax= Maximum edge size. (**val** is a double precision value. It defaults to the diameter of the domain to be meshed)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

periodic= use: builds an adapted periodic mesh, you can write `periodic=[[4,y],[2,y],[1,x]]` to build a biperiodic mesh of a square. (see periodic finite element spaces 4, and see `sphere.edp` for a full example)

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)

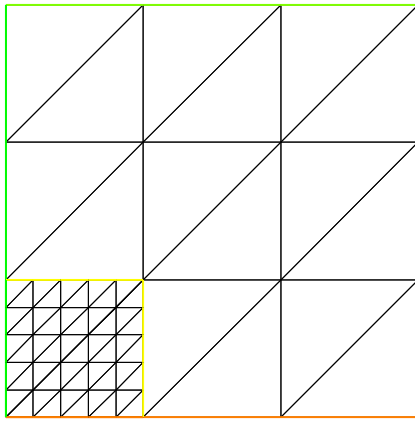
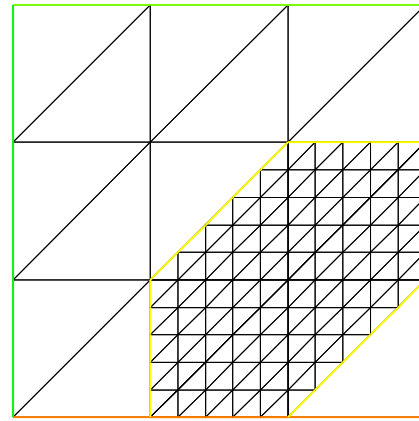
split= sets the level n of triangle splitting. each triangle is splitted in $n \times n$ (one by default).

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

```
mesh Th=square(3,3);
fespace Vh(Th,P1);
Vh u;
int i,n=u.n;
u=0;
for (i=0;i<n;i++) // all degree of freedom
{
  u[][i]=1; // the basic function i
  plot(u,wait=1);
  mesh Sh1=trunc(Th,abs(u)>1.e-10,split=5,label=2);
  plot(Th,Sh1,wait=1,ps="trunc"+i+".eps"); // plot the mesh of
// the function's support
  u[][i]=0; // reset
}
```

Figure 3.6: 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 splitmesh

A other way to split mesh triangle:

```
{
//      new stuff 2004 splitmesh (version 1.37)
assert(version>=1.37);
border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
plot(Th,wait=1,ps="nosplitmesh.eps"); //      see figure 3.8
mesh Th=buildmesh(a(20));
plot(Th,wait=1);
Th=splitmesh(Th,1+5*(square(x-0.5)+y*y));
plot(Th,wait=1,ps="splitmesh.eps"); //      see figure 3.9
}
```

3.9 build empty mesh

The idea is when you want to defined Finite Element space on boundary you can use a mesh with no internal points (call empty mesh). It is can by useful we you have a Lagrange multiplier defined on the border.

So the function emptymesh remove all the internal point of a mesh expect if the point is on internal boundary.

```
{
//      new stuff 2004 emptymesh (version 1.40)
//      -- useful to build Multiplicator space
//      build a mesh without internal point
//      with the same boundary
//      -----
assert(version>=1.40);
border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
mesh Th=buildmesh(a(20));
Th=emptymesh(Th);
plot(Th,wait=1,ps="emptymesh-1.eps"); //      see figure 3.10
}
```

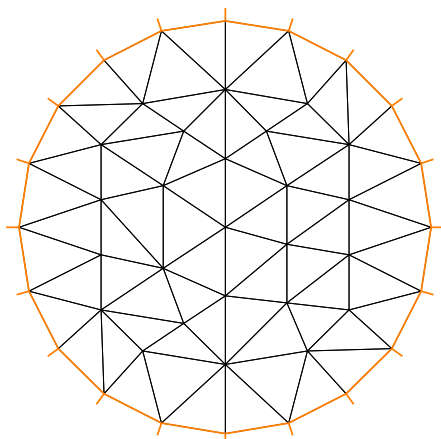
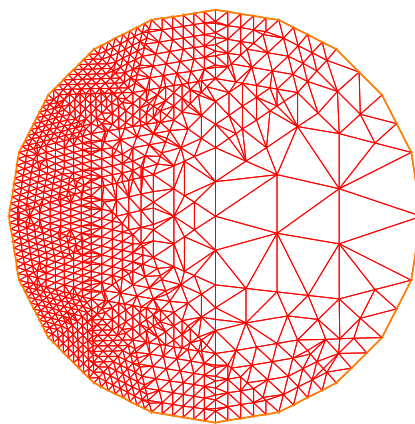


Figure 3.8: initial mesh

Figure 3.9: all left mesh triangle is split conformally in $\text{int}(1+5*(\text{square}(x-0.5)+y*y)^2)$ triangles.

}

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

```
{ // new stuff 2004 emptymesh (version 1.40)
  // -- useful to build Multiplier space
  // build a mesh without internal point
  // of pseudo sub domain
  // -----
  assert(version>=1.40);
  mesh Th=square(10,10);
  int[int] ssd(Th.nt);
  for(int i=0;i<ssd.n;i++) // build the pseudo region numbering
  { int iq=i/2; // because 2 triangle per quad
    int ix=iq%10; //
    int iy=iq/10; //
    ssd[i]= 1 + (ix>=5) + (iy>=5)*2;
  }
  Th=emptymesh(Th,ssd); // build empty with
  // all edge e = T1 ∩ T2 and ssd[T1] ≠ ssd[T2]
  plot(Th,wait=1,ps="emptymesh-2.eps"); // see figure 3.11
  savemesh(Th,"emptymesh-2.msh");
}
```

3.10 Get Mesh numbering

```
{ // get mesh information (version 1.37)
```

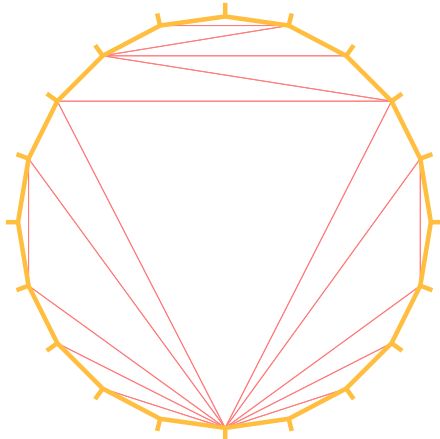


Figure 3.10: The empty mesh with boundary

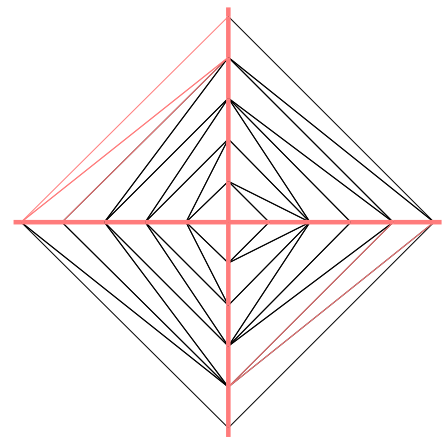


Figure 3.11: An empty mesh defined from a pseudo region numbering of triangle

```

mesh Th=square(2,2);
// get data of the mesh
int nbtriangles=Th.nt;
for (int i=0;i<nbtriangles;i++)
  for (int j=0; j <3; j++)
    cout << i << " " << j << " Th[i][j] = "
      << Th[i][j] << " x = "<< Th[i][j].x << " , y= "<< Th[i][j].y
      << " , label=" << Th[i][j].label << endl;
}

```

the output is:

```

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

```

3.11 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=[[1a_1,sa_1],[1b_1,sb_1],  
                ...  
                [1a_k,sa_k],[1b_k,sb_k]] ) ;
```

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 `[1ai,sai],[1bi,sbi]`. The `int` expressions `1ai` and `1bi` are defined the 2 labels of the piece of the boundary to be equivalence, and the `real` expressions `sai` and `sbi` give two common abscissa on the two boundary curve, and two points are identify if the two abscissa are equal.

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

P0 piecewise constante discontinuous finite element

$$P0_h = \{v \in L^2(\Omega) ; \forall K \in \mathcal{T}_h \exists \alpha_K \in \mathbb{R} : v|_K = \alpha_K\} \quad (4.1)$$

P1 piecewise linear continuous finite element

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

P1dc piecewise linear discontinuous finite element

$$P1dc_h = \{v \in L^2(\Omega) ; \forall K \in \mathcal{T}_h v|_K \in P_1\} \quad (4.3)$$

P1b piecewise linear continuous finite element plus bubble

$$P1b_h = \{v \in H^1(\Omega) ; \forall K \in \mathcal{T}_h v|_K \in P_1 \oplus \text{Span}\{\lambda_0^K \lambda_1^K \lambda_2^K\}\} \quad (4.4)$$

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

P2 piecewise P_2 continuous finite element,

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

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

P2dc piecewise P_2 discontinuous finite element,

$$P2dc_h = \{v \in L^2(\Omega) ; \forall K \in \mathcal{T}_h \quad v|_K \in P_2\} \quad (4.6)$$

RT0 Raviart-Thomas finite element

$$RT0_h = \{\mathbf{v} \in H(\text{div})/\forall K \in \mathcal{T}_h \quad \mathbf{v}|_K(x, y) = \begin{vmatrix} \alpha_K \\ \beta_K \end{vmatrix} + \gamma_K \begin{vmatrix} x \\ y \end{vmatrix}\} \quad (4.7)$$

where $H(\text{div})$ 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

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

$$X_{ph} = \{v \in X_h ; v(\cdot|_0) = v(\cdot|_1), v(\cdot|_0) = v(\cdot|_1)\}$$

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

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

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
fespace Mh(Th,P2); // scalar FE
fespace Rh(Th,RT0); // vectorial FE

```

so X_h, M_h, R_h 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 $U_h, V_h \in R_h$ 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

$$U_h = \begin{vmatrix} U_{xh} \\ U_{yh} \end{vmatrix} \quad \text{and} \quad V_h = \begin{vmatrix} V_{xh} \\ V_{yh} \end{vmatrix}$$

Like in the previous version, `freefem+`, the finite element functions (type FE functions) are both functions from \mathbb{R}^2 to \mathbb{R}^N with $N = 1$ for scalar function 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; // interpolate vh = ((x-1/2)^2 + Y^2)

```

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

```

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

```

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

```

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

```

Warning for no scalar finite element function `[Uxh,Uyh]` the two array `Uxh[]` and `Uyh[]` are the same array, because the degree of freedom can touch more than one component.

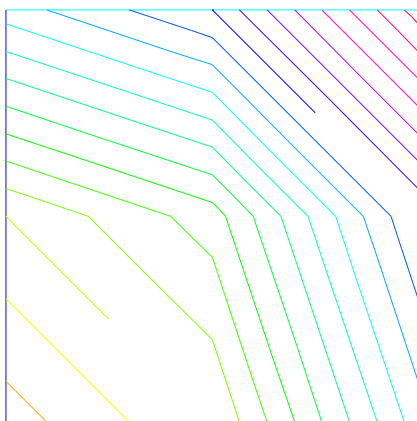
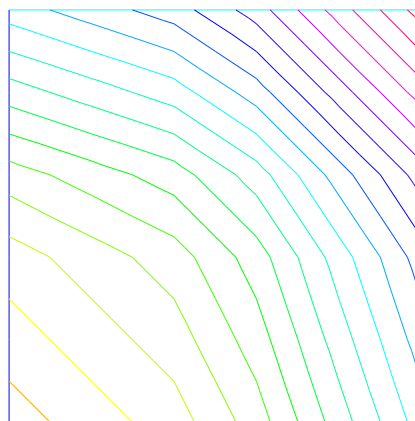
The other way to set a FE function is to solve a ‘problem’ (see below).

Note 6 *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` can be rebuild if the associated mesh have changed 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 always 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: v_h Iso on mesh 2×2 Figure 4.2: v_h 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:

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

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);
Vh uh,vh;
func f=1;
func g=0;

solve laplace(uh,vh) =
  int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) )
+ int2d(Th)( -f*vh )
+ on(1,2,3,4,uh=g) ;

f=x+y;
laplace;
plot(uh,ps="Laplace.eps",value=true);

```

Note 7 Using the keyword `problem` in place of `solve` would define the problem only and not solve it.

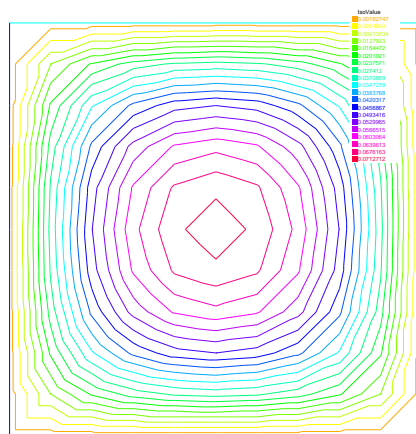


Figure 4.3: Isovalues of the solution

A laplacian in mixed finite formulation .

```

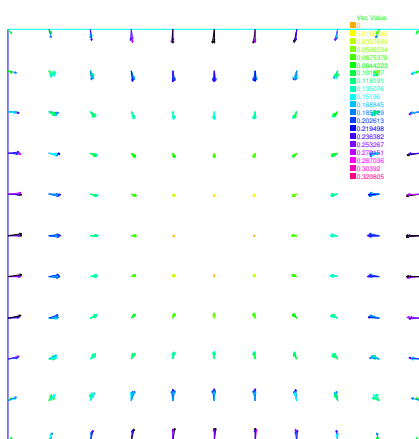
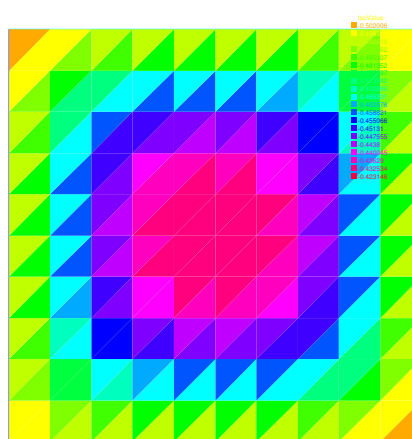
mesh Th=square(10,10);
fespace Vh(Th,RT0);
fespace Ph(Th,P0);

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

```

Figure 4.4: Flux (u_1, u_2) Figure 4.5: Isovalue of p

Note 8 To make programs more readable we stop now using blue color on dx, dy .

An other formulation of the Laplace equation with Discontinuous Galerkin formulation with P_2 discontinuous can be write (see LapDC.epd and [17])

```

// file: LapDG2.edp
// solve  $-\Delta u = f$  on  $\Omega$  and  $u = g$  on  $\Gamma$ 
macro dn(u) (N.x*dx(u)+N.y*dy(u) ) // def the normal derivative

mesh Th = square(10,10); // unite square
fespace Vh(Th,P2dc); // Discontinuous P2 finite element
// if pena = 0 => Vh must be P2 otherwise we need some penalisation
real pena=0; // a paramater to add penalisation
varf Ans(u,v)=
  int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v) )
+ intalledges(Th)( // loop on all edge of all triangle
// the edge are see nTonEdge times so we / nTonEdge
// remark: nTonEdge =1 on border edge and =2 on internal
// we are in a triange th normal is the exterior normal
// def: jump = external - internal value; on border exter value =0
// average = (external + internal value)/2, on border just internal
value
  ( jump(v)*average(dn(u)) - jump(u)*average(dn(v)) + pena*jump(u)*jump(v) )
/ nTonEdge
) ;
func f=1;
func g=0;
Vh u,v;
Xh uu,vv;
problem A(u,v,solver=UMFPACK) //
= Ans
- int2d(Th)(f*v)
- int1d(Th)(g*dn(v) + pena*g*v)
;
problem A1(uu,vv,solver=CG) //
=
int2d(Th)(dx(uu)*dx(vv)+dy(uu)*dy(vv)) - int2d(Th)(f*vv) + on(1,2,3,4,uu=g);

A; // solve DG

```

where

- $nTonEdge$ is the number on triangle which see the current edge, with $nTonEdge==2$ on internal edge and $nTonEdge==1$ on boundary edge,
- $jump$ give the jump in the direction of the external normal: external minus internal value on internal edges, and minus the internal value
- $average$ is the half of the external plus internal value on internal edges and the internal value on boundary edges.

4.2 Parameter Description for solve and problem

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

Note 9 *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,UMFPACK ...

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

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

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

if it is positive then the stopping test is

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

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

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

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

tgvs= Huge value (10^{30}), to lock boundary conditions

4.3 Problem definition

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

- a name; this is the name given to the variational form (type `varf`) for possible reuse.

- the bilinear form term:

$$\text{-) } \text{int2d}(\text{Th})(\mathbf{K}^*\mathbf{v}^*\mathbf{w}) = \sum_{T \in \text{Th}} \int_T K v w$$

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

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

$$\text{-) } \text{intalldedges}(\text{Th})(\mathbf{K}^*\mathbf{v}^*\mathbf{w}) = \sum_{T \in \text{Th}} \int_{\partial T} K v w$$

$$\text{-) } \text{intalldedges}(\text{Th}, 1)(\mathbf{K}^*\mathbf{v}^*\mathbf{w}) = \sum_{T \in \text{Th}, T \subset \Omega_1} \int_{\partial T} K v w$$

-) a sparse matrix of type `matrix`

- the linear form term:

$$\text{-) } \text{int1d}(\text{Th})(\mathbf{K}^*\mathbf{w}) = \sum_{T \in \text{Th}} \int_T K w$$

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

$$\text{-) } \text{intalldedges}(\text{Th})(\mathbf{f}^*\mathbf{w}) = \sum_{T \in \text{Th}} \int_{\partial T} f w$$

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

- The boundary condition form term :

- An "on" form (for Dirichlet) : `on(1, u = g)`

- a linear form on Γ (for Neumann) `-int1d(Th)(f*w)` or `-int1d(Th,3)(f*w)`

- a bilinear form on Γ or Γ_2 (for Robin) `int1d(Th)(K*v*w)` or `int1d(Th,2)(K*v*w)`.

