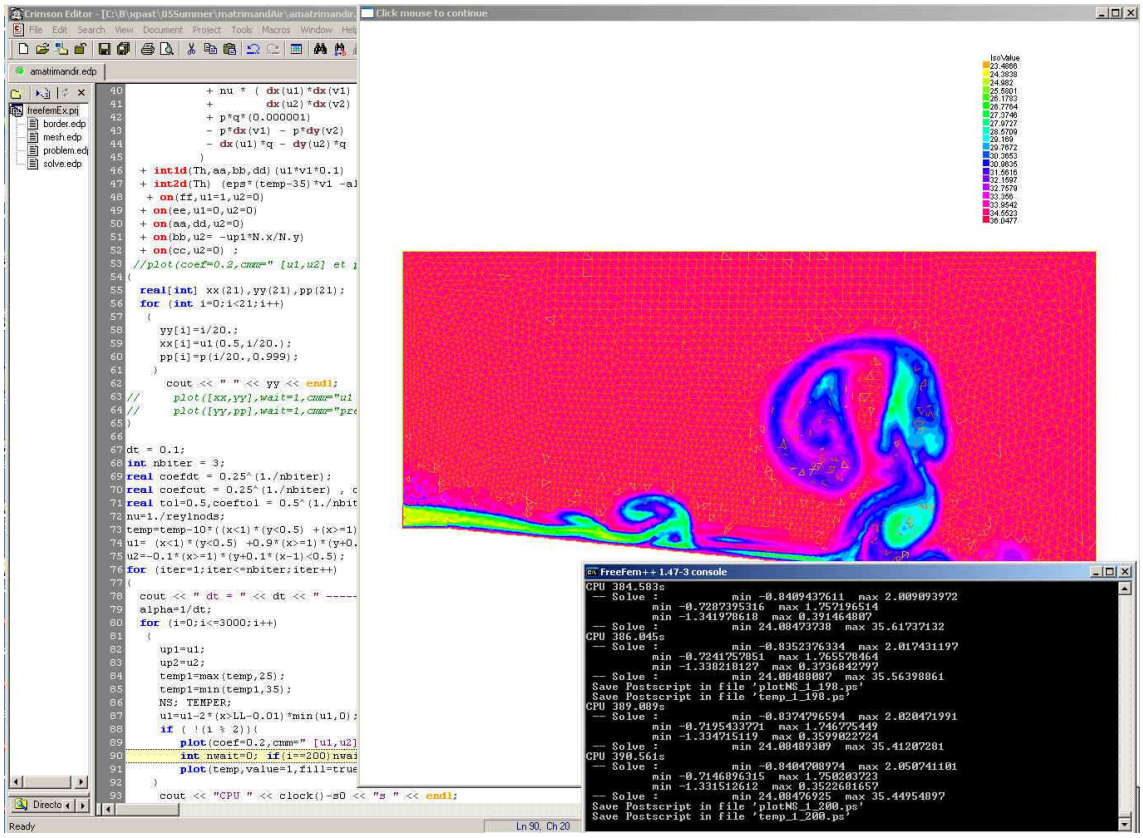
Freefem++

Second Edition, Version 2.23-2

http://www.freefem.org/ff++

F. Hecht, O. Pironneau
A. Le Hyaric, K. Ohtsuka



Laboratoire Jacques-Louis Lions, Université Pierre et Marie Curie, Paris

FreeFem++

Second Edition, Version 2.23-2

http://www.freefem.org/ff++

Frédéric Hecht[1,4]
mailto:frederic.hecht@upmc.fr
http://www.ann.jussieu.fr/~hecht

Olivier Pironneau[1,3]
mailto:olivier.pironneau@upmc.fr
http://www.ann.jussieu.fr/pironneau

Antoine Le Hyaric[1,2]
mailto:lehyaric@ann.jussieu.fr
http://www.ann.jussieu.fr/~lehyaric/

Kohji Ohtsuka[5]
mailto:ohtsuka@barnard.cs.hkg.ac.jp
http://barnard.cs.hkg.ac.jp/

1. Laboratoire Jacques-Louis Lions, 175 rue du Chevaleret , 75013 Paris, France
   Université Pierre et Marie Curie, 4 Place Jussieu, 75252 PARIS cedex 05, France.
2. CNRS, UMR 7598, Laboratoire Jacques-Louis Lions, 175 rue du Chevaleret , 75013 Paris, France.
3. Institut de France, Académie des sciences, 23, quai de Conti, 75006 Paris, France.
4. INRIA, Projet Gamma, Domaine de Voluceau, Rocquencourt, B.P. 105, 78153 Le Chesnay Cedex, France.
5. Hiroshima Kokusai Gakuin University, Hiroshima, Japan.

Acknowledgments We are very grateful to l'École Polytechnique (Palaiseau, France) for printing the second edition of this manual (http://www.polytechnique.fr), and to l'Agence Nationale de la Recherche (Paris, France) for funding of the extension of FreeFem++ to a parallel tridimensional version (http://www.agence-nationale-recherche.fr).

Contents

Preface

ॐ पूर्णमदः पूर्णमिदं पूर्णात् पूर्णमुदच्यते।
पूर्णस्य पूर्णमादाय पूर्णमेवावशिष्यते ॥
ॐ शान्तिः शान्तिः शान्तिः ॥

Fruit of a long maturing process freefem, in its last avatar, `FreeFem++`, is a high level integrated development environment (IDE) for numerically solvong partial differential equations (PDE). It is the ideal tool for teaching the finite element method but it is also perfect for research to quickly test new ideas or multi-physics and complex applications.

`FreeFem++` has an advanced automatic mesh generator, capable of a posteriori mesh adaptation; it has a general purpose elliptic solver interfaced with fast algorithms such as the multi-frontal method UMFPACK. Hyperbolic and parabolic problems are solved by iterative algorithms prescribed by the user with the high level language of `FreeFem++`. It has several triangular finite elements, including discontinuous elements. Finally everything is there in `FreeFem++` to prepare research quality reports: color display online with zooming and other features and postscript printouts.

This book is ideal for students at Master level, for researchers at any level, and for engineers, and also in financial mathematics.

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; it is not a package, it is an integrated product with its own high level programming language. This software runs on all unix OS (with g++ 2.95.2 or later, and X11R6) , on Window95, 98, 2000, NT, XP, and MacOS X.

Moreover `FreeFem++` is highly adaptive. Many phenomena involve several coupled system, for example: fluid-structure interactions, Lorenz forces for aluminium casting and ocean-atmosphere problems are three such systems. These require different finite element approximations degrees, possibly on different meshes. Some algorithms like Schwarz' domain decomposition method also require data interpolation on multiple meshes within one program. `FreeFem++` can handle these difficulties, i.e. *arbitrary finite element spaces on arbitrary unstructured and adapted bidimensional meshes*.

The characteristics of `FreeFem++` are:

- Problem description (real or complex valued) by their variational formulations, with access to the internal vectors and matrices if needed.
- Multi-variables, multi-equations, bi-dimensional (or 3D axisymmetric) , static or time dependent, linear or nonlinear coupled systems; however the user is required to describe the iterative procedures which reduce the problem to a set of linear problems.
- Easy geometric input by analytic description of boundaries by pieces; however this software is not a CAD system; for instance when two boundaries intersect, the user must specify the intersection points.
- Automatic mesh generator, based on the Delaunay-Voronoi algorithm. Inner point density is proportional to the density of points on the boundary [7].
- Metric-based anisotropic mesh adaptation. The metric can be computed automatically from the Hessian of any `FreeFem++` function [8].

- High level user friendly typed input language with an algebra of analytic and finite element functions.
- Multiple finite element meshes within one application with automatic interpolation of data on different meshes and possible storage of the interpolation matrices.
- A large variety of triangular finite elements : linear and quadratic Lagrangian elements, discontinuous P1 and Raviart-Thomas elements, elements of a non-scalar type, mini-element,... (but no quadrangles).
- Tools to define discontinuous Galerkin formulations via finite elements `P0`, `P1dc`, `P2dc` and keywords: `jump`, `mean`, `intalledges`.
- A large variety of linear direct and iterative solvers (LU, Cholesky, Crout, CG, GMRES, UMFPACK) and eigenvalue and eigenvector solvers.
- Near optimal execution speed (compared with compiled C++ implementations programmed directly).
- Online graphics, generation of `,.txt,.eps,.gnu, mesh` files for further manipulations of input and output data.
- Many examples and tutorials: elliptic, parabolic and hyperbolic problems, Navier-Stokes flows, elasticity, Fluid structure interactions, Schwarz's domain decomposition method, eigenvalue problem, residual error indicator, ...
- An experimental parallel version using `mpi`

1.1 Installation

First open the following web page

`http://www.freefem.org/ff++/`

And choose your architecture files: Linux, Windows, MacOS X, or go to the end of the page to get the full list of download.
Remark: Binaries are available for Microsoft Windows and Apple Mac OS X and Linux.

1.1.1 Installation from sources

Only for those who need to recompile `FreeFem++` or install it from the source code:
To compile the documentation and the application under MS-Windows we have used the LaTeX and the `cygwin` environment from

`http://www.cygwin.com`

and under MacOS X we have used the apple Developer Tools Xcode, LaTeX from `http://www.ctan.org/system/mac/texmac`.
`FreeFem++` must be compiled and installed from the source archive. This archive is available from:
To extract files from the compressed archive `freefem++-(VERSION).tar.gz` a directory called

`freefem++-(VERSION)`

enter the following commands in a shell window :

```
tar zxvf freefem++-(VERSION).tar.gz
cd freefem++-(VERSION)
```

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) :
After installation, The list of application (depending of the system and the compiling option) can be :

1. `FreeFem++`, standard version, with a graphical interface based on X11, Win32
2. `FreeFem++-nw`, postscript plot output only (batch version, no windows)
3. `FreeFem++-mpi`, parallel version, postscript output only
4. `FreeFem++-glx`, graphics using OpenGL and X11
5. `FreeFem++-cs`, integrated development environment (please see chapter Graphical User Interface 11 for more details).
6. `/Applications/FreeFem++.app`, Drag and Drop CoCoa MacOs Application
7. `FreeFem++-CoCoa`, MacOS Shell script for MacOS OpenGL version (MacOS 10.2 or better) (note: it uses /Applications/FreeFem++.app)
8. `bamg` , the bamg mesh generator
9. `cvmsh2` , a mesh file convertor
10. `drawbdmesh` , a mesh file viewer

Remark, in most of the case you can set the level of the output (verbosity) to value `nn` with adding the parameters `-v nn` at the command line.
As an installation test, under unix: go into the directory `examples++-tutorial` and run `FreeFem++` on the example script `LaplaceP1.edp` with the command :

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

And if you are using `nedit` as your text editor, Do one time `nedit -import edp.nedit` to have coloring syntax for yours `.edp` files.

1.1.2 Windows binaries install

First download the windows installation file, execute the download file to install FreeFem++, after that you have three new icons on your destop:

- FreeFem++ (VERSION).exe the classical FreeFem++ application.
- FreeFem++ (VERSION) GUI.exe the GUI FreeFem++ application, see section 11 for more information.
- FreeFem++ (VERSION) Examples a link to theFreeFem++ directory examples.

where (VERSION) is the version of the files for example 2.3-0-P4.
By default, the installed files are in

C:\Programs Files\FreeFem++

In this directory, you have all the .dll files and and other application FreeFem++-nw.exe the FreeFem++ application without graphic windows.

Remark: if you add to the environment variable PATH C:\Program Files\FreeFem++ you can use all the programs in the window Command-line shell. (Use System in Control Panel, On the Advanced tab, click Environment Variables, then add the correct string).
The systaxe of tools in command-line are

- FreeFem++.exe [-vnn] [-b] [-s] [-n] [-h] [-f] [filepath] where the

  -b no color (black and white plot)

  -n no edit ????

  -s not wait at end of execution

  -vnn set the level of verbosity to nn before execution of the script.

  if no file path then you get a dialog box to choose the edp file.

- FreeFem++-nw.exe [-v nn] [[-f] filepath]

where the part in [] is facultative.

Link with other text editor

Crimson Editor at http://www.crimsoneditor.com/ and adapt it as follows:

- Go to the Tools/Preferences/File association menu and add the .edp extension set
- In the same panel in Tools/User Tools, add a FreeFem++ item (1st line) with the path to freefem++.exe on the second line and $(FilePath) and $(FileDir) on third and fourth lines. Tick the 8.3 box.
- for color syntax, extract file from crimson-freefem.zip and put files in the corresponding sub-folder of Crimson folder (C:\Program Files\Crimson Editor).

winedt for Windows : this is the best but it could be tricky to set up. Download it from

`http://www.winedt.com`

this is a multipurpose text editor with advance features such as syntax coloring; a macro is available on ww.freefem.org to localize winedt to `FreeFem++` without disturbing the winedt functional mode for LateX, TeX, C, etc. However winedt is not free after the trial period.

TeXnicCenter for Windows: this is the easiest and will be the best once we find a volonteer to program the color syntax. Download it from

`http://www.texniccenter.org/`

It is also an editor for TeX/LaTeX but it has a "'tool"' menu which can be configure to launch `FreeFem++` programs: do the following:

- Select the `Tools/Customize` menu will bring a dialog box.
- Select the `Tools` tab and create a new item: call it `freefem`.
- in the 3 lines below,
  1. search for `FreeFem++.exe`
  2. select Main file with further option then Full path and click also on the `8.3 box`
  3. select main file full directory path with 8.3

nedit on the Mac OS, Cygwin/Xfree and linux, to import the color syntax do

```
nedit -import edp.nedit
```

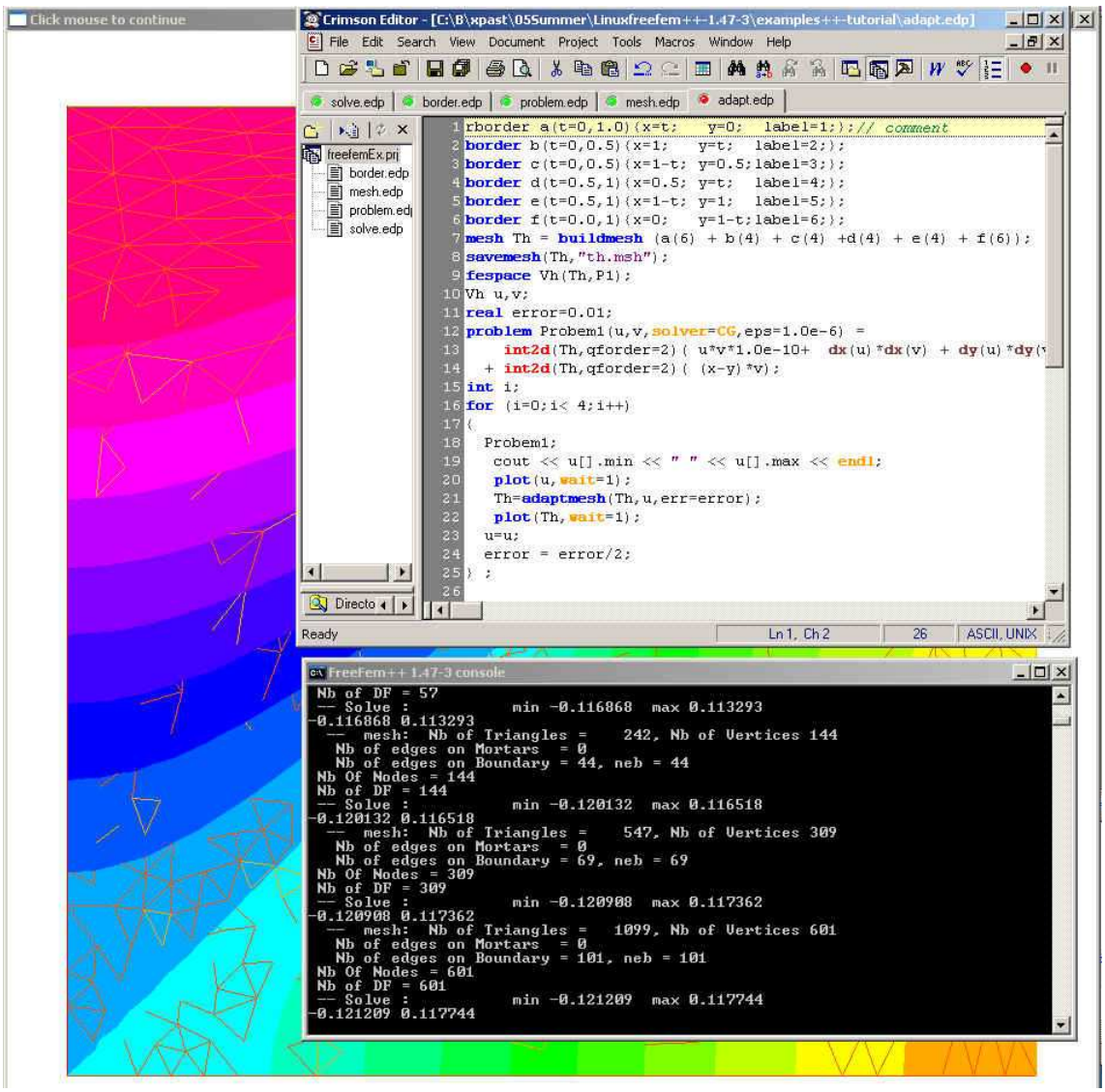
1.1.3 MacOS X binaries install

Download the MacOS X binaries version files , extract all the file with a double click on the icon of the file, go the the directory and put the `FreeFem+.app` application in the `/Applications` directory. If you want a terminal acces to `FreeFem++` just copy the file `FreeFem++-CoCoa` in the directory of your `$PATH` shell environment variable.
If you want to automatically launch the `FreeFem++.app` if you double click on a `.edp` file icon. Under the finder pick a `.edp` in directory `examples++-tutorial` for example, select menu `File -> Get Info` an `change Open with:` (choose FreeFem++.app) and click on button `change All...`.

1.2 How to use `FreeFem++`

Under Windows with Graphic Interfaces The executable `freefem++.exe` opens a dialog box for choosing the input file; then it executes its content and produces graphics and output files.

To write or modify the input file a text editor can be used.




```
rborder a(t=0,1.0)(x=t;   y=0;  label=1;);// comment
border b(t=0,0.5)(x=1;   y=t;  label=2;);
border c(t=0,0.5)(x=1-t; y=0.5;label=3;);
border d(t=0.5,1)(x=0.5; y=t;  label=4;);
border e(t=0.5,1)(x=1-t; y=1;  label=5;);
border f(t=0.0,1)(x=0;   y=1-t;label=6;);
mesh Th = buildmesh (a(6) + b(4) + c(4) +d(4) + e(4) + f(6));
savemesh(Th,"th.msh");
fespace Vh(Th,P1);
Vh u,v;
real error=0.01;
problem Problem1(u,v,solver=CG,eps=1.0e-6) =
    int2d(Th,qforder=2)( u*v*1.0e-10+  dx(u)*dx(v) + dy(u)*dy(v
  + int2d(Th,qforder=2)( (x-y)*v);
int i;
for (i=0;i< 4;i++)
{
  Problem1;
   cout << u[].min << " " << u[].max << endl;
   plot(u,wait=1);
   Th=adaptmesh(Th,u,err=error);
   plot(Th,wait=1);
  u=u;
  error = error/2;
} ;
```

```
Nb of DF = 57
-- Solve :           min -0.116868  max 0.113293
-0.116868 0.113293
  --  mesh:  Nb of Triangles =    242, Nb of Vertices 144
   Nb of edges on Mortars  = 0
   Nb of edges on Boundary = 44, neb = 44
Nb Of Nodes = 144
Nb of DF = 144
-- Solve :           min -0.120132  max 0.116518
-0.120132 0.116518
  --  mesh:  Nb of Triangles =    547, Nb of Vertices 309
   Nb of edges on Mortars  = 0
   Nb of edges on Boundary = 69, neb = 69
Nb Of Nodes = 309
Nb of DF = 309
-- Solve :           min -0.120908  max 0.117362
-0.120908 0.117362
  --  mesh:  Nb of Triangles =   1099, Nb of Vertices 601
   Nb of edges on Mortars  = 0
   Nb of edges on Boundary = 101, neb = 101
Nb Of Nodes = 601
Nb of DF = 601
-- Solve :           min -0.121209  max 0.117744
-0.121209 0.117744
```

Figure 1.1: The 3 panels of the integrated environment built with the `Crimson Editor` with `FreeFem++` in action. The Tools menu has an item to launch `FreeFem++` by a Ctrl+1 command.

An integrated environment (FreeFem++ GUI), written by A. Le Hyaric, is also included with the distribution. the application name is FreeFem++-cs; note however that the graphics display much slower in this mode. There are other ways to have an integrated environment. TeX users have usually an editor installed; if it is winedt or TeXnicCenter then these can be programmed to handle the edit-run-correct cycle of FreeFem++ with color syntax and automatic launch of freefem. If you don't have one of these installed the easiest is to download the freeware Crimson Editor.

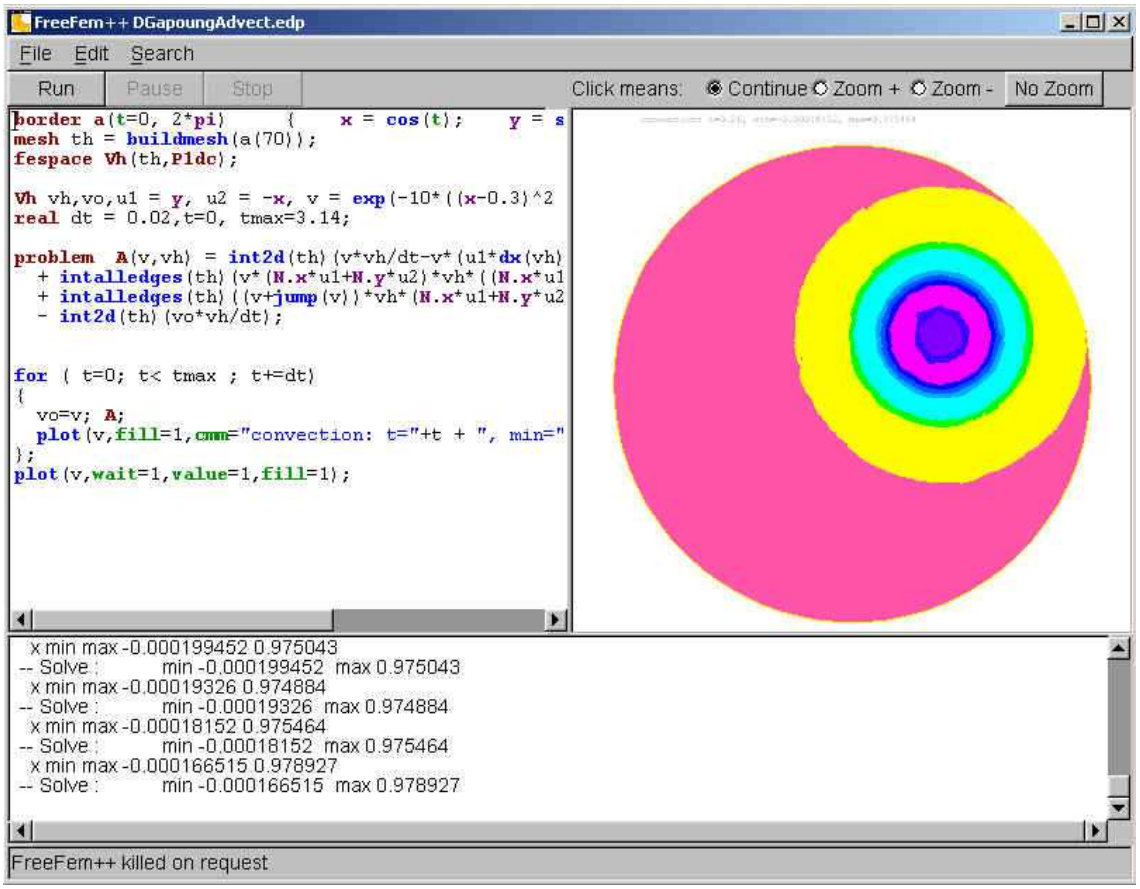


Figure 1.2: The 3 panels of the integrated environment freefem++-cs: to the top left one sus the program (which can be edited?), on the top right ons the graphic window, and on the bottom window text message are displayed.

Under MacOS X with Graphic Interfaces For testing or running a .edp file, and juste drag and drop the file icon on The MacOS application FreeFem++.app icon, but your can also use the menu: File → Open after launching the application.
Oe and one of the best way on MacOS is using the text editor mi.app

http://www.mimikaki.net/en/

and using the edp mode store in mode-mi-edp.zip. So just download the mi editor install it, unzip the file and put crated Folder in the Folder openned with the mi.app menu Option->Open Mode Folder menu and set mi is the Default application. for all the .edp files.

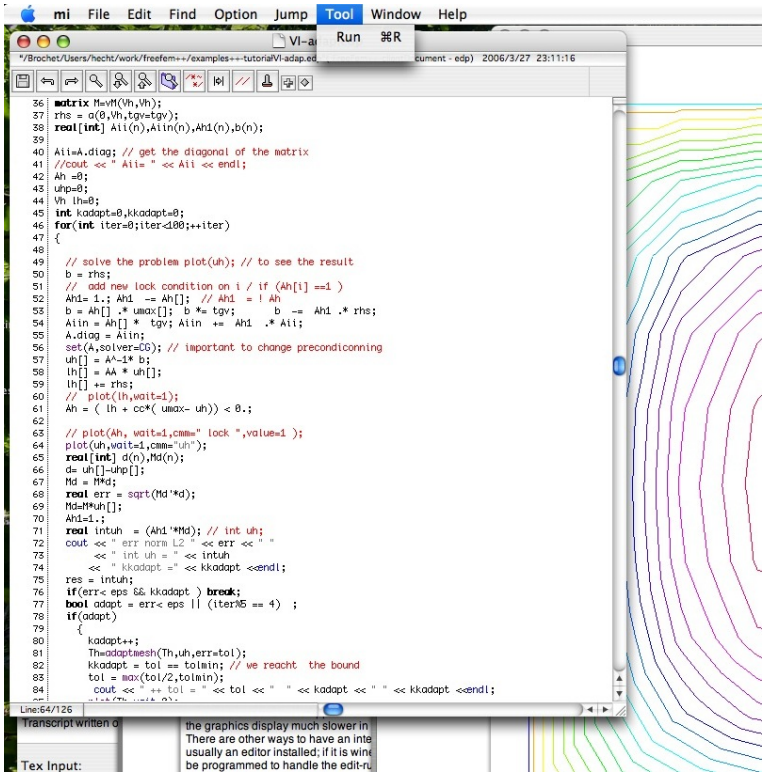


Figure 1.3: Screen of edp with `mi` text editor

Under terminal First choose the type of application `FreeFem++`, `FreeFem++-nw`, ... depending on your pleasure, system, etc.... So for example your can enter

```
FreeFem++ your-edp-file-path
```

1.3 Environment variable, and init file

`FreeFem++` reads an user's init file named `freefem++.pref` to can change value of global variable: `verbosity`, `includepath`, `loadpath`.
Remark: `verbosity` variable change the level of internal printing (0, nothing (except mistake), 1 fews, 10 lots, etc ...), the default value is 2. The include files are search in the `includepath` list and the load files are search in `loadpath` list.
The syntaxe of the file is:

```
verbosity= 5
loadpath += "/Library/FreeFem++/lib"
loadpath += "/Users/hecht/Library/FreeFem++/lib"
includepath  += "/Library/FreeFem++/edp"
includepath  += "/Users/hecht/Library/FreeFem++/edp"
#  comment
load += "funcTemplate"
load += "myfunction"
```

The possible paths for this file are

- under unix and MacOs

```
/etc/freefem++.pref
   $(HOME)/.freefem++.pref
   freefem++.pref
```

- under windows

```
freefem++.pref
```

We can also use shell environment variable to change , verbosity and the search rule.

```
export FF_VERBOSITY=50
export FF_INCLUDEPATH="dir;;dir2"
export FF_LOADPATH="dir;;dir3""
```

Remark: the separtor of directory is ";" and not ":" because ":" is used under Windows.

1.4 History

The project has evolved from `MacFem, PCfem`, written in Pascal. The first C version lead to `freefem 3.4`; it offered mesh adaptativity on a single mesh only.

A thorough rewriting in C++ led to `freefem+` (`freefem+ 1.2.10` was its last release), which included interpolation over multiple meshes (functions defined on one mesh can be used on any other mesh); this software is no longer maintained but still in use because it handles a problem description using the strong forms of the PDEs. 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[17] for example). Doing it in a minimum number of operations was the challenge. Our implementation is $O(n \log n)$ and based on a quadtree. This version also grew out of hand because of the evolution of the template syntax in C++.

We have been working for a few years now on `FreeFem++` , entirely re-written again in C++ with a thorough usage of `template` and generic programming for coupled systems of unknown size at compile time. Like all versions of `freefem` it has a high level user friendly input language which is not too far from the mathematical writing of the problems.

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 [25], templates and `bison` for its implementation; more detail can be found in [11]. 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. Therefore the library of finite elements available in `FreeFem++` will grow with the version number and with the number of users who program more new elements. So far we have discontinuous P_0 elements,linear P_1 and quadratic P_2 Lagrangian elements, discontinuous P_1 and Raviart-Thomas elements and a few others like bubble elements.

Chapter 2

Getting Started

To illustrate with an example, let us explain how FreeFem++ solves the **Poisson**'s equation: *for a given function* $f(x,y)$*, find a function* $u(x,y)$ *satisfying*

$$-\Delta u(x,y) = f(x,y) \quad \text{for all } (x,y) \in \Omega, \tag{2.1}$$

$$u(x,y) = 0 \quad \text{for all } (x,y) \text{ on } \partial\Omega, . \tag{2.2}$$

Here $\partial\Omega$ is the boundary of the bounded open set $\Omega \subset \mathbb{R}^2$ and $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$.
The following is a FreeFem++ program which computes u when $f(x,y) = xy$ and Ω is the unit disk. The boundary $C = \partial\Omega$ is

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

Note that in FreeFem++ the domain Ω is assumed to described by its boundary that is on the left side of its boundary oriented by the parameter. As illustrated in Fig. 2.2, we can see the isovalue of u by using `plot` (see line 13 below).

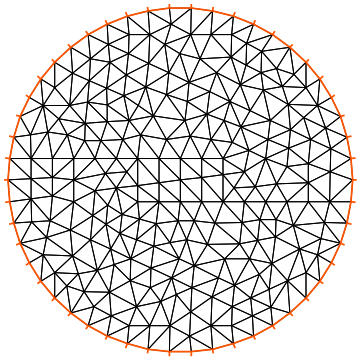


Figure 2.1: mesh Th by build(C(50))

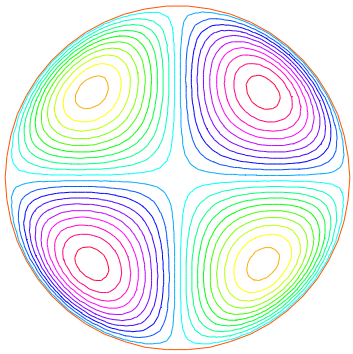


Figure 2.2: isovalue by plot(u)

Example 2.1

```
    //    defining the bourndary
1: border C(t=0,2*pi){x=cos(t); y=sin(t);}
```

```
   //    the triangulated domain Th is on the left side of its boundary
2: mesh Th = buildmesh (C(50));
            //    the finite element space defined over Th is called here Vh
3; fespace Vh(Th,P1);
4: Vh u,v;          //    defines u and v as piecewise-P1 continuous functions
5: func f= x*y;                          //    definition of a called f function
6: real cpu=clock();                                 //    get the clock is second
7: solve Poisson(u,v,solver=LU) =                      //    defines the PDE
8:    int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v))               //    bilinear part
9:    - int2d(Th)( f*v)                                  //    right hand side
10:    + on(C,u=0)  ;                        //    Dirichlet boundary condition
11: plot(u);
12: cout << " CPU time = " << clock()-cpu << endl;
```

Note that the qualifier `solver=LU` is not required and by default a multi-frontal LU would have been used. Note also that the lines containing `clock` are equally not required. Finally note how close to the mathematics `FreeFem++` input language is. `8` and `9` corresponds to the mathematical variational equation

$$\int_{T_h} (\frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y})\mathrm{d}x\mathrm{d}y = \int_{T_h} f v \mathrm{d}x\mathrm{d}y$$

for all v which are in the finite element space V_h and zero on the boundary C.

Exercise : Change P1 into P2 and run the program.

2.0.1 FEM by `FreeFem++` : how does it work?

This first example shows how `FreeFem++` executes with no effort all the usual steps required by the finite element method (FEM). Let us go through them one by one.

1st line: the boundary Γ is described analytically by a parametric equation for x and for y. When $\Gamma = \sum_{j=0}^{J} \Gamma_j$ then each curve Γ_j, must be specified and crossings of Γ_j are not allowed except at end points .
The keyword "label" can be added to define a group of boundaries for later use (boundary conditions for instance). Hence the circle could also have been described as two half circle with the same label:

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

Boundaries can be referred to either by name (`Gamma1` for example) or by label (`C` here) or even by its internal number here 1 for the first half circle and 2 for the second (more examples are in Section 5.8).

2nd line: the triangulation $\mathcal{T}_h$ of Ω is automatically generated by `buildmesh(C(50))` using 50 points on `C` as in Fig. 2.1.
The domain is assumed to be on the left side of the boundary which is implicitly oriented by the parametrization. So an elliptic hole can be added by

```
border C(t=2*pi,0){x=0.1+0.3*cos(t); y=0.5*sin(t);}
```

If by mistake one had written

```
border C(t=0,2*pi){x=0.1+0.3*cos(t); y=0.5*sin(t);}
```

then the inside of the ellipse would be triangulated as well as the outside.
Automatic mesh generation is based on the Delaunay-Voronoi algorithm. Refinement of the mesh are done by increasing the number of points on Γ, for example, `buildmesh(C(100))`, because inner vertices are determined by the density of points on the boundary. Mesh adaptation can be performed also against a given function `f` by calling `adaptmesh(Th,f)`.
Now the name $\mathcal{T}_h$ (`Th` in `FreeFem++`) refers to the family $\{T_k\}_{k=1,\cdots,n_t}$ of triangles shown in figure 2.1. Traditionally h refers to the mesh size, n_t to the number of triangles in $\mathcal{T}_h$ and n_v to the number of vertices, but it is seldom that we will have to use them explicitly. If Ω is not a polygonal domain, a "skin" remains between the exact domain Ω and its approximation $\Omega_h = \cup_{k=1}^{n_t} T_k$. However, we notice that all corners of $\Gamma_h = \partial\Omega_h$ are on Γ.