If needed, the different kind of terms in the sum can appear more than once.

Remark: the integral mesh and the mesh associated to test function or unknown function can be different in the case of linear form.

Note 10 N_x and N_y are the normal's components.

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

4.4 Integrals

There are three kinds of integrals:

- surface integral defined with the keyword `int2d`
- integrals on curves `int1d`.
- integrals on the three edges of all triangles `intalledges`, remark the edges are see two times generally (the variable `nTonEdge` give this number).

the syntaxe is :

```
keyword(domain_parameters)( function )
```

where the `(domain_parameters)` given the definition of the domain of integration and the kind of quadrature formulae.

The `(domain_parameters)` can be :

- `(Th) ???`
- `(Th,1) ???`
- `(Th,1,qforder=2) ???`
- `(Th,1,qft= qf3pT, qfe= qf2pE) ???`

where where `Th` is a mesh.

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 quadrature formulae on triangles are:

name (qft=)	on	order qforder=	exact	number of quadrature points
<code>qf1pT</code>	triangle	2	1	1
<code>qf2pT</code>	triangle	3	2	3
<code>qf3pT</code>	triangle	6	4	7
<code>qf1pTlump</code>	triangle		4	3
<code>qf2pT4P1</code>	triangle		2	9

The quadrature formulae on edges are:

name (qfe=)	on	order qforder=	exact	number of quadrature points
<code>qf1pE</code>	segment	2	1	1
<code>qf2pE</code>	segment	3	2	2
<code>qf3pE</code>	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) ;

```

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

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)
matrix By= by(Xh,Mh); // By = (By_ij) and By_ij = by(b_j^x, b_j^m)
// where b_j^x is a basis of Xh, and b_j^m is a basis of Mh.

```

Note 11 The line of the matrix corresponding to test function on the bilinear form.

Note 12 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

```

Note 13 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 k th vector will contain the real part and the $k + 1$ th vector the imaginary part of the corresponding complex conjugate eigenvectors.

¹<http://www.caam.rice.edu/software/ARPACK/>

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

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

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, pi[^2
//      -----
//      we use the inverse shift mode
//      the shift is given with the real sigma
//      -----
//      find lambda and u_lambda in H_0^1(Omega) such that:
//
//
//      \int_{\Omega} \nabla u_{\lambda} \nabla v = \lambda \int_{\Omega} u_{\lambda} v, \forall v \in H_0^1(\Omega)
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=tdv on the diagonal term of the lock degre of freedom.
//      So take dirichlet boundary condition just on a variationnal form
//      and not on b variationnal form.
//      because we solve w = OP^-1 * B * v

int nev=20; //      number of computed eigen valeu close to sigma

```

```

real[int] ev(nev); // to store the nev eigenvalue
Vh[int] eV(nev); // to store the nev eigenvector

int k=EigenValue(OP,B,sym=true,sigma=sigma,value=ev,vector=eV,
                tol=1e-10,maxit=0,ncv=0);

// tol= the tolerace
// maxit= the maximum iteration see arpack doc.
// ncv see arpack doc. http://www.caam.rice.edu/software/ARPACK/
// the return value is number of converged eigen value.

for (int i=0;i<k;i++)
{
  ul=eV[i];
  real gg = int2d(Th)(dx(ul)*dx(ul) + dy(ul)*dy(ul));
  real mm= int2d(Th)(ul*ul) ;
  cout << " ---- " << i<< " " << ev[i]<< " err= "
        <<int2d(Th)(dx(ul)*dx(ul) + dy(ul)*dy(ul) - (ev[i])*ul*ul) << " --- "<<endl;
  plot(eV[i],cmm="Eigen Vector "+i+" valeur =" + ev[i] ,wait=1,value=1);
}

```

The output of this example is:

```

Nb of edges on Mortars = 0
Nb of edges on Boundary = 80, neb = 80
Nb Of Nodes = 1681
Nb of DF = 1681
Real symmetric eigenvalue problem: A*x - B*x*lambda

```

```

Thanks to ARPACK++ class ARrcSymGenEig
Real symmetric eigenvalue problem: A*x - B*x*lambda
Shift and invert mode sigma=20

```

```

Dimension of the system          : 1681
Number of 'requested' eigenvalues : 20
Number of 'converged' eigenvalues : 20
Number of Arnoldi vectors generated: 41
Number of iterations taken       : 2

```

```

Eigenvalues:
lambda[1]: 5.0002
lambda[2]: 8.00074
lambda[3]: 10.0011
lambda[4]: 10.0011
lambda[5]: 13.002
lambda[6]: 13.0039
lambda[7]: 17.0046
lambda[8]: 17.0048
lambda[9]: 18.0083
lambda[10]: 20.0096
lambda[11]: 20.0096

```

```

lambda[12]: 25.014
lambda[13]: 25.0283
lambda[14]: 26.0159
lambda[15]: 26.0159
lambda[16]: 29.0258
lambda[17]: 29.0273
lambda[18]: 32.0449
lambda[19]: 34.049
lambda[20]: 34.0492

```

```

---- 0 5.0002 err= -0.000225891 ---
---- 1 8.00074 err= -0.000787446 ---
---- 2 10.0011 err= -0.00134596 ---
---- 3 10.0011 err= -0.00134619 ---
---- 4 13.002 err= -0.00227747 ---
---- 5 13.0039 err= -0.004179 ---
---- 6 17.0046 err= -0.00623649 ---
---- 7 17.0048 err= -0.00639952 ---
---- 8 18.0083 err= -0.00862954 ---
---- 9 20.0096 err= -0.0110483 ---
---- 10 20.0096 err= -0.0110696 ---
---- 11 25.014 err= -0.0154412 ---
---- 12 25.0283 err= -0.0291014 ---
---- 13 26.0159 err= -0.0218532 ---
---- 14 26.0159 err= -0.0218544 ---
---- 15 29.0258 err= -0.0311961 ---
---- 16 29.0273 err= -0.0326472 ---
---- 17 32.0449 err= -0.0457328 ---
---- 18 34.049 err= -0.0530978 ---
---- 19 34.0492 err= -0.0536275 ---

```

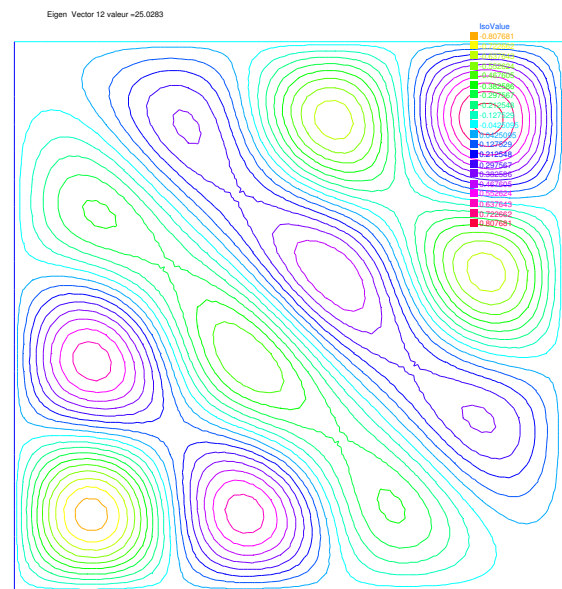
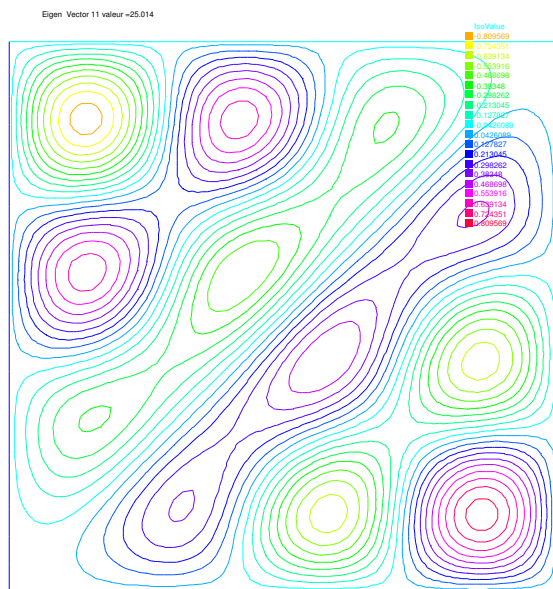


Figure 4.6: Isovalue of 11th eigenvector $u_{4,3} - u_{3,4}$

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,
 - g** to toggle to grey or color ,
 - v** to toggle the plotting of value,
 - p** to save to a postscript file,
 - ?** to show all actives keyboard char,
- to redraw, otherwise we continue.

ps= string expression to save the plot on postscript file

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

fill= to fill between isovalues.

cmm= string expression to write in the graphic window

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

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

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

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

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

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

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

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

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

For example:

```

real[int] xx(10),yy(10);
mesh Th=square(5,5);
fespace Vh(Th,P1);
Vh uh=x*x+y*y,vh=-y^2+x^2;
int i;

for (i=0;i<10;i++) // compute a cut
{
  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
// zoom on box defined by the two corner points [0.1,0.2] and [0.5,0.6]
plot(uh,[uh,vh],bb=[[0.1,0.2],[0.5,0.6]],
wait=true,grey=1,fill=1,value=1,ps="threeg.eps"); // figure 4.9
plot([xx,yy],ps="likegnu.eps",wait=true); // figure 4.10

```

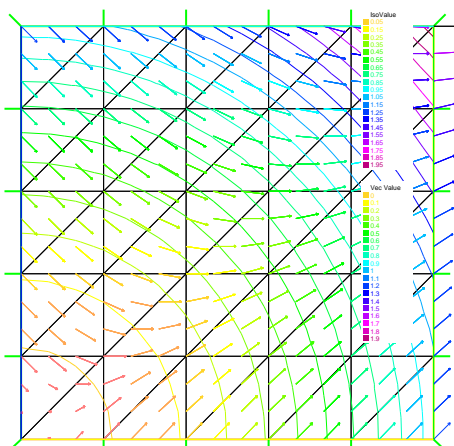


Figure 4.8: mesh, isovalue, and vector

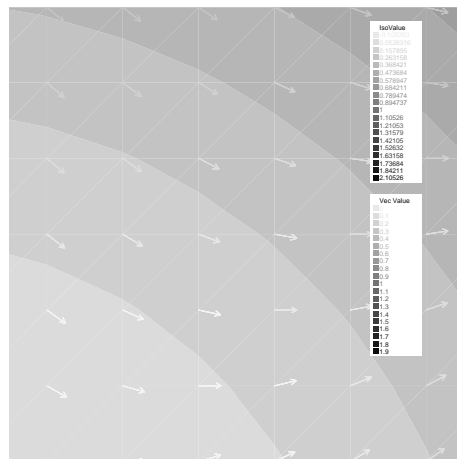


Figure 4.9: enlargement in grey of isovalue, and vector

4.8 link with gnuplot

First this work only if gnuplot² is installed , and only on unix computer.
You just and to the previous example:

²<http://www.gnuplot.info/>

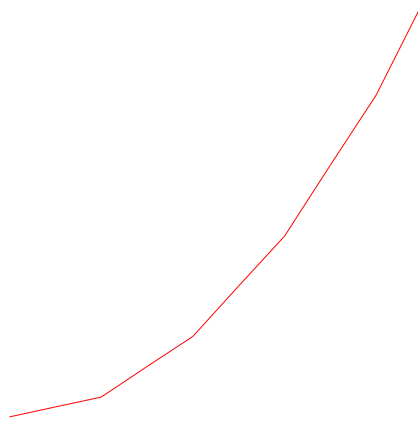


Figure 4.10: Plots a cut of u_h . Note that a refinement of the same can be obtained in combination with gnuplot

```

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

// to call gnuplot command and wait 5 second (tanks to unix command)
// and make postscript plot

exec("echo 'plot \"plot.gp\" w l \
pause 5 \
set term postscript \
set output \"gnuplot.eps\" \
replot \
quit' | gnuplot");

```

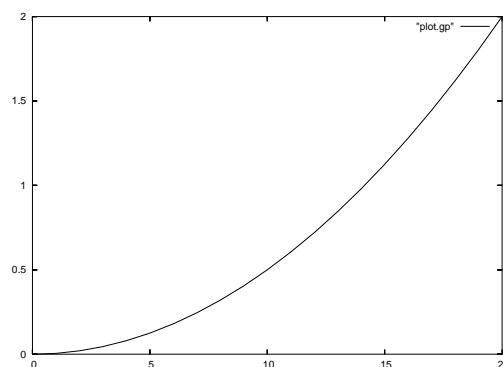


Figure 4.11: Plots a cut of u_h with gnuplot

4.9 link with medit

First this work only if medit ³ software is installed.

```

// build square ]-1,1[2
mesh Th=square(10,10,[2*x-1,2*y-1]);
fespace Vh(Th,P1);
Vh u=2-x*x-y*y;

savemesh(Th, "mm", [x,y,u*.5]); // save mm.points and mm.faces file
// for medit
// build a mm.bb file

{ ofstream file("mm.bb");
file << "2 1 1 " << u[.n << " 2 \n";
int j;
for (j=0;j<u[.n ; j++)
file << u[][j] << endl;
}

// call medit command
exec("medit mm");

// clean files on unix OS
exec("rm mm.bb mm.faces mm.points");

```

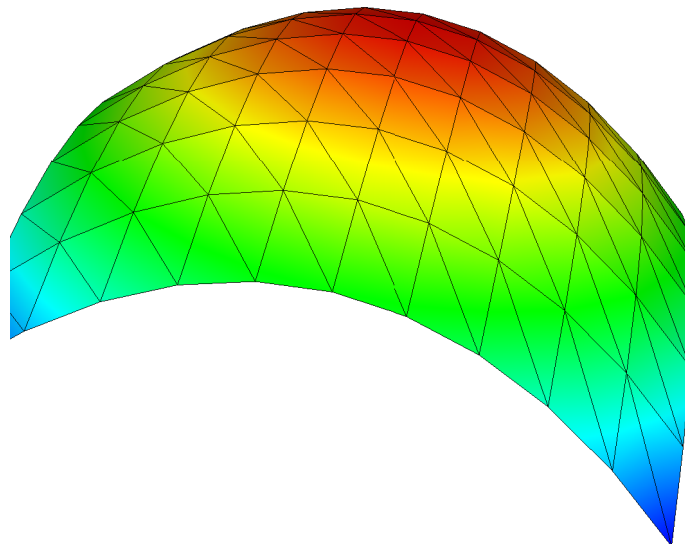


Figure 4.12: medit plot

³<http://www-rocq.inria.fr/gamma/medit/medit.html>

4.10 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, $\phi^n \circ X^n$ 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**⁴ the time step,

<exp4> is the name of the finite element function which is convected (u in the example 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.

⁴The minus is due to the backwark schema

Chapter 5

algorithm

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

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 convex 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.

remark, you can use the macro tools (see 6.13) to build easily the differential see example `Newtow.edp`.

Two version are available:

linearCG linear case , the functional J is quadratic.

NLCG non linear case (the function is just convex).

The named parameter of this two function are:

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

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

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

veps= set 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 useful 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 [16] 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

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

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 fempl(disk,P1); // define the fempl space  
fempl u,v; // introduce 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
femplot 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 Component - 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 fonction (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 Component - 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 the curve abscissa
// 1 and 2 are bottom and upper side with x the curve abscissa
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) = // definition 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);

```

An over example is in `periodic4.edp` file, with periodic condition no parallel to the axis. Solve a diamond with a hole:

```

real r=0.25; // a diamond with a hole

border a(t=0,1){x=-t+1; y=t;label=1;};
border b(t=0,1){ x=-t; y=1-t;label=2;};
border c(t=0,1){ x=t-1; y=-t;label=3;};
border d(t=0,1){ x=t; y=-1+t;label=4;};
border e(t=0,2*pi){ x=r*cos(t); y=-r*sin(t);label=0;};
int n = 10;
mesh Th= buildmesh(a(n)+b(n)+c(n)+d(n)+e(n));
plot(Th,wait=1);
real r2=1.732;
func abs=sqrt(x^2+y^2);
// warning for periodic condition:

```

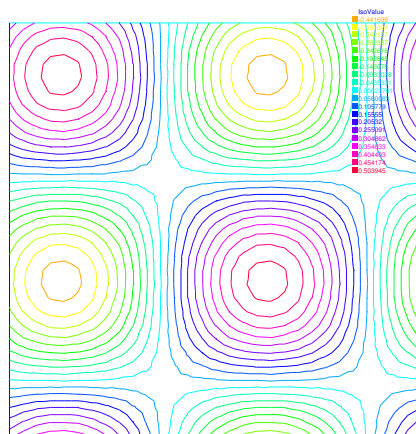


Figure 6.1: The isovalue of solution u with periodic boundary condition

```
// side a and c
// on side a (label 1)  $x \in [0,1]$  or  $x-y \in [-1,1]$ 
// on side c (label 3)  $x \in [-1,0]$  or  $x-y \in [-1,1]$ 
//           // so the common abscissa can be respectively  $x$  and  $x+1$ 
//           // or you can try curviline abscissa  $x-y$  and  $x-y$ 
// 1 first way
// fespace Vh(Th,P2,periodic=[[2,1+x],[4,x],[1,x],[3,1+x]]);
// 2 second way
fespace Vh(Th,P2,periodic=[[2,x+y],[4,x+y],[1,x-y],[3,x-y]]);

Vh uh,vh;

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

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;};
```

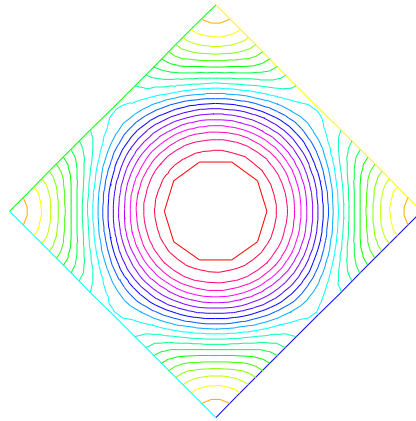


Figure 6.2: The isovalue of solution u for $\Delta u = ((y+x)^2 + 1)((y-x)^2 + 1) - k$, in Ω and $\partial_n u = 0$ on hole, and with two periodic boundary condition on external border

```
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 the 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 adaptindicatorP2.edp

In this example, we do metric mesh adaption and we computationj the classical residual error indicator η_K on the element K for the Poisson problem.

First, we solve the same problem as in a previous example.

```
border ba(t=0,1.0){x=t; y=0; label=1;}; // comment
border bb(t=0,0.5){x=1; y=t; label=2;};
border bc(t=0,0.5){x=1-t; y=0.5;label=3;};
border bd(t=0.5,1){x=0.5; y=t; label=4;};
border be(t=0.5,1){x=1-t; y=1; label=5;};
border bf(t=0.0,1){x=0; y=1-t;label=6;};
mesh Th = buildmesh (ba(6) + bb(4) + bc(4) +bd(4) + be(4) + bf(6));

fespace Vh(Th,P2);
Vh u,v;

real error=0.01;
func f=(x-y);
problem Probem1(u,v,solver=CG,eps=1.0e-6) =
  int2d(Th,qforder=5)( u*v*1.0e-10+ dx(u)*dx(v) + dy(u)*dy(v))
  + int2d(Th,qforder=5)( -f*v);
```

Now, the local error indicator η_K is:

$$\eta_K = \left(h_K^2 \|f - \Delta u_h\|_{L^2(K)}^2 + \sum_{e \in \mathcal{E}_K} h_e \left\| \left[\frac{\partial u_h}{\partial n_k} \right] \right\|_{L^2(e)}^2 \right)^{\frac{1}{2}}$$

where h_K is the longest's edge of K , \mathcal{E}_K is the set of K edge not on $\Gamma = \partial\Omega$, n_K is the outside unit normal to K , h_e is the length of edge e , $[g]$ is the jump of the function g across edge (left value minus righth value).

Of course, we can use a variational form to compute η_K^2 , with test function constante function by triangle.

```

fespace Nh(Th,P0);          // the set function constante function by triangle
Nh eta,logeta;
varf indicator2(uu,chiK) =
    intalldges(Th)(chiK*lenEdge*square( jump(N.x*dx(u)+N.y*dy(u)))
    +int2d(Th)(chiK*square(hTriangle*(f-dxx(u)-dyy(u))) );

for (int i=0;i< 4;i++)
{
  Probem1;
  cout << u[].min << " " << u[].max << endl;
  plot(u,wait=1);
  cout << " indicator2 " << endl;

  eta[] = indicator2(0,Nh);
  eta=sqrt(eta);
  cout << eta[].min << " " << eta[].max << endl;
  plot(eta,fill=1,wait=1,cmm="indicator density ",ps="rhoP2.eps");
  Th=adaptmesh(Th,[dx(u),dy(u)],err=error,anisomax=1);
  plot(Th,wait=1);
  u=u;
  eta=eta;
  error = error/2;
} ;

```

If the method is correct, we expect an almost constant function η , as you can see on the graphics .labelfig rhop2.

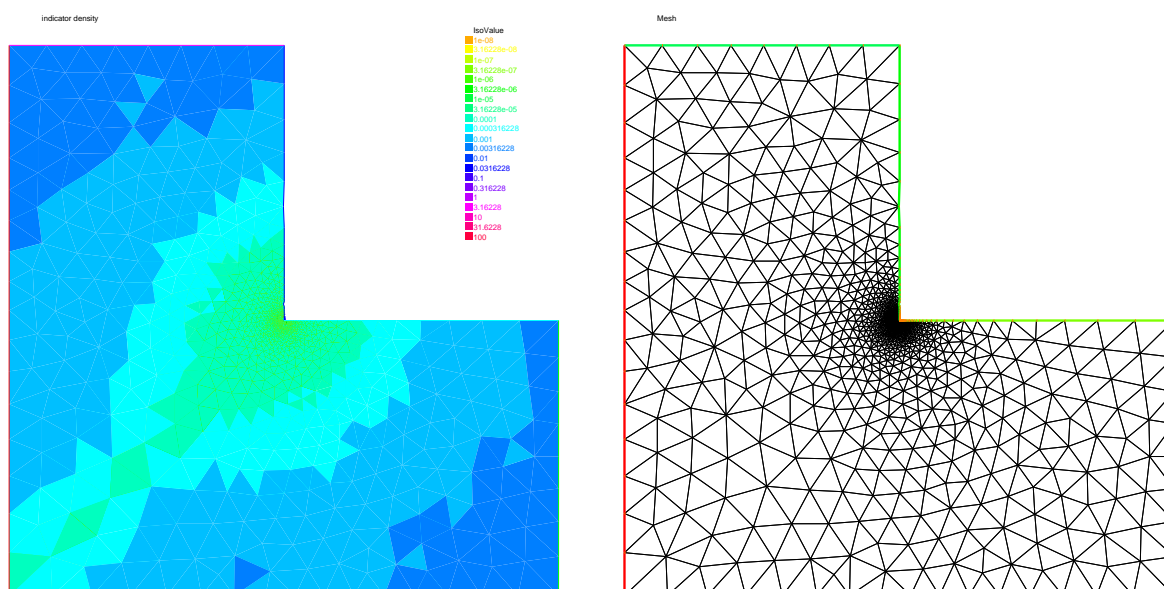


Figure 6.3: Density of the error indicator with isotropic P^2 metric

6.5 Algo.edp

We propose to solve the following non-linear academic problem of minimization of a functional

$$J(u) = \int_{\Omega} f(|\nabla u|^2) - u * b$$

where u is function of $H_0^1(\Omega)$. and where f is defined by

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

6.5.1 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) - ∫Ω ub
// 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

$$\forall v_h \in V_{0h}, \quad \int_{\Omega} \alpha \nabla u_h \cdot \nabla v_h = \int_{\Omega} b v_h$$

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 preconditionner 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.

6.5.2 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 introduce 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''(|\nabla u|^2) optimisation

// the variational form of evaluate dJ = \nabla 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 = \nabla^2 J
// 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");

```

6.6 Stokes and Navier-Stokes

The Stokes equations are:

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

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

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

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

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

and

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

The FE spaces and functions are constructed by

```
fespace Xh(Th,P2);           // definition of the velocity component space
fespace Mh(Th,P1);         // definition of the pressure space
Xh u2,v2;
Xh u1,v1;
Xh p,q;
```

The Stokes operator is implemented as a system-solve for the velocity (u_1, u_2) and the pressure p . The test function for the velocity is (v_1, v_2) 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.

Note 14 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 $\text{rot}\psi = u$ or better,

$$-\Delta\psi = \nabla \times u$$

```
Xh psi,phi;
```

```
solve streamlines(psi,phi) =
  int2d(Th)( dx(psi)*dx(phi) + dy(psi)*dy(phi))
+ int2d(Th)( -phi*(dy(u1)-dx(u2)))
```

```
+ on(1,2,3,4,psi=0);
```

Now the Navier-Stokes equations are solved

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

with the same boundary conditions and with initial conditions $u = 0$.

This is implemented by using the convection operator `convect` for the term $\frac{\partial u}{\partial t} + u \cdot \nabla u$, giving a discretization in time

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

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=Croust,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 $\nabla \cdot u(p) = 0$. The last problem is linear, symmetric negative, so we can use the conjugate gradient algorithm .

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 $\text{divup } p \rightarrow \nabla \cdot u(p)$.

```

func real[int] divup(real[int] & pp)
{
    b[] = Bx'*pp; b[] += bc1[] ; u1[] = A^-1*b[]; // compute u1(pp)
    b[] = By'*pp; b[] += bc2[] ; u2[] = A^-1*b[]; // compute u2(pp)
    // div(u1,u2) = Bx'*u1[] + By'*u2[];
    ppp[] = Bx*u1[]; // ppp = tBxu1
    ppp[] += By*u2[]; // + tByu2
    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 preconditioner is that in a periodic domain, all differential operators commute and the Uzawa algorithm comes to solving the linear operator $\nabla \cdot ((\alpha Id + \nu \Delta)^{-1} \nabla)$, 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)wi
// 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 absolute 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 => absolute 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 "formatted Marrocco"
savemesh(th,"toto.Th");                              // format database db mesh "bamg"
savemesh(th,"toto.msh");                             // format freefem
savemesh(th,"toto.nopo");                            // modulef format see [7]
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 "bang : a bidimensional 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 examples, of domain decomposition technique: first, Schwarz algorithm with overlapping, second Schwarz algorithm without overlapping (also call Shur complement), and last we show to use the conjugate gradient to solve the boundary problem of the Shur complement.

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.

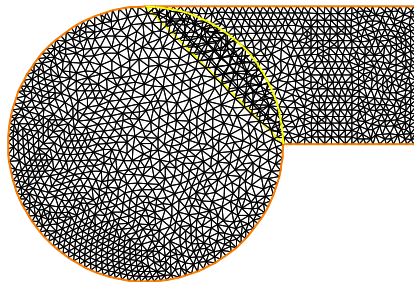


Figure 6.4: The 2 overlapping mesh TH and th

```

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);
};

```

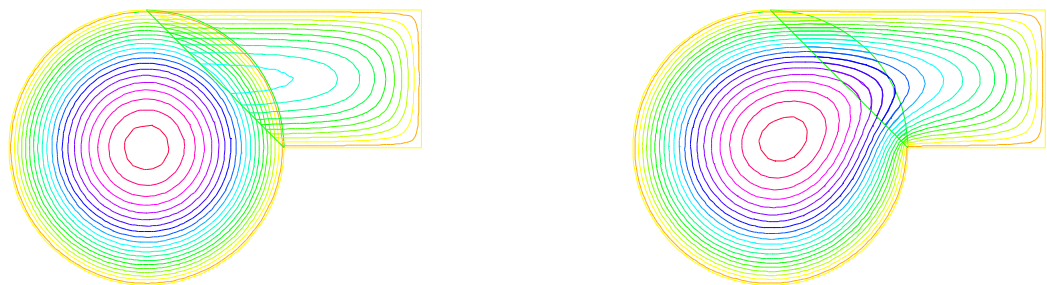


Figure 6.5: Isovalues of the solution at iteration 0 and iteration 9

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

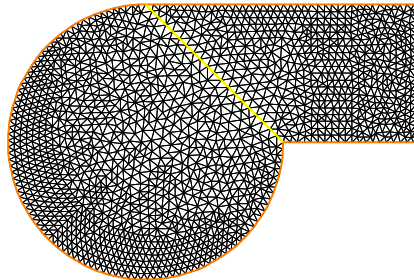


Figure 6.6: The two none overlapping mesh TH and th

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 problem 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 upgrading λ 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");

```

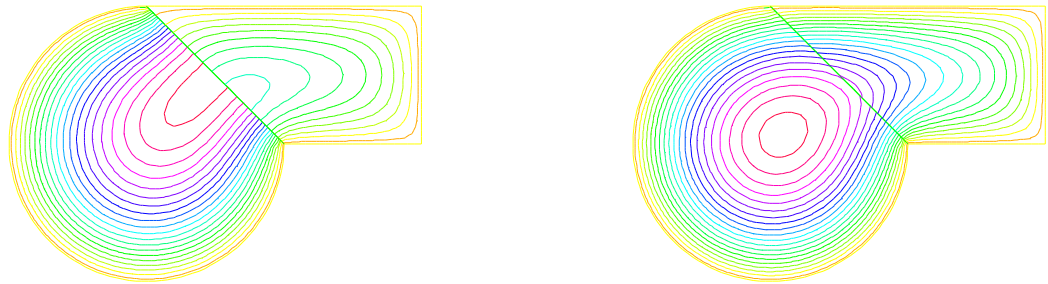


Figure 6.7: Isovalues of the solution at iteration 0 and iteration 9 without overlapping

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 problem 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 component. The border problem is solve with conjugate gradient.

First, we construct the two domain

```

// Schwarz without overlapping (Shur complement Neumann -> Dirichet)
real cpu=clock();
int inside = 2;
int outside = 1;

border Gamma1(t=1,2){x=t;y=0;label=outside;};
border Gamma2(t=0,1){x=2;y=t;label=outside;};
border Gamma3(t=2,0){x=t ;y=1;label=outside;};

border GammaInside(t=1,0){x = 1-t; y = t;label=inside;};

border GammaArc(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;

// build the mesh of  $\Omega_1$  and  $\Omega_2$ 
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  $\lambda \in V_{h1}$ 
```

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[]=1; // make FE function form 1
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, originally 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 :

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

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 \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;}; // righth beam
border d(t=0,10) { x=10-t; y=2; label=3;}; // top beam
real E = 21.5;
real sigma = 0.29;
real mu = E/(2*(1+sigma));

```

```

real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -0.05;
mesh th = buildmesh( b(20)+c(5)+d(20)+a(5));
fespace Vh(th,[P1,P1]);
Vh [uu,vv], [w,s];
cout << "lambda,mu,gravity ="<<lambda<< " " << mu << " " << gravity << endl;
//      deformation of a beam under its own weight
solve bb([uu,vv],[w,s]) =
    int2d(th)(
        2*mu*(dx(uu)*dx(w)+dy(vv)*dy(s)+ ((dx(vv)+dy(uu))*(dx(s)+dy(w)))/2
    )
        + lambda*(dx(uu)+dy(vv))*(dx(w)+dy(s))
    )
    + int2d(th) (-gravity*s)
    + on(1,uu=0,vv=0)
    ;

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

```

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 :

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

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, originally 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 :

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

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

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

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;}; //      righth 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 at 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 uul=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 keyword
//      construct a mesh with 4 regions (sub-domains)

border a(t=0,1){x=t;y=0;};
border b(t=0,0.5){x=1;y=t;};
border c(t=0,0.5){x=1-t;y=0.5;};
border d(t=0.5,1){x=0.5;y=t;};
border e(t=0.5,1){x=1-t;y=1;};
border f(t=0,1){x=0;y=1-t;};

//      internal boundary

border i1(t=0,0.5){x=t;y=1-t;};
border i2(t=0,0.5){x=t;y=t;};
border i3(t=0,0.5){x=1-t;y=t;};

mesh th = buildmesh (a(6) + b(4) + c(4) +d(4) + e(4) +
    f(6)+i1(6)+i2(6)+i3(6));
fespace Ph(th,P0); //      constant discontinuous functions / element
fespace Vh(th,P1); //      P1 ontinuous functions / element

Ph reg=region; //      defined the P0 function associated to region number
plot(reg,fill=1,wait=1,value=1);

```

`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:

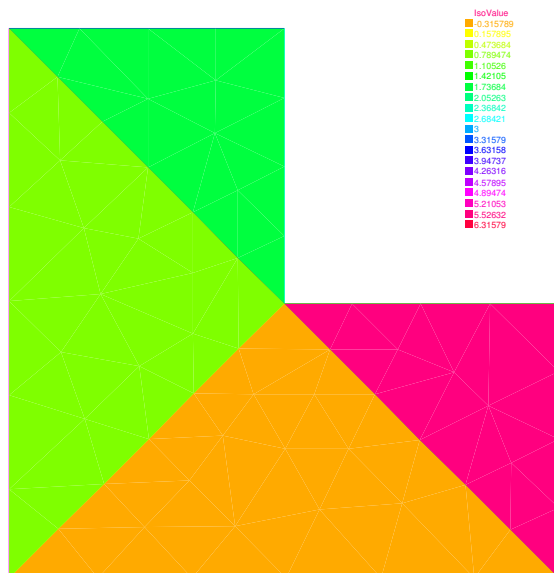


Figure 6.8: the function reg

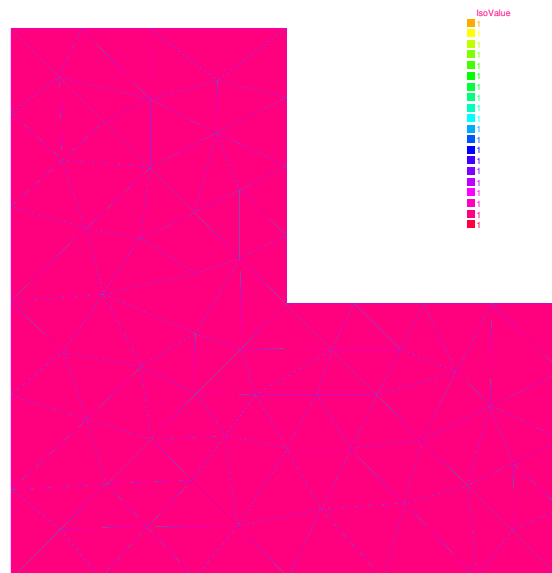


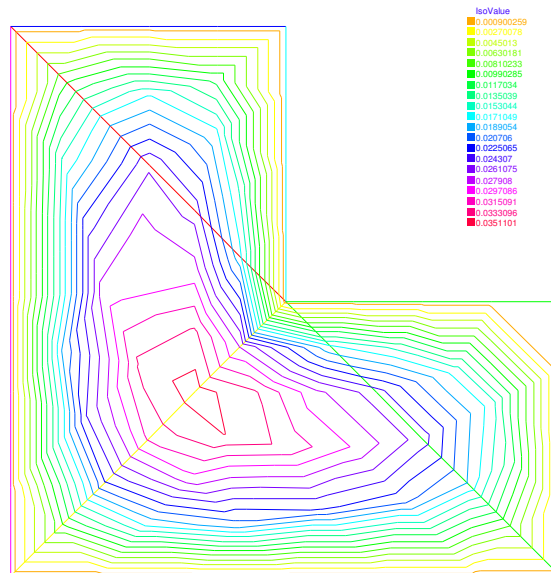
Figure 6.9: the function nu

```
int nupper=reg(0.4,0.9);           // get the region number of point (0.4,0.9)
int nlower=reg(0.9,0.1);          // get the region number of point (0.4,0.1)
cout << " nlower " << nlower << ", nupper = " << nupper<< endl;
// defined the characteristics functions of upper and lower region
Ph nu=1+5*(region==nlower) + 10*(region==nupper);
plot(nu,fill=1,wait=1);
```

This is particularly useful to define discontinuous functions such as might occur when one part of the domain is copper and the other one is iron, for example.

We this in mind we proceed to solve a Laplace equation with discontinuous coefficients (ν is 1, 6 and 11 below).