3rd line: A finite element space is, usually, a space of polynomial functions on elements, triangles here only, with certain matching properties at edges, vertices etc. Here `fespace Vh(Th,P1)` defines V_h to be the space of continuous functions which are affine in x, y on each triangle of T_h. As it is a linear vector space of finite dimension, basis can be found. The canonical basis is made of functions, called the *hat functions* ϕ_k which are continuous piecewise affine and are equal to 1 on one vertex and 0 on all others. A typical hat function is shown on figure 2.4 [1]. Then

$$V_h(\mathcal{T}_h, P_1) = \left\{ w(x,y) \;\middle|\; w(x,y) = \sum_{k=1}^{M} w_k \phi_k(x,y),\ w_k \text{ are real numbers} \right\} \tag{2.3}$$

where M is the dimension of V_h, i.e. the number of vertices. The w_k are called the *degree of freedom* of w and M the number of the degree of freedom. It is said also that the *nodes* of this finite element method are the vertices.
Currently `FreeFem++` implements the following elements , (see section 6 for the full description)
`P0` piecewise constant,
`P1` continuous piecewise linear,
`P2` continuous piecewise quadratic,
`RT0` Raviart-Thomas piecewise constant,
`P1nc` piecewise linear non-conforming,
`P1dc` piecewise linear discontinuous,
`P2dc` piecewise quadratic discontinuous,
`P1b` piecewise linear continuous plus bubble,

[1] The easiest way to define ϕ_k is by making use of the *barycentric coordinates* $\lambda_i(x,y)$, $i = 1,2,3$ of a point $q = (x,y) \in T$, defined by

$$\sum_i \lambda_i = 1, \quad \sum_i \lambda_i \boldsymbol{q}^i = \boldsymbol{q}$$

where q^i, $i = 1,2,3$ are the 3 vertices of T. Then it is easy to see that the restriction of ϕ_k on T is precisely λ_k.

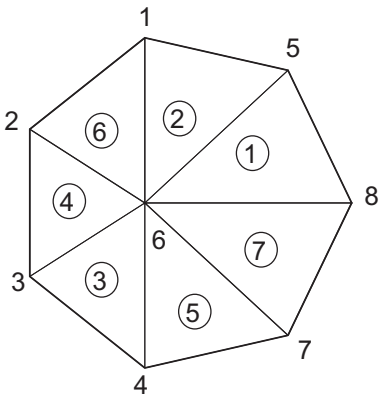


Figure 2.3: mesh `Th`

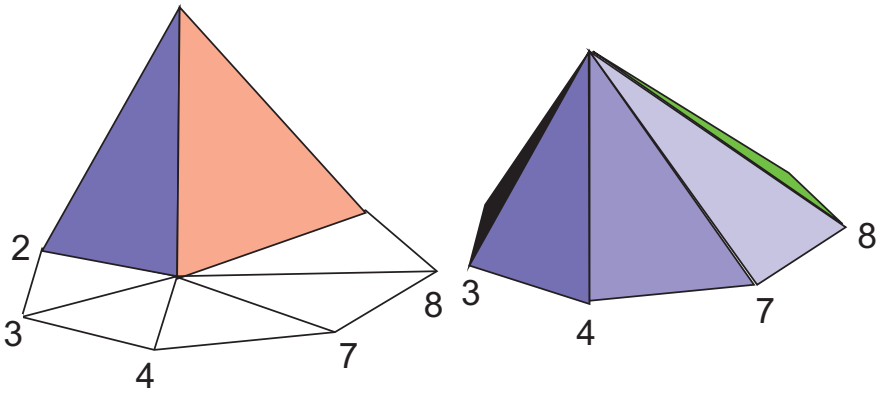


Figure 2.4: Graph of ϕ_1 (left hand side) and ϕ_6

`P2b` piecewise quadratic continuous plus bubble.
...
To get the full list do in a unix terminal, in directory `examples++-tutorial` do

```
FreeFem++ dumptable.edp
grep TypeOfFE lestables
```

The user can add other elements fairly easily if required.

Step3: Setting the problem
4th line: `Vh u,v` declares that u and v are approximated as above, namely

$$u(x,y) \simeq u_h(x,y) = \sum_{k=0}^{M-1} u_k \phi_k(x,y) \tag{2.4}$$

5th line: the right hand side `f` is defined analytically using the keyword **`func`**.
7th–9th lines: defines the bilinear form of equation (2.1) and its Dirichlet boundary conditions (2.2).
This *variational formulation* is derived by multiplying (2.1) by $v(x,y)$ and integrating the result over Ω:

$$-\int_\Omega v\Delta u \,\mathrm{d}x\mathrm{d}y = \int_\Omega vf \,\mathrm{d}x\mathrm{d}y$$

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

$$a(u,v) - \ell(f,v) = 0 \qquad \forall v \text{ satisfying } v = 0 \text{ on } \partial\Omega. \tag{2.5}$$

$$\text{with } a(u,v) = \int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x\mathrm{d}y, \quad \ell(f,v) = \int_\Omega fv \,\mathrm{d}x\mathrm{d}y \tag{2.6}$$

In `FreeFem++` the problem **Poisson** can be declared only (see below) for future use or declared and solved at the same time in which case

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

and (2.5) is written with `dx`(u) = $\partial u/\partial x$, `dy`(u) = $\partial u/\partial y$ and

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

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

In `FreeFem++` there is no need to distinguish the bilinear form a from the linear form ℓ, as long as the terms are inside different integrals, `FreeFem++` find out which one is the bilinear form by checking where both terms `u` and `v` are present.

The other way is to defined the problem and after we solve it, write :

```
Vh u,v; problem Poisson(u,v) =
  ...
Poisson;                                    //    the problem now is solve here
```

Step4: Solution and visualization

6th line: The current time in second is stored into the real-valued variable `cpu`.
7th line The problem is solved
11th line: The visualization is done as illustrated in Fig. 2.2 (see Section 7.1 for zoom, postscript and other commands).
12th line: The computing time (not counting graphics) is written on the console Notice the C++-like syntax; the user needs not study C++ for using `FreeFem++` , but it helps to guess what is allowed in the language.

Access to matrices and vectors
Internally `FreeFem++` will solve a linear system of the type

$$\sum_{j=0}^{M-1} A_{ij} u_j - F_i = 0, \quad i = 0, \cdots, M-1; \qquad F_i = \int_\Omega f \phi_i \, \mathrm{d}x\mathrm{d}y \tag{2.7}$$

which is found by using (2.4) and replacing v by ϕ_i in (2.5). And the Dirichlet conditions are implemented by penalty namely by setting $A_{ii} = 10^{30}$ and $F_i = 10^{30} * 0$ if i is a boundary degree of freedom. Note, that the number 10^{30} is called `tgv` (*très grande valeur*) and it is generally possible to change this value , see the index item `solve!tgv=`.

The matrix $A = (A_{ij})$ is called *stiffness matrix* .
If the user wants to access A directly he can do so by using

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

The vector F in (2.7) can also be constructed manually

```
varf l(unused,v) = int2d(Th)(f*v)+on(C,u=0);
Vh F;  F[] = l(0,V);       //    F[] is the vector associated to the function F
```

The problem can then be solved at the level of algebra by

```
u[]=A^-1*F[];              //    u[] is the vector associated to the function u
```

Note 2.1 *Here* `u` *and* `F` *are finite element function, and* `u[]` *and* `F[]` *give the array of value associated (* `u[]` $\equiv (u_i)_{i=0,\dots,M-1}$ *and* `F[]` $\equiv (F_i)_{i=0,\dots,M-1}$*). So we have*

$$\mathtt{u}(x,y) = \sum_{i=0}^{M-1} \mathtt{u[][}i\mathtt{]}\phi_i(x,y), \qquad \mathtt{F}(x,y) = \sum_{i=0}^{M-1} \mathtt{F[][}i\mathtt{]}\phi_i(x,y)$$

where $\phi_i, i = 0\dots, , M-1$ *are the basis functions of* `Vh` *like in equation (2.3), and* $M =$ `Vh.ndof` *is the number of degree of freedom (i.e. the dimension of the space* `Vh`*).*

The linear system (2.7) is solved by `UMFPACK` unless another option is mentioned specifically as in

```
Vh u,v; problem Poisson(u,v,solver=CG) = int2d(...
```

meaning that `Poisson` is declared only here and when it is called (by simply writing `Poisson;`) then (2.7) will be solved by the Conjugate Gradient method.

2.0.2 Some Features of `FreeFem++`

The language of `FreeFem++` is typed, polymorphic and reentrant with macro generation (see 9.12). Every variable must be typed and declared in a statement each statement separated from the next by a semicolon ";". The syntax is that of C++ by default augmented with something that is more akin to TEX. For the specialist, one key guideline is that `FreeFem++` rarely generates an internal finite element array; this was adopted for speed and consequently `FreeFem++` could be hard to beat in terms of execution speed, except for the time lost in the interpretation of the language (which can be reduced by a systematic usage of `varf` and matrices instead of `problem`.

2.1 The Development Cycle: Edit–Run/Visualize–Revise

An integrated environment is provided with `FreeFem++` written by A. Le Hyaric; Many examples and tutorials are also given along with this documentation and it is best to study them and learn by example. Explanations for some of these examples are given in this book in the next chapter. If you are a beginner of FEM, you may want to read also a book on variational formulations.
The development cycle will have the following steps:

Modeling: From strong forms of PDE to weak form, one must know the variational formulations to use `FreeFem++` ; one should also have an eye on the reusability of the variational formulation so as to keep the same internal matrices; a typical example is the time dependent heat equation with an implicit time scheme: the internal matrix can be factorized only once and `FreeFem++` can be taught to do so.

Programming: Write the code in `FreeFem++` language using a text editor such as the one provided in the integrated environment.

Run: Run the code (e.g. written in file mycode.edp). If not from the integrated environment it can be done at the console level by

```
% FreeFem++ mycode.edp
```

Note, the name of the command `FreeFem++` can be depend of your installation.

Visualization: Use the keyword `plot` to display functions while `FreeFem++` is running. Use the plot-parameter `wait=1` to stop the program to have time to see the plot. Use the plot-parameter `ps="toto.eps"` to generate a postscript file to archive the results.

Debugging: A global variable "debug" (for example) can help as in `wait=true` to `wait=false`.

```
bool debug = true;
mesh Th = square(10,10,[-1+2*x,-1+2*y]);          //    ]-1,1[^2 with 10x10 mesh
plot(Th,wait=debug);                          //    plot Th then needs a mouse click
fespace Vh(Th,P2);
Vh f = sin(pi*x)*cos(pi*y);
plot(f,wait=debug);                                  //    plot the function f
Vh g = sin(pi*x + cos(pi*y));
plot(g,wait=debug);                                  //    plot the function g
```

Changing debug to false will make the plots flow continuously; drinking coffee and watching the flow of graph on the screen can then become a pleasant experience.

Error messages are displayed in the console window. They are not always very explicit because of the template structure of the C++ code, sorry! Nevertheless they are displayed at the right place. For example, if you forget parenthesis as in

```
bool debug = true;
mesh Th = square(10,10;
plot(Th);
```

then you will get the following message from `FreeFem++`,

```
    2 : mesh Th = square(10,10;
 Error line number 2, in file bb.edp, before  token ;
parse error
  current line = 2
Compile error : parse error
        line number :2, ;
error Compile error : parse error
        line number :2, ;
 code = 1
```

If you use the same symbol twice as in

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

then you will get the message

```
2 : real aaa; The identifier aaa exist
      the existing type is <Pd>
      the new  type is <Pd>
```

If you find that the program isn't doing what you want you may also use **cout** to display in text format on the console window the value of variables.

The following example works:

```
...;
fespace Vh...; Vh u;...
cout<<u;...
matrix A=a(Vh,Vh);...
cout<<A;
```

Another trick is to *comment in and out* by using the" // " as in C++. For example

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

Chapter 3

Learning by Examples

This chapter is for those, like us, who don't like manuals. A number of simple examples cover a good deal of the capacity of `FreeFem++` and are self-explanatory. For the modelling part this chapter continues at Chapter 9 where some PDEes of physics, engineering and finance are studied in greater depth.

3.1 Membranes

Summary *Here we shall learn how to solve a Dirichlet and/or mixed Dirichlet Neumann problem for the Laplace operator with application to the equilibrium of a membrane under load. We shall also check the accuracy of the method and interface with other graphics packages.*

An elastic membrane Ω is to a planar rigid support Γ, but a force $f(x)dx$ is exerted on each surface element $\mathrm{d}x = \mathrm{d}x_1\mathrm{d}x_2$. The vertical membrane displacement, $\varphi(x)$, is obtained by solving Laplace's equation:

$$-\Delta\varphi = f \text{ in } \Omega.$$

As the membrane is fixed to its planar support, one has:

$$\varphi|_\Gamma = 0.$$

If the support wasn't planar but at an elevation $z(x_1, x_2)$ then the boundary conditions would be of non-homogeneous Dirichlet type.

$$\varphi|_\Gamma = z.$$

If a part Γ_2 of the membrane border Γ is not fixed to the support but is left hanging then due to membrane's rigidity the angle with the normal vector n is zero; thus the boundary conditions are

$$\varphi|_{\Gamma_1} = z, \quad \frac{\partial\varphi}{\partial n}|_{\Gamma_2} = 0$$

where $\Gamma_1 = \Gamma - \Gamma_2$; recall that $\frac{\partial\varphi}{\partial n} = \nabla\varphi \cdot n$. Let us recall also that the Laplace operator Δ is defined by:

$$\Delta\varphi = \frac{\partial^2\varphi}{\partial x_1^2} + \frac{\partial^2\varphi}{\partial x_2^2}.$$

With such "mixed boundary conditions" the problem has a unique solution (see (1987), Dautray-Lions (1988), Strang (1986) and Raviart-Thomas (1983)); the easiest proof is to notice that φ is the state of least energy, i.e.

$$E(\phi) = \min_{\varphi - z \in V} E(v), \quad \text{with} \quad E(v) = \int_\Omega (\frac{1}{2}|\nabla v|^2 - fv)$$

and where V is the subspace of the Sobolev space $H^1(\Omega)$ of functions which have zero trace on Γ_1. Recall that ($x \in \mathbb{R}^d$, $d = 2$ here)

$$H^1(\Omega) = \{u \in L^2(\Omega) \ : \ \nabla u \in (L^2(\Omega))^d\}$$

Calculus of variation shows that the minimum must satisfy, what is known as the weak form of the PDE or its variational formulation (also known here as the theorem of virtual work)

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

Next an integration by parts (Green's formula) will show that this is equivalent to the PDE when second derivatives exist.

WARNING Unlike `freefem+` which had both weak and strong forms, `FreeFem++` implements only weak formulations. It is not possible to go further in using this software if you don't know the weak form (i.e. variational formulation) of your problem: either you read a book, or ask help form a colleague or drop the matter. Now if you want to solve a system of PDE like $A(u, v) = 0$, $B(u, v) = 0$ don't close this manual, because in weak form it is

$$\int_\Omega (A(u, v)w_1 + B(u, v)w_2) = 0 \ \ \forall w_1, w_2...$$

Example Let an ellipse have the length of the se $a = 2$, and unitary the semiminor axis Let the surface force be $f = 1$. Programming this case with `FreeFem++` gives:

Example 3.1 (membrane.edp)

```
//    file membrane.edp
real theta=4.*pi/3.;
real a=2.,b=1.;      //    the length of the semimajor axis and semiminor axis
func z=x;

border Gamma1(t=0,theta)    { x = a * cos(t); y = b*sin(t); }
border Gamma2(t=theta,2*pi) { x = a * cos(t); y = b*sin(t); }
mesh Th=buildmesh(Gamma1(100)+Gamma2(50));

fespace Vh(Th,P2);                          //    P2 conforming triangular FEM
Vh phi,w, f=1;

solve Laplace(phi,w)=int2d(Th)(dx(phi)*dx(w) + dy(phi)*dy(w))
                - int2d(Th)(f*w) + on(Gamma1,phi=z);
plot(phi,wait=true, ps="membrane.eps");                          //   Plot phi
plot(Th,wait=true, ps="membraneTh.eps");                          //   Plot Th

savemesh(Th,"Th.msh");
```

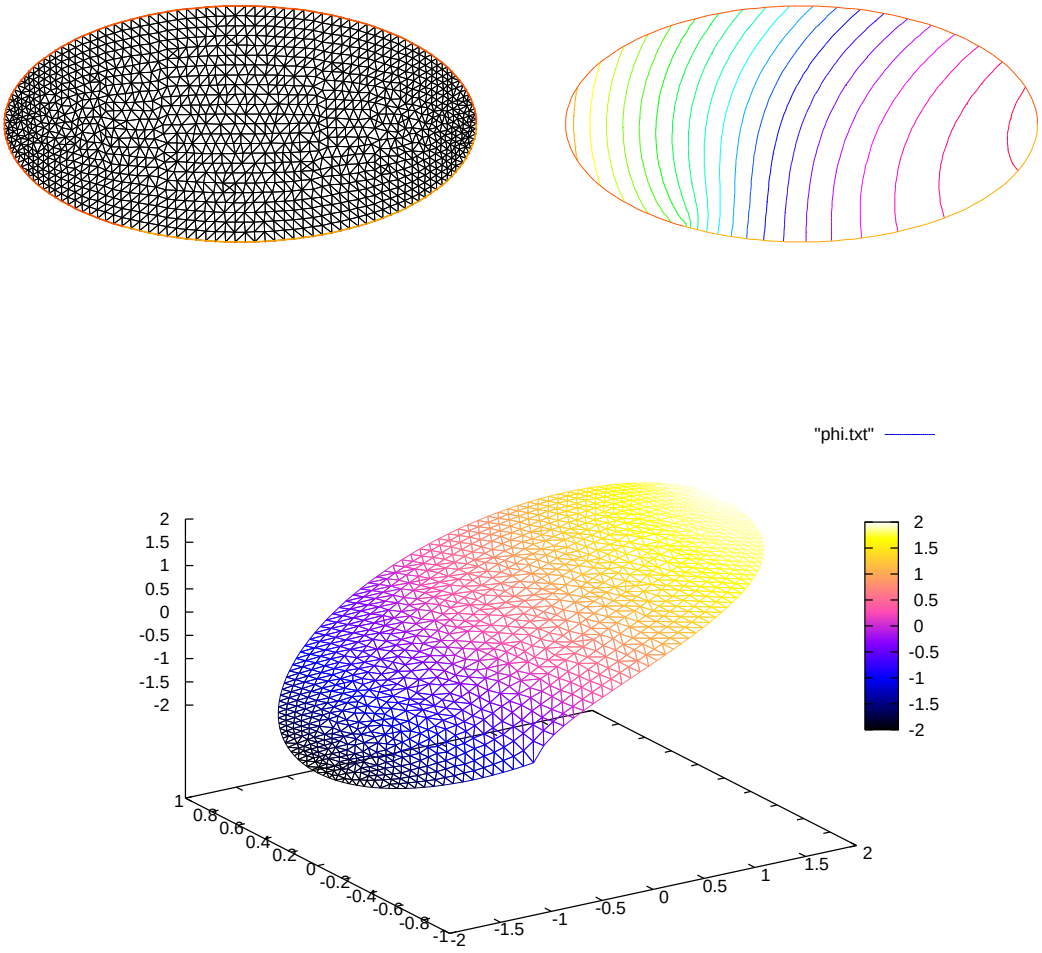


Figure 3.1: Mesh and level lines of the membrane deformation. Below: the same in 3D drawn by `gnuplot` from a file generated by `FreeFem++` .

A triangulation is built by the keyword `buildmesh`. The keyword calls a triangulation subroutine based on the Delaunay test, which first triangulates with only the boundary points, then adds internal points by subdividing the edges. How fine the triangulation becomes is controlled by the size of the closest boundary edges.

The PDE is then discretized using the triangular second order finite element method on the triangulation; as was briefly indicated in the previous chapter, a linear system is derived from the discrete formulation whose size is the number of vertices plus number mid-edges in the triangulation. The system is solved by a multi-frontal Gauss LU factorization implemented in the package `UMFPACK`. The keyword plot will display both $\mathbb{T}_h$ and φ (remove `Th` if φ only is desired) and the qualifier `fill=true` replaces the default option (colored level lines) by a full color display. Results are on figure 3.1.

```
plot(phi,wait=true,fill=true);              //   Plot phi with full color display
```

Next we would like to check the results!
One simple way is to adjust the parameters so as to know the solutions. For instance on the unit circle `a=1` , $\varphi_e = \sin(x^2 + y^2 - 1)$ solves the problem when

$$z = 0,\ f = -4(\cos(x^2 + y^2 - 1) - (x^2 + y^2)\sin(x^2 + y^2 - 1))$$

except that on Γ_2 $\partial_n\varphi = 2$ instead of zero. So we will consider a non-homogeneous Neumann condition and solve

$$\int_\Omega (\nabla\varphi \cdot \nabla w = \int_\Omega fw + \int_{\Gamma_2} 2w \quad \forall w \in V$$

We will do that with two triangulations, compute the L^2 error:

$$\epsilon = \int_\Omega |\varphi - \varphi_e|^2$$

and print the error in both cases as well as the log of their ratio an indication of the rate of convergence.

Example 3.2 (membranerror.edp)

```
//    file membranerror.edp
verbosity =0;                                  //    to remove all default output
real theta=4.*pi/3.;
real a=1.,b=1.;       //    the length of the semimajor axis and semiminor axis
border Gamma1(t=0,theta)     { x = a * cos(t); y = b*sin(t); }
border Gamma2(t=theta,2*pi) { x = a * cos(t); y = b*sin(t); }

func f=-4*(cos(x^2+y^2-1) -(x^2+y^2)*sin(x^2+y^2-1));
func phiexact=sin(x^2+y^2-1);

real[int] L2error(2);                                  //    an array two values
for(int n=0;n<2;n++)
{
  mesh Th=buildmesh(Gamma1(20*(n+1))+Gamma2(10*(n+1)));
  fespace Vh(Th,P1);
   Vh phi,w;

  solve laplace(phi,w)=int2d(Th)(dx(phi)*dx(w) + dy(phi)*dy(w))
    - int2d(Th)(f*w) - int1d(Th,Gamma2)(2*w)+ on(Gamma1,phi=0);
```

```
  plot(Th,phi,wait=true,ps="membrane.eps");                    //   Plot Th and phi

  L2error[n]= sqrt(int2d(Th)((phi-phiexact)^2));
}

for(int n=0;n<2;n++)
 cout << " L2error " << n << " = "<<  L2error[n] <<endl;

cout <<" convergence rate = "<< log(L2error[0]/L2error[1])/log(2.)  <<endl;
```

the output is

```
 L2error 0 = 0.00699498
 L2error 1 = 0.00182818
 convergence rate = 1.93591
times: compile 0.02s, execution 6.94s
```

We find a rate of 1.93591, which is not close enough to the 3 predicted by the theory. The Geometry is always a polygon so we lose one order due to the geometry approximation in $O(h^2)$

Now if you are not satisfied with the `.eps` plot generated by `FreeFem++` and you want to use other graphic facilities. Then you must store the solution in a file very much like in `C++`. It will be useless if you don't save the triangulation as well, consequently you must do

```
     {
       ofstream ff("phi.txt");
       ff << phi[];
      }
    savemesh(Th,"Th.msh");
```

For the triangulation the name is important: it is the extension that determines the format.
Still that may not take you where you want. Here is an interface with gnuplot to produce the right part of figure 3.2.

```
                                                    //    to build a gnuplot data file
{ ofstream ff("graph.txt");
   for (int i=0;i<Th.nt;i++)
   { for (int j=0; j <3; j++)
       ff<<Th[i][j].x  << "    "<< Th[i][j].y<< "  "<<phi[][Vh(i,j)]<<endl;
    ff<<Th[i][0].x  << "    "<< Th[i][0].y<< "  "<<phi[][Vh(i,0)]<<"\n\n\n"
   }
}
```

We use the finite element numbering, where `Wh(i,j)` is the global index of j^{Th} degrees of freedom of triangle number i.
Then open `gnuplot` and do

```
        set palette rgbformulae 30,31,32
        splot "graph.txt" w l pal
```

This works with `P2` and `P1`, but not with `P1nc` because the 3 first degrees of freedom of `P2` or `P2` are on vertices and not with `P1nc`.

3.2 Heat Exchanger

Summary *Here we shall learn more about geometry input and triangulation files, as well as read and write operations.*

The problem Let $\{C_i\}_{1,2}$, be 2 thermal conductors within an enclosure C_0. The first one is held at a constant temperature u_1 the other one has a given thermal conductivity κ_2 5 times larger than the one of C_0. We assume that the border of enclosure C_0 is held at temperature $20°C$ and that we have waited long enough for thermal equilibrium.
In order to know $u(x)$ at any point x of the domain Ω, we must solve

$$\nabla \cdot (\kappa \nabla u) = 0 \quad \text{in } \Omega, \qquad u_{|\Gamma} = g$$

where Ω is the interior of C_0 minus the conductors C_1 and Γ is the boundary of Ω, that is $C_0 \cup C_1$ Here g is any function of x equal to u_i on C_i. The second equation is a reduced form for:

$$u = u_i \text{ on } \quad C_i, \quad i = 0, 1.$$

The variational formulation for this problem is in the subspace $H_0^1(\Omega) \subset H^1(\Omega)$ of functions which have zero traces on Γ.

$$u - g \in H_0^1(\Omega) \ : \ \int_\Omega \nabla u \nabla v = 0 \quad \forall v \in H_0^1(\Omega)$$

Let us assume that C_0 is a circle of radius 5 centered at the origin, C_i are rectangles, C_1 being at the constant is at temperature $u_1 = 60°C$.

Example 3.3 (heatex.edp)

```
//    file heatex.edp
int C1=99, C2=98;              //    could be anything such that ≠0 and C1≠C2
border C0(t=0,2*pi){x=5*cos(t); y=5*sin(t);}

border C11(t=0,1){ x=1+t;  y=3;      label=C1;}
border C12(t=0,1){ x=2;    y=3-6*t;  label=C1;}
border C13(t=0,1){ x=2-t;  y=-3;     label=C1;}
border C14(t=0,1){ x=1;    y=-3+6*t; label=C1;}

border C21(t=0,1){ x=-2+t; y=3;      label=C2;}
border C22(t=0,1){ x=-1;   y=3-6*t;  label=C2;}
border C23(t=0,1){ x=-1-t; y=-3;     label=C2;}
border C24(t=0,1){ x=-2;   y=-3+6*t; label=C2;}

plot(    C0(50)                              //    to see the border of the domain
        + C11(5)+C12(20)+C13(5)+C14(20)
        + C21(-5)+C22(-20)+C23(-5)+C24(-20),
        wait=true, ps="heatexb.eps");

mesh Th=buildmesh(    C0(50)
                    + C11(5)+C12(20)+C13(5)+C14(20)
                    + C21(-5)+C22(-20)+C23(-5)+C24(-20));
plot(Th,wait=1);

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

```
Vh kappa=1+2*(x<-1)*(x>-2)*(y<3)*(y>-3);
solve a(u,v)= int2d(Th)(kappa*(dx(u)*dx(v)+dy(u)*dy(v)))
                +on(C0,u=20)+on(C1,u=60);
plot(u,wait=true, value=true, fill=true, ps="heatex.eps");
```

Note the following:

- `C0` is oriented counterclockwise by t, while `C1` is oriented clockwise and `C2` is oriented counterclockwise. This is why `C1` is viewed as a hole by `buildmesh`.
- `C1` and `C2` are built by joining pieces of straight lines. To group them in the same logical unit to input the boundary conditions in a readable way we assigned a label on the boundaries. As said earlier borders have an internal number corresponding to their order in the program (check it by adding a `cout<<C22;` above). This is essential to understand how a mesh can be output to a file and re-read (see below).
- As usual the mesh density is controlled by the number of vertices assigned to each boundary. It is not possible to change the (uniform) distribution of vertices but a piece of boundary can always be cut in two or more parts, for instance `C12` could be replaced by `C121+C122`:

```
                                //    border C12(t=0,1) x=2; y=3-6*t; label=C1;
border C121(t=0,0.7){ x=2;    y=3-6*t;  label=C1;}
border C122(t=0.7,1){ x=2;    y=3-6*t;  label=C1;}
... buildmesh(.../* C12(20) */ + C121(12)+C122(8)+...);
```

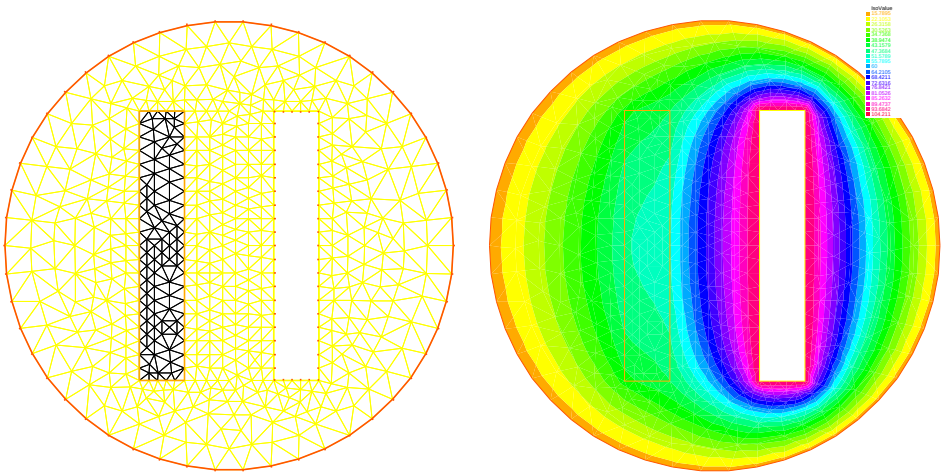


Figure 3.2: The heat exchanger

Exercise Use the symmetry of the problem with respect to the axes; triangulate only one half of the domain, and set Dirichlet conditions on the vertical axis, and Neumann conditions on the horizontal axis.

Writing and reading triangulation files Suppose that at the end of the previous program we added the line

```
savemesh(Th,"condensor.msh");
```

and then later on we write a similar program but we wish to read the mesh from that file. Then this is how the condenser should be computed:

```
mesh Sh=readmesh("condensor.msh");
fespace Wh(Sh,P1); Wh us,vs;
solve b(us,vs)= int2d(Sh)(dx(us)*dx(vs)+dy(us)*dy(vs))
                +on(1,us=0)+on(99,us=1)+on(98,us=-1);
plot(us);
```

Note that the names of the boundaries are lost but either their internal number (in the case of `C0`) or their label number (for `C1` and `C2`) are kept.

3.3 Acoustics

Summary *Here we go to grip with ill posed problems and eigenvalue problems*
Pressure variations in air at rest are governed by the wave equation:

$$\frac{\partial^2 u}{\partial t^2} - c^2 \Delta u = 0.$$

When the solution wave is monochromatic (and that depend on the boundary and initial conditions), u is of the form $u(x,t) = Re(v(x)e^{ikt})$ where v is a solution of Helmholtz's equation:

$$\begin{aligned} &k^2 v + c^2 \Delta v = 0 \ \text{ in } \ \Omega, \\ &\frac{\partial v}{\partial n}|_\Gamma = g. \end{aligned} \tag{3.1}$$

where g is the source. Note the "+" sign in front of the Laplace operator and that $k > 0$ is real. This sign may make the problem ill posed for some values of $\frac{c}{k}$, a phenomenon called "resonance".
At resonance there are non-zero solutions even when $g = 0$. So the following program may or may not work:

Example 3.4 (sound.edp)

```
//    file sound.edp
real kc2=1;
func g=y*(1-y);

border a0(t=0,1) { x= 5; y= 1+2*t ;}
border a1(t=0,1) { x=5-2*t; y= 3 ;}
border a2(t=0,1) { x= 3-2*t; y=3-2*t ;}
border a3(t=0,1) { x= 1-t; y= 1 ;}
border a4(t=0,1) { x= 0; y= 1-t ;}
border a5(t=0,1) { x= t; y= 0  ;}
border a6(t=0,1) { x= 1+4*t; y= t ;}

mesh Th=buildmesh( a0(20) + a1(20) + a2(20)
        + a3(20) + a4(20) + a5(20) + a6(20));
fespace Vh(Th,P1);
```

```
Vh u,v;

solve sound(u,v)=int2d(Th)(u*v * kc2 - dx(u)*dx(v) - dy(u)*dy(v))
                  - int1d(Th,a4)(g*v);
plot(u, wait=1, ps="sound.eps");
```

Results are on Figure 3.3. But when $kc2$ is an eigenvalue of the problem, then the solution is not unique: if $u_e \not\equiv 0$ is an eigen state, then for any given solution $u + u_e$ is another a solution. To find all the u_e one can do the following

```
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 );
varf b([u1],[u2]) = int2d(Th)( u1*u2 ) ;    //   no Boundary condition see note
9.1

matrix OP= op(Vh,Vh,solver=Crout,factorize=1);
matrix B= b(Vh,Vh,solver=CG,eps=1e-20);

int nev=2;                        //    number of requested eigenvalues near 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);
cout<<ev(0)<<" 2 eigen values "<<ev(1)<<endl;
v=eV[0];
plot(v,wait=1,ps="eigen.eps");
```

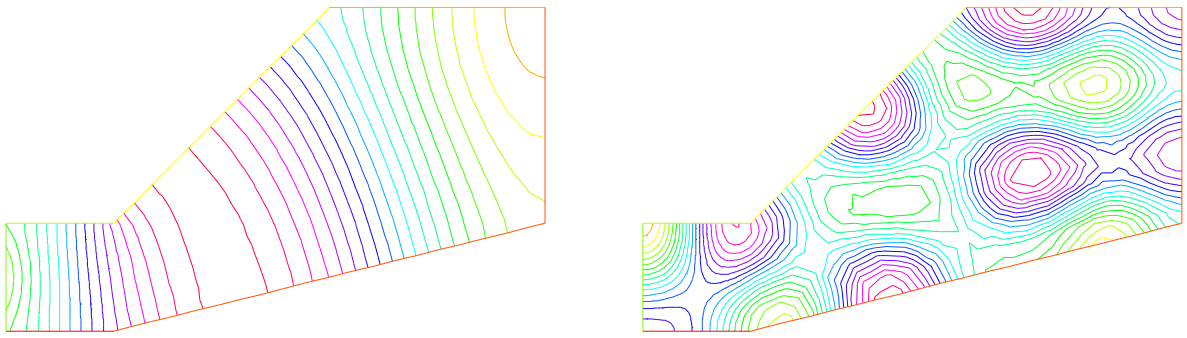


Figure 3.3: Left:Amplitude of an acoustic signal coming from the left vertical wall. Right: first eigen state ($\lambda = (k/c)^2 = 19.4256$) close to 20 of eigenvalue problem :$-\Delta\varphi = \lambda\varphi$ and $\frac{\partial\varphi}{\partial n} = 0$ on Γ

3.4 Thermal Conduction

Summary *Here we shall learn how to deal with a time dependent parabolic problem. We shall also show how to treat an axisymmetric problem and show also how to deal with a nonlinear problem.*

How air cools a plate We seek the temperature distribution in a plate $(0, Lx) \times (0, Ly) \times (0, Lz)$ of rectangular cross section $\Omega = (0, 6) \times (0, 1)$; the plate is surrounded by air at temperature u_e and initially at temperature $u = u_0 + \frac{x}{L}u_1$. In the plane perpendicular to the plate at $z = Lz/2$, the temperature varies little with the coordinate z; as a first approximation the problem is 2D.

We must solve the temperature equation in Ω in a time interval (0,T).

$$\begin{aligned} &\partial_t u - \nabla \cdot (\kappa \nabla u) = 0 \text{ in } \Omega \times (0, T), \\ &u(x, y, 0) = u_0 + x u_1 \\ &\kappa \frac{\partial u}{\partial n} + \alpha(u - u_e) = 0 \text{ on } \Gamma \times (0, T). \end{aligned} \tag{3.2}$$

Here the diffusion κ will take two values, one below the middle horizontal line and ten times less above, so as to simulate a thermostat. The term $\alpha(u - u_e)$ accounts for the loss of temperature by convection in air. Mathematically this boundary condition is of Fourier (or Robin, or mixed) type.

The variational formulation is in $L^2(0, T; H^1(\Omega))$; in loose terms and after applying an implicit Euler finite difference approximation in time; we shall seek $u^n(x, y)$ satisfying for all $w \in H^1(\Omega)$:

$$\int_\Omega \left(\frac{u^n - u^{n-1}}{\delta t} w + \kappa \nabla u^n \nabla w\right) + \int_\Gamma \alpha(u^n - u_u e) w = 0$$