```
Ph nu=1+5*(region==nlower) + 10*(region==nupper);
plot(nu,fill=1,wait=1);
problem lap(u,v) = int2d(th)( dx(u)*dx(v)*dy(u)*dy(v)) + int2d(-1*v) + on(a,b,c,d,e,f);
plot(u);
```

Figure 6.10: the isovalue of the solution u

6.12 FreeBoundary.edp

The domain Ω is defined with:

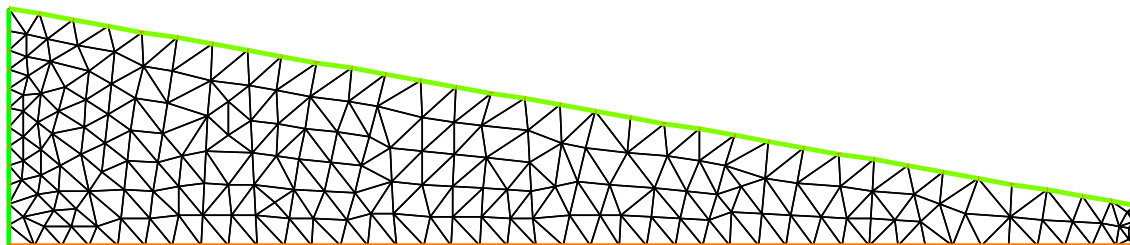
```

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 trapèze
border a(t=0,L){x=t;y=0;}; // bottom:  $\Gamma_a$ 
border b(t=0,h1){x=L;y=t;}; // right:  $\Gamma_b$ 
border f(t=L,0){x=t;y=t*(h1-h)/L+h;}; // free surface:  $\Gamma_f$ 
border d(t=h,0){x=0;y=t;}; // left:  $\Gamma_d$ 

int n=4;
mesh Th=buildmesh (a(10*n)+b(6*n)+f(8*n)+d(3*n));
plot(Th,ps="dTh.eps");

```

Figure 6.11: The mesh of the domain Ω

The free boundary problem is:

Find u and Ω such that:

$$\left\{ \begin{array}{ll} -\Delta u = 0 & \text{in } \Omega \\ u = y & \text{on } \Gamma_b \\ \frac{\partial u}{\partial n} = 0 & \text{on } \Gamma_d \cup \Gamma_a \\ \frac{\partial u}{\partial n} = \frac{q}{K} n_x \text{ and } u = y & \text{on } \Gamma_f \end{array} \right.$$

We use a fixe point method; $\Omega^0 = \Omega$

in two step, fist we solve the classical following problem:

$$\left\{ \begin{array}{ll} -\Delta u = 0 & \text{in } \Omega^n \\ u = y & \text{on } \Gamma_b^n \\ \frac{\partial u}{\partial n} = 0 & \text{on } \Gamma_d^n \cup \Gamma_a^n \\ u = y & \text{on } \Gamma_f^n \end{array} \right.$$

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:

$$\left\{ \begin{array}{ll} -\Delta v = 0 & \text{in } \Omega^n \\ v = 0 & \text{on } \Gamma_a^n \\ \frac{\partial v}{\partial n} = 0 & \text{on } \Gamma_b^n \cup \Gamma_d^n \\ \frac{\partial v}{\partial n} = \frac{\partial u}{\partial n} - \frac{q}{K} n_x & \text{on } \Gamma_f^n \end{array} \right.$$

The varitional formulation is:

find v on V , such than $v = 0$ on Γ_a^n

$$\int_{\Omega^n} \nabla v \nabla v' = \int_{\Gamma_f^n} \left(\frac{\partial u}{\partial n} - \frac{q}{K} n_x \right) v', \quad \forall v' \in V \text{ with } v' = 0 \text{ on } \Gamma_a^n$$

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.;
  }

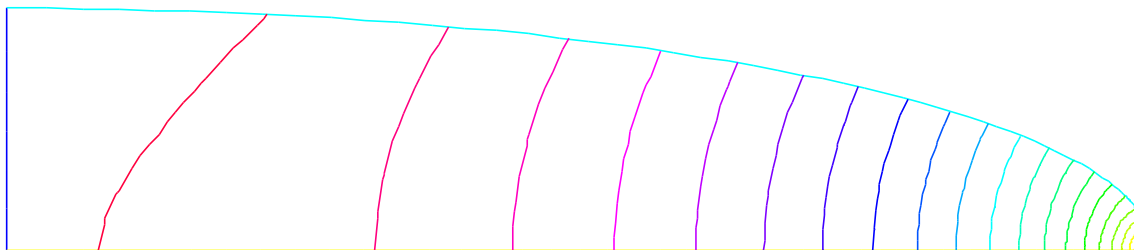
  while (1)
  {
    real mint = checkmovemesh(Th,[x,y-v*coef]);

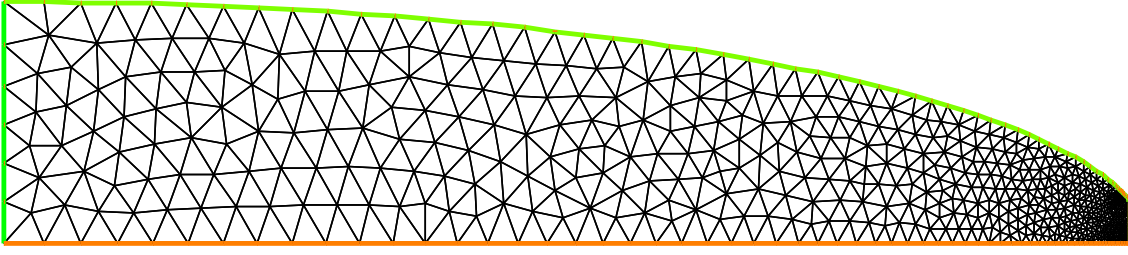
    if (mint>mintcc) break;

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

Figure 6.13: The adapted mesh of the domain Ω^{72}

6.13 nolinear-elas.edp

The nonlinear elasticity problem is find the displacement (u_1, u_2) minimizing J

$$\min J(u_1, u_2) = \int_{\Omega} f(F2) - \int_{\Gamma_p} P_a u_2$$

where $F2(u_1, u_2) = A(E[u_1, u_2], E[u_1, u_2])$ and $A(X, Y)$ is bilinear sym. positive form with respect two matrix X, Y . where f is a given C^2 function, and $E[u_1, u_2] = (E_{ij})_{i=1,2, j=1,2}$ is the Green-Saint Venant deformation tensor defined with:

$$E_{ij} = 0.5(\partial_i u_j + \partial_j u_i) + \sum_k \partial_i u_k \times \partial_j u_k$$

The differential of J is

$$DJ(u_1, u_2)(v_1, v_2) = \int 2A(E[u_1, u_2], DE[u_1, u_2](v_1, v_2))f'(F2(u_1, u_2)) - \int_{\Gamma_p} P_a u_2$$

denote $\mathbf{u} = u_1, u_2$, $\mathbf{v} = v_1, v_2$, $\mathbf{w} = (w_1, w_2)$ and the second order differential is

$$\begin{aligned} D^2 J(\mathbf{u})((\mathbf{v}), (\mathbf{w})) &= A(E[\mathbf{u}], DE[\mathbf{u}](\mathbf{v}))A(E[\mathbf{u}], DE[\mathbf{u}](\mathbf{w}))f''(F2(\mathbf{u})) \\ &+ A(DE[\mathbf{u}](\mathbf{v}), DE[\mathbf{u}](\mathbf{w}))f'(F2(\mathbf{u})) \\ &+ A(DE[\mathbf{u}], D^2 E[\mathbf{u}]((\mathbf{v}), (\mathbf{w})))f'(F2(\mathbf{u})) \end{aligned}$$

where DE and $D^2 E$ are the first and second differential of E .

The Newton Method is

choose $n = 0$, and u_0, v_0 the initial displacement

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

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

- $u_{n+1} = u_n - du, \quad v_{n+1} = v_n - dv$
- until (du, dv) small is enough


```

// remark :
// dEE(di,dj,dui,duj,u1,u2,du1,du2) is "the formal differential of EE"
// where du1 =  $\delta u_1$  , du2 =  $\delta u_2$ 
// ddEE(di,dj,dui,duj,u1,u2,du1,du2) is "the formal differential of dEE"
// where ddu1 =  $\delta^2 u_1$  , ddu2 =  $\delta^2 u_2$ 
// ---

// the macro corresponding to the 3 componente of E
macro E1(u,v) /*E11*/EE(dx,dx,u,u,u,v) //
macro E2(u,v) /*E12*/EE(dx,dy,u,v,u,v) //
macro E3(u,v) /*E22*/EE(dy,dy,v,v,u,v) //

macro dE1(u,v,uu,vv) /*dE11*/dEE(dx,dx,uu,uu,u,v,uu,vv) //
macro dE2(u,v,uu,vv) /*dE12*/dEE(dx,dy,uu,vv,u,v,uu,vv) //
macro dE3(u,v,uu,vv) /*dE22*/dEE(dy,dy,vv,vv,u,v,uu,vv) //
macro ddE1(u,v,uu,vv,uuu,vvv) /*ddE11*/ddEE(dx,dx,uu,vv,uuu,vvv) //
macro ddE2(u,v,uu,vv,uuu,vvv) /*ddE12*/ddEE(dx,dy,uu,vv,uuu,vvv) //
macro ddE3(u,v,uu,vv,uuu,vvv) /*ddE22*/ddEE(dy,dy,uu,vv,uuu,vvv) //

// a formal bilinear term
macro PP(A,B,u,v) (A(u,v)*B(u,v)) //

// a formal diff bilinear term
macro dPP(A,B,dA,dB,u,v,uu,vv) (dA(u,v,uu,vv)*B(u,v) + A(u,v)*dB(u,v,uu,vv)) //

// a formal diff2 bilinear term
macro ddPP(A,B,dA,dB,ddA,ddB,u,v,uu,vv,uuu,vvv) (
  dA(u,v,uu,vv)*dB(u,v,uuu,vvv) + dA(u,v,uuu,vvv)*dB(u,v,uu,vv)
  + ddA(u,v,uu,vv,uuu,vvv)*B(u,v) + A(u,v)*ddB(u,v,uu,vv,uuu,vvv)
) //
// so the matrix A is 6 coef //

// a11 a12 a13
// a12 a22 a23
// a13 a23 a33
macro F2(u,v) /* F2 */ (
  a11*PP(E1,E1,u,v)
  + a22*PP(E2,E2,u,v)
  + a33*PP(E3,E3,u,v)
  + a13*PP(E1,E3,u,v)
  + a13*PP(E3,E1,u,v)
  + a12*PP(E1,E2,u,v)
  + a12*PP(E2,E1,u,v)
  + a23*PP(E2,E3,u,v)
  + a23*PP(E3,E2,u,v)
) // end macro F2

macro dF2(u,v,uu,vv) /* dF2 */ (
  a11*dPP(E1,E1,dE1,dE1,u,v,uu,vv)
  + a12*dPP(E1,E2,dE1,dE2,u,v,uu,vv)
  + a13*dPP(E1,E3,dE1,dE3,u,v,uu,vv)
  + a21*dPP(E2,E1,dE2,dE1,u,v,uu,vv)
  + a22*dPP(E2,E2,dE2,dE2,u,v,uu,vv)
  + a23*dPP(E2,E3,dE2,dE3,u,v,uu,vv)
  + a31*dPP(E3,E1,dE3,dE1,u,v,uu,vv)
)

```

```

+ a32*dPP(E3,E2,dE3,dE2,u,v,uu,vv)
+ a33*dPP(E3,E3,dE3,dE3,u,v,uu,vv)
) // end macro dF2 (DF2)

macro ddF2(u,v,uu,vv,uuu,vvv) /* ddF2 */ (
  a11*ddPP(E1,E1,dE1,dE1,ddE1,ddE1,u,v,uu,vv,uuu,vvv)
+ a12*ddPP(E1,E2,dE1,dE2,ddE1,ddE2,u,v,uu,vv,uuu,vvv)
+ a13*ddPP(E1,E3,dE1,dE3,ddE1,ddE3,u,v,uu,vv,uuu,vvv)
+ a21*ddPP(E2,E1,dE2,dE1,ddE2,ddE1,u,v,uu,vv,uuu,vvv)
+ a22*ddPP(E2,E2,dE2,dE2,ddE2,ddE2,u,v,uu,vv,uuu,vvv)
+ a23*ddPP(E2,E3,dE2,dE3,ddE2,ddE3,u,v,uu,vv,uuu,vvv)
+ a31*ddPP(E3,E1,dE3,dE1,ddE3,ddE1,u,v,uu,vv,uuu,vvv)
+ a32*ddPP(E3,E2,dE3,dE2,ddE3,ddE2,u,v,uu,vv,uuu,vvv)
+ a33*ddPP(E3,E3,dE3,dE3,ddE3,ddE3,u,v,uu,vv,uuu,vvv)
) // end macro ddF2 (D2F2)

// differential of J:

// for hyper elasticity problem
// -----

macro f(u) (u) // end of macro
macro df(u) (1) // end of macro df = f'
macro ddf(u) (0) // end of macro ddf = f''

// -- du caouchouc --- CF cours de Herve Le Dret.
// -----

real mu = 0.012e5; // kg/cm2
real lambda = 0.4e5; // kg/cm2
//
//  $\sigma = 2\mu E + \lambda \text{tr}(E) Id$ 
//
// ( a b )
// ( b c )
//
// tr*Id : (a,b,c) -> (a+c,0,a+c)
// so the associated matrix is:
// ( 1 0 1 )
// ( 0 0 0 )
// ( 1 0 1 )
// ----- the coef
real a11= 2*mu + lambda ;
real a22= 2*mu ;
real a33= 2*mu + lambda ;
real a12= 0 ;
real a13= lambda ;
real a23= 0 ;

// symmetric part

real a21= a12 ;
real a31= a13 ;
real a32= a23 ;
real Pa=1e2; // a pressure of 100 Pa
// -----

int n=30,m=10;
mesh Th= square(n,m,[x,.3*y]); // label: 1 bottom, 2 right, 3 up, 4 left;

```

```

int bottom=1, right=2,upper=3,left=4;

plot(Th);

fespace Wh(Th,P1dc);
fespace Vh(Th,[P1,P1]);
fespace Sh(Th,P1);

Wh e2,fe2,dfe2,ddfe2; // optimisation
Wh ett,ezz,err,erz; // optimisation

Vh [uu,vv], [w,s],[un,vn];
[un,vn]=[0,0]; // intialisation
[uu,vv]=[0,0];

varf vmass([uu,vv],[w,s],solver=CG) = int2d(Th)( uu*w + vv*s );
matrix M=vmass(Vh,Vh);

problem NonLin([uu,vv],[w,s],solver=LU)=
  int2d(Th,qforder=1)( // (D2J(un)) part
    ddF2(un,vn,uu,vv,w,s)* dfe2
    + dF2(un,vn,uu,vv)*dF2(un,vn,w,s)*ddfe2
  )
  -int2d(Th,<1)( // (DJ(un)) part
    dF2(un,vn,w,s) * dfe2 )
  - int1d(Th,3)(Pa*s)
  + on(right,left,uu=0,vv=0);
; // Newton's method
// -----

Sh u1,v1;
for (int i=0;i<10;i++)
{
  cout << "Loop " << i << endl;
  e2 = F2(un,vn);
  dfe2 = df(e2);
  ddfe2 = ddf(e2);
  cout << " e2 max " <<e2[.max << " , min" << e2[.min << endl;
  cout << " de2 max " << dfe2[.max << " , min" << dfe2[.min << endl;
  cout << " dde2 max " << ddfe2[.max << " , min" << ddfe2[.min << endl;
  NonLin; // compute [uu,vv] = (D2J(un))-1(DJ(un))

  w[] = M*uu[];
  real res = sqrt(w[]' * uu[]); // norme L2 of [uu,vv]
  u1 = uu;
  v1 = vv;
  cout << " L2 residual = " << res << endl;
  cout << " u1 min =" <<u1[.min << " , u1 max= " << u1[.max << endl;
  cout << " v1 min =" <<v1[.min << " , v2 max= " << v1[.max << endl;
  plot([uu,vv],wait=1,cmm=" uu, vv " );
  un[] -= uu[];
  plot([un,vn],wait=1,cmm=" displacement " );
  if (res<1e-5) break;
}

plot([un,vn],wait=1);

```

```
mesh th1 = movemesh(Th, [x+un, y+vn]);  
plot(th1,wait=1); // see figure 6.14
```

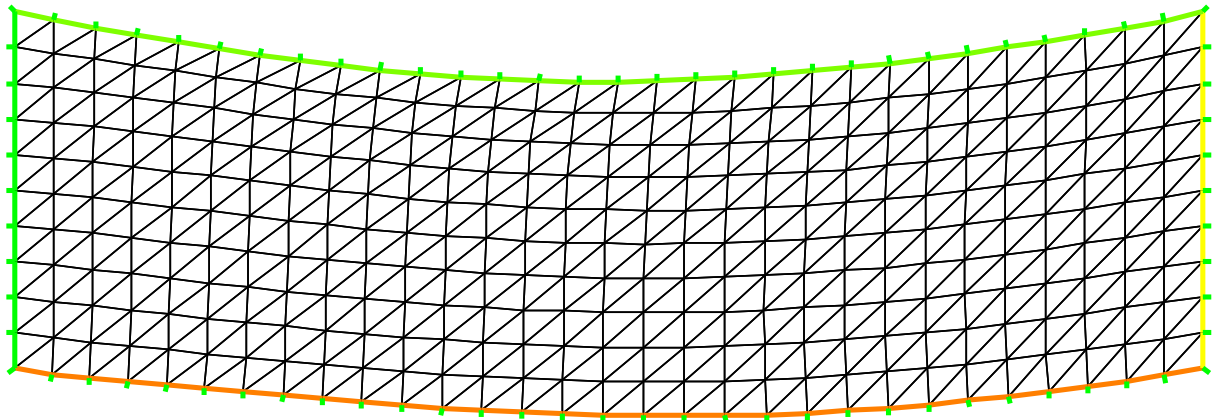


Figure 6.14: The deformed domain

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

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

broadcast a function to broadcast from a processor to all other a data

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

7.1 Schwarz in parallel

If example is just the rewriting of example schwarz-overlap in section 6.8.1.

How to use

```
[examples++-mpi] hecht%lamboot
```

LAM 6.5.9/MPI 2 C++/ROMIO - Indiana University

```
[examples++-mpi] hecht% mpirun -np 2 FreeFem++-mpi schwarz-c.edp
```

```
//    a new coding verion c, methode de schwarz in parallele
//    with 2 proc.
//    -----
//    F.Hecht december 2003
//    -----
//    to test the broadcast instruction
//    and array of mesh
//    add add the stop test
//    -----
```

```

if ( mpisize != 2 ) {
  cout << " sorry number of processeur !=2 " << endl;
  exit(1);}
verbosity=3;
real pi=4*atan(1);
int inside = 2;
int outside = 1;
border a(t=1,2){x=t;y=0;label=outside;};
border b(t=0,1){x=2;y=t;label=outside;};
border c(t=2,0){x=t ;y=1;label=outside;};
border d(t=1,0){x = 1-t; y = t;label=inside;};
border e(t=0, pi/2){ x= cos(t); y = sin(t);label=inside;};
border e1(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;
mesh[int] Th(mpisize);
if (mpirank == 0)
  Th[0] = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
else
  Th[1] = buildmesh ( e(5*n) + e1(25*n) );

broadcast(processor(0),Th[0]);
broadcast(processor(1),Th[1]);

fespace Vh(Th[mpirank],P1);
fespace Vhother(Th[1-mpirank],P1);

Vh u=0,v;
Vhother U=0;
int i=0;

problem pb(u,v,init=i,solver=Cholesky) =
  int2d(Th[mpirank])( dx(u)*dx(v)+dy(u)*dy(v) )
- int2d(Th[mpirank])( v)
+ on(inside,u = U) + on(outside,u= U) ;

for ( i=0 ;i< 20; i++)
{
  cout << mpirank << " loop " << i << endl;
  pb;
  // send u to the other proc, receive in U
  processor(1-mpirank) << u[]; processor(1-mpirank) >> U[];
  real err0,err1;
  err0 = int1d(Th[mpirank],inside)(square(U-u)) ;
  // send err0 to the other proc, receive in err1
  processor(1-mpirank)<<err0; processor(1-mpirank)>>err1;
  real err= sqrt(err0+err1);
  cout <<" err = " << err << " err0 = " << err0 << ", err1 = " << err1 << endl;
  if(err<1e-3) break;
};
if (mpirank==0)
  plot(u,U,ps="uU.eps");

```

Chapter 8

The Graphic User Interface (FFedit)

8.1 Unix version

This version is for Linux and MacOSX.

8.1.1 Installation of Tcl and Tk

You have to install tcl8.4.6 and tk8.4.6 for the GUI to work.

First go to <http://www.tcl.tk/software/tcltk/downloadnow84.tml>

and download :

tcl8.4.6-src.tar.gz

and tk8.4.6-src.tar.gz

Then do :

```
tar zxvf tcl8.4.6-src.tar.gz
```

```
tar zxvf tk8.4.6-src.tar.gz
```

It creates two directories : tcl8.4.6 and tk8.4.6

Now do :

```
cd tcl8.4.6
```

```
cd unix (if your OS is Linux or MacOSX)
```

```
./configure
```

```
make
```

```
make install
```

then

```
cd tk8.4.6
```

```
cd unix
```

```
./configure
```

```
make
```

```
make install
```

At the end of installation, you have to find where is your binary “wish” or “wish84” or “wish8.4” by typing :

```
- > which wish (or wish84 or wish8.4)
```

If wish84 does exist it is all. If not, you have to go in the directory where is “wish” or

“wish8.4” (for example /usr/bin/)
and then create a link :
- > ln -s wish wish84
or
- > ln -s wish8.4 wish84

8.1.2 Description

The Graphic User Interface is in the directory called FFedit. You can run it by typing `./FFedit.tcl`

The Graphic User Interface shows a text window with buttons on the left and right side, a horizontal menu bar above and an entry below where you can see the path of the script when it is opened or where you can type the path of a script to run it.

This is the description of the different functions of the GUI :

*New : you can access it by the button “New” on the right side or by selecting it in the menu File on the horizontal bar. It deletes the current script and enables you to type a new script.

*Open : you can access it by the button “Open” on the right side or by selecting it in the menu File or by typing simultaneously `ctrl+o` on the keyboard. It enables you to select a script which has been saved. This script is then opened in the text window. Then, you can modify it, save the changes, save under another name or run it.

*Save : you can access it by the button “Save” on the right side or by selecting it in the menu File or by typing simultaneously `ctrl+s` on the keyboard. It enables you to save the changes of a script.

*Save as : you can access it by the button “Save As” on the right side or by selecting it in the menu File.

It enables you to locate where you want to save your script and to choose your the name of your script.

*Run : you can access it by the button “Run” on the right side or by selecting it in the menu File or by typing simultaneously `ctrl+r` on the keyboard.

It enables you to run the current script.

Warning: when you run by typing `ctrl+r`, the cursor must be in the text window.

*Print : you can access it by the button “Print” on the right side or by selecting it in the menu File or by typing simultaneously `ctrl+p` on the keyboard.

It enables you to print your script. You have to choose the name of the printer.

*Help (under construction) : you can access it by the button “Help” on the left side. When an example is opened, it shows you a documentation about this example.

Warning : The bottom of each page is not accessible, you have to print to see the entire

document.

*Example : you can access it by the button “Ex” on the left side. It runs a little example.

*Read Mesh : you can access it by the button “R.M” on the left side. It enables you to read a mesh which has been saved. It adds the command in your script so that you can use it in your script.

*Polygonal Border : you can access it by the button “P.B” on the left side.

It enables you to build a polygonal border.

When you click on this button, a new window is opened : you have to click on the button “Border” then it asks you how many borders you want. When you enter a number and click on “OK” the exact number of couple of entries enable you to enter the coordinates of the vertices. And then you have to enter the number of points on each border. The border number 1 is the segment between the vertex number 1 and the vertex number 2 ...etc...

You must turn in the opposite sens of needles of a watch.

When you have finished, you have to click on the button “Build”. It builds the domain with polygonal border and shows the result.

Then you can save the result by clicking on the button “S.Mesh”. Choose the extension .msh for the name.

*Navier Stokes : you can access it by the button “N.S” on the left side.

A new window is opened. You have to build your polygonal domain by clicking on the button “Border” it works like the Polygonal Border function.

You can save by clicking on the button “S.Mesh”. Choose the extension .msh for the name.

Then you have to choose the Limits Condition by clicking on the buton “L.C”.

First choose between “Free” or “Imposed”, then click on the button “Validate” and enter the expression of Imposed condition. You have to enter the tangential and the normal component of the velocity. Then click on “Validate” again.

The expression of the limit condition can be a number but a mathematical expression as well.

*emc2 : you can access it by the button “emc2”. It runs emc2 the mesh building software (<http://www-rocq1.inria.fr/gamma/cdrom/www/emc2/fra.htm>)

*Set up : you can access it by the button “set up”. It is the first thing you have to do when you use FFedit for the first time.

When you click on this button, a new window is opened, with two entries where you have to enter the path of the binary of FreeFem++ (where you compiled or installed) and the path of the scripts (programs written with FreeFem++) for instance the examples provided in FreeFem++.

*Undo : you can access it by selecting in the menu Edit or by typing simultaneously ctrl+z. It is an unlimited undo function.

*Redo : you can access it by selecting in the menu Edit or by typing simultaneously ctrl+e.

It is an unlimited redo function.

*Cut : you can access it by selecting in the menu Edit or by typing simultaneously ctrl+x on the keyboard.

*Copy : you can access it by selecting in the menu Edit or by typing simultaneously ctrl+c on the keyboard.

*Paste : you can access it by selectiog in the menu Edit or by typing simultaneously ctrl+y on the keyboard.

*Delete : you can access it by selecting in the Edit menu. It is a Delete function.

*Select all : you can access it by selecting in the Edit menu or by typing simultaneously ctrl+l. This function select all the text you have written, so you can delete, cut copy paste etc...

*Background : you can access it by selecting in the menu Color. You can then choose the color of the background.

*Foreground : you can access it by selecting in the menu Color. You can then choose the color of the foreground.

*Syntax Color : you can access it by selecting in the menu Color. It enables the syntax coloring of your FreeFem++ script.

*Family : you can access it by selecting in the menu Font. You can then choose the style of your characters.

*Size : you can access it by selecting in the menu Font. You can then choose the size of your characters.

*Find : you can access it by selecting in the menu Search. It enables you to find a word in the whole text. (This function doesn't work yet.)

*Find next : you can access it by slecting in the menu Search. It enables you to find a word from the position of the cursor.(This function doesn't work yet.)

*Replace : you can access it by selecting in the menu Search. It enables you to replace a word by another in the whole text.

8.1.3 Cygwin version

This version is for Windows.

8.1.4 Installation of Cygwin

First you have to go to the site:

<http://www.cygwin.com>

Click on “Install or Update now”

A window appears.

Click on “Open” then “Next” and then choose “Install from Internet”

Click on “Next” twice and choose “Direct Connection”.

Then choose one of the download sites.

For instance : <ftp://ftp-stud.fht-esslingen.de>

Then choose in each category the options to install.

(click on the symbol + for each, you can then see the options appear)

Here are the required options for FreeFem++ and TCL TK to work :

+ all options of “Devel” (it includes gcc ...etc...)

+ all options of “Graphics” (specially “Gnuplot” to be able to see the results of FreeFem++ on a graphic)

+ all options of X11 and XFree86

You can install “Xemacs” and others editors in the category “Editors”.

Then click on “Next” and wait that Cygwin is installed.

Then an icône appears on yours Desktop.

Click on it and the Cygwin window is opened. It is like an Unix terminal.

To work on a terminal X type :

```
- > startX
```

You will have to work on a terminal X to have Gnuplot and visualize the results of FreeFem++ scripts)

Now you have to compile and install FreeFem++.

8.1.5 Compilation and Installation of FreeFem++ under Cygwin

go to the site :

<http://www.freefem.org>

click on FreeFem++

and download FreeFem++ (the version when I wrote this is 1.38)

Choose to download in your cygwin/home/you directory.

Then on your terminal Cygwin type :

```
- > tar zxvf freefem++.tgz
```

to uncompress this file.

Go into FreeFem++v1.38 directory by typing:

```
- > cd FreeFem++v1.38
```

Now you have to compile FreeFem++.

type:

```
- > make all HOSTTYPE=i-386
```

to compile FreeFem++

If there is an error : "... -ldl : no such file or directory"

Then you have to modify the Makefile-i386 which is in the directory src:

```
- > cd src
```

Edit it (with xemacs for example):

```
- > xemacs Makefile-i386
```

at line 1 : replace "LIBLOCAL = -ldl" by "#LIBLOCAL = -ldl"

It will comment this line because -ldl is not on your machine.

Then return to the main directory

```
- > cd
```

and type:

```
- > make all HOSTTYPE=i-386
```

At the end of compilation, a directory called "c-i386" is created.

In this directory you can find the binary FreeFem++.

You can now run an example:

First open an X terminal:

```
- > startX
```

In this terminal go in to FreeFem++v1.38:

```
- > cd FreeFem++v1.38
```

and type:

```
- > c-i386/FreeFem++ examples++-tutorial/adapt.edp
```

You can then see the gnuplot window with the graphical results.

8.1.6 Compilation and Installation of tcl8.4.0 and tk8.4.0 under Cygwin

Now if you want to use the Graphical User Interface of FreeFem++ (called FFedit)

you have to install the language TCL TK in which FFedit has been written.

The version which works under cygwin is tcl8.4.0 and tk8.4.0

The latest version when I wrote this is tcl8.4.6 and tk8.4.6

DON'T USE IT

It works under Linux and MacOSX but not under Cygwin.

You have to download tcl8.4.0 and tk8.4.0 :

For instance, go to Google.fr and type download tcl tk 8.4.0

And choose the "Sourceforge.net: Project Filelist".

Choose :

tcl8.4.0-src.tar.gz and tk8.4.0-src.tar.gz

When the download is finished you have to uncompress these directories:

```
- > tar zxvf tcl8.4.0-src.tar.gz
```

```
- > tar zxvf tk8.4.0-src.tar.gz
```

tcl8.4.0 and tk8.4.0 will work under Cygwin only if you apply a patch on both:

These patches are on the site:

<http://www.xraylith.wisc.edu/khan/software/tcl>

Choose “Tcl/Tk8.4.0 for Cygwin

Click on “very preliminary Cygwin ports of Tcl/Tk8.4.0

You are then on the site ftp

Follow the instructions of the README or follow these instructions:

1) Run :

```
- > xemacs tcl-8.4.0-cygwin.diff
```

By doing this, you create a new file called “tcl-8.4.0-cygwin.diff”

on the site ftp click on “tcl-8.4.0-cygwin.diff”

Do a Copy/Paste of the content into your xemacs window and save it.

2) Do the same with “tk-8.4.0-cygwin.diff”

Now you have to apply the patch in tcl8.4.0 and tk8.4.0

The two previous patch files (.diff) must be respectively in tcl8.4.0 and tk8.4.0 directories.