```
func u0 =10+90*x/6;
func k = 1.8*(y<0.5)+0.2;
real ue = 25, alpha=0.25, T=5, dt=0.1 ;

mesh Th=square(30,5,[6*x,y]);
fespace Vh(Th,P1);
Vh u=u0,v,uold;

problem thermic(u,v)= int2d(Th)(u*v/dt + k*(dx(u) * dx(v) + dy(u) * dy(v)))
                + int1d(Th,1,3)(alpha*u*v)
                - int1d(Th,1,3)(alpha*ue*v)
                - int2d(Th)(uold*v/dt) + on(2,4,u=u0);
ofstream ff("thermic.dat");
for(real t=0;t<T;t+=dt){
    uold=u;     //    uold ≡ u^{n-1} = u^n ≡u
    thermic;  //    here solve the thermic problem
    ff<<u(3,0.5)<<endl;
    plot(u);
}
```

Notice that we must separate by hand the bilinear part from the linear one.

Notice also that the way we store the temperature at point (3,0.5) for all times in file `thermic.dat`. Should a one dimensional plot be required, the same procedure can be used. For instance to print $x \mapsto \frac{\partial u}{\partial y}(x, 0.9)$ one would do

```
for(int i=0;i<20;i++) cout<<dy(u)(6.0*i/20.0,0.9)<<endl;
```

Results are shown on Figure 3.4.

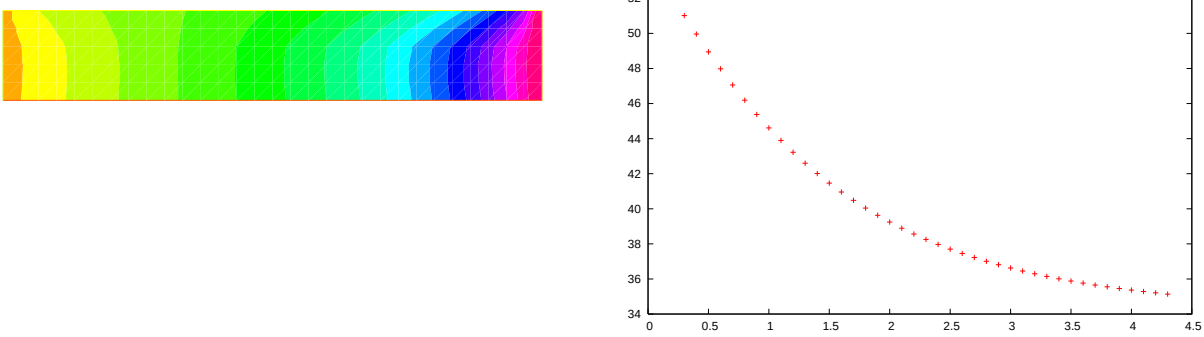


Figure 3.4: Temperature at T=4.9. Right: decay of temperature versus time at x=3, y=0.5

3.4.1 Axisymmetry: 3D Rod with circular section

Let us now deal with a cylindrical rod instead of a flat plate. For simplicity we take $\kappa = 1$. In cylindrical coordinates, the Laplace operator becomes (r is the distance to the axis, z is the distance along the axis, θ polar angle in a fixed plane perpendicular to the axis):

$$\Delta u = \frac{1}{r}\partial_r(r\partial_r u) + \frac{1}{r^2}\partial^2_{\theta\theta} u + \partial^2_{zz}.$$

Symmetry implies that we loose the dependence with respect to θ; so the domain Ω is again a rectangle $]0, R[\times]0, |[$. We take the convention of numbering of the edges as in `square()` (1 for the bottom horizontal ...); the problem is now:

$$\begin{aligned} &r\partial_t u - \partial_r(r\partial_r u) - \partial_z(r\partial_z u) = 0 \text{ in } \Omega, \\ &u(t=0) = u_0 + \frac{z}{L_z}(u_1 - u_) \\ &u|_{\Gamma_4} = u_0, \quad u|_{\Gamma_2} = u_1, \quad \alpha(u - u_e) + \frac{\partial u}{\partial n}|_{\Gamma_1 \cup \Gamma_3} = 0. \end{aligned} \tag{3.3}$$

Note that the PDE has been multiplied by r.

After discretization in time with an implicit scheme, with time steps `dt`, in the `FreeFem++` syntax r becomes x and z becomes y and the problem is:

```
problem thermaxi(u,v)=int2d(Th)((u*v/dt + dx(u)*dx(v) + dy(u)*dy(v))*x)
                + int1d(Th,3)(alpha*x*u*v) - int1d(Th,3)(alpha*x*ue*v)
                - int2d(Th)(uold*v*x/dt) + on(2,4,u=u0);
```

Notice that the bilinear form degenerates at $x = 0$. Still one can prove existence and uniqueness for u and because of this degeneracy no boundary conditions need to be imposed on Γ_1.

3.4.2 A Nonlinear Problem : Radiation

Heat loss through radiation is a loss proportional to the absolute temperature to the fourth power (Stefan's Law). This adds to the loss by convection and gives the following boundary condition:

$$\kappa\frac{\partial u}{\partial n} + \alpha(u - u_e) + c[(u + 273)^4 - (u_e + 273)^4] = 0$$

The problem is nonlinear, and must be solved iteratively. If m denotes the iteration index, a semi-linearization of the radiation condition gives

$$\frac{\partial u^{m+1}}{\partial n} + \alpha(u^{m+1} - u_e) + c(u^{m+1} - u_e)(u^m + u_e + 546)((u^m + 273)^2 + (u_e + 273)^2) = 0,$$

because we have the identity $a^4 - b^4 = (a - b)(a + b)(a^2 + b^2)$. The iterative process will work with $v = u - u_e$.

```
...
fespace Vh(Th,P1);                                    //    finite element space
real rad=1e-8, uek=ue+273;                          //    def of the physical constants
Vh vold,w,v=u0-ue,b;
problem thermradia(v,w)
    = int2d(Th)(v*w/dt + k*(dx(v) * dx(w) + dy(v) * dy(w)))
                + int1d(Th,1,3)(b*v*w)
                - int2d(Th)(vold*w/dt) + on(2,4,v=u0-ue);

for(real t=0;t<T;t+=dt){
    vold=v;
    for(int m=0;m<5;m++){
       b= alpha + rad * (v + 2*uek) * ((v+uek)^2 + uek^2);
       thermradia;
    }
}
vold=v+ue; plot(vold);
```

3.5 Irrotational Fan Blade Flow and Thermal effects

Summary *Here we will learn how to deal with a multi-physics system of PDEs on a Complex geometry, with multiple meshes within one problem. We also learn how to manipulate the region indicator and see how smooth is the projection operator from one mesh to another.*

Incompressible flow Without viscosity and vorticity incompressible flows have a velocity given by:

$$u = \begin{pmatrix} \frac{\partial \psi}{\partial x_2} \\ -\frac{\partial \psi}{\partial x_1} \end{pmatrix}, \quad \text{where } \psi \text{ is solution of } \quad \Delta \psi = 0$$

This equation expresses both incompressibility ($\nabla \cdot u = 0$) and absence of vortex ($\nabla \times u = 0$). As the fluid slips along the walls, normal velocity is zero, which means that ψ satisfies:

$$\psi \text{ constant on the walls.}$$

One can also prescribe the normal velocity at an artificial boundary, and this translates into non constant Dirichlet data for ψ.

Airfoil Let us consider a wing profile S in a uniform flow. Infinity will be represented by a large circle C where the flow is assumed to be of uniform velocity; one way to model this problem is to write

$$\Delta \psi = 0 \text{ in } \Omega, \qquad \psi|_S = 0, \quad \psi|_C = u_\infty y, \tag{3.4}$$

where $\partial\Omega = C \cup S$

The NACA0012 Airfoil An equation for the upper surface of a NACA0012 (this is a classical wing profile in aerodynamics) is:

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

Example 3.5 (potential.edp) `// file potential.edp`

```
real S=99;
border C(t=0,2*pi) {  x=5*cos(t);  y=5*sin(t);}
border Splus(t=0,1){  x = t; y = 0.17735*sqrt(t)-0.075597*t
        - 0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4); label=S;}
border Sminus(t=1,0){  x =t; y= -(0.17735*sqrt(t)-0.075597*t
        -0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4)); label=S;}
mesh Th= buildmesh(C(50)+Splus(70)+Sminus(70));
fespace Vh(Th,P2); Vh psi,w;

solve potential(psi,w)=int2d(Th)(dx(psi)*dx(w)+dy(psi)*dy(w))+
  on(C,psi = y) + on(S,psi=0);

plot(psi,wait=1);
```

A zoom of the streamlines are shown on Figure 3.5.

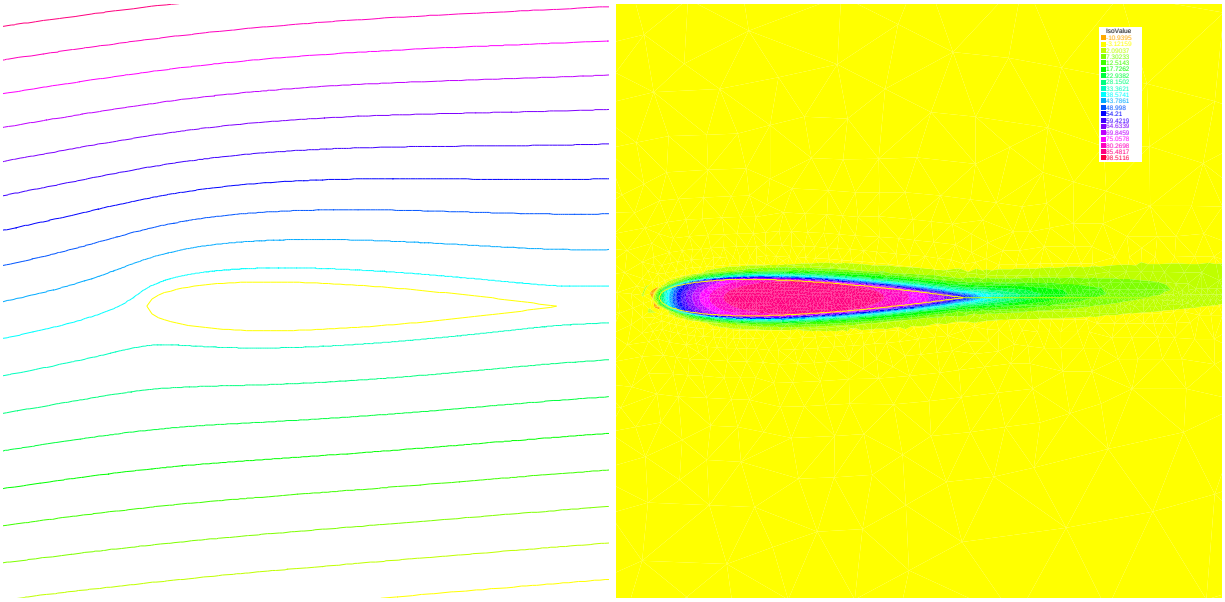


Figure 3.5: Zoom around the NACA0012 airfoil showing the streamlines (curve ψ = constant). To obtain such a plot use the interactive graphic command: "+" and p. Right: temperature distribution at time T=25 (now the maximum is at 90 instead of 120). Note that an incidence angle has been added here (see Chapter 9).

3.5.1 Heat Convection around the airfoil

Now let us assume that the airfoil is hot and that air is there to cool it. Much like in the previous section the heat equation for the temperature u is

$$\partial_t v - \nabla \cdot (\kappa \nabla v) + u \cdot \nabla v = 0, \quad v(t=0) = v_0, \quad \frac{\partial v}{\partial n}|_C = 0$$

But now the domain is outside AND inside S and κ takes a different value in air and in steel. Furthermore there is convection of heat by the flow, hence the term $u \cdot \nabla v$ above. Consider the following, to be plugged at the end of the previous program:

```
...
border D(t=0,2){x=1+t;y=0;}              //    added to have a fine mesh at trail
mesh Sh = buildmesh(C(25)+Splus(-90)+Sminus(-90)+D(200));
fespace Wh(Sh,P1); Wh v,vv;
int steel=Sh(0.5,0).region, air=Sh(-1,0).region;
fespace W0(Sh,P0);
W0 k=0.01*(region==air)+0.1*(region==steel);
W0 u1=dy(psi)*(region==air), u2=-dx(psi)*(region==air);
Wh vold = 120*(region==steel);
real dt=0.05, nbT=50;
int i;
problem thermic(v,vv,init=i,solver=LU)= int2d(Sh)(v*vv/dt
                 + k*(dx(v) * dx(vv) + dy(v) * dy(vv))
                 + 10*(u1*dx(v)+u2*dy(v))*vv)- int2d(Sh)(vold*vv/dt);
for(i=0;i<nbT;i++){
    v=vold; thermic;
```

```
 plot(v);
}
```

Notice here

- how steel and air are identified by the mesh parameter region which is defined when buildmesh is called and takes an integer value corresponding to each connected component of Ω;
- how the convection terms are added without upwinding. Upwinding is necessary when the Pecley number $|u|L/\kappa$ is large (here is a typical length scale), The factor 10 in front of the convection terms is a quick way of multiplying the velocity by 10 (else it is too slow to see something).
- The solver is Gauss' LU factorization and when `init`$\neq 0$ the LU decomposition is reused so it is much faster after the first iteration.

3.6 Pure Convection : The Rotating Hill

Summary *Here we will present two methods for upwinding for the simplest convection problem. We will learn about Characteristics-Galerkin and Discontinuous-Galerkin Finite Element Methods.*
Let Ω be the unit disk centered at 0; consider the rotation vector field

$$u_1 = y, \quad u_2 = -x.$$

Pure convection by u is

$$\partial_t c + u.\nabla c = 0 \ \text{ in } \ \Omega \times (0,T) \quad c(t=0) = c^0 \ \text{ in } \ \Omega.$$

The exact solution is $c(x,y,t) = c^0(X(0), Y(0))$ where $\boldsymbol{X} := (X,Y)$ are solutions of ($\boldsymbol{u} = (u_1, u_2)$

$$\frac{\mathrm{d}\boldsymbol{X}}{\mathrm{d}\tau} = u(\boldsymbol{X}, \tau), \quad \boldsymbol{X}(t) = (x,y).$$

The game consists in solving the equation until $T = 2\pi$, that is for a full revolution and to compare the final solution with the initial one; they should be equal.

Solution by a Characteristics-Galerkin Method In `FreeFem++` there is an operator called `convect([u1,u2],dt,c)` which solves exactly the equation for $\boldsymbol{X}^m := (X(t), Y(t))$ at $t = m\delta t$:

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{X}}\tau(X,Y) = u(\boldsymbol{X}, \tau), \quad \boldsymbol{X}(t - \mathrm{d}t) = (x,y).$$

When u is piecewise constant; this is possible because (X,Y) is then a polygonal curve which can be computed exactly and the solution exists always when u is divergence free; convect returns $c(\boldsymbol{X}^m, t - \mathrm{d}t)$.

Example 3.6 (convects.edp) `//  file convects.edp`

```
border C(t=0, 2*pi) { x=cos(t);  y=sin(t); };
mesh Th = buildmesh(C(100));
fespace Uh(Th,P1);
Uh cold, c = exp(-10*((x-0.3)^2 +(y-0.3)^2));

real dt = 0.17,t=0;
Uh u1 = y, u2 = -x;
for (int m=0; m<2*pi/dt ; m++) {
    t += dt;      cold=c;
    c=convect([u1,u2],-dt,cold);
    plot(c,cmm=" t="+t + ", min=" + c[].min + ", max=" +  c[].max);
}
```

The method is very powerful but has two limitations: a/ it is not conservative, b/ it may diverge in rare cases when $|u|$ is too small due to quadrature error.

Solution by Discontinuous-Galerkin FEM Discontinuous Galerkin methods take advantage of the discontinuities of c at the edges to build upwinding. There are may formulations possible. We shall implement here the so-called dual-P_1^{DC} formulation (see Ern[10]):

$$\int_\Omega (\frac{c^{n+1}-c^n}{\delta t} + u\cdot\nabla c)w + \int_E (\alpha|n\cdot u| - \frac{1}{2}n\cdot u)[c]w = \int_{E_\Gamma^-} |n\cdot u|cw \quad \forall w$$

where E is the set of inner edges and E_Γ^- is the set of boundary edges where $u\cdot n < 0$ (in our case there is no such edges). Finally $[c]$ is the jump of c across an edge with the convention that c^+ refers to the value on the right of the oriented edge.

Example 3.7 (convects_end.edp) `//  file convects.edp`

```
...
fespace Vh(Th,P1dc);

Vh w, ccold, v1 = y, v2 = -x, cc = exp(-10*((x-0.3)^2 +(y-0.3)^2));
real u, al=0.5;  dt = 0.05;

macro n(N.x*v1+N.y*v2)                          //  problem Adual(cc,w) =
int2d(Th)((cc/dt+(v1*dx(cc)+v2*dy(cc)))*w)
  + intalledges(Th)((1-nTonEdge)*w*(al*abs(n)-n/2)*jump(cc))
          //  - int1d(Th,C)((n<0)*abs(n)*cc*w) // unused because cc=0 on ∂Ω⁻
  - int2d(Th)(ccold*w/dt);

for ( t=0; t< 2*pi ; t+=dt)
{
  ccold=cc; Adual;
  plot(cc,fill=1,cmm="t="+t + ", min=" + cc[].min + ", max=" +  cc[].max);
};
real [int] viso=[-0.2,-0.1,0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.1];
plot(c,wait=1,fill=1,ps="convectCG.eps",viso=viso);
plot(c,wait=1,fill=1,ps="convectDG.eps",viso=viso);
```

Notice the new keywords, intalledges to integrate on all edges, nTonEdge which is one if the triangle has a boundary edge and zero otherwise, `jump` to implement $[c]$. Results of both methods are shown on Figure 3.6 with identical levels for the level line; this is done with the plot-modifier viso.
Notice also the macro where the parameter u is not used (but the syntax needs one) and which ends with a //; it simply replaces the name `n` by `(N.x*v1+N.y*v2)`. As easily guessed `N.x,N.y` is the normal to the edge.

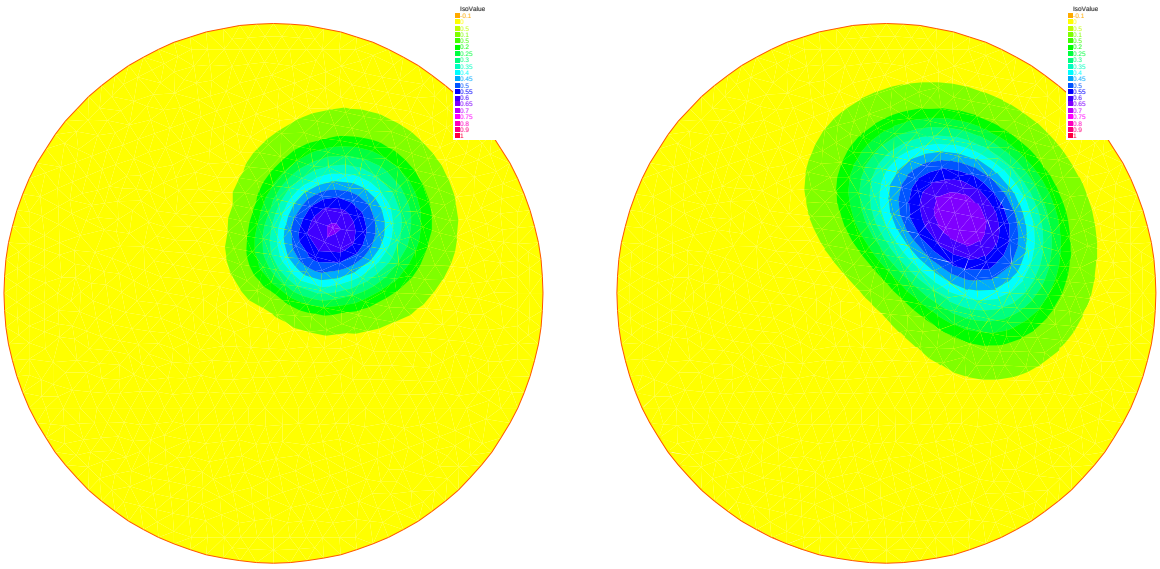


Figure 3.6: The rotated hill after one revolution, left with Characteristics-Galerkin, on the right with Discontinuous P_1 Galerkin FEM.

Now if you think that DG is too slow try this

```
                                                //    the same DG very much faster
varf aadual(cc,w) = int2d(Th)((cc/dt+(v1*dx(cc)+v2*dy(cc)))*w)
        + intalledges(Th)((1-nTonEdge)*w*(al*abs(n)-n/2)*jump(cc));
varf bbdual(ccold,w) =  - int2d(Th)(ccold*w/dt);
matrix  AA= aadual(Vh,Vh);
matrix BB = bbdual(Vh,Vh);
set (AA,init=t,solver=UMFPACK);
Vh rhs=0;
for ( t=0; t< 2*pi ; t+=dt)
{
  ccold=cc;
  rhs[] = BB* ccold[];
  cc[] = AA^-1*rhs[];
  plot(cc,fill=0,cmm="t="+t + ", min=" + cc[].min + ", max=" +  cc[].max);
};
```

Notice the new keyword set to specify a solver in this framework; the modifier init is used to tel the solver that the matrix has not changed (init=true), and the name parameter are the same that in problem definition (see. 6.7) .

Finite Volume Methods can also be handled with `FreeFem++` but it requires programming. For instance the $P_0 - P_0$ Finite Volume Method of Dervieux et al associates to each P_0 function c^1 a P_0 function c^0 with constant value around each vertex q^i equal to $c^1(q^i)$ on the cell σ_i made by all the medians of all triangles having q^i as vertex. Then upwinding is done by taking left or right values at the median:

$$\int_{\sigma_i} \frac{1}{\delta t}(c^{1^{n+1}} - c^{1^n}) + \int_{\partial\sigma_i} u \cdot nc^- = 0 \quad \forall i$$

It can be programmed as

```
load "mat_dervieux";                    //    external module in C++ must be loaded
border a(t=0, 2*pi){ x = cos(t); y = sin(t);  }
mesh th = buildmesh(a(100));
fespace Vh(th,P1);

Vh vh,vold,u1 = y, u2 = -x;
Vh v = exp(-10*((x-0.3)^2 +(y-0.3)^2)), vWall=0, rhs =0;

real dt = 0.025;
                                        //    qf1pTlump means mass lumping is used
problem  FVM(v,vh) = int2d(th,qft=qf1pTlump)(v*vh/dt)
                   - int2d(th,qft=qf1pTlump)(vold*vh/dt)
      + int1d(th,a)(((u1*N.x+u2*N.y)<0)*(u1*N.x+u2*N.y)*vWall*vh)
+ rhs[] ;

matrix A;
MatUpWind0(A,th,vold,[u1,u2]);

for ( int t=0; t< 2*pi ; t+=dt){
  vold=v;
  rhs[] = A * vold[] ; FVM;
  plot(v,wait=0);
};
```

the mass lumping parameter forces a quadrature formula with Gauss points at the vertices so as to make the mass matrix diagonal; the linear system solved by a conjugate gradient method for instance will then converge in one or two iterations.
The right hand side `rhs` is computed by an external C++ function `MatUpWind0(...)` which is programmed as

```
                    //    computes matrix a on a triangle for the Dervieux FVM
int   fvmP1P0(double q[3][2],              //    the 3 vertices of a triangle T
              double u[2],                 //    convection velocity on T
              double c[3],                 //    the P1 function on T
              double a[3][3],              //    output matrix
              double where[3] )      //    where>0 means we're on the boundary
{
  for(int i=0;i<3;i++) for(int j=0;j<3;j++) a[i][j]=0;

    for(int i=0;i<3;i++){
        int ip = (i+1)%3, ipp =(ip+1)%3;
        double unL =-((q[ip][1]+q[i][1]-2*q[ipp][1])*u[0]
                     -(q[ip][0]+q[i][0]-2*q[ipp][0])*u[1])/6;
        if(unL>0) { a[i][i] += unL; a[ip][i]-=unL;}
            else{ a[i][ip] += unL; a[ip][ip]-=unL;}
```

```
        if(where[i]&&where[ip]){                    //    this is a boundary edge
            unL=((q[ip][1]-q[i][1])*u[0] -(q[ip][0]-q[i][0])*u[1])/2;
            if(unL>0) { a[i][i]+=unL; a[ip][ip]+=unL;}
        }
    }
  return 1;
}
```

It must be inserted into a larger .cpp file, shown in Appendix A, which is the load module linked to `FreeFem++` .

3.7 A Projection Algorithm for the Navier-Stokes equations

Summary *Fluid flows require good algorithms and good triangultions. We show here an example of a complex algorithm and or first example of mesh adaptation.*

An incompressible viscous fluid satisfies:

$$\partial_t u + u \cdot \nabla u + \nabla p - \nu \Delta u = 0, \quad \nabla \cdot u = 0 \quad \text{in } \Omega \times]0, T[,$$

$$u|_{t=0} = u^0, \quad u|_\Gamma = u_\Gamma.$$

A possible algorithm, proposed by Chorin, is

$$\frac{1}{\delta t}[u^{m+1} - u^m o X^m] + \nabla p^m - \nu \Delta u^m = 0, \quad u|_\Gamma = u_\Gamma,$$

$$-\Delta p^{m+1} = -\nabla \cdot u^m o X^m, \quad \partial_n p^{m+1} = 0,$$

where $uoX(x) = u(x - u(x)\delta t)$ since $\partial_t u + u \cdot \nabla u$ is approximated by the method of characteristics, as in the previous section.

An improvement over Chorin's algorithm, given by Rannacher, is to compute a correction, q, to the pressure (the overline denotes the mean over Ω)

$$-\Delta q = \nabla \cdot \boldsymbol{u} - \overline{\nabla \cdot \boldsymbol{u}}$$

and define

$$u^{m+1} = \tilde{u} + \nabla q \delta t, \quad p^{m+1} = p^m - q - \overline{p^m - q}$$

where $\tilde{u}$ is the (u^{m+1}, v^{m+1}) of Chorin's algorithm.

The backward facing step The geometry is that of a channel with a backward facing step so that the inflow section is smaller than the outflow section. This geometry produces a fluid recirculation zone that must be captured correctly.
This can only be done if the triangulation is sufficiently fine, or well adapted to the flow.

Example 3.8 (NSprojection.edp)

```
border a0(t=1,0){ x=0;      y=t;          label=1;}
border a1(t=0,1){ x=2*t;    y=0;          label=2;}
```

```
border a2(t=0,1){ x=2;       y=-t/2;          label=2;}
border a3(t=0,1){ x=2+18*t^1.2;  y=-0.5;      label=2;}
border a4(t=0,1){ x=20;      y=-0.5+1.5*t;    label=3;}
border a5(t=1,0){ x=20*t;    y=1;             label=4;}
int n=1;
mesh Th= buildmesh(a0(3*n)+a1(20*n)+a2(10*n)+a3(150*n)+a4(5*n)+a5(100*n));
plot(Th);
fespace Vh(Th,P1);
real nu = 0.0025, dt = 0.2;                                    //    Reynolds=400
Vh w,u = 4*y*(1-y)*(y>0)*(x<2), v =0, p = 0, q=0;
real area= int2d(Th)(1.);

for(int n=0;n<100;n++){
  Vh uold = u,  vold = v, pold=p;
  Vh f=convect([u,v],-dt,uold),  g=convect([u,v],-dt,vold);

  solve pb4u(u,w,init=n,solver=LU)
        =int2d(Th)(u*w/dt +nu*(dx(u)*dx(w)+dy(u)*dy(w)))
        -int2d(Th)((f/dt-dx(p))*w)
        + on(1,u = 4*y*(1-y)) + on(2,4,u = 0)+ on(3,u=f);
  plot(u);

  solve pb4v(v,w,init=n,solver=LU)
        = int2d(Th)(v*w/dt +nu*(dx(v)*dx(w)+dy(v)*dy(w)))
        -int2d(Th)((g/dt-dy(p))*w)
        +on(1,2,3,4,v = 0);

 real meandiv = int2d(Th)(dx(u)+dy(v))/area;

 solve pb4p(q,w,init=n,solver=LU)= int2d(Th)(dx(q)*dx(w)+dy(q)*dy(w))
    - int2d(Th)((dx(u)+ dy(v)-meandiv)*w/dt)+ on(3,q=0);

 real meanpq = int2d(Th)(pold - q)/area;
 if(n==50){
    Th = adaptmesh(Th,u,v,q); plot(Th, wait=true);
 }
 p = pold-q-meanpq;
 u = u + dx(q)*dt;
 v = v + dy(q)*dt;
}
```

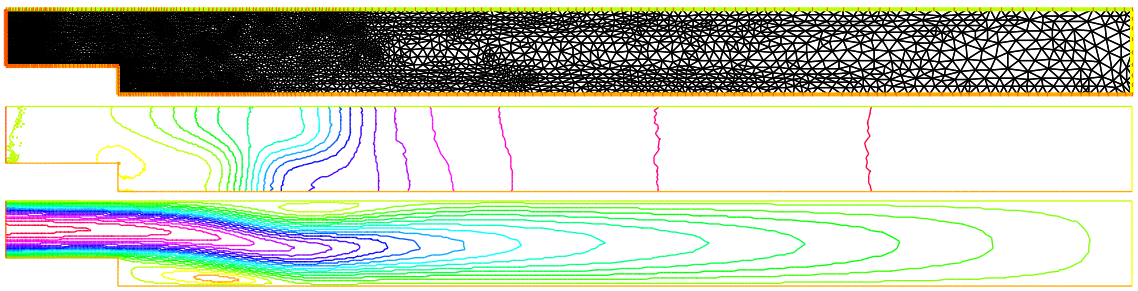


Figure 3.7: Rannacher's projection algorithm: result on an adapted mesh (top) showing the pressure (middle) and the horizontal velocity u at Reynolds 400.

We show in figure 3.7 the numerical results obtained for a Reynolds number of 400 where mesh adaptation is done after 50 iterations on the first mesh.

3.8 The System of elasticity

Elasticity Solid objects deform under the action of applied forces: a point in the solid, originally at (x, y, z) will come to (X, Y, Z) after some time; the vector $\mathbf{u} = (u_1, u_2, u_3) = (X - x, Y - y, Z - z)$ is called the displacement. When the displacement is small and the solid is elastic, Hooke's law gives a relationship between the stress tensor $\sigma(u) = (\sigma_{ij}(u))$ and the strain tensor $\epsilon(u) = \epsilon_{ij}(u)$

$$\sigma_{ij}(u) = \lambda \delta_{ij} \nabla.\mathbf{u} + 2\mu\epsilon_{ij}(u),$$

where the Kronecker symbol $\delta_{ij} = 1$ if $i = j$, 0 otherwise, with

$$\epsilon_{ij}(u) = \frac{1}{2}(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}),$$

and where λ, μ are two constants that describe the mechanical properties of the solid, and are themselves related to the better known constants E, Young's modulus, and ν, Poisson's ratio:

$$\mu = \frac{E}{2(1+\nu)}, \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}.$$

Lamé's system Let us consider a beam with axis Oz and with perpendicular section Ω. The components along x and y of the strain $\mathbf{u}(x)$ in a section Ω subject to forces $\mathbf{f}$ perpendicular to the axis are governed by

$$-\mu\Delta\mathbf{u} - (\mu + \lambda)\nabla(\nabla.\mathbf{u}) = \mathbf{f} \text{ in } \Omega,$$

where λ, μ are the Lamé coefficients introduced above.
Remark, we do not used this equation because the associated variationnal form does not give the right boundary condition, we simply use

$$-div(\sigma) = \mathbf{f} \quad \text{in}\Omega$$

where the corresponding variationnal form is:

$$\int_\Omega \sigma(u) : \epsilon(\mathbf{v})\, dx - \int_\Omega \mathbf{v} f\, dx = 0;$$

where : denote the tensor scalar product, i.e. $a : b = \sum_{i,j} a_{ij} b_{ij}$.
So the variationnal form can be written as :

$$\int_\Omega \lambda \nabla.u \nabla.v + 2\mu\epsilon(\mathbf{u}) : \epsilon(\mathbf{v})\, dx - \int_\Omega \mathbf{v} f\, dx = 0;$$

Example Consider elastic plate with the undeformed rectangle shape $[0,20]\times[-1,1]$. The body force is the gravity force $\boldsymbol{f}$ and the boundary force $\boldsymbol{g}$ is zero on lower, upper and right sides. The left vertical sides of the beam is fixed. The boundary conditions are

$$\begin{aligned} \sigma.\mathbf{n} &= g = 0 \text{ on } \Gamma_1,\Gamma_4,\Gamma_3, \\ \mathbf{u} &= \mathbf{0} \text{ on } \Gamma_2 \end{aligned}$$

Here $\mathbf{u} = (u, v)$ has two components.

The above two equations are strongly coupled by their mixed derivatives, and thus any iterative solution on each of the components is risky. One should rather use `FreeFem++` 's system approach and write:

Example 3.9 (lame.edp)

```
//    file lame.edp
mesh Th=square(10,10,[20*x,2*y-1]);
fespace Vh(Th,P2);
Vh u,v,uu,vv;
real sqrt2=sqrt(2.);
macro epsilon(u1,u2)  [dx(u1),dy(u2),(dy(u1)+dx(u2))/sqrt2]          //    EOM
 //   the sqrt2 is because we want:  epsilon(u1,u2)'* epsilon(v1,v2) == ϵ(u) : ϵ(v)
macro div(u,v)  ( dx(u)+dy(v) )                                     //    EOM


real E = 21e5, nu = 0.28, mu= E/(2*(1+nu));
real lambda = E*nu/((1+nu)*(1-2*nu)), f = -1;                              //

solve lame([u,v],[uu,vv])= int2d(Th)(
        lambda*div(u,v)*div(uu,vv)
        +2.*mu*( epsilon(u,v)'*epsilon(uu,vv) ) )
        - int2d(Th)(f*vv)
        + on(4,u=0,v=0);
real coef=100;
plot([u,v],wait=1,ps="lamevect.eps",coef=coef);

mesh th1 = movemesh(Th, [x+u*coef, y+v*coef]);
plot(th1,wait=1,ps="lamedeform.eps");
real dxmin  = u[].min;
real dymin  = v[].min;

cout << " - dep.  max   x = "<< dxmin<< " y=" << dymin << endl;
cout << "   dep.  (20,0)  = " << u(20,0) << " " << v(20,0) << endl;
```

The numerical results are shown on figure 3.8 and the output is:

```
 -- square mesh : nb vertices  =121 ,  nb triangles = 200 ,  nb boundary edges 40
 -- Solve :           min -0.00174137  max 0.00174105
          min -0.0263154  max 1.47016e-29
 - dep.  max   x = -0.00174137 y=-0.0263154
   dep.  (20,0)  = -1.8096e-07 -0.0263154
times: compile 0.010219s, execution 1.5827s
```

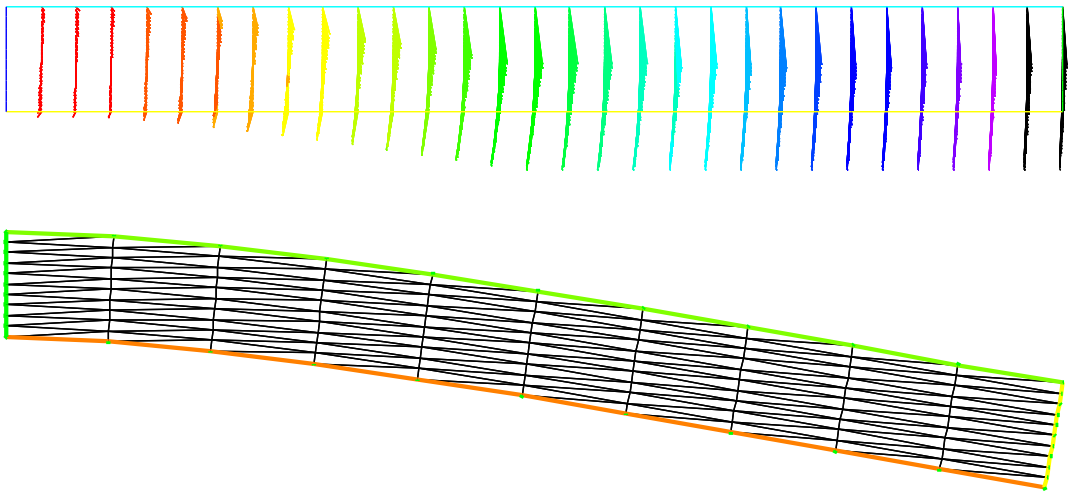


Figure 3.8: Solution of Lamé's equations for elasticity for a 2D beam deflected by its own weight and clamped by its left vertical side; result are shown with a amplification factor equal to 100. *Remark: the size of the arrow is automatically bound, but the color gives the real length*

3.9 The System of Stokes for Fluids

In the case of a flow invariant with respect to the third coordinate (two-dimensional flow), flows at low Reynolds number (for instance micro-organisms) satisfy,

$$-\Delta u + \nabla p = 0$$
$$\nabla \cdot \boldsymbol{u} = 0$$

where $\boldsymbol{u} = (u_1, u_2)$ is the fluid velocity and p its pressure.
The driven cavity is a standard test. It is a box full of liquid with its lid moving horizontally at speed one. The pressure and the velocity must be discretized in compatible fintie element spaces for the LBB conditions to be satisfied:

$$\sup_{p \in P_h} \frac{(\boldsymbol{u}, \nabla p)}{|p|} \geq \beta |\boldsymbol{u}| \quad \forall \boldsymbol{u} \in U_h$$

```
//   file stokes.edp
int n=3;
mesh Th=square(10*n,10*n);
fespace Uh(Th,P1b); Uh u,v,uu,vv;
fespace Ph(Th,P1);  Ph p,pp;

solve stokes([u,v,p],[uu,vv,pp]) =
    int2d(Th)(dx(u)*dx(uu)+dy(u)*dy(uu) + dx(v)*dx(vv)+ dy(v)*dy(vv)
            + dx(p)*uu + dy(p)*vv + pp*(dx(u)+dy(v)))
            + on(1,2,4,u=0,v=0) + on(3,u=1,v=0);
plot([u,v],p,wait=1);
```

Results are shown on figure 3.9

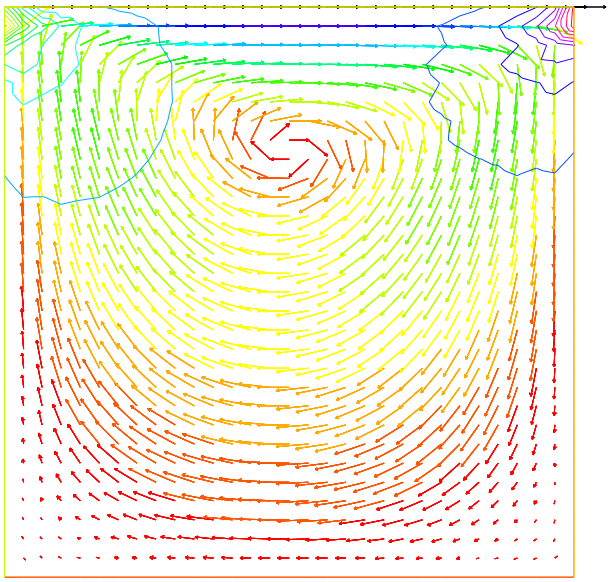


Figure 3.9: Solution of Stokes' equations for the driven cavity problem, showing the velocity field and the pressure level lines.

3.10 A Large Fluid Problem

A friend of one of us in Auroville-India was building a ramp to access an air conditioned room. As I was visiting the construction site he told me that he expected to cool air escaping by the door to the room to slide down the ramp and refrigerate the feet of the coming visitors. I told him "no way" and decided to check numerically. The results are on the front page of this book.
The fluid velocity and pressure are solution of the Navier-Stokes equations with varying density function of the temperature.
The geometry is trapezoidal with prescribed inflow made of cool air at the bottom and warm air above and so are the initial conditions; there is free outflow, slip velocity at the top (artificial) boundary and no-slip at the bottom. However the Navier-Stokes cum temperature equations have a RANS $k-\epsilon$ model and a Boussinesq approximation for the buoyancy. This comes to

$$
\begin{aligned}
&\partial_t\theta + u\nabla\theta - \nabla\cdot(\kappa_T^m\nabla\theta) = 0\\
&\partial_t u + u\nabla u - \nabla\cdot(\mu_T\nabla u) + \nabla p + e(\theta-\theta_0)\boldsymbol{e}_2,\quad \nabla\cdot u = 0\\
&\mu_T = c_\mu\frac{k^2}{\epsilon},\quad \kappa_T = \kappa\mu_T\\
&\partial_t k + u\nabla k + \epsilon - \nabla\cdot(\mu_T\nabla k) = \frac{\mu_T}{2}|\nabla u + \nabla u^T|^2\\
&\partial_t\epsilon + u\nabla\epsilon + c_2\frac{\epsilon^2}{k} - \frac{c_\epsilon}{c_\mu}\nabla\cdot(\mu_T\nabla\epsilon) = \frac{c_1}{2}k|\nabla u + \nabla u^T|^2 = 0
\end{aligned}
\tag{3.5}
$$

We use a time discretization which preserves positivity and uses the method of characteristics ($X^m(x)\approx x - u^m(x)\delta t$)

$$
\begin{aligned}
&\frac{1}{\delta t}(\theta^{m+1} - \theta^m\circ X^m) - \nabla\cdot(\kappa_T^m\nabla\theta^{m+1}) = 0\\
&\frac{1}{\delta t}(u^{m+1} - u^m\circ X^m) - \nabla\cdot(\mu_T^m\nabla u^{m+1}) + \nabla p^{m+1} + e(\theta^{m+1}-\theta_0)\boldsymbol{e}_2,\quad \nabla\cdot u^{m+1} = 0
\end{aligned}
$$

$$\frac{1}{\delta t}(k^{m+1} - k^m \circ X^m) + k^{m+1}\frac{\epsilon^m}{k^m} - \nabla\cdot(\mu_T^m \nabla k^{m+1}) = \frac{\mu_T^m}{2}|\nabla u^m + \nabla u^{mT}|^2$$
$$\frac{1}{\delta t}(\epsilon^{m+1} - \epsilon^m \circ X^m) + c_2\epsilon^{m+1}\frac{\epsilon^m}{k^m} - \frac{c_\epsilon}{c_\mu}\nabla(\mu_T^m \nabla \epsilon^{m+1}) = \frac{c_1}{2}k^m|\nabla u^m + \nabla u^{mT}|^2$$
$$\mu_T^{m+1} = c_\mu \frac{k^{m+1^2}}{\epsilon^{m+1}}, \quad \kappa_T^{m+1} = \kappa\mu_T^{m+1} \tag{3.6}$$

In variational form and with appropriated boundary conditions the problem is:

```
real L=6;
border aa(t=0,1){x=t; y=0 ;}
border bb(t=0,14){x=1+t; y= - 0.1*t ;}
border cc(t=-1.4,L){x=15; y=t ;}
border dd(t=15,0){x= t ; y = L;}
border ee(t=L,0.5){ x=0; y=t ;}
border ff(t=0.5,0){ x=0; y=t ;}
int n=8;
mesh Th=buildmesh(aa(n)+bb(9*n) + cc(4*n) + dd(10*n)+ee(6*n) + ff(n));
real s0=clock();

fespace Vh2(Th,P1b);                                          //    velocity space
fespace Vh(Th,P1);                                            //    pressure space
fespace V0h(Th,P0);                                           //    for gradients
Vh2 u2,v2,up1=0,up2=0;
Vh2 u1,v1;
Vh  u1x=0,u1y,u2x,u2y, vv;

real reylnods=500;
                  //   cout << " Enter the reynolds number :"; cin >> reylnods;
assert(reylnods>1 && reylnods < 100000);
up1=0;
up2=0;
func g=(x)*(1-x)*4;                                                  //    inflow
Vh p=0,q, temp1,temp=35, k=0.001,k1,ep=0.0001,ep1;
V0h muT=1,prodk,prode, kappa=0.25e-4, stress;
real alpha=0, eee=9.81/303, c1m = 1.3/0.09 ;
real  nu=1, numu=nu/sqrt( 0.09), nuep=pow(nu,1.5)/4.1;
int i=0,iter=0;
real dt=0;
problem TEMPER(temp,q) =                                  //    temperature equation
    int2d(Th)(
             alpha*temp*q + kappa * ( dx(temp)*dx(q) + dy(temp)*dy(q) ))
                                                  //    + int1d(Th,aa,bb)(temp*q* 0.1)
  + int2d(Th) ( -alpha*convect([up1,up2],-dt,temp1)*q )
   + on(ff,temp=25)
  + on(aa,bb,temp=35) ;

problem kine(k,q)=                          //    get the kinetic turbulent energy
    int2d(Th)(
             (ep1/k1+alpha)*k*q + muT * ( dx(k)*dx(q) + dy(k)*dy(q) ))
                                                  //    + int1d(Th,aa,bb)(temp*q*0.1)
  + int2d(Th) ( prodk*q-alpha*convect([up1,up2],-dt,k1)*q )
   + on(ff,k=0.0001)  + on(aa,bb,k=numu*stress) ;

 problem viscturb(ep,q)=         //    get the rate of turbulent viscous energy
    int2d(Th)(
```

```
              (1.92*ep1/k1+alpha)*ep*q + c1m*muT * ( dx(ep)*dx(q) + dy(ep)*dy(q)
))
                                                    //    + int1d(Th,aa,bb)(temp*q*0.1)
  + int2d(Th) ( prode*q-alpha*convect([up1,up2],-dt,ep1)*q )
   + on(ff,ep= 0.0001) + on(aa,bb,ep=nuep*pow(stress,1.5)) ;


 solve NS ([u1,u2,p],[v1,v2,q]) = //    Navier-Stokes k-epsilon and Boussinesq
    int2d(Th)(
              alpha*( u1*v1 + u2*v2)
             + muT * (dx(u1)*dx(v1)+dy(u1)*dy(v1)+dx(u2)*dx(v2)+dy(u2)*dy(v2))
    //     ( 2*dx(u1)*dx(v1) + 2*dy(u2)*dy(v2)+(dy(u1)+dx(u2))*(dy(v1)+dx(v2)))
             + p*q*(0.000001)
             - p*dx(v1) - p*dy(v2)
             - dx(u1)*q - dy(u2)*q
            )
  + int1d(Th,aa,bb,dd)(u1*v1* 0.1)
  + int2d(Th) (eee*(temp-35)*v1 -alpha*convect([up1,up2],-dt,up1)*v1
                               -alpha*convect([up1,up2],-dt,up2)*v2 )
   + on(ff,u1=3,u2=0)
  + on(ee,u1=0,u2=0)
  + on(aa,dd,u2=0)
  + on(bb,u2= -up1*N.x/N.y)
  + on(cc,u2=0) ;
 plot(coef=0.2,cmm=" [u1,u2] et p  ",p,[u1,u2],ps="StokesP2P1.eps",value=1,wait=1);
{
  real[int] xx(21),yy(21),pp(21);
  for (int i=0;i<21;i++)
   {
     yy[i]=i/20.;
     xx[i]=u1(0.5,i/20.);
     pp[i]=p(i/20.,0.999);
    }
      cout << " " << yy << endl;
                                  //    plot([xx,yy],wait=1,cmm="u1 x=0.5 cup");
                           //    plot([yy,pp],wait=1,cmm="pressure y=0.999 cup");
}

dt = 0.05;
int nbiter = 3;
real coefdt = 0.25^(1./nbiter);
real coefcut = 0.25^(1./nbiter) , cut=0.01;
real tol=0.5,coeftol = 0.5^(1./nbiter);
nu=1./reylnods;

for (iter=1;iter<=nbiter;iter++)
{
 cout << " dt = " << dt << " ------------------------ " << endl;
  alpha=1/dt;
  for (i=0;i<=500;i++)
   {
     up1=u1;
     up2=u2;
     temp1=max(temp,25);
     temp1=min(temp1,35);
     k1=k; ep1=ep;
```

```
    muT=0.09*k*k/ep;
     NS; plot([u1,u2],wait=1);                          //    Solves Navier-Stokes
    prode =0.126*k*(pow(2*dx(u1),2)+pow(2*dy(u2),2)+2*pow(dx(u2)+dy(u1),2))/2;
    prodk= prode*k/ep*0.09/0.126;
    kappa=muT/0.41;
    stress=abs(dy(u1));
    kine; plot(k,wait=1);
    viscturb; plot(ep,wait=1);
    TEMPER;                                       //    solves temperature equation
    if ( !(i % 5)){
        plot(temp,value=1,fill=true,ps="temp_"+iter+"_"+i+".ps");
        plot(coef=0.2,cmm=" [u1,u2] et p  ",p,[u1,u2],ps="plotNS_"+iter+"_"+i+".ps");
    }
    cout << "CPU " << clock()-s0 << "s " << endl;
  }

  if (iter>= nbiter) break;
   Th=adaptmesh(Th,[dx(u1),dy(u1),dx(u1),dy(u2)],splitpbedge=1,
                abserror=0,cutoff=cut,err=tol, inquire=0,ratio=1.5,hmin=1./1000);
 plot(Th,ps="ThNS.eps");
  dt = dt*coefdt;
  tol = tol *coeftol;
  cut = cut *coefcut;
}
cout << "CPU " <<clock()-s0 << "s " << endl;
```

3.11 An Example with Complex Numbers

In a microwave oven heat comes from molecular excitation by an electromagnetic field. For a plane monochromatic wave, amplitude is given by Helmholtz's equation:

$$\beta v + \Delta v = 0.$$

We consider a rectangular oven where the wave is emitted by part of the upper wall. So the boundary of the domain is made up of a part Γ_1 where $v = 0$ and of another part $\Gamma_2 = [c, d]$ where for instance $v = \sin(\pi\frac{y-c}{c-d})$.
Within an object to be cooked, denoted by B, the heat source is proportional to v^2. At equilibrium, one has

$$-\Delta\theta = v^2 I_B, \quad \theta_\Gamma = 0$$

where I_B is 1 in the object and 0 elsewhere.
Results are shown on figure 3.10
In the program below $\beta = 1/(1 - I/2)$ in the air and $2/(1 - I/2)$ in the object ($i = \sqrt{-1}$):

Example 3.10 (muwave.edp)

```
                                                          //    file muwave.edp
real a=20, b=20, c=15, d=8, e=2, l=12, f=2, g=2;
border a0(t=0,1) {x=a*t; y=0;label=1;}
border a1(t=1,2) {x=a; y= b*(t-1);label=1;}
border a2(t=2,3) { x=a*(3-t);y=b;label=1;}
```

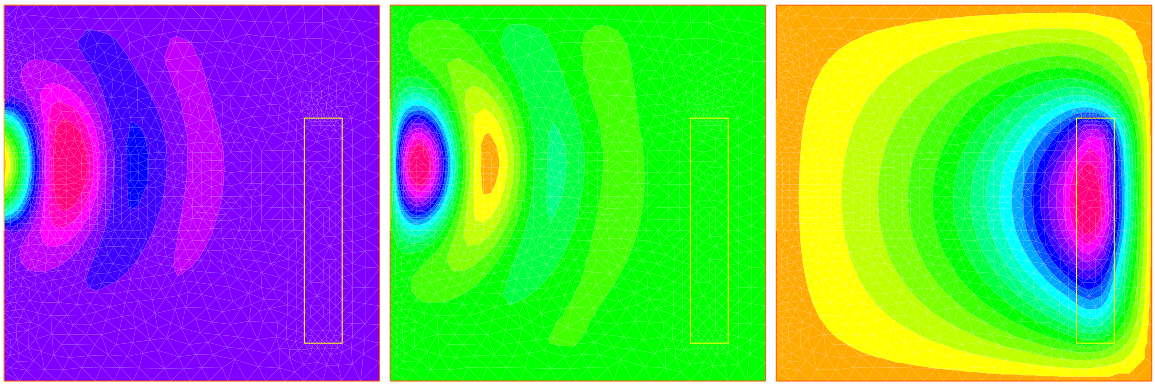


Figure 3.10: A microwave oven: real (left) and imaginary (middle) parts of wave and temperature (right).

```
border a3(t=3,4){x=0;y=b-(b-c)*(t-3);label=1;}
border a4(t=4,5){x=0;y=c-(c-d)*(t-4);label=2;}
border a5(t=5,6){ x=0; y= d*(6-t);label=1;}

border b0(t=0,1) {x=a-f+e*(t-1);y=g; label=3;}
border b1(t=1,4) {x=a-f; y=g+l*(t-1)/3; label=3;}
border b2(t=4,5) {x=a-f-e*(t-4); y=l+g; label=3;}
border b3(t=5,8) {x=a-e-f; y= l+g-l*(t-5)/3; label=3;}
int n=2;
mesh Th = buildmesh(a0(10*n)+a1(10*n)+a2(10*n)+a3(10*n)
        +a4(10*n)+a5(10*n)+b0(5*n)+b1(10*n)+b2(5*n)+b3(10*n));
plot(Th,wait=1);
fespace Vh(Th,P1);
real meat =  Th(a-f-e/2,g+l/2).region, air= Th(0.01,0.01).region;
Vh R=(region-air)/(meat-air);

Vh<complex> v,w;
solve muwave(v,w) = int2d(Th)(v*w*(1+R)
                -(dx(v)*dx(w)+dy(v)*dy(w))*(1-0.5i))
   + on(1,v=0) + on(2, v=sin(pi*(y-c)/(c-d)));
Vh vr=real(v), vi=imag(v);
plot(vr,wait=1,ps="rmuonde.ps", fill=true);
plot(vi,wait=1,ps="imuonde.ps", fill=true);

fespace Uh(Th,P1); Uh u,uu, ff=1e5*(vr^2 + vi^2)*R;

solve temperature(u,uu)= int2d(Th)(dx(u)* dx(uu)+ dy(u)* dy(uu))
     - int2d(Th)(ff*uu) + on(1,2,u=0);
plot(u,wait=1,ps="tempmuonde.ps", fill=true);
```

3.12 Optimal Control

Thanks to the function `BFGS` it is possible to solve complex nonlinear optimization problem within `FreeFem++`. For example consider the following inverse problem

$$\min_{b,c,d\in R} J = \int_E (u-u_d)^2 \ : \ -\nabla(\kappa(b,c,d)\cdot\nabla u) = 0, \ \ u|_\Gamma = u_\Gamma$$

where the desired state u_d, the boundary data u_Γ and the observation set $E \subset \Omega$ are all given. Furthermore let us assume that

$$\kappa(x) = 1 + bI_B(x) + cI_C(x) + dI_D(x) \quad \forall x \in \Omega$$

where B, C, D are separated subsets of Ω.

To solve this problem by the quasi-Newton BFGS method we need the derivatives of J with respect to b, c, d. We self explanatory notations, if $\delta b, \delta c, \delta d$ are variations of b, c, d we have

$$\delta J \approx 2\int_E (u-u_d)\delta u, \quad -\nabla(\kappa\cdot\nabla\delta u) \approx \nabla(\delta\kappa\cdot\nabla u) \ \ \delta u|_\Gamma = 0$$

Obviously J'_b is equal to δJ when $\delta b = 1, \delta c = 0, \delta d = 0$, and so on for J'_c and J'_d.

All this is implemented in the following program

```
                                                        //    file optimcontrol.edp
border aa(t=0, 2*pi) {    x = 5*cos(t);    y = 5*sin(t);  };
border bb(t=0, 2*pi) {    x = cos(t);    y = sin(t);  };
border cc(t=0, 2*pi) {    x = -3+cos(t);    y = sin(t);  };
border dd(t=0, 2*pi) {    x = cos(t);    y = -3+sin(t);  };
mesh th = buildmesh(aa(70)+bb(35)+cc(35)+dd(35));
fespace Vh(th,P1);
Vh Ib=((x^2+y^2)<1.0001),
   Ic=(((x+3)^2+ y^2)<1.0001),
   Id=((x^2+(y+3)^2)<1.0001),
   Ie=(((x-1)^2+ y^2)<=4),
   ud,u,uh,du;
real[int] z(3);
problem A(u,uh) =int2d(th)((1+z[0]*Ib+z[1]*Ic+z[2]*Id)*(dx(u)*dx(uh)
                    +dy(u)*dy(uh))) + on(aa,u=x^3-y^3);
z[0]=2; z[1]=3; z[2]=4;
A; ud=u;
ofstream f("J.txt");
func real J(real[int] & Z)
{
    for (int i=0;i<z.n;i++)z[i]=Z[i];
    A; real s= int2d(th)(Ie*(u-ud)^2);
    f<<s<<"   "; return s;
}

real[int] dz(3), dJdz(3);

problem B(du,uh)
  =int2d(th)((1+z[0]*Ib+z[1]*Ic+z[2]*Id)*(dx(du)*dx(uh)+dy(du)*dy(uh)))
```

```
  +int2d(th)((dz[0]*Ib+dz[1]*Ic+dz[2]*Id)*(dx(u)*dx(uh)+dy(u)*dy(uh)))
  +on(aa,du=0);

func real[int] DJ(real[int] &Z)
    {
      for(int i=0;i<z.n;i++)
        { for(int j=0;j<dz.n;j++) dz[j]=0;
          dz[i]=1; B;
          dJdz[i]= 2*int2d(th)(Ie*(u-ud)*du);
        }
     return dJdz;
 }

 real[int] Z(3);
 for(int j=0;j<z.n;j++) Z[j]=1;
 BFGS(J,DJ,Z,eps=1.e-6,nbiter=15,nbiterline=20);
 cout << "BFGS: J(z) = " << J(Z) <<  endl;
 for(int j=0;j<z.n;j++) cout<<z[j]<<endl;
 plot(ud,value=1,ps="u.eps");
```

In this example the sets B, C, D, E are circles of boundaries bb, cc, dd, ee are the domain Ω is the circle of boundary aa. The desired state u_d is the solution of the PDE for $b = 2, c = 3, d = 4$. The unknowns are packed into array z. Notice that it is necessary to recopy Z into z because one is a local variable while the other one is global. The program found $b = 2.00125, c = 3.00109, d = 4.00551$. Figure 3.11 shows u at convergence and the successive function evaluations of J. Note

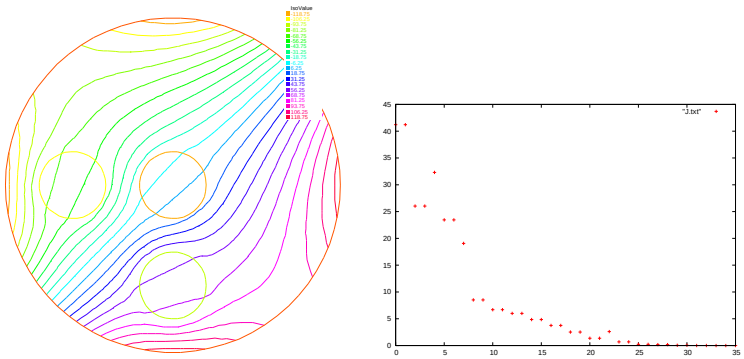


Figure 3.11: On the left the level lines of u. On the right the successive evaluations of J by BFGS (5 values above 500 have been removed for readability)

that an *adjoint state* could have been used. Define p by

$$-\nabla \cdot (\kappa \nabla p) = 2I_E(u - u_d), \quad p|_\Gamma = 0$$

Consequently

$$\begin{aligned} \delta J &= -\int_\Omega (\nabla \cdot (\kappa \nabla p))\delta u \\ &= \int_\Omega (\kappa \nabla p \cdot \nabla \delta u) = -\int_\Omega (\delta\kappa \nabla p \cdot \nabla u) \end{aligned} \tag{3.7}$$

Then the derivatives are found by setting $\delta b = 1, \delta c = \delta d = 0$ and so on:

$$J'_b = -\int_B \nabla p \cdot \nabla u, \quad J'_c = -\int_C \nabla p \cdot \nabla u, \quad J'_d = -\int_D \nabla p \cdot \nabla u$$

Remark As BFGS stores an $M \times M$ matrix where M is the number of unknowns, it is dangerously expensive to use this method when the unknown x is a Finite Element Function. One should use another optimizer such as the NonLinear Conjugate Gradient `NLCG` (also a key word of `FreeFem++`). See the file algo.edp in the examples folder.

3.13 A Flow with Shocks

Compressible Euler equations should be discretized with Finite Volumes or FEM with flux upwinding scheme but these are not implemented in `FreeFem++` . Nevertheless acceptable results can be obtained with the method of characteristiscs provided that the mean values $\bar{f} = \frac{1}{2}(f^+ + f^-)$ are used at shocks in the scheme.

$$\begin{aligned}
&\partial_t \rho + \bar{u}\nabla\rho + \bar{\rho}\nabla \cdot u = 0 \\
&\bar{\rho}(\partial_t u + \frac{\overline{\rho u}}{\bar{\rho}}\nabla u + \nabla p = 0 \\
&\partial_t p + \bar{u}\nabla p + (\gamma - 1)\bar{p}\nabla \cdot u = 0
\end{aligned} \tag{3.8}$$

One possibility is to couple u, p and then update ρ, i.e.

$$\begin{aligned}
&\frac{1}{(\gamma-1)\delta t \bar{p}^m}(p^{m+1} - p^m \circ X^m) + \nabla \cdot u^{m+1} = 0 \\
&\frac{\bar{\rho}^m}{\delta t}(u^{m+1} - u^m \circ \tilde{X}^m) + \nabla p^{m+1} = 0 \\
&\rho^{m+1} = \rho^m \circ X^m + \frac{\bar{\rho}^m}{(\gamma-1)\bar{p}^m}(p^{m+1} - p^m \circ X^m)
\end{aligned} \tag{3.9}$$

A numerical result is given on Figure 3.12 and the `FreeFem++` script is

```
verbosity=1;
int anew=1;
real x0=0.5,y0=0, rr=0.2;
border ccc(t=0,2){x=2-t;y=1;};
border ddd(t=0,1){x=0;y=1-t;};
border aaa1(t=0,x0-rr){x=t;y=0;};
border cercle(t=pi,0){ x=x0+rr*cos(t);y=y0+rr*sin(t);}
border aaa2(t=x0+rr,2){x=t;y=0;};
border bbb(t=0,1){x=2;y=t;};

int m=5; mesh Th;
if(anew) Th = buildmesh (ccc(5*m) +ddd(3*m) + aaa1(2*m) + cercle(5*m)
              + aaa2(5*m) + bbb(2*m) );
      else Th = readmesh("Th_circle.mesh"); plot(Th,wait=0);

real dt=0.01, u0=2, err0=0.00625, pena=2;
fespace Wh(Th,P1);
```

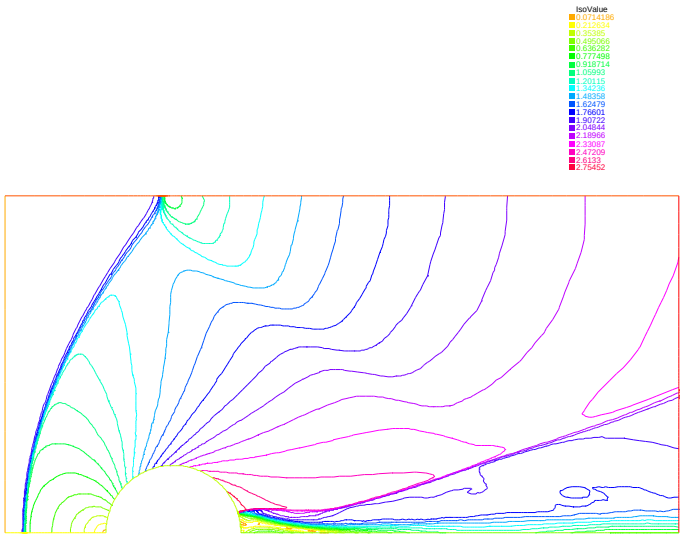


Figure 3.12: Pressure for a Euler flow around a disk at Mach 2 computed by (3.9)

```
fespace Vh(Th,P1);
Wh u,v,u1,v1,uh,vh;
Vh r,rh,r1;
macro dn(u) (N.x*dx(u)+N.y*dy(u) )              //    def the normal derivative

if(anew){ u1= u0; v1= 0; r1 = 1;}
else {
    ifstream g("u.txt");g>>u1[];
    ifstream gg("v.txt");gg>>v1[];
    ifstream ggg("r.txt");ggg>>r1[];
    plot(u1,ps="eta.eps", value=1,wait=1);
    err0=err0/10; dt = dt/10;
}

problem  eul(u,v,r,uh,vh,rh)
   = int2d(Th)(  (u*uh+v*vh+r*rh)/dt
                  + ((dx(r)*uh+ dy(r)*vh) - (dx(rh)*u + dy(rh)*v))
               )
 + int2d(Th)(-(rh*convect([u1,v1],-dt,r1) + uh*convect([u1,v1],-dt,u1)
                + vh*convect([u1,v1],-dt,v1))/dt)
  +int1d(Th,6)(rh*u)                                    //    +int1d(Th,1)(rh*v)
 + on(2,r=0) + on(2,u=u0) + on(2,v=0);

int j=80;
for(int k=0;k<3;k++)
{
    if(k==20){ err0=err0/10; dt = dt/10; j=5;}
    for(int i=0;i<j;i++){
```

```
        eul; u1=u; v1=v; r1=abs(r);
         cout<<"k="<<k<<"  E="<<int2d(Th)(u^2+v^2+r)<<endl;
         plot(r,wait=0,value=1);
}
Th = adaptmesh (Th,r, nbvx=40000,err=err0,
      abserror=1,nbjacoby=2, omega=1.8,ratio=1.8, nbsmooth=3,
      splitpbedge=1, maxsubdiv=5,rescaling=1) ;
 plot(Th,wait=0);
 u=u;v=v;r=r;

savemesh(Th,"Th_circle.mesh");
ofstream f("u.txt");f<<u[];
ofstream ff("v.txt");ff<<v[];
ofstream fff("r.txt");fff<<r[];
r1 = sqrt(u*u+v*v);
plot(r1,ps="mach.eps", value=1);
r1=r;
}
```

3.14 Classification of the equations

Summary *It is usually not easy to determine the type of a system. Yet the approximations and algorithms suited to the problem depend on its type:*

- *Finite Elements compatible (LBB conditions) for elliptic systems*
- *Finite difference on the parabolic variable and a time loop on each elliptic subsystem of parabolic systems; better stability diagrams when the schemes are implicit in time.*
- *Upwinding, Petrov-Galerkin, Characteristics-Galerkin, Discontinuous-Galerkin, Finite Volumes for hyperbolic systems plus, possibly, a time loop.*

When the system changes type, then expect difficulties (like shock discontinuities)!

Elliptic, parabolic and hyperbolic equations A partial differential equation (PDE) is a relation between a function of several variables and its derivatives.

$$F(\varphi(x), \frac{\partial\varphi}{\partial x_1}(x), \cdots, \frac{\partial\varphi}{\partial x_d}(x), \frac{\partial^2\varphi}{\partial x_1^2}(x), \cdots, \frac{\partial^m\varphi}{\partial x_d^m}(x)) = 0 \quad \forall x \in \Omega \subset \mathcal{R}^d.$$

The range of x over which the equation is taken, here Ω, is called the *domain* of the PDE. The highest derivation index, here m, is called the *order*. If F and φ are vector valued functions, then the PDE is actually a *system* of PDEs.
Unless indicated otherwise, here by convention *one* PDE corresponds to one scalar valued F and φ. If F is linear with respect to its arguments, then the PDE is said to be *linear*.
The general form of a second order, linear scalar PDE is $\frac{\partial^2\varphi}{\partial x_i \partial x_j}$ and $A : B$ means $\sum_{i,j=1}^{d} a_{ij} b_{ij}$.

$$\alpha\varphi + a \cdot \nabla\varphi + B : \nabla(\nabla\varphi) = f \quad \text{in} \quad \Omega \subset \mathcal{R}^d,$$

where $f(x), \alpha(x) \in \mathcal{R}, a(x) \in \mathcal{R}^d, B(x) \in \mathcal{R}^{d\times d}$ are the PDE *coefficients*. If the coefficients are independent of x, the PDE is said to have *constant coefficients*.

To a PDE we associate a quadratic form, by replacing φ by 1, $\partial\varphi/\partial x_i$ by z_i and $\partial^2\varphi/\partial x_i\partial x_j$ by z_iz_j, where z is a vector in $\mathcal{R}^d$:

$$\alpha + a \cdot z + z^T Bz = f.$$

If it is the equation of an ellipse (ellipsoid if $d \geq 2$), the PDE is said to be *elliptic*; if it is the equation of a parabola or a hyperbola, the PDE is said to be *parabolic* or *hyperbolic*. If $A \equiv 0$, the degree is no longer 2 but 1, and for reasons that will appear more clearly later, the PDE is still said to be hyperbolic.

These concepts can be generalized to systems, by studying whether or not the polynomial system $P(z)$ associated with the PDE system has branches at infinity (ellipsoids have no branches at infinity, paraboloids have one, and hyperboloids have several).
If the PDE is not linear, it is said to be *non linear*. Those are said to be locally elliptic, parabolic, or hyperbolic according to the type of the linearized equation.
For example, for the non linear equation

$$\frac{\partial^2\varphi}{\partial t^2} - \frac{\partial\varphi}{\partial x}\frac{\partial^2\varphi}{\partial x^2} = 1,$$

we have $d = 2, x_1 = t, x_2 = x$ and its linearized form is:

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial u}{\partial x}\frac{\partial^2\varphi}{\partial x^2} - \frac{\partial\varphi}{\partial x}\frac{\partial^2 u}{\partial x^2} = 0,$$

which for the unknown u is locally elliptic if $\frac{\partial\varphi}{\partial x} < 0$ and locally hyperbolic if $\frac{\partial\varphi}{\partial x} > 0$.

Examples Laplace's equation is elliptic:

$$\Delta\varphi \equiv \frac{\partial^2\varphi}{\partial x_1^2} + \frac{\partial^2\varphi}{\partial x_2^2} + \cdots + \frac{\partial^2\varphi}{\partial x_d^2} = f, \quad \forall x \in \Omega \subset \mathcal{R}^d.$$

The *heat* equation is parabolic in $Q = \Omega\times]0, T[\subset \mathcal{R}^{d+1}$:

$$\frac{\partial\varphi}{\partial t} - \mu\Delta\varphi = f \quad \forall x \in \Omega \subset \mathcal{R}^d, \quad \forall t \in]0, T[.$$

If $\mu > 0$, the *wave* equation is hyperbolic:

$$\frac{\partial^2\varphi}{\partial t^2} - \mu\Delta\varphi = f \quad \text{in} \quad Q.$$

The *convection diffusion* equation is parabolic if $\mu \neq 0$ and hyperbolic otherwise:

$$\frac{\partial\varphi}{\partial t} + a\nabla\varphi - \mu\Delta\varphi = f.$$

The *biharmonic* equation is elliptic:

$$\Delta(\Delta\varphi) = f \quad \text{in} \quad \Omega.$$

Boundary conditions A relation between a function and its derivatives is not sufficient to define the function. Additional information on the boundary $\Gamma = \partial\Omega$ of Ω, or on part of Γ is necessary. Such information is called a *boundary condition*. For example,

$$\varphi(x) \text{ given, } \forall x \in \Gamma,$$

is called a *Dirichlet boundary condition*. The *Neumann* condition is

$$\frac{\partial \varphi}{\partial n}(x) \text{ given on } \Gamma \quad (\text{or } n \cdot B\nabla\varphi, \text{given on } \Gamma \text{ for a general second order PDE})$$

where n is the normal at $x \in \Gamma$ directed towards the exterior of Ω (by definition $\frac{\partial \varphi}{\partial n} = \nabla\varphi \cdot n$). Another classical condition, called a *Robin* (or *Fourier*) condition is written as:

$$\varphi(x) + \beta(x)\frac{\partial \varphi}{\partial n}(x) \text{ given on } \Gamma.$$

Finding a set of boundary conditions that defines a unique φ is a difficult art.
In general, an elliptic equation is well posed (*i.e.* φ is unique) with one Dirichlet, Neumann or Robin conditions on the whole boundary.
Thus, Laplace's equations is well posed with a Dirichlet or Neumann condition but also with

$$\varphi \text{ given on } \Gamma_1, \quad \frac{\partial \varphi}{\partial n} \text{ given on } \Gamma_2, \quad \Gamma_1 \cup \Gamma_2 = \Gamma, \quad \dot{\Gamma_1} \cap \dot{\Gamma_2} = \emptyset.$$

Parabolic and hyperbolic equations rarely require boundary conditions on all of $\Gamma\times]0, T[$. For instance, the heat equation is well posed with

$$\varphi \text{ given at } t = 0 \text{ and Dirichlet or Neumann or mixed conditions on } \partial\Omega.$$

Here t is time so the first condition is called an initial condition. The whole set of conditions are also called Cauchy conditions.
The wave equation is well posed with

$$\varphi \text{ and } \frac{\partial \varphi}{\partial t} \text{ given at } t = 0 \text{ and Dirichlet or Neumann or mixed conditions on } \partial\Omega.$$

Chapter 4

Syntax

4.1 Data Types

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

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

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

Example 4.1

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

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

4.2 List of major types

bool is used for logical expression and flow-control. The result of a comparison is a boolean type as in

```
bool fool=(1<2);
```

which makes fool to be true. Similar examples can be built with ==, <=, >=, <, >, ! =.

int declares an integer.

string declare the variable to store a text enclosed within double quotes, such as:

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

real declares the variable to store a number such as “12.345”.

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

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

Here’s the output;

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

ofstream to declare an output file .

ifstream to declare an input file .

real[int] declares a variable that stores multiple real numbers with integer index.

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

This produces the output;

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

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

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

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

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

Remark that the function's type is given by the expression's type. Raising functions to a numerical power is done, for instance, by `x^1, y^0.23`.

mesh creates the triangulation, see Section 5.

fespace defines a new type of finite element space, see Section Section 6.

problem declares the weak form of a partial differential problem without solving it.

solve declares a problem and solves it.

varf defines a full variational form.

matrix defines a sparse matrix.

4.3 Global Variables

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

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

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

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

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

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

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

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

lenEdge gives the length of the current edge

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

hTriangle gives the size of the current triangle

nuTriangle gives the index of the current triangle (int value).

nuEdge gives the index of the current edge in the triangle (int value).

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

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

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

cin is the standard input device (default is keyboard). (`istream`value).

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

true means "true" in `bool` value.

false means "false" in `bool` value.

pi is the `real`value approximation value of π.

4.4 System Commands

Here is how to show all the types, and all the operator and functions of a `FreeFem++` program:

```
dumptable(cout);
```

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

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

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

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

Another useful system command is assert() to make sure something is true.