```
- > cp tcl-8.4.0-cygwin.diff tcl8.4.0
```

```
- > cp tk-8.4.0-cygwin.diff tk8.4.0
```

(If the two files are one level under tcl8.4.0 and tk8.4.0)

Now apply the patches:

type:

```
- > cd tcl8.4.0
```

```
- > patch -p0 -s < tcl-8.4.0-cygwin.diff
```

```
- > cd
```

```
- > cd tk8.4.0
```

```
- > patch -p0 -s < tk-8.4.0-cygwin.diff
```

Now you can compile and install TCL TK under Cygwin:

* compilation and installation of tcl8.4.0

Go in to the directory tcl8.4.0/win

- > cd tk8.4.0

- > cd win

Then type :

- > ./configure

The two steps remaining are make and make install
type

- > make

The compilation starts, when finished install by typing:

- > make install

When finished try to see if it works by typing:

- > tclsh84

if ok quit by typing ctrl-c

*Compilation and installation of tk8.4.0

Go in to the directory tk8.4.0/win

- > cd tk8.4.0

- > cd win

Then type :

- > ./configure

The two steps remaining are make and make install
type

- > make

The compilation starts, if you have errors like :

```
windres -o tk.res.o -include "C:/cygwin/home/ly/tk8.4.0/generic"
```

```
-include "C Option-I is deprecated for setting the input format,  
please use -J instead"
```

```
windres : can't open icon file 'tk.ico' : no such file or directory
```

This file 'tk.ico' is in fact in the directory win/rc

You have to copy it in the directory 'generic':

Be in tk8.4.0

type :

```
- > cp win/rc/tk.ico generic/
```

If you compile again you will see that there is the same
errors with the files:

```
"buttons.bmp" "cursor00.cur" "cursor02.cur" ...etc...
```

```
"wish.exe.manifest" and "wish.ico"
```

Do the same for these files.

For the cursor*.cur files do once the command:

```
- > cp win/rc/cursor*.cur generic/
```

when finished install by typing:

- > make install

When finished try to see if it works by typing:

- > wish84

if ok quit by typing ctrl-c

8.1.7 Use

Now everything is ok to use FFedit and FreeFem++ under Windows by Cygwin.
WARNING: if you work on the Cygwin terminal you will not be able to see the graphical results of FreeFem++.

You have to run an X terminal and run FFedit under this X terminal:

To run an X terminal under cygwin type on your Cygwin terminal:

- > startX

An X terminal runs:

Under this terminal:

type

- > cd FFedit

- > ./FFedit.tcl

Chapter 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
(((R) x_i^j , j=1,dim), (I) $Ref\phi_i^v$, i=1,NbOfVertices)
- Edges (I) NbOfEdges
(@@Vertex_i¹, @@Vertex_i², (I) $Ref\phi_i^e$, i=1, NbOfEdges)
- Triangles (I) NbOfTriangles
((@@Vertex_i^j, j=1,3), (I) $Ref\phi_i^t$, i=1, NbOfTriangles)
- Quadrilaterals (I) NbOfQuadrilaterals
((@@Vertex_i^j, j=1,4), (I) $Ref\phi_i^t$, i=1, NbOfQuadrilaterals)
- Geometry
(C*) FileNameOfGeometricSupport
 - VertexOnGeometricVertex
(I) NbOfVertexOnGeometricVertex
(@@Vertex_i, @@Vertex_i^{geo}, i=1, NbOfVertexOnGeometricVertex)
 - EdgeOnGeometricEdge
(I) NbOfEdgeOnGeometricEdge
(@@Edge_i, @@Edge_i^{geo}, i=1, NbOfEdgeOnGeometricEdge)
- CrackedEdges (I) NbOfCrackedEdges
(@@Edge_i¹, @@Edge_i², i=1, NbOfCrackedEdges)

When the current mesh refers to a previous mesh, we have in addition

- MeshSupportOfVertices
(C*) FileNameOfMeshSupport
 - VertexOnSupportVertex
(I) NbOfVertexOnSupportVertex
(@@Vertex_i, @@Vertex_i^{supp}, i=1,NbOfVertexOnSupportVertex)
 - VertexOnSupportEdge
(I) NbOfVertexOnSupportEdge
(@@Vertex_i, @@Edge_i^{supp}, (R) u_i^{supp}, i=1,NbOfVertexOnSupportEdge)
 - VertexOnSupportTriangle
(I) NbOfVertexOnSupportTriangle
(@@Vertex_i, @@Tria_i^{supp}, (R) u_i^{supp}, (R) v_i^{supp},
i=1, NbOfVertexOnSupportTriangle)
 - VertexOnSupportQuadrilaterals
(I) NbOfVertexOnSupportQuadrilaterals
(@@Vertex_i, @@Quad_i^{supp}, (R) u_i^{supp}, (R) v_i^{supp},
i=1, NbOfVertexOnSupportQuadrilaterals)

9.2 bb File type for Store Solutions

The file is formatted such that:

```
2 nbsol nbv 2
((Uij, ∀i ∈ {1, ..., nbsol}), ∀j ∈ {1, ..., nbv})
```

where

- nbsol is a integer equal to the number of solutions.
- nbv is a integer equal to the number of vertex .
- 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 typesol1 ... typesoln nbv 2
(((Uijk, ∀i ∈ {1, ..., typesolk}), ∀k ∈ {1, ...n}) ∀j ∈ {1, ..., nbv})
```

where

- n is a integer equal to the number of solutions
- typesol^k, type of the solution number *k*, is
 - typesol^k = 1 the solution k is scalare (1 value per vertex)
 - typesol^k = 2 the solution k is vectorial (2 values per unknown)

- `typesolk = 3` the solution k is a 2×2 symmetric matrix (3 values per vertex)
- `typesolk = 4` the solution k is a 2×2 matrix (4 values per vertex)
- `nbv` is a integer equal to the number of vertices
- 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`

`hi $\forall i \in \{1, \dots, \text{nbv}\}$`

where

- `nbv` is a integer equal to the number of vertices.
- `hi` 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`

`a11i, a21i, a22i $\forall i \in \{1, \dots, \text{nbv}\}$`

where

- `nbv` is a integer equal to the number of vertices,
- `a11i, a12i, a22i` 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:

`nbv` is the number of vertices.

`nbv` is the number of vertices.

`nu(1:3, 1:nbv)` 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.

`refv(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)),refl(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

Add new finite element

10.1 Some notation

The values of finite element function are in \mathbb{R}^N .

We denote ω_i^K the i basis function for $i \in \{0, \dots, NbDoF - 1\}$, and the component j are denote ω_{ij}^K .

We denote Π_h the interpolator of the finite element. We have the following equality $\omega_i^K = \Pi_h \omega_i^K$.

Let $\mathbf{f} = (f_j)_{j=0, \dots, N-1}$ a function to interpolate, formally we can defined the operator Π_h the following formular:

$$\Pi_h \mathbf{f} = \sum_{k=0}^{kPi-1} \alpha_k \mathbf{f}_{j_k}(P_{p_k}) \omega_{i_k}^K \quad (10.1)$$

where P_p is a set of $npPi$ points,

Remark, in a formula (10.1), the list p_k, j_k, i_k depend just of the type of finite element not of the element, but the coefficient α_k can be depending of the element.

Example 1: classical scalar Lagrange finite element ($N = 1$), first we have $kPi = npPi = NbOfNode$ and

- P_p is the point of the nodal points
- the $\alpha_k = 1$, because we take the value of the function at the point P_k
- $p_k = k$, $j_k = k$ because we have one node per function.
- $j_k = 0$ because $N = 1$

Example 2: The Raviart-Thomas finite element:

$$RT0_h = \{ \mathbf{v} \in H(div) / \forall K \in \mathcal{T}_h \quad \mathbf{v}|_K(x, y) = \left| \begin{matrix} \alpha_K \\ \beta_K \end{matrix} + \gamma_K \begin{matrix} x \\ y \end{matrix} \right\} \quad (10.2)$$

The degree of freedom are the flux throw an edge e of the mesh, where the flux of the function $\mathbf{f} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is $\int_e \mathbf{f} \cdot n_e$, n_e is the unit normal of edge e (this implies a orientation of all the edges of the mesh, for exemple we can use the global numbering of the edge vertices and we just go to small to large number).

To compute this flux, we use an quadrature formular with one point, the middle point of the edge. let a triangle T with three vertices $(\mathbf{a}, \mathbf{b}, \mathbf{c})$. Let denote, the vertices number are i_a, i_b, i_c , the three edge vector $\mathbf{e}^0, \mathbf{e}^1, \mathbf{e}^2$ are defined by $\text{sgn}(i_b - i_c)(\mathbf{b} - \mathbf{c})$, $\text{sgn}(i_c - i_a)(\mathbf{c} - \mathbf{a})$, $\text{sgn}(i_a - i_b)(\mathbf{a} - \mathbf{b})$,

For the finite element the three basis functions are:

$$\omega_0^K = \frac{\text{sgn}(i_b - i_c)}{2|K|}(x - a), \quad \omega_1^K = \frac{\text{sgn}(i_c - i_a)}{2|K|}(x - b), \quad \omega_2^K = \frac{\text{sgn}(i_a - i_b)}{2|K|}(x - c),$$

where $|K|$ is the area of the triangle K .

So we have $N = 2, \text{kPi} = 6(\text{sizeof the formuar}); \text{npPi} = 3$; (number of points) and:

- $P_p = \left\{ \frac{\mathbf{b}+\mathbf{c}}{2}, \frac{\mathbf{a}+\mathbf{c}}{2}, \frac{\mathbf{b}+\mathbf{a}}{2} \right\}$
- $\alpha_0 = -e_2^0, \alpha_1 = e_1^0, \alpha_2 = -e_2^1, \alpha_3 = e_1^1, \alpha_4 = -e_2^2, \alpha_5 = e_1^2$ (effectively, the vector $(-e_2^m, e_1^m)$ is orthogonal to the edge $\mathbf{e}^m = (e_1^m, e_2^m)$ with a length equal to the side of the edge or equal to $\int_{e^m} 1$).
- $i_k = \{0, 0, 1, 1, 2, 2\}$,
- $p_k = \{0, 0, 1, 1, 2, 2\}$, $j_k = \{0, 1, 0, 1, 0, 1, 0, 1\}$.

10.2 Which class of add

Add file `FE_ADD.cpp` in directory `src/femlib` for exemple first to initialize :

```
#include "error.hpp"
#include "rgraph.hpp"
using namespace std;
#include "RNM.hpp"
#include "fem.hpp"
#include "FESpace.hpp"
```

```
namespace Fem2D {
```

Second, you are just a class which derive for `public TypeOfFE` like:

```
class TypeOfFE_RTortho : public TypeOfFE { public:
    static int Data[]; // some numbers
    TypeOfFE_RTortho():
        TypeOfFE( 0+3+0, // nb degree of freedom on element
                2, // dimension N of vectorial FE (1 if scalar FE)
                Data, // the array data
                1, // nb of subdivision for plotting
                1, // nb of sub finite element (generaly 1)
                6, // number kPi of coef to build the interpolator (10.1)
                3, // number npPi of integration point to build interpolator
                0 // an array to store the coef  $\alpha_k$  to build interpolator
                // here this array is no constant so so we have
                // to rebuidt for each element.
        )
    {
        const R2 Pt[] = { R2(0.5,0.5), R2(0.0,0.5), R2(0.5,0.0) };
    }
};
```

```

// the set of Point in  $\hat{K}$ 
for (int p=0, kk=0; p<3; p++) {
    P_Pi_h[p]=Pt[p];
    for (int j=0; j<2; j++)
        pij_alpha[kk++]= IPJ(p,p,j); } // definition of  $i_k, p_k, j_k$  in (10.1)

void FB(const bool * watdd, const Mesh & Th, const Triangle & K,
        const R2 & PHat, RNMK_ & val) const;

void Pi_h_alpha(const baseFEElement & K, KN_<double> & v) const ;

} ;

```

where the array data is form with the concatenation of five array of size NbDoF and one array of size N.

This array is:

```

int TypeOfFE_RTortho::Data[]={
    // for each df 0,1,3 :
    3,4,5, // the support of the node of the df
    0,0,0, // the number of the df on the node
    0,1,2, // the node of the df
    0,0,0, // the df come from which FE (generally 0)
    0,1,2, // which are de df on sub FE
    0,0 }; // for each compontant  $j=0, N-1$  it give the sub FE
associated

```

where the support is a number 0, 1, 2 for vertex support, 3, 4, 5 for edge support, and finally 6 for element support.

The function to defined the function ω_i^K , this function return the value of all the basics function or this derivatives in array `val`, computed at point `PHat` on the reference triangle corresponding to point `R2 P=K(PHat)`; on the current triangle `K`.

The index i, j, k of the array $val(i, j, k)$ corresponding to:

i is basic function number on finite element $i \in [0, NoF[$

j is the value of component $j \in [0, N[$

k is the type of computed value $f(P), dx(f)(P), dy(f)(P), \dots i \in [0, last_operatortype[$.

Remark for optimisation, this value is computed only if $whatd[k]$ is true, and the numbering is defined with

```

enum operatortype { op_id=0,
    op_dx=1, op_dy=2,
    op_dxx=3, op_dyy=4,
    op_dyx=5, op_dxy=5,
    op_dz=6,
    op_dzz=7,
    op_dzx=8, op_dxz=8,
    op_dzy=9, op_dyz=9
};
const int last_operatortype=10;

```

The shape function :

```

void TypeOfFE_RTortho::FB(const bool *whatd,const Mesh & Th,const Triangle & K,
                          const R2 & PHat,RNMK_ & val) const
{
  R2 P(K(PHat));
  R2 A(K[0]), B(K[1]),C(K[2]);
  R l0=1-P.x-P.y,l1=P.x,l2=P.y;
  assert(val.N() >=3);
  assert(val.M()==2 );
  val=0;
  R a=1./(2*K.area);
  R a0=  K.EdgeOrientation(0) * a ;
  R a1=  K.EdgeOrientation(1) * a ;
  R a2=  K.EdgeOrientation(2) * a ;

  if (whatd[op_id]) // -----
                  // value of the function
  {
    assert(val.K()>op_id);
    RN_ f0(val('.',0,0)); // value first component
    RN_ f1(val('.',1,0)); // value second component
    f1[0] = (P.x-A.x)*a0;
    f0[0] = -(P.y-A.y)*a0;

    f1[1] = (P.x-B.x)*a1;
    f0[1] = -(P.y-B.y)*a1;

    f1[2] = (P.x-C.x)*a2;
    f0[2] = -(P.y-C.y)*a2;
  }

  if (whatd[op_dx]) // -----
                  // value of the dx of function
  {
    assert(val.K()>op_dx);
    val(0,1,op_dx) = a0;
    val(1,1,op_dx) = a1;
    val(2,1,op_dx) = a2;
  }
  if (whatd[op_dy])
  {
    assert(val.K()>op_dy);
    val(0,0,op_dy) = -a0;
    val(1,0,op_dy) = -a1;
    val(2,0,op_dy) = -a2;
  }

  for (int i= op_dy; i< last_operatortype ; i++)
    if (whatd[op_dx])
      assert(op_dy);
}

```

The function to defined the coefficient α_k :

```

void TypeOfFE_RT::Pi_h_alpha(const baseFEElement & K,KN_<double> & v) const

```

```

{
  const Triangle & T(K.T);

  for (int i=0,k=0;i<3;i++)
  {
    R2 E(T.Edge(i));
    R signe = T.EdgeOrientation(i) ;
    v[k++] = signe * E.y;
    v[k++] = -signe * E.x;
  }
}

```

Now , we just need to add a new key work in FreeFem++, so at the end of the file, we add:

```

//      let the 2 globals variables
static TypeOfFE_RTortho The_TypeOfFE_RTortho; //
//      ----- the name in freefem -----
static ListOfTFE typefemRTortho("RT0Ortho", & The_TypeOfFE_RTortho); //

//      link with FreeFem++ do not work with static library .a
//      FH so add a extern name to call in init_static_FE
//      (see end of FESpace.cpp)
void init_FE_ADD() { };
//      --- end ---
} //      FEM2d namespace

```

To enforce in loading of this new finite element, we have to add the two new lignes close to the end of files `src/femlib/FESpace.cpp` like:

```

//      correct Probleme of static library link with new make file
void init_static_FE()
{
//      list of other FE file.o
  extern void init_FE_P2h() ;
  init_FE_P2h() ;
  extern void init_FE_ADD() ; //      new ligne 1
  init_FE_ADD(); //      new ligne 2
}

```

10.3 How to add

First, create a file `FE_ADD.cpp` contening all this code, like in file `src/femlib/Element_P2h.cpp`, after modifier the `Makefile.am` by adding the name of your file to the variable `EXTRA_DIST` like:

```

# Makefile using Automake + Autoconf
# -----
# $Id: addfe.tex,v 1.2 2004/09/02 14:04:15 hecht Exp $

# This is not compiled as a separate library because its
# interconnections with other libraries have not been solved.

```

```
EXTRA_DIST=BangFreeFem.cpp BangFreeFem.hpp CGNL.hpp CheckPtr.cpp \
ConjuguedGradientNL.cpp DOperator.hpp Drawing.cpp Element_P2h.cpp \
Element_P3.cpp Element_RT.cpp fem3.hpp fem.cpp fem.hpp FESpace.cpp \
FESpace.hpp FESpace-v0.cpp FQuadTree.cpp FQuadTree.hpp gibbs.cpp \
glutdraw.cpp gmres.hpp MatriceCreuse.hpp MatriceCreuse_tpl.hpp \
MeshPoint.hpp mortar.cpp mshptg.cpp QuadratureFormular.cpp \
QuadratureFormular.hpp RefCounter.hpp RNM.hpp RNM_opc.hpp RNM_op.hpp \
RNM_tpl.hpp FE_ADD.cpp
```

and recompile

Chapter 11

Grammar

11.1 Keywords

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

11.2 The bison grammar

```
start:  input ENDOFFILE;
```

```

input:  instructions ;

instructions:  instruction
            | instructions instruction ;

list_of_id_args:
            | id
            | id '=' no_comma_expr
            | FESPACE id
            | type_of_dcl id
            | type_of_dcl '&' id
            | '[' list_of_id_args ']'
            | list_of_id_args ',' id
            | list_of_id_args ',' '[' list_of_id_args ']'
            | list_of_id_args ',' id '=' no_comma_expr
            | list_of_id_args ',' FESPACE id
            | list_of_id_args ',' type_of_dcl id
            | list_of_id_args ',' type_of_dcl '&' id ;

list_of_id1:  id
            | list_of_id1 ',' id ;

id:  ID | FESPACE ;

list_of_dcls:  ID
            | ID '=' no_comma_expr
            | ID '(' parameters_list ')'
            | list_of_dcls ',' list_of_dcls ;

parameters_list:
            no_set_expr
            | FESPACE ID
            | ID '=' no_set_expr
            | parameters_list ',' no_set_expr
            | parameters_list ',' id '=' no_set_expr ;

type_of_dcl:  TYPE
            | TYPE '[' TYPE ']' ;

ID_space:
            ID
            | ID '[' no_set_expr ']'
            | ID '=' no_set_expr
            | '[' list_of_id1 ']'
            | '[' list_of_id1 ']' '[' no_set_expr ']'
            | '[' list_of_id1 ']' '=' no_set_expr ;

ID_array_space:
            ID '(' no_set_expr ')'
            | '[' list_of_id1 ']' '(' no_set_expr ')' ;

fespace:  FESPACE ;

spaceIDa  :      ID_array_space
            |      spaceIDa ',' ID_array_space ;

```

```

spaceIDb :      ID_space
         |      spaceIDb ',' ID_space ;

spaceIDs :   fespace                spaceIDb
         |   fespace '[' TYPE ']' spaceIDa    ;

fespace_def: ID '(' parameters_list ')' ;

fespace_def_list: fespace_def
                 | fespace_def_list ',' fespace_def ;

declaration:  type_of_dcl list_of_dcls ';'
             | 'fespace' fespace_def_list    ';'
             | spaceIDs ';'
             | FUNCTION ID '=' Expr ';'
             | FUNCTION type_of_dcl ID '(' list_of_id_args ')' '{' instructions'}'
             | FUNCTION ID '(' list_of_id_args ')' '=' no_comma_expr ';' ;

begin: '{' ;
end:   '}' ;

for_loop:  'for' ;
while_loop: 'while' ;

instruction: ';'
           | 'include' STRING
           | 'load'  STRING
           | Expr ';'
           | declaration
           | for_loop '(' Expr ';' Expr ';' Expr ')' instruction
           | while_loop '(' Expr ')' instruction
           | 'if' '(' Expr ')' instruction
           | 'if' '(' Expr ')' instruction ELSE instruction
           | begin instructions end
           | 'border' ID border_expr
           | 'border' ID '[' array ']' ';'
           | 'break' ';'
           | 'continue' ';'
           | 'return' Expr ';' ;

bornes: '(' ID '=' Expr ',' Expr ')' ;

border_expr: bornes instruction ;

Expr:  no_comma_expr
      | Expr ',' Expr ;

unop:  '-'
      | '+'
      | '!'
      | '++'
      | '--' ;

```

```

no_comma_expr:
    no_set_expr
    | no_set_expr '=' no_comma_expr
    | no_set_expr '+=' no_comma_expr
    | no_set_expr '-=' no_comma_expr
    | no_set_expr '*=' no_comma_expr
    | no_set_expr '/=' no_comma_expr ;

```

```

no_set_expr:
    unary_expr
    | no_set_expr '*' no_set_expr
    | no_set_expr '.' no_set_expr
    | no_set_expr './' no_set_expr
    | no_set_expr '/' no_set_expr
    | no_set_expr '%' no_set_expr
    | no_set_expr '+' no_set_expr
    | no_set_expr '-' no_set_expr
    | no_set_expr '<<' no_set_expr
    | no_set_expr '>>' no_set_expr
    | no_set_expr '&' no_set_expr
    | no_set_expr '&&' no_set_expr
    | no_set_expr '|' no_set_expr
    | no_set_expr '||' no_set_expr
    | no_set_expr '<' no_set_expr
    | no_set_expr '<=' no_set_expr
    | no_set_expr '>' no_set_expr
    | no_set_expr '>=' no_set_expr
    | no_set_expr '==' no_set_expr
    | no_set_expr '!=' no_set_expr ;

```

```

parameters:
    no_set_expr
    | FESPACE
    | id '=' no_set_expr
    | parameters ',' FESPACE
    | parameters ',' no_set_expr
    | parameters ',' id '=' no_set_expr ;

```

```

array:
    no_comma_expr
    | array ',' no_comma_expr ;

```

```

unary_expr:
    pow_expr
    | unop pow_expr %prec UNARY ;

```

```

pow_expr: primary
    | primary '^' unary_expr
    | primary '_' unary_expr
    | primary '`' ; // transpose

```

```

primary:
    ID
    | LNUM

```

```

|         DNUM
|         CNUM
|         STRING
|         primary '(' parameters ')'
|         primary '[' Expr ']'
|         primary '[' ']'
|         primary '.' ID
|         primary '++'
|         primary '--'
|         TYPE '(' Expr ')' ;
|         '(' Expr ')'
|         '[' array ']' ;

```

11.3 The Types of the languages, and cast

the types

```

--lgElement = <lgElement>
  [, type :<Polymorphic>
  operator :
  ( <lgVertex> : <lgElement>, <long> )

--lgVertex = <lgVertex>
  label, type :<Polymorphic>
  operator. :
  ( <long> : <lgVertex> )

  x, type :<Polymorphic>
  operator. :
  ( <double> : <lgVertex> )

  y, type :<Polymorphic>
  operator. :
  ( <double> : <lgVertex> )

--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>::solveAx>

--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<MGa
( <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<st

--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> *, i

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

--void = <void>

```

11.4 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>
- Plnc, 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>... )

```

```

- intalleges, type :<Polymorphic>
  operator( :
(   <CDomainOfIntegration> :   <Fem2D::Mesh>... )

- jump, type :<Polymorphic>
  operator( :
(   <LinearComb<MDroit, C_F0>> :   <LinearComb<MDroit, C_F0>> )
(   <double> :   <double> )
(   <LinearComb<MGauche, C_F0>> :   <LinearComb<MGauche, C_F0>> )

- label, type :<long *>
- log, type :<Polymorphic>  operator() :
(   <double> :   <double> )
(   <complex> :   <complex> )

- log10, type :<Polymorphic>  operator() :
(   <double> :   <double> )

- max, type :<Polymorphic>  operator() :
(   <double> :   <double>, <double> )
(   <long> :   <long>, <long> )

- mean, type :<Polymorphic>
  operator( :
(   <double> :   <double> )

- min, type :<Polymorphic>  operator() :
(   <double> :   <double>, <double> )
(   <long> :   <long>, <long> )

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

- norm, type :<Polymorphic>
  operator( :
(   <double> :   <std::complex<double>> )

- nuTriangle, type :<long>
- nuEdge, type :<long>
- on, type :<Polymorphic>  operator() :
(   <BC_set<double>> :   <long>... )

- otherside, type :<Polymorphic>
  operator( :
(   <LinearComb<MDroit, C_F0>> :   <LinearComb<MDroit, C_F0>> )
(   <LinearComb<MGauche, C_F0>> :   <LinearComb<MGauche, C_F0>> )

```

```

- pi, type :<double>
- plot, type :<Polymorphic> operator() :
( <long> : ... )

- pow, type :<Polymorphic> operator() :
( <double> : <double>, <double> )
( <complex> : <complex>, <complex> )

- qf1pE, type :<Fem2D::QuadratureFormular1d>
- qf1pT, type :<Fem2D::QuadratureFormular>
- qf1pTlump, type :<Fem2D::QuadratureFormular>
- qf2pE, type :<Fem2D::QuadratureFormular1d>
- qf2pT, type :<Fem2D::QuadratureFormular>
- qf2pT4P1, type :<Fem2D::QuadratureFormular>
- qf3pE, type :<Fem2D::QuadratureFormular1d>
- qf5pT, type :<Fem2D::QuadratureFormular>

- readmesh, type :<Polymorphic> operator() :
( <Fem2D::Mesh> : <string> )

- real, type :<Polymorphic> operator() :
( <double> : <complex> )

- region, type :<long *>
- savemesh, type :<Polymorphic> operator() :
( <Fem2D::Mesh> : <Fem2D::Mesh>, <string>... )

- sin, type :<Polymorphic> operator() :
( <double> : <double> )
( <complex> : <complex> )

- sinh, type :<Polymorphic> operator() :
( <double> : <double> )
( <complex> : <complex> )

- sqrt, type :<Polymorphic> operator() :
( <double> : <double> )
( <complex> : <complex> )

- square, type :<Polymorphic> operator() :
( <Fem2D::Mesh> : <long>, <long> )
( <Fem2D::Mesh> : <long>, <long>, <E_Array> )

- tan, type :<Polymorphic> operator() :
( <double> : <double> )

```

```

- true, type :<bool>
- trunc, type :<Polymorphic> operator() :
(   <Fem2D::Mesh> :   <Fem2D::Mesh>, <bool> )

- verbosity, type :<long *>
- wait, type :<bool *>
- x, type :<double *>
- y, type :<double *>
- z, type :<double *>

```

11.5 History of the software

beginning: november 21, 2001: version 1.08.

2001/11/22: correction of operator == and !=
 2001/11/23 , version: 1.09: correction (with g++)
 template<class A> struct SameType, type of OK must be int and not bool

2001/11/24 add fonctionality in plot, bb=[[x1,y1],[x2,y2]]
 add loop if enter character +,-,=,c,C,r in graphic window.

2001/11/28

correction bug initialization of QuadTree if less than 4 points in the quadtree
 files QuadTree.cpp and FQuadTree.hpp

```

add exec("xxx...");           // to execute on system command. "xxx ... "
add dumptable(cout);          // to show all internal table

```

2001/11/29:

Version ans graphique + ajout d'option dans plot (cf. doc)
 correction ajoute renum() des maillage crees.

2001/12/10

Correct missing check in plot

```

ex: plot(1); trap before now, genere a compile error
Correct in the interpolation is full not conforme FE,
do not prolonged by continuity. change
add after line 479 in file lgfem.cpp
if (outside && !KK.tfe->NbDfOnVertex && !KK.tfe->NbDfOnEdge)
return SetAny<R>(0.0);

```

2001/12/12

correction in gibbs (mesh renum) reconstruct the array of
 triangle for each vertex (PB. of interpolation in non
 convexe domain) Big bug.

Make version 1.14

2001/12/14

correction in trunc mesh, bug if empty mesh is created,
 and move a little the test point in a triangle
 is not exactly the barycenter.

2002/01/14

correction in probem.cpp line 1309 bug when we write qft= ... , in int2d
 if (nargs[0]) return *GetAny<const Fem2D::QuadratureFormular*>((*nargs[1])...

becomes:

```

if (nargs[0]) return *GetAny<const Fem2D::QuadratureFormular*>((*nargs[0])...
2002/01/15
in lgfem.cpp remove line 522 reffecran(); // bug if not graphique some
time
in file MeshGeom.cpp line 108 bug if name == 0 ; add test before the cout like
if(name) cout << " ... " << name << ....
correct the name of the exec file (FreeFee++ -> FreeFem++) in Makefile
2002/02/03
in Mesh2.cpp in preinit() add call to srand(),
to get the same mesh with the same data.
correct
mesh tth=th;

```

Make version 1.15

2002/02/20

Add periodic boundary condition see the manual //

Make version 1.16

Add Parallele Mpi

```

correct bug in SegmentationFault.edp missing placing of delete[] operator
put declaration in for
wait =xx change the default value of wait
ofstream f("foo.txt",append) open a file in append mode
add NoUseOfWait=true; never wait for run test easlyNoUseOfWait}

```

Make version 1.17 2002/03/20

Make default iso value changing with zoom option.

Add GMRES solver dimKrylov= , tgv= //

april 2002

```

Make the current version without CheckPtr, and
correct some bug in string allocation (forgetting +1 in some length)
Improving the speed of the software
add CPUtime global bool variable to print the CPU time of each instruction.
correct some printing without verbosity
adding some checking in array management
Add option in the Makefile (gnumake)

```

Make version 1.18 the 2002/04/08

```

Big correction in automatic cast see bugv1.18.edp
correction in interpolation of label see also bugv1.18.edp
Big correction in construction of non-symetrix matrix (see BUG)

```

Make version 1.19 2002/04/18

Correction in Quadtree integer overflow when
interpolate solution of from one big domain to a very small one.

Correction in embedded function (return problem)
Correction in return type of function (right value an note left value)
Correction cast bool to int, so lots of bug in expression
like (region==2)*5 given alway 0, invisibl before 1.18 (because bug in
automatics cast)

Make version 1.20 2002/04/25

Correction of bug in interpolation on non convexe domain.
correction of bug in periodic boundary condition (pb of sens(in french))
correction some compilation bug under g++ v3.0 in RNM file
correction find common point buildmesh (change threshold value)

Make version 1.21 2002/04/29

Add Non linear Conjugued Gradient (CGNL) routine.
correct bug in linearGC with non zero right and size.

Make version 1.22 2002/05/02

all optimisation tool from // <http://cool.mines.edu/>
Add BFGS optimisation tools form cool //
correct bug in CGNL, correct tgv=, in linear form //
make algo.edp example

Make version 1.23 2002/05/13

correct bug of the symetric matrix are independ of x,y,..
add line at 285, in file problem.cpp:
MeshPointStack(stack)->set(T(pi),pi,Ku);

correct bug in sign of intlD in linearform,
so change in exemple fluidstructure, schwarz-no-overlap, LaplacePl,
aalapacien, lapacienprecon
correct some bug in -= operator with result of *. */
add some operator:
-square(u) = u^2 //
-intalledges(Th)(...) to compute integrale on all edges of all triangle //

for error indicator add global :
-lenEdge the len of the current edge //
-hTriangle the size of the current triangle //
-area the area of the current triangle //

Make version 1.24

add in BamgFreeFem.cpp after line 211 to set name

```

Tn->name= new char[strlen("msh2bamg")+1];
strcpy(Tn->name,"msh2bamg");
add - of linearform and bilinearform see LaplacePlbis.edp

correct: July 8, 2002
Vh u1,u2,v1,v2;
bug in problem(u1,u2,v1,v2,.... ) =
in file: FESpace.hpp line 193:
{ throwassert(dim_which_sub_fem[N-1]>=0 && dim_which_sub_fem[N-1]< nb_sub_fem);
  for(int i=0,n0=0,l=0,i0=0; i<k; i++,n0+=t.N,i0+=t.NbDoF)
    for (int j=0;j<t.pij_alpha.N();j++,l++) {
      pij_alpha[l].i=t.pij_alpha[j].i+i0;
      pij_alpha[l].p=t.pij_alpha[j].p;
      pij_alpha[l].j=t.pij_alpha[j].j+n0;
    }
}

add plot of list of border //

add init array:

real[int] a=[1,2,3,5.6,7,8,9]; //

```

Make version 1.25

13 aout 2002 Major Bug in : FESpace.hpp ligne 199

All the non scalar problem with same kind
of finite element do not work:

FESpace.hpp ligne 199

```

for(int i=0,n0=0,l=0,i0=0; i<k; i++,n0+=t.N,i0+=t.NbDoF)
  for (int j=0;j<t.pij_alpha.N();j++,l++) {
    pij_alpha[l].i=t.pij_alpha[j].i+i0;
    pij_alpha[l].p=t.pij_alpha[j].p;
    pij_alpha[l].j=t.pij_alpha[j].j+n0;
  }

```

become

```

// Warning the componant is moving first
for (int j=0,l=0;j<t.pij_alpha.N();j++) // for all sub DF
  for(int i=0,i0=0; i<k; i++,l++) // for componate
  {
    pij_alpha[l].i=t.pij_alpha[j].i*k+i; // DoF number
    pij_alpha[l].p=t.pij_alpha[j].p; // point of interpolation
    pij_alpha[l].j=t.pij_alpha[j].j+i*t.N; // componente of
interpolation
  }

```

two corrections in problem.cpp :

```

2 missing delete (example laplacienprecon.edp)
2 delete -> delete [] (example testFE.edp)

```

26 septembre 2002

```

add New Finite element RTortho (a conforme //
FE in  $H(curl)$  like RT conforme FE in  $H(div)$ .

```

```

    see exemple aaRT.edp
    correct the computation of dx(v) and dy(u) in [u,v] RT finite element
    function.
    compile on hpux 11 with gcc 3.2 see Makefile-hp9000s700
    correct argument passing in GMRES routine

```

Make version 1.26 le 26 septembre 2002

05 nov. 2002 Correct very small mistake in bamglib part.

19 nov 2002:

```

    add:
    -tanh function //
    -qf2pT4P1 a triangular QuadratureFormular //
(4P1 the qf2pT QuadratureFormular) //
    small change une meshadapt to just compute the metrix see exemple
    convectapt.edp
    make interpolation matrix between to FESpace (not wet finish)

```

3/12/2002

```

    make correction to get a not to bad linenumber in error.
    add add test is used of unset x,y to make a error.
    add checkmovemesh(Th,[x+u,y+v]) function to return the //
    value of the area of the minimal triangle of the movemesh.
    see mesh.edp example.

```

23/12/2002:

```

    add eigen value solver //
    see eigen README_ARPACK to compile
    see examples+-eigen to test.

```

Make version 1.28 2/1/2003

09/1/2003:

```

    add sub array option in array: //
    real[int] tab(100);
    tab(:) = array from 0 to 99=tab.n-1
    tab(2:10) sub array from 2 to 10
    tab(2:10:2) sub array from 2 to 10 by step 2
    tab(2:10:2)[0] = 5 ; => modificaton of tab[2] = 5;
    tab(2:10:2)[1] = 6 ; => modificaton of tab[2+2] = 6;

```

16/01/2003

```

    add macro generation like cpp preprosseur: //
    this is usefull to make automatic diff //
    exemple: //
    real cc=2;
    macro f(u) (cc*(exp(u)-1)) //
    macro df(f) (cc*(exp(f))) //
    real u=1;
    cout << (cc*(exp(u)-1)) << endl;
    cout << f(u) << endl;
    cout << df(f(u)) << endl;

```

see macro.edp for more detail:

22/01/2003

in file lex.cpp add .c_str() for compilation problem
on g++ 2.95

15/04/2003

add new finite element:

```
P1b P1 + Bubble //
P1dc P1 discontinuous //
P2dc P2 discontinuous //
```

correct bug periodic BC with vectorial FESpace,
add the code.

Make version 1.31

23/04/2003

small correction to compile with g++ 2.95.2 in eigen value tools

26/04/2003

```
add function triangulate(filename) to build the //
Delaunay Triangulation of a set of points in R^2.
to build a function form a set of : x y f(x,y)
see exemple mesh.edp
```

make version 1.32 (29/04/2003)

-add optimization in automatique interpolation

```
CPU of all tutorial exemples 172 s( mon my Mac) new version 1.33 (with graphics)
CPU of all tutorial exemples 167 s( mon my Mac) new version 1.33 (without graphics)
CPU of all tutorial exemples 169 s( mon my Mac) new version 1.32 (without graphics)
bofbof.
```

-correct bug in macro generation, a macro existe just in a block {...}.