```
assert(version>=1.40);
```

4.5 Arithmetic

On integers, +, −, ∗ express the usual arithmetic summation (plus), subtraction (minus) and multiplication (times), respectively,The operators / and % yield the quotient and the remainder from the division of the first expression by the second. If the second number of / or % is zero the behavior is undefined. The *maximum* or *minimum* of two integers a, b are obtained by `max(a,b)` of `min(a,b)`. The power a^b of two integers a, b is calculated by writing `a^b`. The classical `C++` "arithmetical if" expression `a ? b : c` is equal to the value of expression `b` if the value of expression `a` is true otherwise is equal to value of expression `c`.

Example 4.2 *Calculations with the integers*

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

b=0;
cout <<a<<"/0"<<" is "<< a/b << endl;
cout <<a<<"%0"<<" is "<< a%b << endl;
```

produce the following result:

```
plus, minus of 12 and 5 are 17, 7
multiplication, quotient of them are 60, 2
remainder from division of 12 by 5 is 2
the minus of 12 is -12
12 plus -5 need bracket :12+(-5)=7
max and min of 12 and 5 is 12,5
5th power of 12 is 248832
min == (a < b ? a : b)   is 5
12/0 : long long long
Fatal error : ExecError  Div by 0 at exec line  9
Exec error : exit
```

By the relation *integer* $\subset$ *real*, the operators "+, −, ∗, /, %" and " **max**, **min**, ^" are also applicable on real-typed variables. However, % calculates the remainder of the integer parts of two real numbers.
The following are examples similar to Example 4.2

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

It gives the following output:

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

By the relation

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

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

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

The imaginary and real parts of complex number `z` is obtained by **`imag`** and **`real`**. The conjugate of $a + bi$ (a, b are real) is defined by $a - bi$, which is denoted by **`conj`**`(a+b*1i)` in `FreeFem++` .

The complex number $z = a + ib$ is considered as the pair (a, b) of real numbers a, b. We can attach to it the point (a, b) in the Cartesian plane where the x-axis is for the real part and the y-axis for the imaginary part. The same point (a, b) has a representation with polar coordinate (r, ϕ), So z his also $z = r(\cos\phi + i\sin\phi)$, $r = \sqrt{a^2 + b^2}$ and $\phi = \tan^{-1}(b/a)$; r is called the *modulus* and ϕ the *argument* of z. In the following example, we shall show them using `FreeFem++` programming, and *de Moivre's formula* $z^n = r^n(\cos n\phi + i\sin n\phi)$.

Example 4.3

```
real a=2.45, b=5.33;
complex  z1=a+b*1i, z2 = a+sqrt(2.)*1i;
func string pc(complex z)         //    printout complex to (real)+i(imaginary)
{
   string r = "("+real(z);
   if (imag(z)>=0) r = r+"+";
   return r+imag(z)+"i)";
}
                      //    printout complex to |z|*(cos(arg(z))+i*sin(arg(z)))
func string toPolar(complex z)
{
   return abs(z)+"*(cos("+arg(z)+")+i*sin("+arg(z)+"))";
}
cout <<"Standard output of the complex "<<pc(z1)<<" is the pair "
     <<z1<<endl;
cout <<"Plus, minus of "<<pc(z1)<<" and "<<pc(z2)<<" are "<< pc(z1+z2)
     <<", "<< pc(z1-z2) << endl;
cout <<"Multiplication, quotient of them are "<<pc(z1*z2)<<", "
     <<pc(z1/z2)<< endl;
cout <<"Real/imaginary part of "<<pc(z1)<<" is "<<real(z1)<<", "
     <<imag(z1)<<endl;
cout <<"Absolute of "<<pc(z1)<<" is "<<abs(z1)<<endl;
cout <<pc(z2)<<" = "<<toPolar(z2)<<endl;
cout <<"  and polar("<<abs(z2)<<","<<arg(z2)<<") = "
     << pc(polar(abs(z2),arg(z2)))<<endl;
cout <<"de Moivre's formula: "<<pc(z2)<<"^3 = "<<toPolar(z2^3)<<endl;
cout <<"conjugate of "<<pc(z2)<<" is "<<pc(conj(z2))<<endl;
cout <<pc(z1)<<"^"<<pc(z2)<<" is "<< pc(z1^z2) << endl;
```

Here's the output from Example 4.3

```
Standard output of the complex (2.45+5.33i) is the pair (2.45,5.33)
Plus, minus of (2.45+5.33i) and (2.45+1.41421i) are (4.9+6.74421i), (0+3.91579i)
Multiplication, quotient of them are (-1.53526+16.5233i), (1.692+1.19883i)
Real/imaginary part of (2.45+5.33i) is 2.45, 5.33
Absolute of (2.45+5.33i) is 5.86612
```

```
(2.45+1.41421i) = 2.82887*(cos(0.523509)+i*sin(0.523509))
  and polar(2.82887,0.523509) = (2.45+1.41421i)
de Moivre's formula: (2.45+1.41421i)^3
                        = 22.638*(cos(1.57053)+i*sin(1.57053))
conjugate of (2.45+1.41421i) is (2.45-1.41421i)
(2.45+5.33i)^(2.45+1.41421i) is (8.37072-12.7078i)
```

4.6 One Variable Functions

Fundamental functions are built into FreeFem++ .

The *power function* x^ y = **pow**(x,y)= x^y; the *exponent function* **exp**(x) (= e^x); the *logarithmic function* **log**(x)(= $\ln x$) or **log10**(x) (= $\log_{10} x$); the *trigonometric functions* **sin**(x), **cos**(x), **tan**(x) depending on angles measured by *radian*; the inverse of $\sin x$, $\cos x$, $\tan x$ called *circular function* or also call *inverse trigonometric function* **asin**(x) (=arcsin x), **acos**(x) (=arccos x), **atan**(x) (=arctan x); the **atan2**(x,y) function computes the principal value of the arc tangent of y/x, using the signs of both arguments to determine the quadrant of the return value;

the *hyperbolic function*,

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

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

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

The real function to round to integer are **floor**(x) round to largest integral value not greater than x, **ceil**(x) round to smallest integral value not less than x, **rint**(x) functions return the integral value nearest to x (according to the prevailing rounding mode) in floating-point format).

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

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

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

Example 4.4 *The following s an example where an elementary function is used to build the border of a domain.* Cardioid

```
real b = 1.;
real a = b;
```

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

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

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

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

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

Random Functions from Mersenne Twister (see page `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html` for full detail). It is A very fast random number generator Of period $2^{219937} - 1$, and the functions are:

- `randint32()` generates unsigned 32-bit integers.
- `randint31()` generates unsigned 31-bit integers.
- `randreal1()` generates uniform real in [0, 1] (32-bit resolution).
- `randreal2()` generates uniform real in [0, 1) (32-bit resolution).
- `randreal3()` generates uniform real in (0, 1) (32-bit resolution).
- `randres53()` generates uniform real in [0, 1) with 53-bit resolution.
- `randinit(seed )` initializes the state vector by using one 32-bit integer "seed", which may be zero.

Library Functions form the mathematical library (version 2.17).

- the functions `j0(x)`, `j1(x)`, `jn(n,x)`, `y0(x)`, `y1(x)`, `yn(n,x)` are the Bessel functions of first and second kind.

  The functions `j0(x)` and `j1(x)` compute the Bessel function of the first kind of the order 0 and the order 1, respectively; the function `jn(n, x)` computes the Bessel function of the first kind of the integer order n.

  The functions `y0(x)` and `y1(x)` compute the linearly independent Bessel function of the second kind of the order 0 and the order 1, respectively, for the positive integer value x (expressed as a real); the function `yn(n, x)` computes the Bessel function of the second kind for the integer order n for the positive integer value x (expressed as a real).

- the function `tgamma(x)` calculates the Γ function of x. `lgamma(x)` calculates the natural logorithm of the absolute value of the Γ function of x.
- The `erf(x)` function calculates the error function, where $\mathtt{erf}(x) = \frac{2}{\sqrt{(pi)}} \int_0^x exp(-t^2)dt$. The `erfc(x)` = function calculates the complementary error function of x, i.e. $\mathtt{erfc}(x) = 1 - \mathtt{erf}(x)$.

4.7 Functions of Two Variables

4.7.1 Formula

The general form of real functions with two independent variables x, y is usually written as $z = f(x,y)$. In `FreeFem++` , `x` and `y` are reserved word in Section 4.3. When two independent variables are `x` and `y`, we can define a function without argument, for example

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

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

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

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

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

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

4.7.2 FE-function

Arithmetic built-in functions are able to construct a new function by the four arithmetic operations and superposition of them (see *elementary functions*), which are called *formulas* to distinguish from FE-functions. We can add *new formulas* easily, if we want. Here, FE-function is an element of finite element space (real or complex) (see Section Section 6). Or to put it another way: *formulas* are the mathematical expressions combining its numerical analogs, but it is independent of meshes (triangulations).
Also, in `FreeFem++`, we can give an arbitrary symbol to FE-function combining numerical calculation by FEM. The interpolation of a formula-function f in a FE-space is done as in

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

The construction of `fh` (=f_h) is explained in Section 6.

Note 4.1 *The command* **`plot`** *is valid only for real FE-functions.*

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

```
mesh Th=square(20,20,[-pi+2*pi*x,-pi+2*pi*y]);                    //    ]-π,π[^2
fespace Vh(Th,P2);
func z=x+y*1i;                                                    //    z = x + iy
func f=imag(sqrt(z));                                             //    f = ℑ √z
func g=abs( sin(z/10)*exp(z^2/10) );                        //    g = |sin z/10 exp z^2/10|
Vh fh = f; plot(fh);                                  //    Fig. 4.1 isovalue of f
Vh gh = g; plot(gh);                                  //    Fig. 4.2 isovalue of g
```

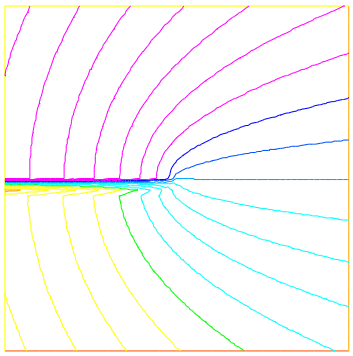


Figure 4.1: $\Im\sqrt{z}$ has branch

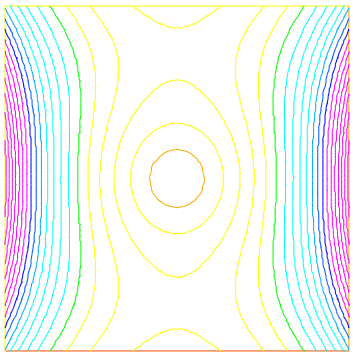


Figure 4.2: $|\sin(z/10)\exp(z^2/10)|$

4.8 Arrays

An *array* stores multiple objects, and there are 2 kinds of arrays: The first is the *vector* that is arrays with *integer indices* and arrays with *string indices*.
In the first case, the size of this array must be know at the execution time, and the implementation is done with the `KN<>` class so all the vector operator of `KN<>` are implemented. The sample

```
real [int] tab(10), tab1(10);                          //    2 array of 10 real
real [int] tab2;                                       //    bug array with no size
tab = 1.03;                                            //    set all the array to 1.03
tab[1]=2.15;
cout << tab[1] << " " << tab[9] << " size of tab = "
     << tab.n << " min: " << tab.min << "  max:" << tab.max
```

```
     << " sum : "   << tab.sum <<   endl;                                  //
tab.resize(12);                                  //    change the size of array tab
                                              //    to 12 with preserving first value
tab(10:11)=3.14;                                          //    set unset value
cout <<" resize tab: " <<  tab << endl;
real [string] tt;
tt["+"]=1.5;
cout<<tt["a"]<<"  "<<tt["+"]<<endl;
real[int]  a(5),b(5),c(5),d(5);
a = 1;
b = 2;
c = 3;
a[2]=0;
d = ( a ? b : c );        //    for i = 0, n-1 :  d[i] = a[i] ?  b[i] :  c[i] ,
cout << " d = ( a ? b : c )  is " << d << endl;
d = ( a ? 1 : c ); //    for i = 0, n-1:  d[i] = a[i] ?  1 :  c[i] , (v2.23-1)
d = ( a ? b : 0 ); //    for i = 0, n-1:  d[i] = a[i] ?  b[i] :  0 , (v2.23-1)
d = ( a ? 1 : 0 ); //    for i = 0, n-1:  d[i] = a[i] ?  0 :  1 ,    (v2.23-1)
tab2.sort;                                //    sort the array tab2 (version 2.18)
```

produce the output

```
2.15 1.03 size of tab = 10 min: 1.03  max:2.15 sum : 11.42
 resize tab: 12
        1.03    2.15    1.03    1.03    1.03
        1.03    1.03    1.03    1.03    1.03
        3.14    3.14
 0  1.5
 d = ( a ? b : c )  is  5
          3       3       2       3       3
```

the all integer array operator are :

```
{
int N=5;
real[int] a(N),b(N),c(N);
a =1;
a(0:4:2) = 2;
a(3:4) = 4;
cout <<" a = " << a << endl;
b = a+ a;
cout <<" b = a+a : " << b << endl;
b += a;
cout <<" b += a : " << b << endl;
b += 2*a;
cout <<" b += 2*a : " << b << endl;
b /= 2;
cout <<" b /= 2 : " << b << endl;
b *= a;                                                    //    same b = b .* a
cout << "b*=a; b =" << b << endl;
b /= a;                                                    //    same b = b ./ a
cout << "b/=a; b =" << b << endl;
c = a+b;
```

```
cout << " c =a+b : c=" << c << endl;
c = 2*a+4*b;
cout << " c =2*a+4b : c= " << c << endl;
c = a+4*b;
cout << " c =a+4b : c= " << c << endl;
c = -a+4*b;
cout << " c =-a+4b : c= " << c << endl;
c = -a-4*b;
cout << " c =-a-4b : c= " << c << endl;
c = -a-b;
cout << " c =-a-b : c= " << c << endl;

c = a .* b;
cout << " c =a.*b  : c= " << c << endl;
c = a ./ b;
cout << " c =a./b  : c= " << c << endl;
c = 2 * b;
cout << " c =2*b   : c= " << c << endl;
c =  b*2 ;
cout << " c =b*2   : c= " << c << endl;

/*  this operator do not exist
c =  b/2 ;
cout << " c =b/2   : c= " << c << endl;
*/

                                                    //    ---- the methods --
cout << " ||a||_1     = " <<  a.l1     << endl;                              //
cout << " ||a||_2     = " <<  a.l2     << endl;                              //
cout << " ||a||_infty = " <<  a.linfty << endl;                              //
cout << " sum a_i     = " <<  a.sum    << endl;                              //
cout << " max a_i     = " <<  a.max    << endl;                              //
cout << " min a_i     = " <<  a.min    << endl;                              //
cout << " a'*a        = " <<  (a'*a)   << endl;                              //
cout << " a quantile 0.2 = " <<  a.quantile(0.2) << endl;                    //
}
```

produce the output

```
5
          3       3       2       3       3

==   3       3       2       3       3
 a = 5
          2       1       2       4       4

 b = a+a : 5
          4       2       4       8       8

 b += a : 5
          6       3       6      12      12

 b += 2*a : 5
         10       5      10      20      20

 b /= 2 : 5
```

```
          5     2.5       5      10      10

b*=a; b =5
         10     2.5      10      40      40

b/=a; b =5
          5     2.5       5      10      10

 c =a+b : c=5
          7     3.5       7      14      14

 c =2*a+4b : c= 5
         24      12      24      48      48

 c =a+4b : c= 5
         22      11      22      44      44

 c =-a+4b : c= 5
         18       9      18      36      36

 c =-a-4b : c= 5
        -22     -11     -22     -44     -44

 c =-a-b : c= 5
         -7     -3.5     -7     -14     -14

 c =a.*b  : c= 5
         10     2.5      10      40      40

 c =a./b  : c= 5
        0.4     0.4     0.4     0.4     0.4

 c =2*b   : c= 5
         10       5      10      20      20

 c =b*2   : c= 5
         10       5      10      20      20

 ||a||_1     = 13
 ||a||_2     = 6.403124237
 ||a||_infty = 4
 sum a_i     = 13
 max a_i     = 4
 min a_i     = 1
 a'*a        = 41
 a quantile 0.2 = 2
```

Note 4.2 *Quantiles are points taken at regular intervals from the cumulative distribution function of a random variable. Here the random is the array value.*
This statisticial function `a.quantile(q)` *and commute v of an array a of size n for a given number* $q \in]0, 1[$ *such than*

$$\#\{i/a[i] < v\} \sim q * n$$

which is equivalent to $v = a[q * n]$ *when the array a is sorted.*

Example of array with renumbering (version 2.3 or better) . The renumbering is always given by an integer array, and if a value in the array is negative, the mining is not image, so we do not set the value.

```
int[int] I=[2,3,4,-1,0];       //    the integer mapping to set the renumbering
b=c=-3;
b= a(I);                    //    for( i=0;i<b.n;i++) if(I[i] >=0) b[i]=a[I[i]];
c(I)= a;                    //    for( i=0;i<I.n;i++) if(I[i] >=0) C(I[i])=a[i];
cout << " b = a(I) : " << b << "\n  c(I) = a " << c << endl;
```

The output is

```
 b = a(I) : 5
          2        4        4       -3        2

  c(I) = a 5
          4       -3        2        1        2
```

4.8.1 Arrays with two integer indices versus matrix

Some example transform full matrices in sparse matrices.

```
  int N=3,M=4;

  real[int,int] A(N,M);
  real[int]  b(N),c(M);
  b=[1,2,3];
  c=[4,5,6,7];

  complex[int,int]  C(N,M);
  complex[int]  cb=[1,2,3],cc=[10i,20i,30i,40i];


  b=[1,2,3];

  int [int] I=[2,0,1];
  int [int] J=[2,0,1,3];

  A=1;                                                  //    set the all matrix
  A(2,:) = 4;                                             //    the full line 2
  A(:,1) = 5;                                           //    the full column 1
  A(0:N-1,2) = 2;                                         //    set the column 2
  A(1,0:2) = 3;                                  //    set the line 1 from 0 to 2

  cout << " A = " << A << endl;
                                                          //    outer product
  C  =  cb*cc';
  C +=  3*cb*cc';
  C -=  5i*cb*cc';
  cout << " C = " << C << endl;
                          //    the way to transform a array to a sparse matrix
  matrix B;
  B = A;
```

```
B=A(I,J);                                          //    B(i,j)= A(I(i),J(j))
B=A(I^-1,J^-1);                                    //    B(I(i),J(j))= A(i,j)

A = 2.*b*c';                                                //    outer product
cout << " A = " << A << endl;
B = b*c';                                    //    outer product B(i,j) = b(i)*c(j)
B = b*c';                                    //    outer product B(i,j) = b(i)*c(j)
B = (2*b*c')(I,J);                  //    outer product B(i,j) = b(I(i))*c(J(j))
B = (3.*b*c')(I^-1,J^-1);           //    outer product B(I(i),J(j)) = b(i)*c(j)
cout << "B = (3.*b*c')(I^-1,J^-1) =  " << B << endl;
```

the output is

```
b = a(I) : 5
         2        4        4       -3        2

 c(I) = a 5
         4       -3        2        1        2

A = 3 4
          1    5    2    1
          3    3    3    1
          4    5    2    4

C = 3 4
        (-50,-40) (-100,-80) (-150,-120) (-200,-160)
        (-100,-80) (-200,-160) (-300,-240) (-400,-320)
        (-150,-120) (-300,-240) (-450,-360) (-600,-480)

A = 3 4
          8   10   12   14
         16   20   24   28
         24   30   36   42
```

4.8.2 Matrix Operations

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

- `'` is unary right transposition of array, matrix in real case or Hermitian conjugation in complex case.
- `.*` is the term to term multiply operator.
- `./` is the term to term divide operator.

there are some compound operator:

- `^-1` is for solving the linear system (example: `b = A^-1 x`)
- `'*` is the compound of transposition and matrix product, so it is the dot product (example `real DotProduct=a'*b`) , in complex case you get the Hermitian product, so mathematically we have `a'*b`= $\overline{a}^T b$.

- `a*b'` is the outer product (example `matrix B=a'*b`)

  To set or get all the indices and coef of the sparse matrix A, let `I,J,C` respectively two `int[int]` array and a `real[int]` array. The three array defined the matrix as follow

$$A = \sum_k \mathtt{C[k]} M_{\mathtt{I[k]},\mathtt{J[k]}} \qquad \text{where} \qquad M_{ab} = (\delta_{ia}\delta_{jb})_{ij}$$

  and you have: M_{ab} is a basic matrix with the only non zero term $m_{ab} = 1$.

  You can write `[I,J,C]=A ;` to get all the term of the matrix `A` (the arrays are automaticaly resize), and `A=[I,J,C] ;` to change all the term matrix. remark the size of the matrix is with `n= I.max` and `m=J.max`. Remark you forget `I,J` when the build a diagonal matrix, and n, m of the matrix is

Example 4.5

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

Vh m0; m0[] = A*f[];
Vh m01; m01[] = A'*f[];
Vh m1; m1[] = f[].*g[];
Vh m2; m2[] = f[]./g[];
cout << "f = " << f[] << endl;
cout << "g = " << g[] << endl;
cout << "A = " << A << endl;
cout << "m0 = " << m0[] << endl;
cout << "m01 = " << m01[] << endl;
cout << "m1 = "<< m1[] << endl;
cout << "m2 = "<< m2[] << endl;
cout << "dot Product = "<< f[]'*g[] << endl;
cout << "hermitien Product = "<< ff[]'*gg[] << endl;
cout << "outer Product = "<< (A=ff[]*gg[]') << endl;
cout << "hermitien outer Product = "<< (AA=ff[]*gg[]') << endl;
real[int] diagofA(A.n);
  diagofA = A.diag;                          //    get the diagonal of the matrix
  A.diag = diagofA ;                         //    set the diagonal of the matrix
                                             //    version 2.17 or better ---
int[int] I(1),J(1); real[int] C(1);
[I,J,C]=A;          //    get of the sparse term of the matrix A (the array are
resized)
```

```
cout << " I= " << I << endl;
cout << " J= " << J << endl;
cout << " C= " << C << endl;
A=[I,J,C];                                                    //    set a new matrix
matrix D=[diagofA] ;     //    set a diagonal matrix D from the array diagofA.
cout << " D = " << D << endl;
```

On the triangulation of Figure 2.4 this produce the following:

$$
\begin{aligned}
A &= \begin{bmatrix}
10^{30} & 0.5 & 0. & 30. & -2.5 & 0. \\
0. & 10^{30} & 0.5 & 0. & 0.5 & -2.5 \\
0. & 0. & 10^{30} & 0. & 0. & 0.5 \\
0.5 & 0. & 0. & 10^{30} & 0. & 0. \\
-2.5 & 0.5 & 0. & 0.5 & 10^{30} & 0. \\
0. & -2.5 & 0. & 0. & 0.5 & 10^{30}
\end{bmatrix} \\
\{v\} = \texttt{f[]} &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0.5 & 1 \end{pmatrix}^T \\
\{w\} = \texttt{g[]} &= \begin{pmatrix} 0 & 1 & 1.2\times10^{-16} & 0 & 1 & 1.2\times10^{-16} \end{pmatrix} \\
\texttt{A*f[]} &= \begin{pmatrix} -1.25 & -2.25 & 0.5 & 0 & 5\times10^{29} & 10^{30} \end{pmatrix}^T \quad (= A\{v\}) \\
\texttt{A'*f[]} &= \begin{pmatrix} -1.25 & -2.25 & 0 & 0.25 & 5\times10^{29} & 10^{30} \end{pmatrix}^T \quad (= A^T\{v\}) \\
\texttt{f[].*g[]} &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0.5 & 1.2\times10^{-16} \end{pmatrix}^T \quad = (v_1 w_1 \quad \cdots \quad v_M w_M)^T \\
\texttt{f[]./g[]} &= \begin{pmatrix} \texttt{-NaN} & 0 & 0 & \texttt{-NaN} & 0.5 & 8.1\times10^{15} \end{pmatrix}^T \quad = (v_1/w_1 \cdots v_M/w_M)^T \\
\texttt{f[]'*g[]} &= 0.5 \quad (= \{v\}^T\{w\} = \{v\}\cdot\{w\})
\end{aligned}
$$

The output of the I, J, C array:

```
 I= 18
          0       0       0       1       1
          1       1       2       2       3
          3       4       4       4       4
          5       5       5
 J= 18
          0       1       4       1       2
          4       5       2       5       0
          3       0       1       3       4
          1       4       5
 C= 18
        1e+30    0.5     -2.5    1e+30    0.5
        0.5     -2.5     1e+30   0.5      0.5
        1e+30   -2.5     0.5     0.5      1e+30
        -2.5     0.5     1e+30
```

The output of a diagonal sparse matrix D (Warning du to fortran interface the indices start on the output at one, but in `FreeFem++` in index as in `C` begin at zero);

```
 D = # Sparce Matrix (Morse)
# first line: n m (is symmetic) nbcoef
# after for each nonzero coefficient:   i j a_ij where (i,j) \in  {1,...,n}x{1,...,m}
6 6 1  6
        1         1 1.000000000000000199e+30
        2         2 1.000000000000000199e+30
        3         3 1.000000000000000199e+30
```

```
4         4 1.000000000000000199e+30
5         5 1.000000000000000199e+30
6         6 1.000000000000000199e+30
```

Note 4.3 *The operators* `^-1` *cannot be used to create a matrix; the following gives an error*

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

4.8.3 Other arrays

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

Example 4.6 *In the following example, Poisson's equation is solved for 3 different given functions* $f = 1$, $\sin(\pi x)\cos(\pi y)$, $|x-1||y-1|$, *whose solutions are stored in an array of FE function.*

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

For the second case, it is just a map of the STL[1][25] so no operations on vector are allowed, except the selection of an item .
The transpose or Hermitian conjugation operator is `'` like Matlab or Scilab, so the way to compute the dot product of two array `a,b` is `real ab= a'*b`.

```
int i;
real [int] tab(10), tab1(10);                        //    2 array of 10 real
    real [int] tab2;                              //    Error:  array with no size
tab = 1;                                             //    set all the array to 1
tab[1]=2;
cout << tab[1] << " " << tab[9] << " size of tab = "
     << tab.n << " " << tab.min << " " << tab.max << " " << endl;
tab1=tab; tab=tab+tab1; tab=2*tab+tab1*5; tab1=2*tab-tab1*5;
tab+=tab; cout << " dot product " << tab'*tab << endl;              //    ttab tab
cout << tab << endl; cout << tab[1] << " "
<< tab[9] <<  endl; real[string] map;                 //    a dynamic array
for (i=0;i<10;i=i+1)
  {
```

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

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

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

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

4.9 Loops

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

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

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

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

The while-loop

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

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

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

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

Example 4.7

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

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

4.10 Input/Output

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

Note 4.4 *The file is closed at the exit of the enclosing block,*

Example 4.8

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

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

  cout << i << endl;
```

We add function to format the output.

- `int nold=f.precision(n)` Sets the number of digits printed to the right of the decimal point. This applies to all subsequent floating point numbers written to that output stream. However, this won't make floating-point "integers" print with a decimal point. It's necessary to use fixed for that effect.
- `f.scientific` Formats floating-point numbers in scientific notation (`d.dddEdd`)
- `f.fixed` Used fixed point notation (`d.ddd`) for floating-point numbers. Opposite of scientific.

- `f.showbase` Converts insertions to an external form that can be read according to the C++ lexical conventions for integral constants. By default, showbase is not set.
- `f.noshowbase` unset `showbase` flags
- `f.showpos` inserts a plus sign (+) into a decimal conversion of a positive integral value.
- `f.noshowpos` unset `showpos` flags
- `f.default` reset all the previous flags (`fmtflags`) to the default expect precision.

Where `f` is output stream descriptor, for example `cout`.
Remark, all this method exept the first return the stream `f`, so you can write

```
cout.scientific.showpos << 3 << endl;
```

4.11 Exception handling

In the version `2.3` of `FreeFem++`, we add exception handing like in `C++`. But to day we only catch all the `C++` exception, and in the `C++` all the errors like `ExecError, assert, exit, ...` are exception so you can have some surprise with the memory management.
The exception handle all `ExecError`:

Example 4.9 *A simple example first to catch a division by zero:*

```
real a;
try {
  a=1./0.;
}
catch  (...)                        //    today only all exceptions are permitted
{
  cout << " Catch an ExecError " << endl;
  a =0;
}
```

The output is

```
1/0 : d d d
  current line = 3
Exec error :  Div by 0
   -- number :1
Try:: catch (...) exception
Catch an ExecError
```

Add a more realistic example:

Example 4.10 *A case with none invertible matrix:*

```
int nn=5        ;
mesh Th=square(nn,nn);
verbosity=5;
```

```
fespace Vh(Th,P1);                                          //    P1 FE space
Vh uh,vh;                                                 //    unkown and test function.
func f=1;                                                  //    right hand side function
func g=0;                                                 //    boundary condition function
real   cpu=clock();
problem laplace(uh,vh,solver=Cholesky,tolpivot=1e-6) =                    //
definion of the problem
            int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) )      //    bilinear form
          + int2d(Th)( -f*vh )                                  //    linear form
  ;

try {
  cout << " Try Cholesky \n";
  laplace;              //    solve the problem plot(uh); // to see the result
  cout << "-- lap Cholesky " << nn << "x" << nn << "  : " <<  -cpu+clock()
       << " s,  max =" << uh[].max << endl;
}
catch(...) {                                                  //    catch all
  cout << " Catch cholesky PB " << endl;
}
```

The output is

```
 -- square mesh : nb vertices  =36 ,  nb triangles = 50 ...
   Nb of edges on Mortars  = 0
   Nb of edges on Boundary = 20, neb = 20
    Nb Mortars 0
 number of real boundary edges 20
    Number of Edges                  = 85
    Number of Boundary Edges         = 20 neb = 20
    Number of Mortars  Edges         = 0
    Nb Of Mortars with Paper Def     = 0 Nb Of Mortars = 0 ...
 Nb Of Nodes = 36
 Nb of DF = 36
 Try Cholesky
   -- Change of Mesh 0  0x312e9e8
   Problem(): initmat 1 VF (discontinuous Galerkin) = 0
  -- SizeOfSkyline =210
   -- size of Matrix 196 Bytes skyline =1
  -- discontinous Galerkin  =0 size of Mat =196 Bytes
  --  int  in   Optimized = 1,    ...
 all
  -- boundary int   Optimized = 1,  all
ERREUR choleskypivot (35)= -1.23124e-13 < 1e-06
  current line = 28
Exec error : FATAL ERREUR dans ../femlib/MatriceCreuse_tpl.hpp
cholesky line:
   -- number :545
 catch an erreur in  solve  =>  set  sol = 0 !!!!!!!
Try:: catch (...) exception
 Catch cholesky PB
```

Chapter 5

Mesh Generation

5.1 Commands for Mesh Generation

border, buildmesh are explained.
All the examples in this section come from the files `mesh.edp` and `tablefunction.edp`.

5.1.1 Square

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

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

generate a 4×5 grid in the unit squre $[0,1]^2$ whose labels are shown in Fig. 5.1. If you want to

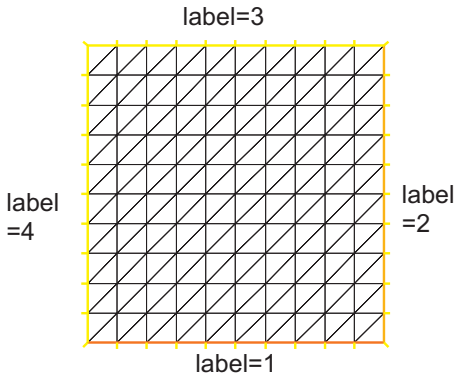


Figure 5.1: Boundary labels of the mesh by `square(10,10)`

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

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

Note 5.1 *Adding the parameter `flags=1`, will produce a Union Jack flag type of mesh.*

```
mesh Th=square(n,m,[x0+(x1-x0)*x,y0+(y1-y0)*y],flags=1);
```

5.1.2 Border

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

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

We can easily check the orientation by drawing the curve $t \mapsto (\varphi_x(t), \varphi_y(t))$, $t_0 \le t \le t_1$. If it is as in Fig. 5.2, then the domain lie on the shaded area, otherwise it lies on the opposite side
The boundaries Γ_j can only intersect at their end points.

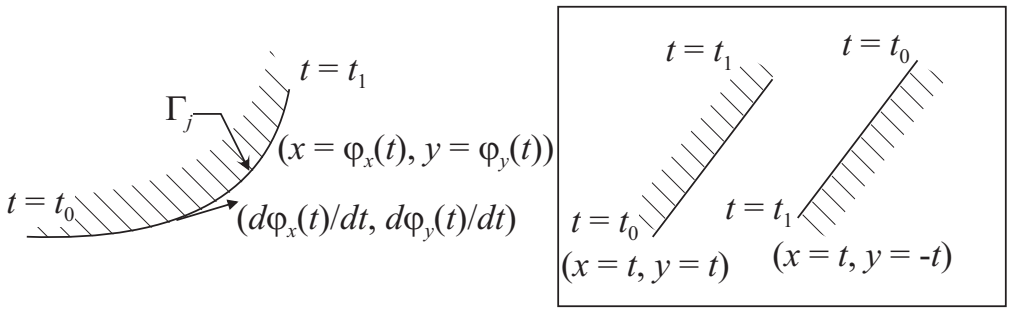


Figure 5.2: Orientation of the boundary defined by $(\phi_x(t), \phi_y(t))$

The general expression to define a triangulation with `buildmesh` is

$$\texttt{mesh Mesh\_Name = buildmesh}\Big(\Gamma_1(m_1) + \cdots + \Gamma_J(m_j)\Big);$$

where m_j are positive or negative numbers to indicate how may point should be put on Γ_j, $\Gamma = \cup_{j=1}^{J}\Gamma_J$. We can change the orientation of boundaries by changing the sign of m_j. The following example shows how to change the orientation. The example generates the unit disk with a small circular hole, and assign "1" to the unit disk ("2" to the circle inside). The boundary label must be non-zero, but it can also be omitted.

```
1: border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
2: border b(t=0,2*pi){ x=0.3+0.3*cos(t); y=0.3*sin(t);label=2;}
3: plot(a(50)+b(+30)) ;                    //    to see a plot of the border mesh
4: mesh Thwithouthole= buildmesh(a(50)+b(+30));
5: mesh Thwithhole   = buildmesh(a(50)+b(-30));
6: plot(Thwithouthole,wait=1,ps="Thwithouthole.eps");          //    figure 5.3
7: plot(Thwithhole,wait=1,ps="Thwithhole.eps");                //    figure 5.4
```

Note 5.2 *You must notice that the orientation is changed by "`b(-30)`" in 5th line. In 7th line, `ps="fileName"` is used to generate a postscript file identification that is shown on screen.*

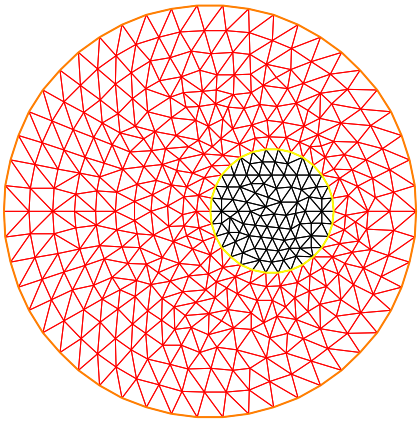


Figure 5.3: mesh without hole

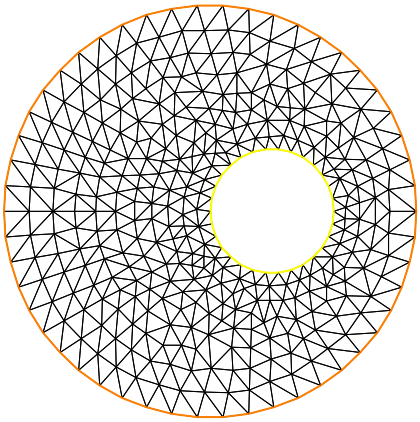


Figure 5.4: mesh with hole

5.1.3 Data Structure and Read/Write Statements for a Mesh

Users who want to use a triangulation made elsewhere should see the file structure of the file generated below:

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

the mesh is shown on Fig. 5.5.

The informations about `Th` are save in the file "mesh_sample.msh". whose structure is shown on Table 5.1.
There n_v denotes the number of vertices, n_t number of triangles and n_s the number of edges on boundary.
For each vertex q^i, $i = 1, \cdots, n_v$, we denote by (q^i_x, q^i_y) the x-coordinate and y-coordinate.
Each triangle $T_k, k = 1, \cdots, 10$ has three vertices q^{k_1}, q^{k_2}, q^{k_3} that are oriented counterclockwise.
The boundary consists of 10 lines L_i, $i = 1, \cdots, 10$ whose end points are q^{i_1}, q^{i_2}.
There are many mesh file formats available for communication with other tools such as emc2, modulef.. (see Section 12), The extension of a file gives the chosen type. More details can be found in the article by F. Hecht "bamg : a bidimentional anisotropic mesh generator" available from the FreeFem web site.

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". Here are some examples:

```
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;};
```

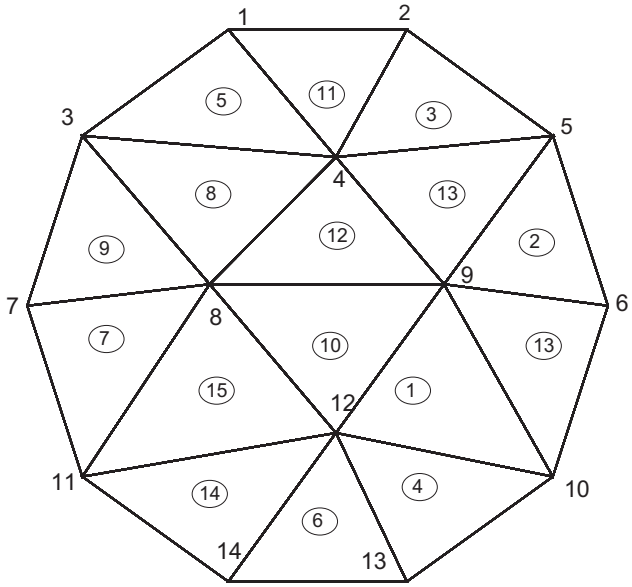


Figure 5.5: mesh by `buildmesh(C(10))`

In the left figure, we have the following.

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

$q^1 = (-0.309016994375,\ 0.951056516295)$
$\vdots \quad \vdots \quad \vdots$
$q^{14} = (-0.309016994375,\ -0.951056516295)$

The vertices of T_1 are q^9, q^{12}, q^{10}.
$\vdots \quad \vdots \quad \vdots$
The vertices of T_{16} are q^9, q^{10}, q^6.

The edge of 1st side L_1 are q^6, q^5.
$\vdots \quad \vdots \quad \vdots$
The edge of 10th side L_{10} are q^{10}, q^6.

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

Table 5.1: The structure of "mesh_sample.msh"

```
int n=10;
mesh th= buildmesh(floor(n)+right(n)+ceiling(n)+left(n));
savemesh(th,"toto.am_fmt");                    //    "formatted Marrocco" format
savemesh(th,"toto.Th");                              //    "bamg"-type mesh
savemesh(th,"toto.msh");                               //    freefem format
savemesh(th,"toto.nopo");                        //    modulef format  see [9]
mesh th2 = readmesh("toto.msh");                          //    read the mesh
```

Example 5.1 (Readmesh.edp)

```
border floor(t=0,1){ x=t; y=0; label=1;};   //    the unit square
border right(t=0,1){ x=1; y=t; label=5;};
border ceiling(t=1,0){ x=t; y=1; label=5;};
border left(t=1,0){ x=0; y=t; label=5;};
int n=10;
mesh th= buildmesh(floor(n)+right(n)+ceiling(n)+left(n));
savemesh(th,"toto.am_fmt");                    //    format "formated Marrocco"
savemesh(th,"toto.Th");                  //    format database db mesh "bamg"
savemesh(th,"toto.msh");                               //    format freefem
savemesh(th,"toto.nopo");                        //    modulef format  see [9]
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);
//    find u such that
//    u + Δu = g  in Ω ,
//    u = 0  on Γ1  and  ∂u/∂n = g  on Γ2
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)                                          //    ∂u/∂n = g  on Γ2
  + on(1,u=0) ;
plot (th2,u);
```

5.1.4 Mesh Connectivity

The following example explains methods to obtain mesh information.

```
{                                          //    get mesh information (version 1.37)
  mesh Th=square(2,2);
                                                          //    get data of the mesh
  int nbtriangles=Th.nt;
  cout << " nb of Triangles = " << nbtriangles << endl;
  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;

//     Th(i) return the vextex i of Th
//     Th[k] return the triangle k of Th

  fespace femp1(Th,P1);
  femp1 Thx=x,Thy=y;                          //    hack of get vertex coordinates
                                                  //    get vertices information :
  int nbvertices=Th.nv;
  cout << " nb of vertices = " << nbvertices << endl;
  for (int i=0;i<nbvertices;i++)
       cout << "Th(" <<i  << ") : "                                  //    << endl;
            << Th(i).x << " " << Th(i).y  << " " << Th(i).label    //   v 2.19
            << "       old method: " << Thx[][i] << " " << Thy[][i] << endl;

                           //    method to find information of point (0.55,0.6)

  int it00 = Th(0.55,0.6).nuTriangle;               //    then triangle number
  int nr00 = Th(0.55,0.6).region;                                    //

                                                        //    info of a triangle
  real area00 = Th[it00].area;                      //    new in version 2.19
  real nrr00 = Th[it00].region;                     //    new in version 2.19
  real nll00 = Th[it00].label;             //    same as region in this case.


//   Hack to get a triangle contening point x,y
//    or region number (old method)
//    ---------------------------------------------------
  fespace femp0(Th,P0);
  femp0 nuT;                        //    a P0 function to get triangle numbering
    for (int i=0;i<Th.nt;i++)
     nuT[][i]=i;
  femp0 nuReg=region;               //    a P0 function to get the region number
                                                                  //    inquire
  int it0=nuT(0.55,0.6);     //   number of triangle Th's contening (0.55,0,6);
  int nr0=nuReg(0.55,0.6); //   number of region of Th's contening (0.55,0,6);


 //    dump
 //    ---------------------------------------------------

  cout << "  point (0.55,0,6) :triangle number " << it00 << " " << it00
```

```
       << ", region = " << nr0 << " == " << nr00 << ",  area K " << area00 << endl;

}
```

the output is:

```
 -- square mesh : nb vertices  =9 ,  nb triangles = 8 ,  nb boundary edges 8
    Nb of Vertices 9 ,  Nb of Triangles 8
    Nb of edge on user boundary  8 ,  Nb of edges on true boundary  8
 number of real boundary edges 8
 nb of Triangles = 8
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
...
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
 Nb Of Nodes = 9
 Nb of DF = 9
 -- vector function's bound  0 1
 -- vector function's bound  0 1
 nb of vertices = 9
Th(0) : 0 0 4       old method: 0 0
Th(1) : 0.5 0 1       old method: 0.5 0
...
Th(7) : 0.5 1 3       old method: 0.5 1
Th(8) : 1 1 3       old method: 1 1
 Nb Of Nodes = 8
 Nb of DF = 8
```

5.1.5 The keyword "triangulate"

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

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

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

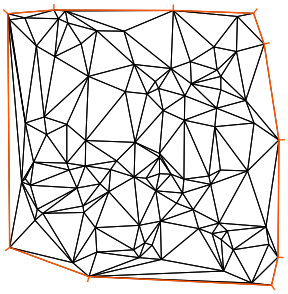


Figure 5.6: Delaunay mesh of the convex hull of point set in file xyf

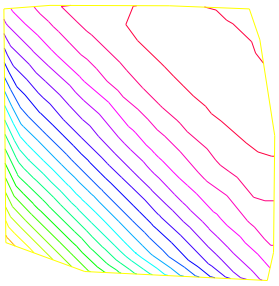


Figure 5.7: Isovalue of table function

```
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 5.6)

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

                                  //    reading the 3rd row to define the function
{ ifstream file("xyf");
   real xx,yy;
   for(int i=0;i<fxy.n;i++)
   file >> xx >>yy >> fxy[][i];                    //    to read third row only.
                                                 //    xx and yy are just skipped
}
plot(fxu,ps="xyf.eps");                  //    plot the function (see figure 5.7)
```

On new way to bluid a mesh from two array one the x values and the other for the y values (version 2.23-2):

```
Vhxy xx=x,yy=y;                        //    to set two arrays for the x's and y's
mesh Th=triangulate(xx[],yy[]);
```

5.2 Boundary FEM Spaces Built as Empty Meshes

To define a Finite Element space on boundary, we came up with the idea of a mesh with no internal points (call empty mesh). It can be useful when you have a Lagrange multiplier definied on the border.
So the function emptymesh remove all the internal points of a mesh except points is on internal boundaries.

```
{                                    //    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 5.8
}
```

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

```
{  //    new stuff 2004 emptymesh (version 1.40)
 //     -- useful to build Multiplicator space
 //     build a mesh without internal point
 //     of peusdo sub domain
 //     -----
  assert(version>=1.40);
  mesh Th=square(10,10);
  int[int] ssd(Th.nt);
  for(int i=0;i<ssd.n;i++)                 //    build the 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 emtpy with
                                        //    all edge e = T1 ∩ T2 and ssd[T1] ≠ ssd[T2]
  plot(Th,wait=1,ps="emptymesh-2.eps");                    //   see figure 5.9
  savemesh(Th,"emptymesh-2.msh");
}
```

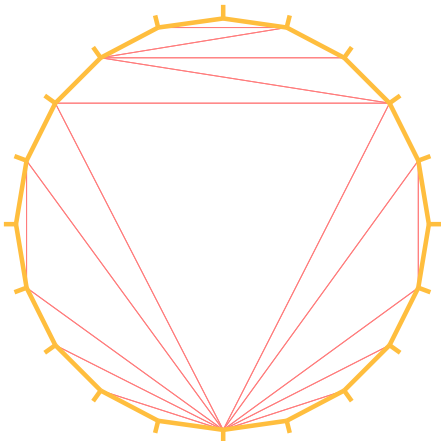


Figure 5.8: The empty mesh with boundary

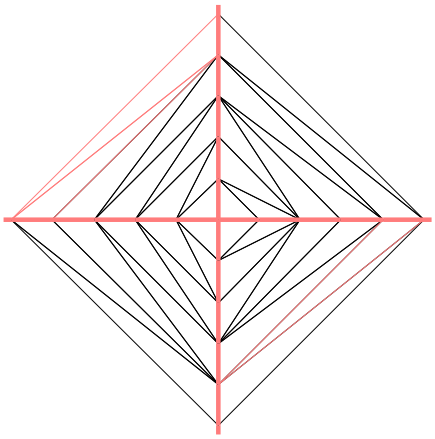


Figure 5.9: An empty mesh defined from a pseudo region numbering of triangle

5.3 Remeshing

5.3.1 Movemesh

Meshes can be translated, rotated and deformed by `movemesh`; this is useful for elasticity to watch the deformation due to the displacement $\mathbf{\Phi}(x,y) = (\Phi_1(x,y), \Phi_2(x,y))$ of shape. It is also useful to handle free boundary problems or optimal shape problems.
If Ω is triangulated as $T_h(\Omega)$, and Φ is a displacement vector then $\Phi(T_h)$ is obtained by

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

Sometimes the moved mesh is invalid because some triangle becomes reversed (with a negative area). This is why we should check the minimum triangle area in the transformed mesh with `checkmovemesh` before any real transformation.

Example 5.2 $\Phi_1(x,y) = x + k * \sin(y * \pi)/10)$, $\Phi_2(x,y) = y + k * \cos(y\pi)/10)$ *for a big number* $k > 1$.

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

mesh Th = buildmesh ( a(6) + b(4) + c(4) +d(4) + e(4) + f(6));
plot(Th,wait=1,fill=1,ps="Lshape.eps");                    //    see figure 5.10
real coef=1;
real minT0= checkmovemesh(Th,[x,y]);                  //    the min triangle area
while(1)                                             //    find a correct move mesh
{
  real minT=checkmovemesh(Th,[x+coef*uu,y+coef*vv]); //   the min triangle area
  if (minT > minT0/5) break ;                          //    if big enough
  coef=/1.5;
}

Th=movemesh(Th,[x+coef*uu,y+coef*vv]);
plot(Th,wait=1,fill=1,ps="movemesh.eps");                  //    see figure 5.11
```

Note 5.3 *Consider a function u defined on a mesh* `Th`*. A statement like* `Th=movemesh(Th...)` *does not change u and so the old mesh still exists. It will be destroyed when no function use it. A statement like u = u redefines u on the new mesh* `Th` *with interpolation and therefore destroys the old* `Th` *if u was the only function using it.*

Example 5.3 (movemesh.edp) *Now, we given an example of moving mesh with a lagrangian function u defined on the moving mesh.*

```
//    simple movemesh example
```

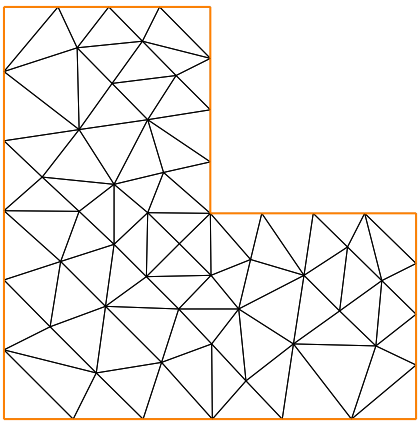


Figure 5.10: L-shape

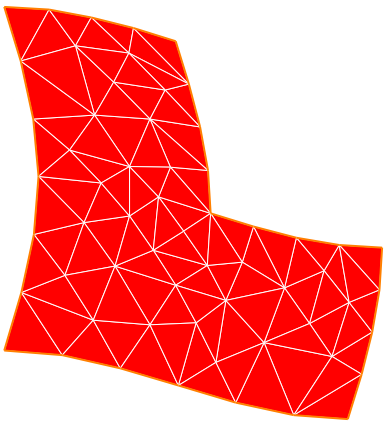


Figure 5.11: moved L-shape

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

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

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

5.4 Regular Triangulation: hTriangle

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

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

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

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

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

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

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

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

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

5.5 Adaptmesh

The function

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

sharply varies in value. However, the initial mesh given by the command in Section 5.1 cannot reflect its sharp variations.

Example 5.4

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

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

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

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

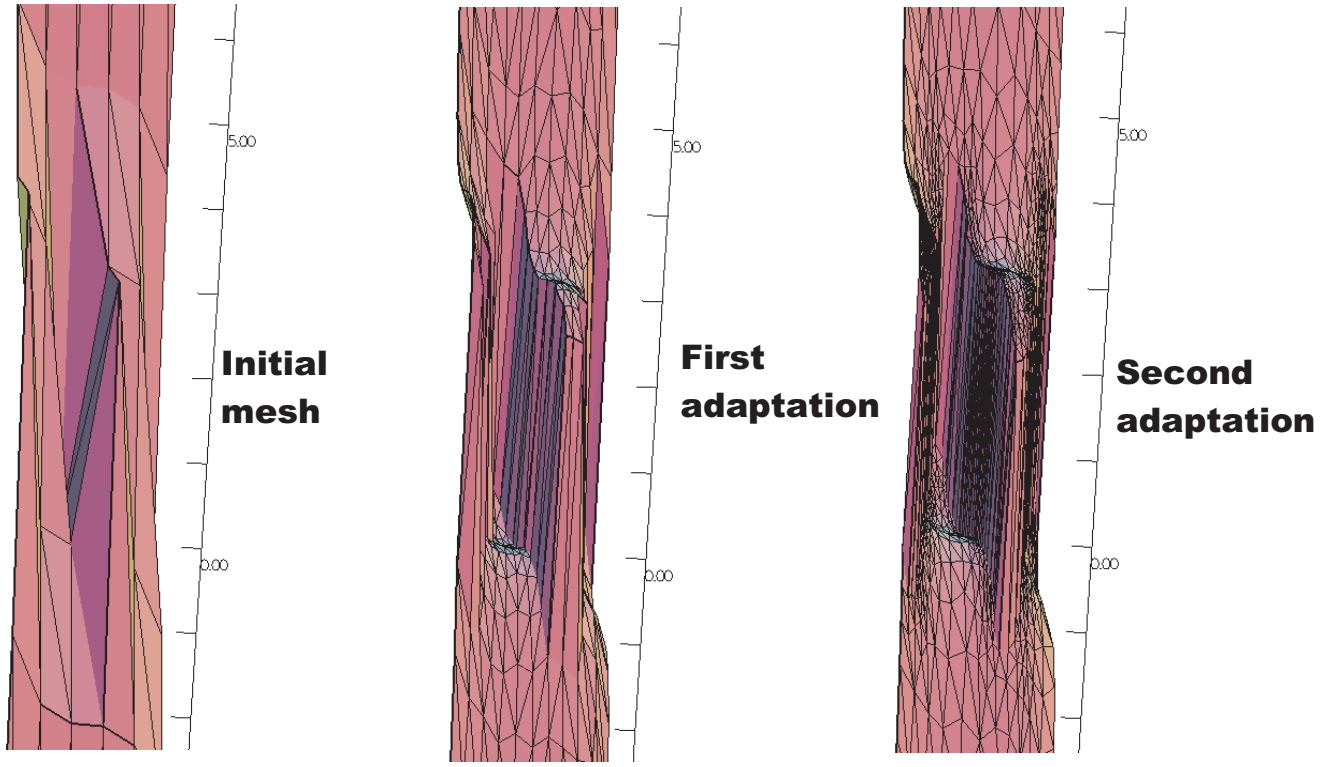


Figure 5.12: 3D graphs for the initial mesh and 1st and 2nd mesh adaptation

create the new mesh `ATh` by the Hessian

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

of a function (formula or FE-function). Mesh adaptation is a very powerful tool when the solution of a problem vary locally and sharply.
Here we solve the problem (2.1)-(2.2), when $f = 1$ and Ω is a L-shape domain.

Example 5.5 (Adapt.edp) *The solution has the singularity $r^{3/2}$, $r = |x - \gamma|$ at the point γ of the intersection of two lines bc and bd (see Fig. 5.13).*

```
border ba(t=0,1.0){x=t;   y=0;  label=1;};
border bb(t=0,0.5){x=1;   y=t;  label=1;};
border bc(t=0,0.5){x=1-t; y=0.5;label=1;};
border bd(t=0.5,1){x=0.5; y=t;  label=1;};
border be(t=0.5,1){x=1-t; y=1;  label=1;};
border bf(t=0.0,1){x=0;   y=1-t;label=1;};
mesh Th = buildmesh ( ba(6)+bb(4)+bc(4)+bd(4)+be(4)+bf(6) );
fespace Vh(Th,P1);                                          //    set FE space
Vh u,v;                                          //    set unknown and test function
func f = 1;
real error=0.1;                                             //    level of error
problem Poisson(u,v,solver=CG,eps=1.0e-6) =
    int2d(Th)(  dx(u)*dx(v) + dy(u)*dy(v))
  - int2d(Th) ( f*v )
  + on(1,u=0)  ;
```

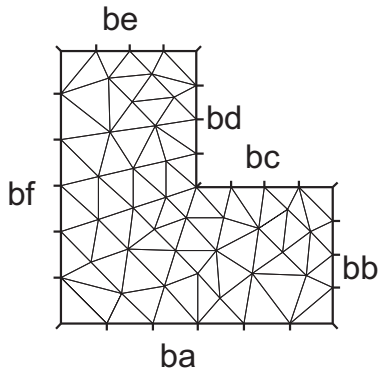


Figure 5.13: L-shape domain and its boundary name

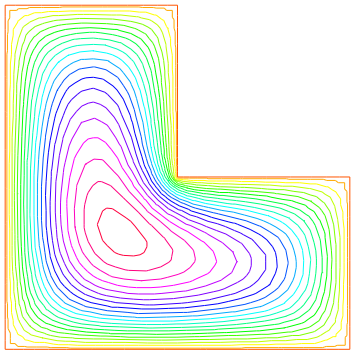


Figure 5.14: Final solution after 4-times adaptation

```
for (int i=0;i< 4;i++)
{
  Poisson;
  Th=adaptmesh(Th,u,err=error);
  error = error/2;
} ;
plot(u);
```

To speed up the adaptation we change by hand a default parameter `err` of `adaptmesh`, which specifies the required precision, so as to make the new mesh finer. The problem is coercive and symmetric, so the linear system can be solved with the conjugate gradient method (parameter `solver=CG` with the stopping criteria on the residual, here `eps=1.0e-6`). By `adaptmesh`, we get good slope of the final solution near the point of intersection of *bc* and *bd* as in Fig. 5.14. This method is described in detail in [8]. It has a number of default parameters which can be modified :

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

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

err= P_1 interpolation error level (0.01 is the default).

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

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

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

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

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

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

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

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

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

$$\mathcal{M} = \left(\frac{1}{\texttt{err coef}^2} \quad \frac{|\mathcal{H}|}{max(\texttt{CutOff}, |\eta|)} \right)^p \tag{5.1}$$

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

$$\mathcal{M} = \left(\frac{1}{\texttt{err coef}^2} \quad \frac{|\mathcal{H}|}{\sup(\eta) - \inf(\eta)} \right)^p . \tag{5.2}$$

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.

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

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

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

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

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

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

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

periodic= As writing `periodic=[[4,y],[2,y],[1,x],[3,x]];` it builds an adapted periodic mesh. The sample build a biperiodic mesh of a square. (see periodic finite element spaces 6, and see `sphere.edp` for a full example)

Example 5.6 *uniformmesh.edp*
We can use the command `adaptmesh` *to build uniform mesh with a contant mesh size. So to buid a mesh with a constant mesh size equal to* $\frac{1}{30}$ *do*

```
mesh Th=square(2,2);                                    //    to have initial mesh
plot(Th,wait=1,ps="square-0.eps");
Th= adaptmesh(Th,1./30.,IsMetric=1,nbvx=10000);                              //
plot(Th,wait=1,ps="square-1.eps");
Th= adaptmesh(Th,1./30.,IsMetric=1,nbvx=10000);   //    more the one time du to
Th= adaptmesh(Th,1./30.,IsMetric=1,nbvx=10000);          //    adaptation bound
maxsubdiv=
plot(Th,wait=1,ps="square-2.eps");
```

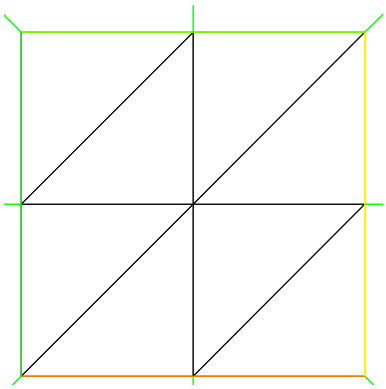


Figure 5.15: Initial mesh

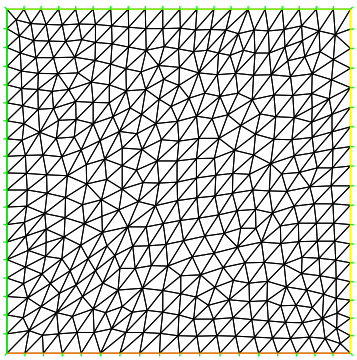


Figure 5.16: first iteration

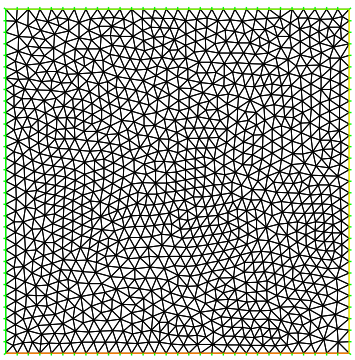


Figure 5.17: last iteration

5.6 Trunc

Two operators have been introduce to remove triangles from a mesh or to divide them. Operator `trunc` has two parameters

`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
 }
```

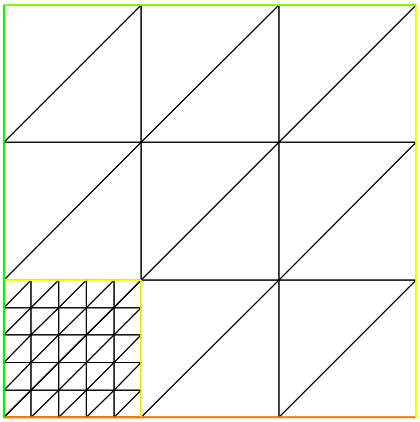


Figure 5.18: mesh of support the function P1 number 0, splitted in 5×5

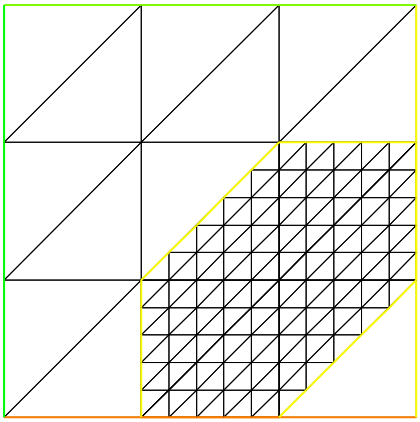


Figure 5.19: mesh of support the function P1 number 6, splitted in 5×5

5.7 Splitmesh

Another way to split mesh triangles is to use `splitmesh`, for example:

```
{                                        //    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 5.20
  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 5.21
}
```

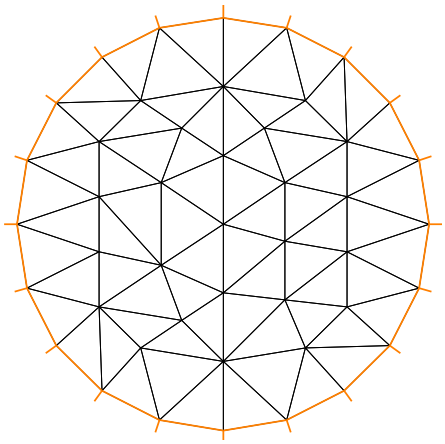


Figure 5.20: initial mesh

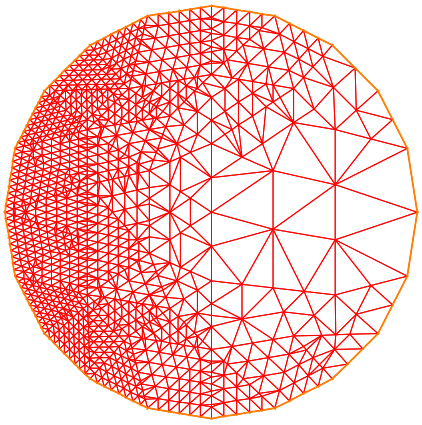


Figure 5.21: all left mesh triangle is split conformaly in `int(1+5*(square(x-0.5)+y*y)^2` triangles.