```
// lex.cpp
if ( ret == '{') { /*cout << " listMacroDef->push_back"<< endl; */
listMacroDef->push_back( MapMacroDef() );}
else if (ret == '}') { /*cout << " listMacroDef->pop_back"<< endl; */
listMacroDef->pop_back( );}
```

-small correction to be compatible with g++ version 3.3

Make version 1.33 01/07/2003

- small correction in parallelempi.cpp add a ; line 159

- comment line 519 in file AFunction.hpp (pb with g++ 3.3.1)

```
// operator Expression() const return f;
(double definition)see line 527
operator const E_F0 * ( ) const {return f;}
```

- correct Makefile for mpi version

21/08/2003:

- Bug in optimization in case on non constant robin boundary condition correct
- add grey=1 ion plot command to make grey plot, and //
correct small mistake (on mac and X11 , ...)

Make version 1.34 (22/08/2003)

16/09/2003

- bug in local variable stack if the size is larger the 8 (complex)
complex a;real b;
a=0;b=1; // here a is 0+i;
- correction is:
- voila la correction dans AFunction.hpp vers la ligne 1486:
- ```
template<class T>
inline Type_Expr NewVariable(aType t,size_t &off)
{
 size_t o= align8(off); // align
 // off += t->un_ptr_type->size;
 // bug off += t->size;
 off += t->un_ptr_type->size; // correction
 return Type_Expr(t,new T(o,t));
}
```

02/11/2003

- correct probleme of renumbering the traingles of a mesh when reading in the bang software (with readmesh command). This implies error when we restore mesh and non P1 finite element solution, see example saveandrestore.edp.

28/11/2003

- correction of bug in window version  
add the umfpack sparce linear solver (this solver have some problem in some case), so I do'nt put this one in default linearsolver. remark, I think is due to the way of taking Dirichlet boundary condition Huge value on diagonal (tgv=1e30) and Stokes matrice. //
- put this file on the web
- put on the web
- the codewarrior projet to build arpack ands umfpack lib.

Make version 1.36

-----

- Change the graphic window in Window xx version

8/12/2003

- repare in mpi version get mesh via mpi
- forget build quadtree soo the interpolation bugs

```

- add 4 line in file parallelempi.cpp in mesh serialization (recivied
 // add 3 line FH 08/12/2003 forget build quadtree sorry
Fem2D::R2 Pn,Px;
a->BoundingBox(Pn,Px);
a->quadtree=new FQuadTree(m,Pn,Px,m->nv);

18/12/2003
- add in mpi version:
broadcast(processor(1),th); // broadcast th from proc 1 to all other.
see the mpi exemple
change the precision to 6 to 12 is savemesh.
- add tool to change the default precision on ostream or ofstream
with the C++ syntaxe
see saverestore.edp exemple.
cout.precision(12);
- add UMFPACK linear solver (not well test)
http: // www.cise.ufl.edu/research/sparse/umfpack
- add full matrix with few operator
real [int,int] A(10,10);
A= 2;
A(5,5) = 2;
cout << A << endl;
-add array of mesh

08/01/2004
- merge all mesh exemple in one file call mesh.edp
- add discontinuous Galerkin method (see LapDG2.edp exemple)
- add syntaxe to get mesh information (see end of mesh.edp exemple)
- add meshsplit function to make conformal recursive locat mesh splitting //

 (see mesh.edp exemple)
- add dynamic load via dlopen see load.edp in example++-load //

Make version 1.37

15/01/2004
- change metrix= in metric= the in adapted mesh metric=[m11,m12,m22] //
 (change the ordre to compatible with the fonction order with IsMetric=1)

02/02/2004
- change printing in gmres algorithme
- print the size of matrix

06/02/2004
- change intalledges in change of discontinuous Galerkin loop also boundary edge
//
jump(u) is external - internal value of u with normal go to internal to external
jump(u) on boundary is -internal value of u //
average(u) on boundary is internal value of u //
- add nTonEdge to get the number of Triangles which see the current edge //

so see the new LapDG2.edp exemple for full detail
07/02/2004
- correct bug in real[string] map; // map array
 change in get_element and operator < of String
- add int array

```

Make Version 1.38

-----

- remove (void) line 14 of throwassert.hpp file (erreur compile g++3.3.5 )
- make change in macro expansion of shell (no perfect to day)

16/04/2004

- add matrix tools
  - bluid interpolation interpolation matrix from a FEspace VH to an other FEspace Vh
  - matrix Ih=matrix B= interplotematrix(VH,Vh);
    - // where Vh correspond to line and Vh to column*
  - the named parameters can be
    - t= true or false (to bluid the transpose or not matrix)
    - op=0 ,1 or 2 (to build the interpolation of value, dx,dy )
    - inside=true or false (to remove or not all outside interpolation quadrature point)
  - build diagonal sparce matrix (type is morse)
  - do the matrice product of to sparce matrix (type is morse)
  - add build a sparce matrix from a full matrix so if you have install UMFPACK solver it is possible to solve systeme with full matrix.
  - add set function to change the solver
  - see exemple sparce-matrix.edp
- correct bug flag outside of mesh
  - add line in file fem.cpp line 864
    - outside=true;
  - so some time the P0 interpolatation is no zero
  - add new king of mesh of multiplicator data :
    - mesh emptyTh=emptymesh(Th); *// to build a mesh with no internal point*
  - see mesh.edp exemple

Make Version 1.40

-----

- add les linear combination of sparce matrice
- set a sparce solver to a sparce matrice (to day just UMFPACK)
- transforme a full matrix in sparce matrix
- add istream f("toto");
  - f.good() or f.EOF to test state the file
  - remark: the state is set after the read so the previous value is wrong

02/06/2004

-----

- correction in call macro with string "... " parameter

24/06/2004

-----

- add tools to bluid periodic adapmesh see sphere.edp of exemple
- add HaveUMFPACK global variable see sparse-matrix.edp exemple
- Change all the structure of the software
- Use ./configure to build all the make file see (README)
- you can build the Window version with g++ under cygwin
- ( http: cygwin.... )

Make Version 1.41

-----

07/07/2004

-----

an anonymous CVS server is available to get FreeFem++ source  
 cvs -d :pserver:anonymousidared.ann.jussieu.fr:/Users/pubcvs/cvs login  
 (password = freefem++)  
 cvs -d :pserver:anonymousidared.ann.jussieu.fr:/Users/pubcvs/cvs freefem++ co  
 the TAG release\_1\_41\_before\_packaging is created

30/08/04

-----

Correct divide by 0 in plot instruction when the bounding box is flat.  
 Set default solver to LU because UMFPACK sometime by bad result.  
 Correct all g++-3.4 error  
 Optimize Choleski and Crout solver (divide cpu time by two on pentium)  
 Add stuff for adding new finite element due to the new software achitecture.

31/08/04

-----

Add string[string] array  
 correct bug in macro expansion  
 macro parameter will be not expanded  
 example of bug:  
 macro a(i) i  
 macro b a(x) // bug here  
 b = 1 ; // we get "i=1" or we want "x =1"

# 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] T. A. DAVIS Algorithm 8xx: UMFPACK V4.1, an unsymmetric-pattern multi-frontal method TOMS, 2003 (under submission) <http://www.cise.ufl.edu/research/sparse/umfpack>
- [3] D. Bernardi, F.Hecht, K. Ohtsuka, O. Pironneau: *freefem+ documentation*, on the web at <ftp://www.freefem.org/freefemplus>.
- [4] D. Bernardi, F.Hecht, O. Pironneau, C. Prud'homme: *freefem documentation*, on the web at <http://www.asci.fr>
- [5] P.L. George: *Automatic triangulation*, Wiley 1996.
- [6] F. Hecht: The mesh adapting software: bamg. INRIA report 1998.
- [7] Modulef ??????????
- [8] J.L. Lions, O. Pironneau: Parallel Algorithms for boundary value problems, Note CRAS. Dec 1998. Also : Superpositions for composite domains (to appear)
- [9] B. Lucquin, O. Pironneau: *Scientific Computing for Engineers* Wiley 1998.
- [10] F. Preparata, M. Shamos; *Computational Geometry* Springer series in Computer sciences, 1984.
- [11] 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
- [12] J.L. Steger: The Chimera method of flow simulation, Workshop on applied CFD, Univ of Tennessee Space Institute, August 1991.
- [13] N. WIRTH: *Algorithms + Data Structures = Programs*, Prentice Hall, 1976
- [14] ison documentation
- [15] The C++ , programming language, Third edition, Bjarne Stroustrup, Addison-Wesley 1997.

- [16] COOOL: a package of tools for writing optimization code and solving optimization problems,
- [17] B. Riviere, M. Wheeler, V. Girault, A priori error estimates for finite element methods based on discontinuous approximation spaces for elliptic problems. *SIAM J. Numer. Anal.* 39 (2001), no. 3, 902–931 (electronic).
- [18] COOL package <http://cool.mines.edu>

# Index

- ' , 14
- . \* , 14, 41
- << , 15, 17
- >> , 17
- [ ] , 33, 40, 74
- \ " , 13
- , 11
- \n , 13
- ^-1 , 41
  
- adaptmesh, 19, 23, 54
  - abserror=, 25
  - cutoff=, 25
  - err=, 24
  - errg=, 24
  - hmax=, 24
  - hmin=, 24
  - inquire=, 25
  - isMetric=, 25
  - iso=, 25
  - keepbackvertices=, 25
  - maxsubdiv=, 25
  - metric=, 26
  - nbjacobian=, 24
  - nbsmooth=, 24
  - nbvx=, 24
  - nomeshgeneration=, 26
  - omega=, 25
  - periodic=, 26
  - powerin=, 25
  - ratio=, 25
  - rescaling=, 25
  - splitin2, 26
  - splitpbedge=, 25
  - verbosity= , 25
- alphanumeric, 11
- append, 17
- area, 14, 129
- array, 16, 40
  - FE function, 13, 16
  - fespace, 32
  - initialisation, 130
  - integer index, 12
  - mesh, 89, 134
  - mesh index, 12
  - string, 134
  - string index, 12
  - sub, 131
- assert, 16
- average, 36, 134
  
- bang, 23, 69
- BFGS, 52, 129
- bool, 12
- border, 19, 53
- boundary condition, 41
- break, 17
- broadcast, 89, 134
- buildmesh, 19, 53
  
- CG, 36, 37
- CGNL, 129
- checkmovemesh, 21, 131
- Cholesky, 37, 61
- cin, 14
- cint, 17
- compiler, 8
- complex, 12
- concatenation, 15, 58
- continue, 17
- convect, 65, 67
- cout, 14, 17
- Crout, 37
  
- Dirichlet, 57, 64
- discontinuous functions, 79
- divide
  - term to term, 14
- domain decomposition, 71, 72
- dot product, 14, 16

- dumptable, 14
- dynamic file names, 58
- dynamic load, 134
- EigenValue, 43
- eigenvalue, 131
- emptymesh, 19, 27
- endl, 14, 17
- exec, 14, 46, 48
- exit, 16
- factorize=, 61
- false, 12–14
- FE function, 32
  - `[]`, **33**
  - `n`, **33**
  - `set`, 33
  - `value`, 33
- FE space, 32
- femp1, 53
- fespace, 12, 31, 53
  - P0, 31
  - P1, 31
  - P1b, 31
  - P1dc, 31
  - P1nc, 32
  - P2, 32
  - P2dc, 32
  - periodic=, 37, 55
  - RT0, 32
- file
  - am, 103, 104
  - am\_fmt, 22, 68, 103, 104
  - amdba, 103, 104
  - bamg, 22, 68, 101
  - data base, 101
  - ftq, 105
  - mesh, 22, 68
  - msh, 104
  - nopo, 23, 68
- finite element, 31
- fluid, 63
- fluid-structure interaction, 75
- for, 17
- func, 12
- function, 61
  - tables, 23
- GC, 40
- GMRES, 37, 128
- gnuplot, 46
- grey=, 133
- hTriangle, 13, 58, 129
- ifstream, 17
- include, 67
- init=, 37
- int, 12
- int1d, 37, 39
- int2d, 34, 37, 39
- intalldges, 37, 39, 58, 129, 134
- integer
  - constant, 13
- interpolation, 33
- jump, 36, 58, 134
- label, 13, 19, 53, 56
- label=, 26
- lagrangian, 22
- lenEdge, 13, 58, 129
- LinearCG, 66
- linearCG, 51
  - eps=, 51
  - nbiter=, 51
  - precon=, 51
  - veps=, 51
- LU, 35, 37
- macro, 84, 131
- matrix, 13, 40, 61
  - =, 67
  - factorize=, 63
  - full, 134
  - integer index, 12
  - solver, 40
  - solver=, 67
- medit, 48
- mesh, 12
  - change, 33
- meshadapt
  - metric=, 134
- meshsplit, 134
- mixed FEM formulation, 35
- movemesh, 19, 21

- mpirank, 89
- mpisize, 89
- N, 13, 38
  - x, 35
  - y, 35
- n, 33
- Navier-Stokes, 65, 67
- Neumann, 35, 64
- Newton, 52, 62
- NLCG, 51, 62
  - eps=, 51
  - nbiter=, 51
  - veps=, 51
- normal, 38
- nTonEdge, 14, 36, 134
- nuEdge, 14
- nuTriangle, 13
- ofstream, 17
  - append, 17
- on, 34, 38
  - intersection, 66
- optimize=, 41
- P, 13
- P0, 12, **31**
- P1, 12, **31**
- P1b, 13, **31**, 132
- P1dc, 12, **31**, 132
- P1nc, 12, **32**
- P2, 12, **32**
- P2dc, 12, **32**, 132
- periodic, 31, 37, 55, 128
- pi, 14
- plot, 45
  - aspectratio =, 45
  - nbiso =, 45
  - bb=, 45
  - border, 20, 130
  - bw=, 46
  - cmm=, 45
  - coef=, 45
  - cut, 46
  - grey=, 46, 133
  - mesh, 20
  - nbarrow=, 45
  - ps=, 45
  - value=, 45
  - varrow=, 46
  - viso=, 45
- postscript, 57
- precision, 134
- precon=, 37, 68
- problem, 13, 34
  - dimKrylov=, 128
  - eps, 24
  - eps=, 37
  - init, 24
  - init=, 37
  - precon=, 37
  - solver=, 24, 35, 37
  - tgvs=, 37, 128, 133
- processor, 89
- product
  - dot, 14, 16
  - term to term, 14
- qf1pE, 39
- qf1pT, 39
- qf2pE, 39
- qf2pT, 39
- qf2pT4P1, 131
- qforder=, 39, 58, 67
- qft=, 39
- read files, 68
- readmesh, 19, 22
- real, 12
  - constant, 13
- region, 13, 78
- Reusable matrices, 65
- RT0, 12, **32**
- RTortho, 130
- savemesh, 22, 57
- schwarz, 89
- shurr, 71, 72
- singularity, 56
- solve, 13, 34
  - eps=, 37
  - init=, 37
  - linear system, 14, 41
  - precon=, 37
  - solver=, 37
  - tgvs=, 37, 128, 133

- solver=, 61
  - CG, 37, 57
  - Cholesky, 37
  - Crout, 37
  - GMRES, 37
  - LU, 37
  - UMFPACK, 37
- split=, 26
- splitmesh, 19
- square, 19, 58, 129
- Stokes, 63
- stokes, 63
- stop test, 37
  - absolue, 67
- streamlines, 64
- string, 12
  - constant, 13
  - string index, 12
- subdomains, 78
  
- tanh, 131
- Taylor-Hood, 66
- transpose, 14, 16, 116
- triangle
  - $\square$ , 28
  - label, 28
- triangulate, 19, 23, 132
- true, 12–14
- trunc, 26
  - label=, 26
  - split=, 26
- type of finite element, 31
  
- UMFPACK, 36, 37, 134
- Uzawa, 66
  
- varf, 13, 37, 40, 67
- variable, 11
- veps=, 62
- vertex
  - label, 28
  - x, 28
  - y, 28
  
- while, 17
- write files, 68
  
- x, 13
- y, 13
- z, 13