5.8 Meshing Examples

Example 5.7 (Two rectangles touching by a side)

```
border a(t=0,1){x=t;y=0;};
border b(t=0,1){x=1;y=t;};
border c(t=1,0){x=t ;y=1;};
border d(t=1,0){x = 0; y=t;};
border c1(t=0,1){x=t ;y=1;};
border e(t=0,0.2){x=1;y=1+t;};
border f(t=1,0){x=t ;y=1.2;};
border g(t=0.2,0){x=0;y=1+t;};
int n=1;
mesh th = buildmesh(a(10*n)+b(10*n)+c(10*n)+d(10*n));
mesh TH = buildmesh ( c1(10*n) + e(5*n) + f(10*n) + g(5*n) );
```

```
plot(th,TH,ps="TouchSide.esp");                                    //   Fig. 5.22
```

Example 5.8 (NACA0012 Airfoil)

```
border upper(t=0,1) { x = t;
     y = 0.17735*sqrt(t)-0.075597*t
  - 0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4); }
border lower(t=1,0) { x = t;
     y= -(0.17735*sqrt(t)-0.075597*t
  -0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4)); }
border c(t=0,2*pi) { x=0.8*cos(t)+0.5;  y=0.8*sin(t); }
mesh Th = buildmesh(c(30)+upper(35)+lower(35));
plot(Th,ps="NACA0012.eps",bw=1);                                   //   Fig. 5.23
```

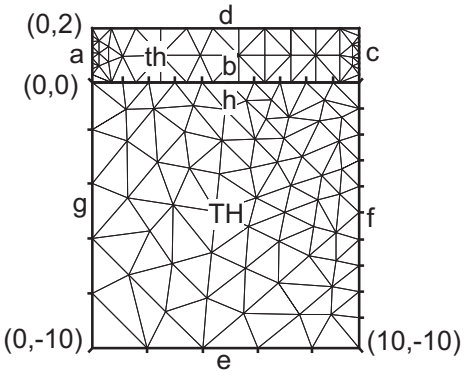


Figure 5.22: Two rectangles touching by a side

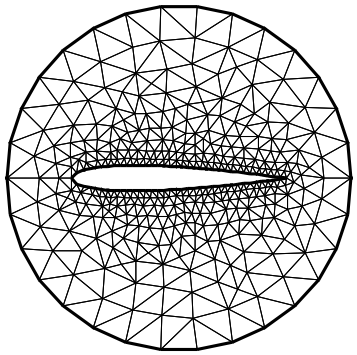


Figure 5.23: NACA0012 Airfoil

Example 5.9 (Cardioid)

```
real b = 1, a = b;
border C(t=0,2*pi) { x=(a+b)*cos(t)-b*cos((a+b)*t/b);
                     y=(a+b)*sin(t)-b*sin((a+b)*t/b); }
mesh Th = buildmesh(C(50));
plot(Th,ps="Cardioid.eps",bw=1);                                   //   Fig. 5.24
```

Example 5.10 (Cassini Egg)

```
border C(t=0,2*pi) { x=(2*cos(2*t)+3)*cos(t);
                     y=(2*cos(2*t)+3)*sin(t); }
mesh Th = buildmesh(C(50));
plot(Th,ps="Cassini.eps",bw=1);                                    //   Fig. 5.25
```

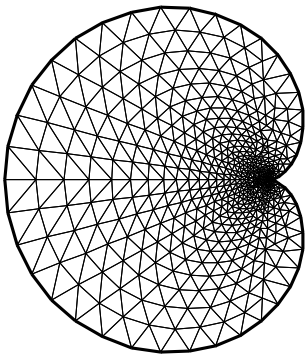


Figure 5.24: Domain with Cardioid curve boundary

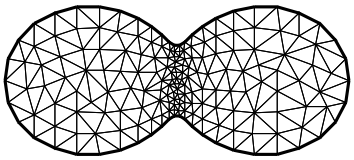


Figure 5.25: Domain with Cassini Egg curve boundary

Example 5.11 (By cubic Bezier curve)

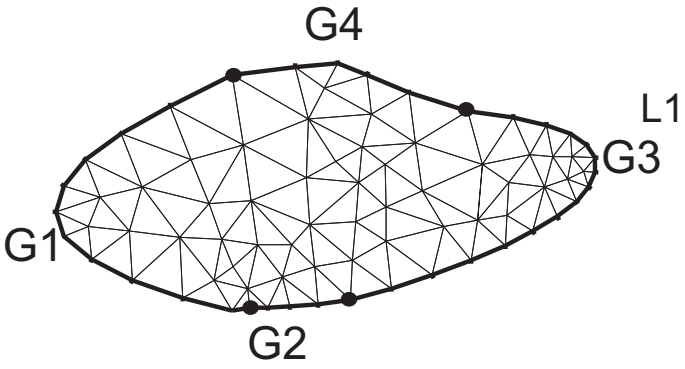
```
      //    A cubic Bezier curve connecting two points with two control points
func real bzi(real p0,real p1,real q1,real q2,real t)
{
  return p0*(1-t)^3+q1*3*(1-t)^2*t+q2*3*(1-t)*t^2+p1*t^3;
}

real[int] p00=[0,1], p01=[0,-1], q00=[-2,0.1], q01=[-2,-0.5];
real[int] p11=[1,-0.9], q10=[0.1,-0.95], q11=[0.5,-1];
real[int] p21=[2,0.7], q20=[3,-0.4], q21=[4,0.5];
real[int] q30=[0.5,1.1], q31=[1.5,1.2];
border G1(t=0,1) { x=bzi(p00[0],p01[0],q00[0],q01[0],t);
                   y=bzi(p00[1],p01[1],q00[1],q01[1],t); }
border G2(t=0,1) { x=bzi(p01[0],p11[0],q10[0],q11[0],t);
                   y=bzi(p01[1],p11[1],q10[1],q11[1],t); }
border G3(t=0,1) { x=bzi(p11[0],p21[0],q20[0],q21[0],t);
                   y=bzi(p11[1],p21[1],q20[1],q21[1],t); }
border G4(t=0,1) { x=bzi(p21[0],p00[0],q30[0],q31[0],t);
                   y=bzi(p21[1],p00[1],q30[1],q31[1],t); }
int m=5;
mesh Th = buildmesh(G1(2*m)+G2(m)+G3(3*m)+G4(m));
plot(Th,ps="Bezier.eps",bw=1);                                    //    Fig 5.26
```

Example 5.12 (Section of Engine)

```
real a= 6., b= 1., c=0.5;
border L1(t=0,1) { x= -a; y= 1+b - 2*(1+b)*t; }
border L2(t=0,1) { x= -a+2*a*t; y= -1-b*(x/a)*(x/a)*(3-2*abs(x)/a );}
border L3(t=0,1) { x= a; y=-1-b + (1+ b )*t; }
border L4(t=0,1) { x= a - a*t;   y=0; }
border L5(t=0,pi) { x= -c*sin(t)/2; y=c/2-c*cos(t)/2; }
border L6(t=0,1) { x= a*t;  y=c; }
border L7(t=0,1) { x= a;  y=c + (1+ b-c )*t; }
border L8(t=0,1) { x= a-2*a*t; y= 1+b*(x/a)*(x/a)*(3-2*abs(x)/a); }
```

```
mesh Th = buildmesh(L1(8)+L2(26)+L3(8)+L4(20)+L5(8)+L6(30)+L7(8)+L8(30));
plot(Th,ps="Engine.eps",bw=1);                                    //    Fig.  5.27
```



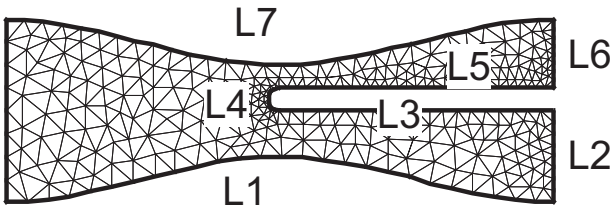


Figure 5.27: Section of Engine

Figure 5.26: Boundary drawed by Bezier curves

Example 5.13 (Domain with U-shape channel)

```
real d = 0.1;                                                    //    width of U-shape
border L1(t=0,1-d) { x=-1; y=-d-t; }
border L2(t=0,1-d) { x=-1; y=1-t; }
border B(t=0,2) { x=-1+t; y=-1; }
border C1(t=0,1) { x=t-1; y=d; }
border C2(t=0,2*d) { x=0; y=d-t; }
border C3(t=0,1) { x=-t; y=-d; }
border R(t=0,2) { x=1; y=-1+t; }
border T(t=0,2) { x=1-t; y=1; }
int n = 5;
mesh Th = buildmesh (L1(n/2)+L2(n/2)+B(n)+C1(n)+C2(3)+C3(n)+R(n)+T(n));
plot(Th,ps="U-shape.eps",bw=1);                                   //    Fig 5.28
```

Example 5.14 (Domain with V-shape cut)

```
real dAg = 0.01;                                                 //    angle of V-shape
border C(t=dAg,2*pi-dAg) { x=cos(t); y=sin(t); };
real[int] pa(2), pb(2), pc(2);
pa[0] = cos(dAg); pa[1] = sin(dAg);
pb[0] = cos(2*pi-dAg); pb[1] = sin(2*pi-dAg);
pc[0] = 0; pc[1] = 0;
border seg1(t=0,1) { x=(1-t)*pb[0]+t*pc[0]; y=(1-t)*pb[1]+t*pc[1]; };
border seg2(t=0,1) { x=(1-t)*pc[0]+t*pa[0]; y=(1-t)*pc[1]+t*pa[1]; };
mesh Th = buildmesh(seg1(20)+C(40)+seg2(20));
plot(Th,ps="V-shape.eps",bw=1);                                  //    Fig.  5.29
```

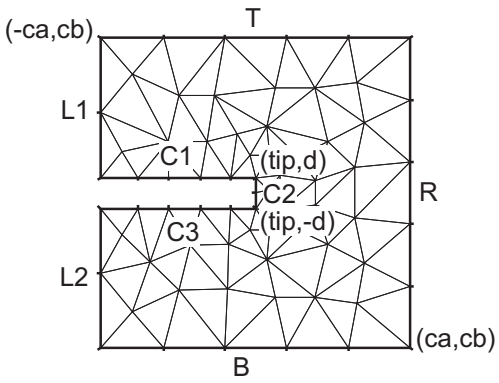


Figure 5.28: Domain with U-shape channel changed by `d`

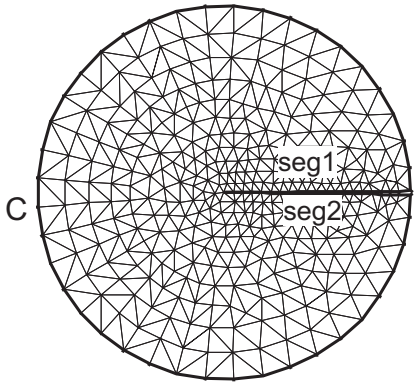


Figure 5.29: Domain with V-shape cut changed by `dAg`

Example 5.15 (Smiling face)

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

border C0(t=0,2*pi) { x=0.1*cos(t); y=0.1*sin(t); }
mesh Th=buildmesh(F(10*m)+C1(2*m)+C2(3)+C3(2*m)+C4(3)
                  +C0(m)+E1(-2*m)+E2(-2*m));
plot(Th,ps="SmileFace.eps",bw=1);                    //    see Fig. 5.30
}
```

Example 5.16 (3point bending)

```
                //    Square for Three-Point Bend Specimens fixed on  Fix1, Fix2
                                                    //    It will be loaded on  Load.
real a=1, b=5, c=0.1;
int n=5, m=b*n;
border Left(t=0,2*a) { x=-b; y=a-t; }
border Bot1(t=0,b/2-c) { x=-b+t; y=-a; }
border Fix1(t=0,2*c) { x=-b/2-c+t; y=-a; }
border Bot2(t=0,b-2*c) { x=-b/2+c+t; y=-a; }
border Fix2(t=0,2*c) { x=b/2-c+t; y=-a; }
border Bot3(t=0,b/2-c) { x=b/2+c+t; y=-a; }
border Right(t=0,2*a) { x=b; y=-a+t; }
```

```
border Top1(t=0,b-c) { x=b-t; y=a; }
border Load(t=0,2*c) { x=c-t; y=a; }
border Top2(t=0,b-c) { x=-c-t; y=a; }
mesh Th = buildmesh(Left(n)+Bot1(m/4)+Fix1(5)+Bot2(m/2)+Fix2(5)+Bot3(m/4)
                    +Right(n)+Top1(m/2)+Load(10)+Top2(m/2));
plot(Th,ps="ThreePoint.eps",bw=1);                                    //    Fig.  5.31
```

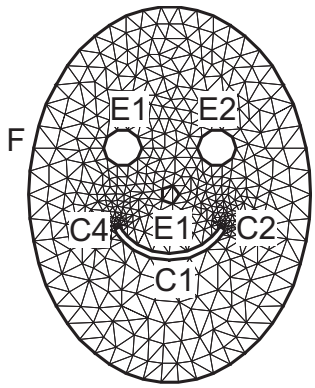


Figure 5.30: Smiling face (Mouth is changeable)

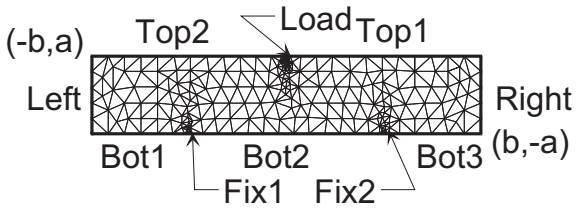


Figure 5.31: Domain for three-point bending test

Chapter 6

Finite Elements

As stated in Section 2. FEM approximates all functions w as

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

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

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

is easily created by

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

or with ℓ pairs of periodic boundary condition

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

where `IDspace` is the name of the space (e.g. `Vh`), `IDmesh` is the name of the associated mesh and `<IDFE>` is a identifier of finite element type. In a pair of periodic boundary condition, if `[la_i,sa_i],[lb_i,sb_i]` is a pair of `int`, the 2 labels `la_i` and `lb_i` define the 2 piece of boundary to be in equivalence; If `[la_i,sa_i],[lb_i,sb_i]` is a pair of `real`, then `sa_i` and `sb_i` give two common abscissa on the two boundary curve, and two points are identified as one if the two abscissa are equal.

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

P0 piecewise constante discontinuous finite element

$$P0_h = \left\{ v \in L^2(\Omega) \,\middle|\, \text{for all } K \in \mathcal{T}_h \ \text{ there is } \alpha_K \in \mathbb{R} : \ \ v_{|K} = \alpha_K \right\} \tag{6.1}$$

P1 piecewise linear continuous finite element

$$P1_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_1 \right\} \tag{6.2}$$

P1dc piecewise linear discontinuous finite element

$$P1dc_h = \left\{ v \in L^2(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_1 \right\} \tag{6.3}$$

P1b piecewise linear continuous finite element plus bubble

$$P1b_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_1 \oplus \mathrm{Span}\{\lambda_0^K \lambda_1^K \lambda_2^K\} \right\} \tag{6.4}$$

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

P2 piecewise P_2 continuous finite element,

$$P2_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_2 \right\} \tag{6.5}$$

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

P2b piecewise P_2 continuous finite element plus bubble,

$$P2_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_2 \oplus \mathrm{Span}\{\lambda_0^K \lambda_1^K \lambda_2^K\} \right\} \tag{6.6}$$

P2dc piecewise P_2 discontinuous finite element,

$$P2dc_h = \left\{ v \in L^2(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h \quad v_{|K} \in P_2 \right\} \tag{6.7}$$

RT0 Raviart-Thomas finite element

$$RT0_h = \left\{ \mathbf{v} \in H(\mathrm{div}) \,\middle|\, \forall K \in \mathcal{T}_h \quad \mathbf{v}_{|K}(x, y) = \begin{vmatrix} \alpha_K \\ \beta_K \end{vmatrix} + \gamma_K \begin{vmatrix} x \\ y \end{vmatrix} \right\} \tag{6.8}$$

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

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

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

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

If we get the finite element spaces

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

$$X_{ph} = \{ v \in X_h | \ v(\begin{vmatrix} 0 \\ \cdot \end{vmatrix}) = v(\begin{vmatrix} 1 \\ \cdot \end{vmatrix}), v(|\,\dot{0}\,) = v(|\,\dot{1}\,) \}$$

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

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

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

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

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

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

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

The functions U_h, V_h have two components so we have

$$U_h = \begin{vmatrix} Uxh \\ Uyh \end{vmatrix} \quad \text{and} \quad V_h = \begin{vmatrix} Vxh \\ Vyh \end{vmatrix}$$

6.1 Lagrange finite element

6.1.1 P0-element

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

$$\phi_k(x,y) = 1 \text{ if } (x,y) \in T_k, \qquad \phi_k(x,y) = 0 \text{ if } (x,y) \notin T_k$$

If we write

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

then for vertices q^{k_i}, $i = 1,2,3$ in Fig. 6.1(a), f_h is built as

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

See Fig. 6.3 for the projection of $f(x,y) = \sin(\pi x)\cos(\pi y)$ on `Vh(Th,P0)` when the mesh `Th` is a 4×4-grid of $[-1,1]^2$ as in Fig. 6.2.

6.1.2 P1-element

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

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

The basis function $\phi_{k_1}(x,y)$ with the vertex q^{k_1} in Fig. 6.1(a) at point $p = (x,y)$ in triangle T_k simply coincide with the *barycentric coordinates* λ_1^k *(area coordinates)* :

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

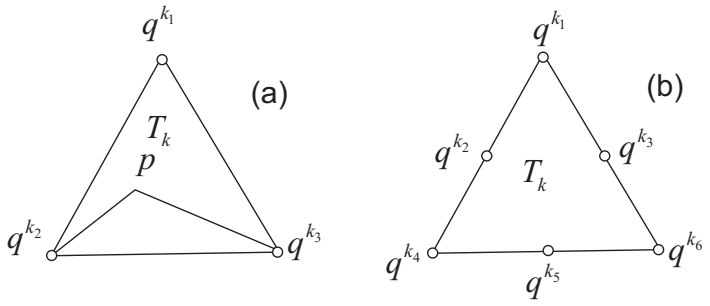


Figure 6.1: P_1 and P_2 degrees of freedom on triangle T_k

If we write

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

then

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

See Fig. 6.4 for the projection of $f(x,y) = \sin(\pi x)\cos(\pi y)$ into `Vh(Th,P1)`.

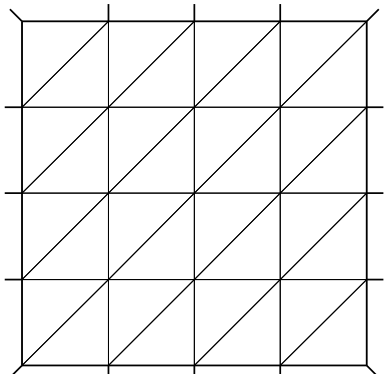


Figure 6.2: Test mesh `Th` for projection

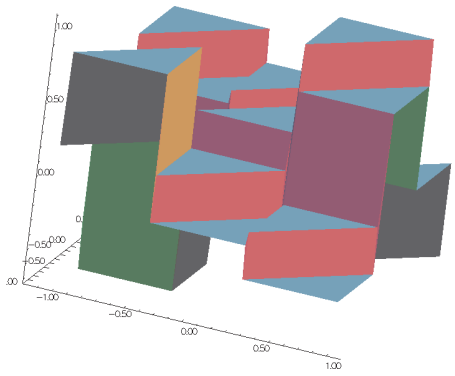


Figure 6.3: projection to `Vh(Th,P0)`

6.1.3 P2-element

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

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

The basis function $\phi_{k_1}(x,y)$ with the vertex q^{k_1} in Fig. 6.1(b) is defined by the *barycentric coordinates*:

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

and for the midpoint q^{k_2}

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

If we write

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

then

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

See Fig. 6.5 for the projection of $f(x,y) = \sin(\pi x)\cos(\pi y)$ into `Vh(Th,P2)`.

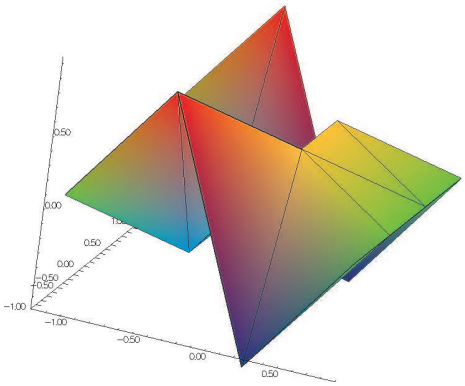


Figure 6.4: projection to `Vh(Th,P1)`

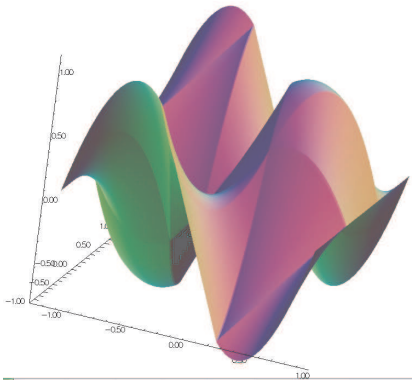


Figure 6.5: projection to `Vh(Th,P2)`

6.2 P1 Nonconforming Element

Refer to [22] for details; briefly, we now consider non-continuous approximations so we shall lose the property

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

If we write

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

then

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

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

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

Strictly speaking $\partial\phi_i/\partial x$, $\partial\phi_i/\partial y$ contain Dirac distribution $\rho\delta_{\partial T_k}$. The numerical calculations will automatically *ignore* them. In [22], there is a proof of the estimation

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

The basis functions ϕ_k have the following properties.

1. For the bilinear form a defined in (2.6) satisfy

$$a(\phi_i,\phi_i)>0,\qquad a(\phi_i,\phi_j)\le 0\quad \text{if } i\neq j$$
$$\sum_{k=1}^{n_v}a(\phi_i,\phi_k)\ge 0$$

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

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

$$\int_\Omega \phi_i\phi_j\,\mathrm{d}x\mathrm{d}y=0\qquad \text{if } i\neq j$$

   which is false for P_1-element.

See Fig. 6.6 for the projection of $f(x,y)=\sin(\pi x)\cos(\pi y)$ into `Vh(Th,P1nc)`. See Fig. 6.6 for the projection of $f(x,y)=\sin(\pi x)\cos(\pi y)$ into `Vh(Th,P1nc)`.

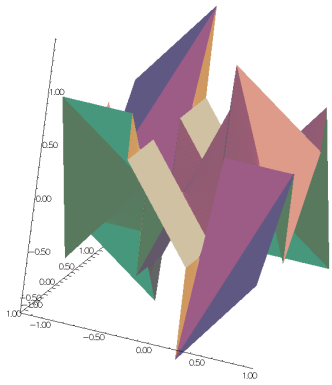


Figure 6.6: projection to `Vh(Th,P1nc)`

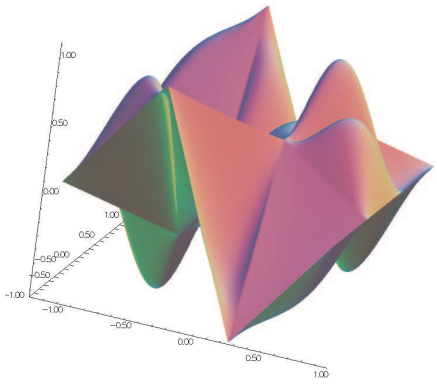


Figure 6.7: projection to `Vh(Th,P1b)`

6.3 Other FE-space

For each triangle $T_k\in\mathcal{T}_h$, let $\lambda_{k_1}(x,y)$, $\lambda_{k_2}(x,y)$, $\lambda_{k_3}(x,y)$ be the area cordinate of the triangle (see Fig. 6.1), and put

$$\beta_k(x,y)=27\lambda_{k_1}(x,y)\lambda_{k_2}(x,y)\lambda_{k_3}(x,y) \tag{6.9}$$

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

1. $\beta_k(x,y) = 0 \quad$ if $(x,y) \in \partial T_k$.
2. $\beta_k(q^{k_b}) = 1$ where q^{k_b} is the barycenter $\frac{q^{k_1}+q^{k_2}+q^{k_3}}{3}$.

If we write

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

then

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

See Fig. 6.7 for the projection of $f(x,y) = \sin(\pi x)\cos(\pi y)$ into `Vh(Th,P1b)`.

6.4 Vector valued FE-function

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

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

make the space

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

6.4.1 Raviart-Thomas element

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

$$\boldsymbol{\phi}_1^k = \frac{sgn(i_b - i_c)}{2|T_k|}(\mathbf{x} - \mathbf{a}), \quad \boldsymbol{\phi}_2^k = \frac{sgn(i_c - i_a)}{2|T_k|}(\mathbf{x} - \mathbf{b}), \quad \boldsymbol{\phi}_3^k = \frac{sgn(i_a - i_b)}{2|T_k|}(\mathbf{x} - \mathbf{c}), \tag{6.10}$$

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

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

then

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

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

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

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

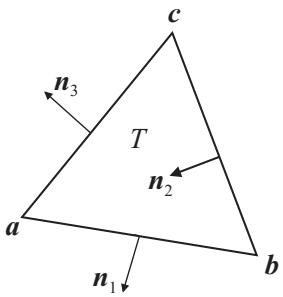


Figure 6.8: normal vectors of each edge

Example 6.1

```
mesh Th=square(2,2);
fespace Xh(Th,P1);
fespace Vh(Th,RT0);
Xh uh,vh;
Vh [Uxh,Uyh];
[Uxh,Uyh] = [sin(x),cos(y)];                    //    ok vectorial FE function
vh= x^2+y^2;                                                    //    vh
Th = square(5,5);                                          //    change the mesh
                                                            //    Xh is unchange
uh = x^2+y^2;                                          //    compute on the new Xh
Uxh = x;                          //    error:  impossible to set only 1 component
                                                       //    of a vector FE function.
vh = Uxh;                                                       //    ok
                                                    //    and now uh use the 5x5 mesh
                                  //    but the fespace of vh is alway the 2x2 mesh
plot(uh,ps="onoldmesh.eps");                                    //    figure 6.9
uh = uh;                          //    do a interpolation of vh (old) of 5x5 mesh
                                      //    to get the new vh on 10x10 mesh.
plot(uh,ps="onnewmesh.eps");                                    //    figure 6.10
vh([x-1/2,y])= x^2 + y^2;                          //    interpolate vh = ((x − 1/2)^2 + y^2)
```

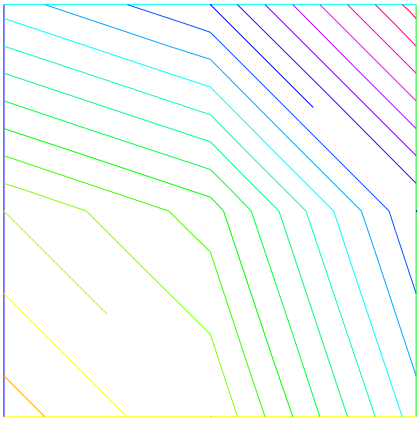


Figure 6.9: vh Iso on mesh 2×2

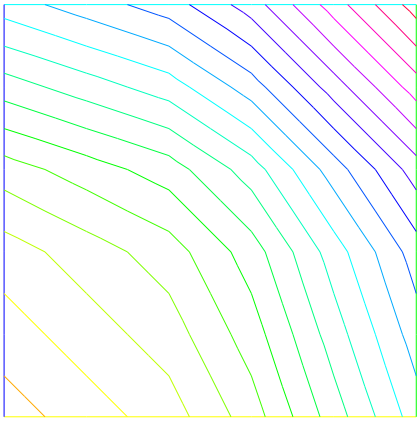


Figure 6.10: vh Iso on mesh 5×5

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

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

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

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

Note 6.1 *For a none 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.*

6.5 A Fast Finite Element Interpolator

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

be two triangulations of a domain Ω. Let

$$V(\mathcal{T}_h^i) = \{C^0(\Omega_h^i) \ : \ f|_{T_k^i} \in P_0\}, \quad i = 0, 1$$

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

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

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

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

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

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

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

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

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

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

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

End

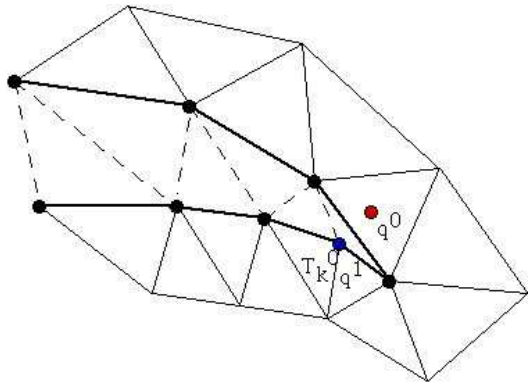


Figure 6.11: To interpolate a function at q^0 the knowledge of the triangle which contains q^0 is needed. The algorithm may start at $q^1 \in T_k^0$ and stall on the boundary (thick line) because the line q^0q^1 is not inside Ω. But if the holes are triangulated too (doted line) then the problem does not arise.

Two problems need to be solved:

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

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

Note 6.2 *Step 3 requires an array of pointers such that each vertex points to one triangle of the triangulation.*

Note 6.3 *The operator = is the interpolation operator of* `FreeFem++` *, The continuous finite functions are extended by continuity to the outside of the domain. Try the following example*

```
mesh Ths= square(10,10);
 mesh Thg= square(30,30,[x*3-1,y*3-1]);
 plot(Ths,Thg,ps="overlapTh.eps",wait=1);
fespace Ch(Ths,P2); fespace Dh(Ths,P2dc);
fespace Fh(Thg,P2dc);
Ch us= (x-0.5)*(y-0.5);
Dh vs= (x-0.5)*(y-0.5);
Fh ug=us,vg=vs;
plot(us,ug,wait=1,ps="us-ug.eps");                    //    see figure 6.12
plot(vs,vg,wait=1,ps="vs-vg.eps");                    //    see figure 6.13
```

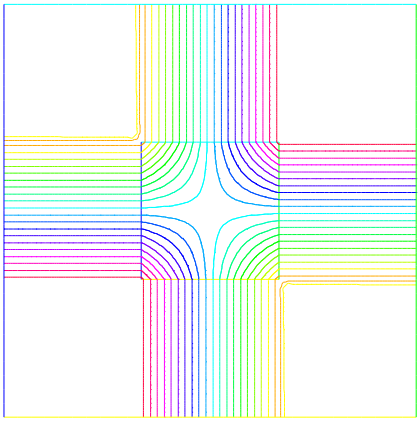


Figure 6.12: Extension of a continuous FE-function

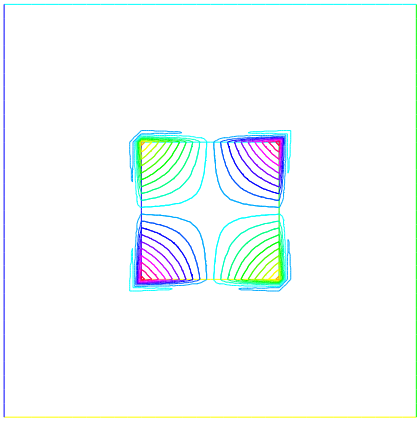


Figure 6.13: Extention of discontinuous FE-function

6.6 Keywords: Problem and Solve

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

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

Note 6.4 *When you want to formulate the problem and to solve it in the same time, you can use the keywork* `solve`.

6.6.1 Weak form and Boundary Condition

To present the principles of Variational Formulations or also called weak forms fr the PDEs, let us take a model problem : a Poisson equation with Dirichlet and Robin Boundary condition .
The problem is: Find u a real function defined on domain Ω of $\mathbb{R}^2$ such that

$$-\nabla.(\nu\nabla u) = f, \quad \text{in} \quad \Omega, \quad au + \nu\frac{\partial u}{\partial n} = b \quad \text{on} \quad \Gamma_r, \quad u = g \quad \text{on} \quad \Gamma_d \tag{6.11}$$

where

- $\nabla.(\nu\nabla u) = \partial_x(\nu\partial_x u) + \partial_y(\nu\partial_y u)$ with $\partial_x u = \frac{\partial u}{\partial x}$ and $\partial_y u = \frac{\partial u}{\partial y}$
- the border $\Gamma = \partial\Omega$ is split in Γ_d and Γ_n such that $\Gamma_d \cup \Gamma_n = \emptyset$ and $\Gamma_d \cap \Gamma_n = \partial\Omega$,
- ν is a given positive function, such that $\exists \nu_0 \in \mathbb{R}, \quad 0 < \nu_0 \leq \nu$.
- a a given non negative function,
- b a given function.

Note 6.5 *This problem, we can be a classical Neumann boundary condition if a is* 0*, and if* Γ_d *is empty. In this case the function is defined just by derivative, so this defined too a constant (if u is a solution then u* + 1 *is also a solution).*

Let v a regular test function null on Γ_d , by integration par part we get

$$-\int_\Omega \nabla.(\nu\nabla u)\, v\, d\omega = \int_\Omega \nu\nabla v.\nabla u\, d\omega = \int_\Omega fv\, d\omega - \int_\Gamma v\nu\frac{\partial u}{\partial n}\, d\gamma, \tag{6.12}$$

where $\nabla v.\nabla u = \frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y}$, and where n is the unitary outside normal of $\partial\Omega$.
Now we note that $\nu\frac{\partial u}{\partial n} = -au + g$ on Γ_r and $v = 0$ on Γ_d and $\partial\Omega = \Gamma_d \cup \Gamma_n$ thus

$$-\int_{\partial\Omega} v\nu\frac{\partial u}{\partial n} = \int_{\Gamma_r} auv - \int_{\Gamma_r} bv$$

The problem become:
Find $u \in V_g = \{v \in H^1(\Omega)/v = g \text{ on } \Gamma_d\}$ such that

$$\int_\Omega \nu\nabla v.\nabla u\, d\omega + \int_{\Gamma_r} auv\, d\gamma = \int_\Omega fv\, d\omega + \int_{\Gamma_r} bv\, d\gamma, \quad \forall v \in V_0 \tag{6.13}$$

where $V_0 = \{v \in H^1(\Omega)/v = 0 \text{ on } \Gamma_d\}$
The problem (6.13) is generally well posed if we do not have only Neumann boundary condition (ie. $\Gamma_d = \emptyset$ and $a = 0$).

Note 6.6 If we have only Neumann boundary condition, then solution is not unique and linear algebra tells us that the right hand side must be orthogonal to the kernel of the operator. Here the problem is defined to a constant, and since $1 \in V_0$ one way of writing the compatibility condition is: $\int_\Omega f\, d\omega + \int_\Gamma b\, d\gamma$ and a way to fix the constant is to solve for $u \in H^1(\Omega)$ such that:

$$\int_\Omega \varepsilon uv\ d\omega\ +\ \nu\nabla v.\nabla u\, d\omega = \int_\Omega fv\, d\omega + \int_{\Gamma_r} bv\, d\gamma, \quad \forall v \in H^1(\Omega) \tag{6.14}$$

where ε is a small parameter ($\sim 10^{-10}$).
Remark that if the solution is of order $\frac{1}{\varepsilon}$ then the compatibility condition is unsatisfied, otherwise we get the solution such that $\int_\Omega u = 0$.

In FreeFem++, the problem (6.13) become

```
problem Pw(u,v) =
      int2d(Th)( nu* ( dx(u)*dx(u) +  dy(u)*dy(u)))   //   ∫Ω ν∇v.∇u dω
    + int1d(Th,gn)( a * u*v )                         //   ∫Γr auv dγ
    - int2d(Th)(f*v)                                  //   ∫Ω fv dω
    - int1d(Th,gn)( b * v )                           //   ∫Γr bv dγ
    + on(gd)(u= g) ;                                  //   u = g  on Γd
```

where Th is a mesh of the domain Ω, and gd and gn are respectively the boundary label of boundary Γ_d and Γ_n.

6.7 Parameters affecting `solve` and `problem`

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

Note 6.7 *If the functions are a part of vectoriel FE then you must give all the functions of the vectorial FE in the same order (see laplaceMixte problem for example).*

Note 6.8 *Don't mix complex and real parameters FE function.*

Bug: 1 *The mixing of `fespace` with different periodic boundary condition is not implemented. So all the finite element spaces used for test or unknown 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 parameters are:

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

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

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

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

if it is positive then the stopping test is

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

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

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

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

tgv= Huge value (10^{30}) used to implement Dirichlet boundary conditions.

tolpivot= set the tolerence of the pivot in UMFPACK ($10^{-}1$) and, LU, Crout, Cholesky factorisation (10^{-20}).

tolpivotsym= set the tolerence of the pivot sym in UMFPACK

strategy= set the integer UMFPACK strategy (0 by default).

6.8 Problem definition

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

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

  Remark, that the name in the "varf" of the unknow of test function is forgotten, we just used the order in argument list to recall name as in a `C++` function, see note 6.12,

- the terms of the bilinear form itself: if K is a given function,

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

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

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

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

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

  -) they contribute to the sparse matrix of type `matrix` which, whether declared explicitly or not is contructed by `FreeFem++` .

- the right handside of the PDE, volumic terms of the linear form: for given functions K, f:

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

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

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

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

- The boundary condition terms :
  - An "on" form (for Dirichlet) : `on(1, u = g )`
    The meaning is for all node i the this associated boundary, the diagonal term of the matrix $a_{ii} = tgv$ with the *terrible geant value* `tgv` (=10^{30} by default) and the right hand side $b[i]$ = "$g[i]$" $\times\, tgv$, where the "$g[i]$" is the boundary node value given by the interpolation of 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)`.

Note 6.9 • *If needed, the different kind of terms in the sum can appear more than once.*

- *the integral mesh and the mesh associated to test function or unknown function can be different in the case of linear form.*
- `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)` .

6.9 Numerical Integration

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

$$\int_D f(\boldsymbol{x}) = \sum_{\ell=1}^{L} c_\ell f(\boldsymbol{\xi}_\ell) \tag{6.15}$$

then we have the error estimation (see Crouzeix-Mignot (1984)), and then there exists a constant $C > 0$ such that,

$$\left| \int_D f(\boldsymbol{x}) - \sum_{\ell=1}^{L} \omega_\ell f(\boldsymbol{\xi}_\ell) \right| \le C|D|h^{r+1} \tag{6.16}$$

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

$$\{(x, y)|\ x = (1 - t)q_x^i + tq_x^j,\ y = (1 - t)q_y^i + tq_y^j,\ 0 \le t \le 1\}).$$

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

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

where * stands for the name of the quadrature formula or the precision (order) of the Gauss formula. where $|q^iq^j|$ is the length of segment $\overline{q^iq^j}$. For a part Γ_1 of Γ_h with the label "1", we can

L	(qfe=)	qforder=	point in $[q^iq^j](=t)$	ω_ℓ	exact on $P_k, k=$
1	qf1pE	2	$1/2$	$\lvert q^iq^j\rvert$	1
2	qf2pE	3	$(1\pm\sqrt{1/3})/2$	$\lvert q^iq^j\rvert/2$	3
3	**qf3pE**	6	$(1\pm\sqrt{3/5})/2$ $1/2$	$(5/18)\lvert q^iq^j\rvert$ $(8/18)\lvert q^iq^j\rvert$	5
4	**qf4pE**	8	$(1\pm\frac{\sqrt{525+70\sqrt{30}}}{35})/2.$ $(1\pm\frac{\sqrt{525-70\sqrt{30}}}{35})/2.$	$\frac{18-\sqrt{30}}{72}\lvert q^iq^j\rvert$ $\frac{18+\sqrt{30}}{72}\lvert q^iq^j\rvert$	7
5	**qf5pE**	10	$(1\pm\frac{\sqrt{245+14\sqrt{70}}}{21})/2$ $1/2$ $(1\pm\frac{\sqrt{245-14\sqrt{70}}}{21})/2$	$\frac{322-13\sqrt{70}}{1800}\lvert q^iq^j\rvert$ $\frac{64}{225}\lvert q^iq^j\rvert$ $\frac{322+13\sqrt{70}}{1800}\lvert q^iq^j\rvert$	9
2	qf1pElump	2	-1 $+1$	$\lvert q^iq^j\rvert/2$ $\lvert q^iq^j\rvert/2$	1

calculate the integral over Γ_1 by

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

The integral over Γ_1, Γ_3 are given by

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

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

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

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

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

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

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

L	qft=	qforder=	point in T_k	ω_ℓ	degree of exact
1	qf1pT	2	$\left(\frac{1}{3},\frac{1}{3}\right)$	$\lvert T_k\rvert$	1
3	qf2pT	3	$\left(\frac{1}{2},\frac{1}{2}\right)$ $\left(\frac{1}{2},0\right)$ $\left(0,\frac{1}{2}\right)$	$\lvert T_k\rvert/3$ $\lvert T_k\rvert/3$ $\lvert T_k\rvert/3$	2
7	**qf5pT**	6	$\left(\frac{1}{3},\frac{1}{3}\right)$ $\left(\frac{6-\sqrt{15}}{21},\frac{6-\sqrt{15}}{21}\right)$ $\left(\frac{6-\sqrt{15}}{21},\frac{9+2\sqrt{15}}{21}\right)$ $\left(\frac{9+2\sqrt{15}}{21},\frac{6-\sqrt{15}}{21}\right)$ $\left(\frac{6+\sqrt{15}}{21},\frac{6+\sqrt{15}}{21}\right)$ $\left(\frac{6+\sqrt{15}}{21},\frac{9-2\sqrt{15}}{21}\right)$ $\left(\frac{9-2\sqrt{15}}{21},\frac{6+\sqrt{15}}{21}\right)$	$0.225\lvert T_k\rvert$ $\frac{(155-\sqrt{15})\lvert T_k\rvert}{1200}$ $\frac{(155-\sqrt{15})\lvert T_k\rvert}{1200}$ $\frac{(155-\sqrt{15})\lvert T_k\rvert}{1200}$ $\frac{(155+\sqrt{15})\lvert T_k\rvert}{1200}$ $\frac{(155+\sqrt{15})\lvert T_k\rvert}{1200}$ $\frac{(155+\sqrt{15})\lvert T_k\rvert}{1200}$	5
3	qf1pTlump		$(0,0)$ $(1,0)$ $(0,1)$	$\lvert T_k\rvert/3$ $\lvert T_k\rvert/3$ $\lvert T_k\rvert/3$	1
9	qf2pT4P1		$\left(\frac{1}{4},\frac{3}{4}\right)$ $\left(\frac{3}{4},\frac{1}{4}\right)$ $\left(0,\frac{1}{4}\right)$ $\left(0,\frac{3}{4}\right)$ $\left(\frac{1}{4},0\right)$ $\left(\frac{3}{4},0\right)$ $\left(\frac{1}{4},\frac{1}{4}\right)$ $\left(\frac{1}{4},\frac{1}{2}\right)$ $\left(\frac{1}{2},\frac{1}{4}\right)$	$\lvert T_k\rvert/12$ $\lvert T_k\rvert/12$ $\lvert T_k\rvert/12$ $\lvert T_k\rvert/12$ $\lvert T_k\rvert/12$ $\lvert T_k\rvert/12$ $\lvert T_k\rvert/6$ $\lvert T_k\rvert/6$ $\lvert T_k\rvert/6$	1
15	qf7pT	8	see [35] for detail		7
21	qf9pT	10	see [35] for detail		9

Note 6.10 *By default, we use the formula which is exact for polynomes of degrees* 5 *on triangles or edges (in bold in two tables).*

6.10 Variational Form, Sparse Matrix, PDE Data Vector

First, it is possible to define variational forms, and use this forms to build matrix and vector to make very fast script (4 times faster here).
For example solve the Thermal Conduction problem of section 3.4.
The variational formulation is in $L^2(0,T;H^1(\Omega))$; we shall seek u^n satisfying

$$\forall w \in V_0; \qquad \int_\Omega \frac{u^n - u^{n-1}}{\delta t} w + \kappa \nabla u^n \nabla w) + \int_\Gamma \alpha(u^n - u_{ue}) w = 0$$

where $V_0 = \{w \in H^1(\Omega)/w_{|\Gamma_{24}} = 0\}$.

So the to code the method with the matrices $A = (A_{ij})$, $M = (M_{ij})$, and the vectors $u^n, b^n, b', b'', b_{cl}$ (notation if w is a vector then w_i is a component of the vector).

$$u^n = A^{-1}b^n, \qquad b' = b_0 + Mu^{n-1}, \quad b'' = \nu_\infty\, b_{cl}, \quad b_i^n = \begin{cases} b''_i & \text{if } i \in \Gamma_{24} \\ b'_i & \text{else if } \notin \Gamma_{24} \end{cases} \tag{6.17}$$

Where with ν_∞ = `tgv` = 10^{30} :

$$A_{ij} = \begin{cases} \nu_\infty & \text{if } i \in \Gamma_{24}, \text{and} j = i \\ \int_\Omega w_j w_i/dt + k(\nabla w_j.\nabla w_i) + \int_{\Gamma_{13}} \alpha w_j w_i & \text{else if } i \notin \Gamma_{24}, \text{or} j \neq i \end{cases} \tag{6.18}$$

$$M_{ij} = \begin{cases} \nu_\infty & \text{if } i \in \Gamma_{24}, \text{and} j = i \\ \int_\Omega w_j w_i/dt & \text{else if } i \notin \Gamma_{24}, \text{or} j \neq i \end{cases} \tag{6.19}$$

$$b_{0,i} = \int_{\Gamma_{13}} \alpha u_{ue} w_i \tag{6.20}$$

$$b_{cl} = u^0 \quad \text{the initial data} \tag{6.21}$$

```
//    file thermal-fast.edp in examples++-tutorial

func fu0 =10+90*x/6;
func k = 1.8*(y<0.5)+0.2;
real ue = 25. , alpha=0.25, T=5, dt=0.1 ;

mesh Th=square(30,5,[6*x,y]);
fespace Vh(Th,P1);

Vh u0=fu0,u=u0;
```

Create three variational formulation, and build the matrices A,M.

```
varf vthermic (u,v)= int2d(Th)(u*v/dt + k*(dx(u) * dx(v) + dy(u) * dy(v)))
  +  int1d(Th,1,3)(alpha*u*v)
  + on(2,4,u=1);

varf vthermic0(u,v) =   int1d(Th,1,3)(alpha*ue*v);

varf vMass (u,v)= int2d(Th)( u*v/dt)  + on(2,4,u=1);

real tgv = 1e30;
matrix A= vthermic(Vh,Vh,tgv=tgv,solver=CG);
matrix M= vMass(Vh,Vh);
```

Now, to build the right hand size we need 4 vectors.

```
real[int]  b0  = vthermic0(0,Vh);                  //    constant part of the RHS
real[int]  bcn = vthermic(0,Vh);   //    tgv on Dirichlet boundary node ( !=0 )
                                   //    we have for the node i :  i ∈ Γ24  ⇔  bcn[i] ≠ 0
real[int]  bcl=tgv*u0[];           //    the Dirichlet boundary condition part
```

Note 6.11 *The boundary condition is implemented by penalization and vector* `bcn` *contains the contribution of the boundary condition* $u_{=}1$ *, so to change the boundary condition, we have just to multiply the vector bc*[] *by the current value f of the new boundary condition term by term with the operator* `.*`. *Section 9.6.2* `Examples++-tutorial/StokesUzawa.edp` *gives a real example of using all this features.*

And the new version of the algorithm:

```
ofstream ff("thermic.dat");
for(real t=0;t<T;t+=dt){
    real[int] b = b0 ;                                          //    for the RHS
    b += M*u[];                                  //    add the the time dependant part
                                                 //    lock boundary part:
    b = bcn ? bcl  : b ;          //    do ∀i:  b[i] = bcn[i] ?  bcl[i] :  b[i] ;
    u[] = A^-1*b;
    ff << t <<" "<<u(3,0.5)<<endl;
    plot(u);
}
for(int i=0;i<20;i++)
  cout<<dy(u)(6.0*i/20.0,0.9)<<endl;
plot(u,fill=true,wait=1,ps="thermic.eps");
```

Note 6.12 *The functions appearing in the variational form are formal; they must be declared but not necessarily defined, the only important think in the order in the parameter list, like in*

```
varf vb1([u1,u2],q) = int2d(Th)( (dy(u1)+dy(u2)) *q) + int2d(Th)(1*q);
varf vb2([v1,v2],p) = int2d(Th)( (dy(v1)+dy(v2)) *p) + int2d(Th)(1*p);
```

To build matrix A from the bilinear part the the variational form a of type `varf` do simply

```
A = a(Vh,Wh [, ...] );
        //    where Vh is "unkown fespace" with a correct number of component
          //    where Wh is "test fespace" with a correct number of component
```

The possible named parameter `" [, ... ] "` of the construction are

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

The default solver is `GMRES`. 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.

factorize = if true then do the matrix factorization for `LU`, `Cholesky` or `Crout`, the default value is $false$.

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

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

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

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

tgv= Huge value (10^{30}) used to implement Dirichlet boundary conditions.

tolpivot= set the tolerence of the pivot in UMFPACK ($10^{-}1$) and, LU, Crout, Cholesky factorisation (10^{-20}).

tolpivotsym= set the tolerence of the pivot sym in UMFPACK

strategy= set the integer UMFPACK strategy (0 by default).

Note 6.13 *The line of the matrix corresponding to the space* `Wh` *and the column of the matrix corresponding to the space* `Vh`.

To build the dual vector b (of type real[int]) from the linear part of the variational form a do simply

```
real b(Vh.ndof);
b = a(0,Vh);
```

A first example to compute the area of each triangle K of mesh Th, just do:

```
fespace Nh(Th,P0);                //    the space function contant / triangle
Nh areaK;
varf varea(unused,chiK) =  int2d(Th)(chiK);
etaK[]= varea(0,Ph);
```

Effectively, the basic functions of space Nh, are the characteristic function of the element of Th, and the numbering is the numeration of the element, so by construction:

$$\texttt{etaK}[i] = \int 1_{|K_i} = \int_{K_i} 1;$$

Now, we can use this to compute error indicator like in examples `AdaptResidualErrorIndicator.edp` in directory `examples++-tutorial`.
First to compute a continuous approximation to the function h "density mesh size" of the mesh Th.

```
fespace Vh(Th,P1);
Vh h ;
real[int]  count(Th.nv);  varf vmeshsizen(u,v)=intalledges(Th,qfnbpE=1)(v);
varf vedgecount(u,v)=intalledges(Th,qfnbpE=1)(v/lenEdge);
                                          //    computation of the mesh size
                                          //    ----------------------------
count=vedgecount(0,Vh);                     //    number of edge / vertex
h[]=vmeshsizen(0,Vh);                     //    sum length edge / vertex
h[]=h[]./count;                           //    mean lenght edge / vertex
```

To compute error indicator for Poisson equation :

$$\eta_K = \int_K h_K^2 |(f + \Delta u_h)|^2 + \int_{\partial K} h_e |[\frac{\partial u_h}{\partial n}]|^2$$

where h_K is size of the longest edge (`hTriangle`), h_e is the size of the current edge (`lenEdge`), n the normal.

```
fespace Nh(Th,P0);              //    the space function contant / triangle
Nh etak;
varf vetaK(unused,chiK) =
    intalledges(Th)(chiK*lenEdge*square(jump(N.x*dx(u)+N.y*dy(u))))
   +int2d(Th)(chiK*square(hTriangle*(f+dxx(u)+dyy(u))) );

etak[]= vetaK(0,Ph);
```

We add automatic expression optimization by default, if this optimization creates problems, it can be removed with the keyword `optimize` as in the following example :

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

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

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


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

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

and after some operations on sparse matrices are available for example

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

6.11 Interpolation matrix

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

`inside=` set true to create zero-extension.

`t=` set true to get the transposed matrix

`op=` set an integer written below

- **0** the default value and interpolate of the function
- **1** interpolate the ∂_x
- **2** interpolate the ∂_y

. .

Example 6.2 (mat_interpol.edp)

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

matrix IV= interpolate(Vh,Vh4);                    //    here the function is
                                                   //    exended by continuity
cout << " IV Vh<-Vh4 " << IV << endl;
Vh v, vv;           Vh4 v4=x*y;
v=v4;               vv[]= IV*v4[];
                                                   //    here v == vv =>
real[int]  diff= vv[] - v[];
cout << " || v - vv || = " <<  diff.linfty << endl;
assert( diff.linfty<= 1e-6);

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

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

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

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

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

6.12 Finite elements connectivity

Here, we show how get the informations of a finite element space $W_h(\mathcal{T}_n, *)$, where "*" denotes P1, P2, P1nc, etc.

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

See the following for an example:

Example 6.3 (FE.edp)
```
mesh Th=square(5,5);
fespace Wh(Th,P2);
cout << " nb of degree of freedom           : " << Wh.ndof << endl;
cout << " nb of degree of freedom / ELEMENT : " << Wh.ndofK << endl;
 int k= 2;                                                    //    element 2
 int kdf= Wh.ndofK ;
 cout << " df of element " << k << ":" ;
 for (int i=0;i<kdf;i++)
    cout << Wh(k,i) << " ";
 cout << endl;
```

and the output is:

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

Chapter 7

Visualization

Results created by FEM are huge data, so it is very important to render them visible. There are two ways of visualization in FreeFem++ : One, the default view, supports the drawing of meshes, isovalues of real FE-functions and of vector fields, all by the command **plot** (see Section 7.1 below). For later use, FreeFem++ can store these plots as postscript files.
Another method is to use the external tools, for example, gnuplot (see Section 7.2), medit (see Section 7.3) using the command **system** to launch them and/or to save the data in text files.

7.1 Plot

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

Note 7.1 *The length of a arrow is always bound to be in* $[5‰, 5\%]$ *of the screen size, to see something not a porcupine.*

The parameters are

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

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

p to save to a postscript file,

? to show all actives keyboard char,

to redraw, otherwise we continue.

ps= string expression to save the plot on postscript file

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

fill= to fill between isovalues.

cmm= string expression to write in the graphic window

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

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

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

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

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

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

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

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

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

hsv= (array of float) to defined color of 3*n value in HSV color model declare with for example

```
real[int] colors = [h1,s1,v1,... ,  hn,vn,vn];
```

where `hi,si,vi` is the ith color to defined the color table.

boundary= (bool) to plot or not the boundary of the domain (true by default).

For example:

```
real[int] xx(10),yy(10);
mesh Th=square(5,5);
fespace Vh(Th,P1);
Vh uh=x*x+y*y,vh=-y^2+x^2;
int i;
                                                    //    compute a cut
for (i=0;i<10;i++)
 {
   x=i/10.; y=i/10.;
   xx[i]=i;
   yy[i]=uh;                          //    value of uh at point (i/10.  , i/10.)
```

```
}
plot(Th,uh,[uh,vh],value=true,ps="three.eps",wait=true);      //    figure 7.1
   //    zoom on box defined by the two corner points [0.1,0.2] and [0.5,0.6]
plot(uh,[uh,vh],bb=[[0.1,0.2],[0.5,0.6]],
       wait=true,grey=1,fill=1,value=1,ps="threeg.eps");      //    figure 7.2
plot([xx,yy],ps="likegnu.eps",wait=true);                     //    figure 7.3
```

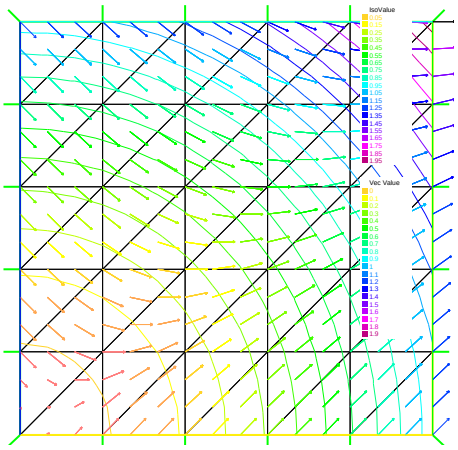


Figure 7.1: mesh, isovalue, and vector

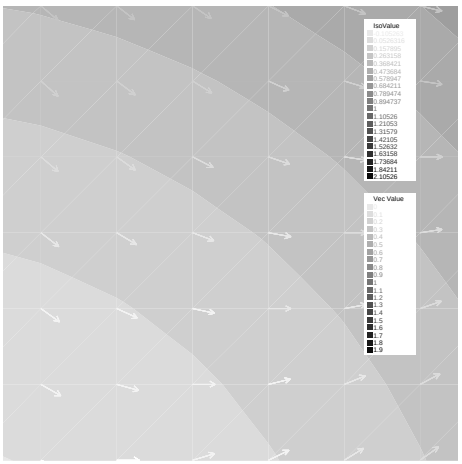


Figure 7.2: enlargement in grey of isovalue, and vector

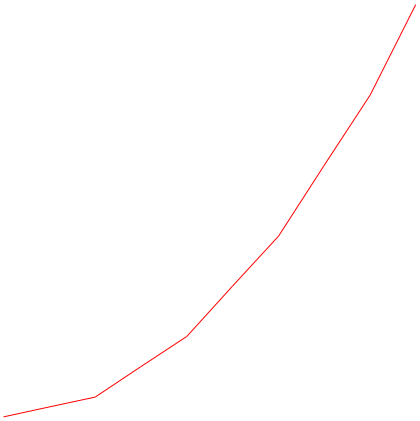


Figure 7.3: Plots a cut of uh. Note that a refinement of the same can be obtained in combination with gnuplot

To change the color table and to choose the value of iso line you can do :

```
//    from:  http://en.wikipedia.org/wiki/HSV_color_space
//   The HSV (Hue, Saturation, Value) model,
//    defines a color space in terms of three constituent components:
//
//   HSV color space as a color wheel 7.4
```

```
//   Hue, the color type (such as red, blue, or yellow):
//    Ranges from 0-360 (but normalized to 0-100% in some applications Here)
//   Saturation, the "vibrancy" of the color:  Ranges from 0-100%
//    The lower the saturation of a color, the more "grayness" is present
//    and the more faded the color will appear.
//   Value, the brightness of the color:
//    Ranges from 0-100%
//
real[int] colorhsv=[                                          //    color hsv model
  4./6., 1 , 0.5,                                             //    dark blue
  4./6., 1 , 1,                                                //    blue
  5./6., 1 , 1,                                                //    magenta
  1, 1. , 1,                                                   //    red
  1, 0.5 , 1                                                   //    light red
   ];
 real[int] viso(31);

 for (int i=0;i<viso.n;i++)
   viso[i]=i*0.1;

 plot(uh,viso=viso(0:viso.n-1),value=1,fill=1,wait=1,hsv=colorhsv);
```

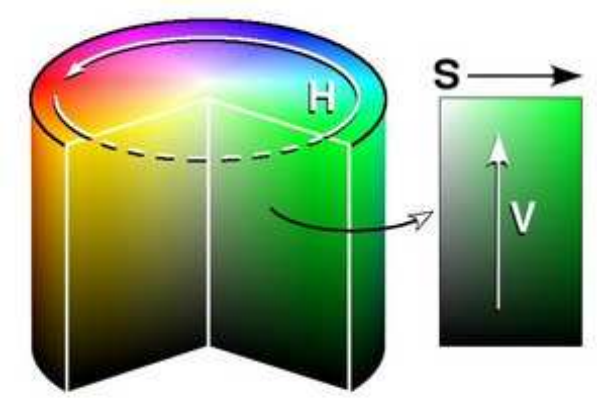


Figure 7.4: hsv color cylinder

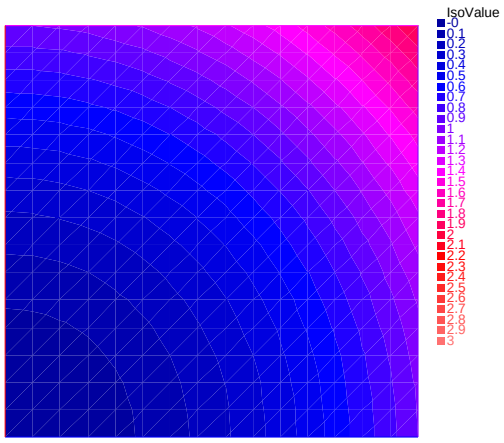


Figure 7.5: isovalue with an other color table

7.2 link with gnuplot

Example 3.2 shows how to generate a gnu-plot from a FreeFem++ file. Let us present here another technique which has the advantage of being online, i.e. one doesn't need to quit FreeFem++ to generate a gnu-plot. But this work only if gnuplot[1] is installed , and only on unix computer. Add to the previous example:

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

      //    to call gnuplot command and wait 5 second (thanks 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");
```

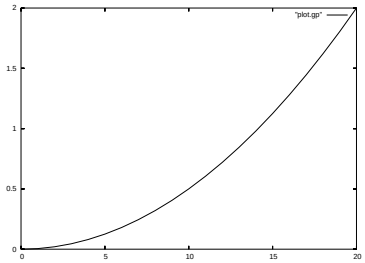


Figure 7.6: Plots a cut of uh with gnuplot

7.3 link with medit

First you must install medit [2], a freeware display package by Pascal Frey using OpenGL. Then you may run the follwoing example.

```
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
```

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

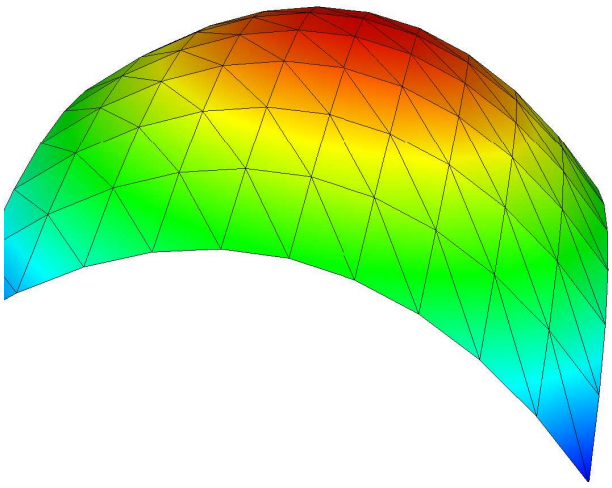


Figure 7.7: medit plot

```
{ ofstream file("mm.bb");
 file << "2 1 1 "<< u[].n << " 2 \n";
 for (int 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");
```

Chapter 8

Algorithms

The complete example is in `algo.edp` file.

8.1 conjugate Gradient/GMRES

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

$$\nabla J(x) = \left(\frac{\partial J}{\partial x_i}(\boldsymbol{x})\right) = 0 \tag{8.1}$$

where J is a functional (to minimize for example) from $\mathbb{R}^n$ to $\mathbb{R}$.
If the function is convex we can use the conjugate gradient to solve the problem, and we just need the function (named `dJ` for example) which compute ∇J, so the two parameters are the name of the function with prototype `func real[int] dJ(real[int] & xx)` which compute ∇J, a vector `x` of type `real[int]` to initialize the process and get the result.
Given an initial value $\boldsymbol{x}^{(0)}$, a maximum number $i_{\max}$ of iterations, and an error tolerance $0 < \epsilon < 1$: Put $\boldsymbol{x} = \boldsymbol{x}^{(0)}$ and write

`NLCG(`∇J`, `$\boldsymbol{x}$`, precon=`M`, nbiter=`$i_{\max}$`, eps=`ϵ`);`

will give the solution of $\boldsymbol{x}$ of $\nabla J(\boldsymbol{x}) = 0$. We can omit parameters `precon, nbiter, eps`. Here M is the preconditioner whose default is the identity matrix. The stopping test is

$$\|\nabla J(\boldsymbol{x})\|_P \leq \epsilon \|\nabla J(\boldsymbol{x}^{(0)})\|_P$$

Writing the minus value in `eps=`, i.e.,

`NLCG(`∇J`, `$\boldsymbol{x}$`, precon=`M`, nbiter=`$i_{\max}$`, eps=`$-\epsilon$`);`

we can use the stopping test

$$\|\nabla J(\boldsymbol{x})\|_P^2 \leq \epsilon$$

The parameters of these three functions are:

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

`precon=` set the preconditioner 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 $\|\nabla J(x)\|_P \le \varepsilon\|\nabla J(x_0)\|_P$, otherwise the absolute test is $\|\nabla J(x)\|_P^2 \le |\varepsilon|$.

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

Example 8.1 (algo.edp) *For a given function b, let us find the minimizer u of the functional*

$$J(u) = \frac{1}{2}\int_\Omega f(|\nabla u|^2) - \int_\Omega ub$$
$$f(x) = ax + x - \ln(1+x), \quad f'(x) = a + \frac{x}{1+x}, \quad f''(x) = \frac{1}{(1+x)^2}$$

under the boundary condition $u = 0$ on $\partial\Omega$.

```
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

```
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)(0.5*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 preconditioner C with solving the problem: find $u_h \in V_{0h}$ such that*

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

*where $\alpha = f'(|\nabla u|^2)$. */*

```
varf alap(uh,vh,solver=Cholesky,init=iter)=
   int2d(Th)( alpha *( dx(uh)*dx(vh) + dy(uh)*dy(vh) )) + on(1,2,3,4,uh=0);

varf amass(uh,vh,solver=Cholesky,init=iter)=
   int2d(Th)( uh*vh)  + on(1,2,3,4,uh=0);

matrix Amass = alap(Vh,Vh,solver=CG);                                    //

matrix Alap=  alap(Vh,Vh,solver=Cholesky,factorize=1);                   //

                                                 //    the preconditionner function
func real[int] C(real[int] & x)
{
   real[int] u(x.n);
   u=Amass*x;
   x = Alap^-1*u;
   x = x .* Cl[];
   return x;                                 //    no return of local array variable
}
```

/ To solve the problem, we make 10 iteration of the conjugate gradient, recompute the preconditioner 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");
```

For a given symmetric positive matrix A, consider the quadratic form

$$J(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}$$

then $J(\boldsymbol{x})$ is minimized by the solution $\boldsymbol{x}$ of $A\boldsymbol{x} = \boldsymbol{b}$. In this case, we can use the function `LinearCG`

`LinearCG(`A`,` $\boldsymbol{x}$`, precon=`M`, nbiter=`$i_{\max}$`, eps=`$\pm\epsilon$`);`

If A is not symmetric, we can use GMRES(Generalized Minimum Residual) algorithm by

`LinearGMRES(`A`,` $\boldsymbol{x}$`, precon=`M`, nbiter=`$i_{\max}$`, eps=`$\pm\epsilon$`);`

Also, we can use the non-linear version of GMRES algorithm (the functional J is just convex)

`LinearGMRES(`∇J`,` $\boldsymbol{x}$`, precon=`M`, nbiter=`$i_{\max}$`, eps=`$\pm\epsilon$`);`

For detail of these algorithms, refer to [13][Chapter IV, 1.3].

8.2 Optimization

Two algorithms of COOOL a package [26] are interfaced with the Newton Raphson method (call `Newton`) and the `BFGS` method. Be careful of these algorithms, because their implementation use full matrices.
Example of utilization

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

Chapter 9

Mathematical Models

Summary *This chapter goes more in depth into a number of problems that* `FreeFem++` *can solve. It is a complement to chapter 3 which was only an introduction. Users are invited to contribute to make this data base of problem solutions grow.*

9.1 Static Problems

9.1.1 Soap Film

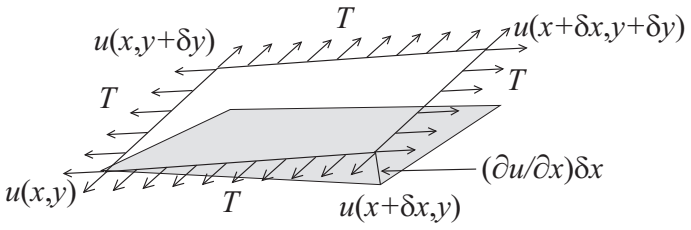
Our starting point here will be the mathematical model to find the shape of **soap film** which is glued to the ring on the $xy-$plane

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

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

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

Similarly, for the edges AB and DC we have



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

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

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

Assuming small displacements, we have

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

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

$$\mu dxdy\partial^2 u/\partial x^2 + \mu dxdy\partial^2 u/\partial y^2 + pdxdy = 0.$$

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

$$-\Delta u = f \quad \text{in } \Omega \tag{9.1}$$

where $f = p/\mu$, $\Omega = \{(x,y);\ x^2+y^2 < 1\}$. Poisson's equation (2.1) appear also in **electrostatics** taking the form of $f = \rho/\epsilon$ where ρ is the charge density, ϵ the dielectric constant and u is named as electrostatic potential. The soap film is glued to the ring $\partial\Omega = C$, then we have the boundary condition

$$u = 0 \quad \text{on } \partial\Omega \tag{9.2}$$

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

Example 9.1 (a_tutorial.edp)

```
 1 : border a(t=0,2*pi){ x = cos(t); y = sin(t);label=1;};
 2 :
 3 : mesh disk = buildmesh(a(50));
 4 : plot(disk);
 5 : fespace femp1(disk,P1);
 6 : femp1 u,v;
 7 : func f = -1;
 8 : problem laplace(u,v) =
 9 :     int2d(disk)( dx(u)*dx(v) + dy(u)*dy(v) )          //    bilinear form
10 :   - int2d(disk)( f*v )                                  //    linear form
11 :   + on(1,u=0) ;                                  //    boundary condition
12 : func ue = (x^2+y^2-1)/4;                         //    ue:  exact solution
13 : laplace;
14 : femp1 err = u - ue;
15 :
16 : plot (u,ps="aTutorial.eps",value=true,wait=true);
17 : plot(err,value=true,wait=true);
18 :
19 : cout << "error L2=" << sqrt(int2d(disk)( err^2) )<< endl;
20 : cout << "error H10=" << sqrt( int2d(disk)((dx(u)-x/2)^2)
21 :                              + int2d(disk)((dy(u)-y/2)^2))<< endl;
22 :
23 : disk = adaptmesh(disk,u,err=0.01);
24 : plot(disk,wait=1);
25 :
26 : laplace;
27 :
```

```
28 : plot (u,value=true,wait=true);
29 : err = u - ue;                          //    become FE-function on adapted mesh
30 : plot(err,value=true,wait=true);
31 : cout << "error L2=" << sqrt(int2d(disk)( err^2) )<< endl;
32 : cout << "error H10=" << sqrt(int2d(disk)((dx(u)-x/2)^2)
33 :                                + int2d(disk)((dy(u)-y/2)^2))<< endl;
```

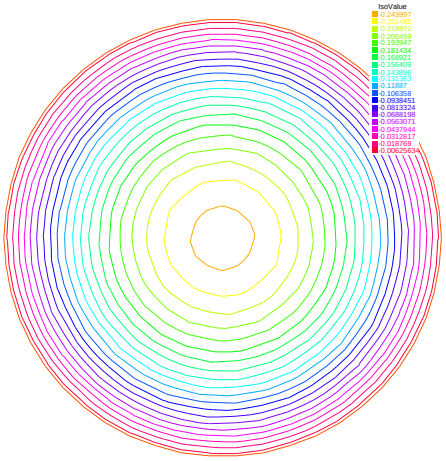


Figure 9.1: isovalue of u

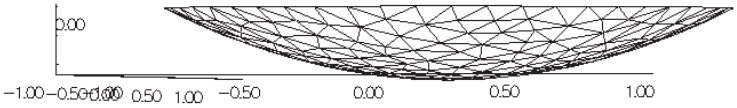


Figure 9.2: a side view of u

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

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

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

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

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

9.1.2 Electrostatics

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

$$\mathrm{div}\boldsymbol{E} = \rho/\epsilon, \quad \mathrm{curl}\boldsymbol{E} = 0 \tag{9.3}$$

where ρ is the charge density and ϵ is called the permittivity of free space. From the second equation in (9.3), we can introduce the electrostatic potential such that $\boldsymbol{E} = -\nabla\phi$. Then we have Poisson equation $-\Delta\phi = f$, $f = -\rho/\epsilon$. We now obtain the equipotential line which is the level curve of ϕ, when there are no charges except conductors $\{C_i\}_{1,\cdots,K}$. Let us assume K conductors $C_1, \cdots, C_K$ within an enclosure C_0. Each one is held at an electrostatic potential φ_i. We assume

that the enclosure $C0$ is held at potential 0. In order to know $\varphi(x)$ at any point x of the domain Ω, we must solve

$$-\Delta\varphi = 0 \quad \text{in } \Omega, \tag{9.4}$$

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

$$\varphi = \varphi_i \text{ on } C_i,\ i = 1...N, \varphi = 0 \text{ on } C_0. \tag{9.5}$$

Example 9.2 *First we give the geometrical informations;* $C_0 = \{(x,y);\ x^2 + y^2 = 5^2\}$, $C_1 = \{(x,y) : \frac{1}{0.3^2}(x-2)^2 + \frac{1}{3^2}y^2 = 1\}$, $C_2 = \{(x,y) : \frac{1}{0.3^2}(x+2)^2 + \frac{1}{3^2}y^2 = 1\}$. *Let* Ω *be the disk enclosed by* C_0 *with the elliptical holes enclosed by* C_1 *and* C_2. *Note that* C_0 *is described counterclockwise, whereas the elliptical holes are described clockwise, because the boundary must be oriented so that the computational domain is to its left.*

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

mesh Th = buildmesh(C0(60)+C1(-50)+C2(-50));
plot(Th,ps="electroMesh");                                    //    figure 9.3
fespace Vh(Th,P1);                                           //    P1 FE-space
Vh uh,vh;                                          //    unknown and test function.
problem Electro(uh,vh) =                            //    definition of the problem
    int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) )                    //    bilinear
    + on(C0,uh=0)                                    //    boundary condition on C0
    + on(C1,uh=1)                                          //    +1 volt on C1
    + on(C2,uh=-1) ;                                       //    -1 volt on C2

Electro;              //    solve the problem, see figure 9.4 for the solution
plot(uh,ps="electro.eps",wait=true);                          //    figure 9.4
```

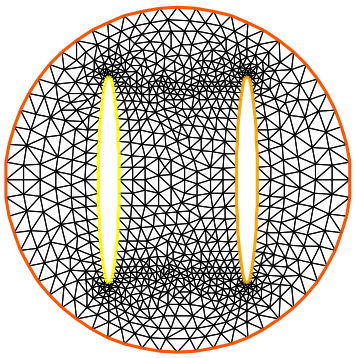


Figure 9.3: Disk with two elliptical holes

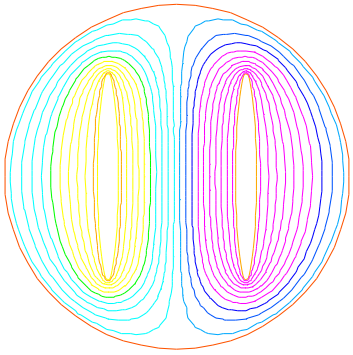


Figure 9.4: Equipotential lines, where C_1 is located in right hand side

9.1.3 Aerodynamics

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

$$\Delta\varphi = 0 \quad \text{in } \Omega, \quad \varphi|_S = c, \quad \varphi|_{\Gamma_\infty} = u_{\infty 1x} - u_{\infty 2x} \tag{9.6}$$

where Ω is the area occupied by the fluid, u_∞ is the air speed at infinity, c is a constant to be determined so that $\partial_n\varphi$ is continuous at the trailing edge P of S (so-called Kutta-Joukowski condition). Lift is proportional to c. To find c we use a superposition method. As all equations in (9.6) are linear, the solution φ_c is a linear function of c

$$\varphi_c = \varphi_0 + c\varphi_1, \tag{9.7}$$

where φ_0 is a solution of (9.6) with $c = 0$ and φ_1 is a solution with $c = 1$ and zero speed at infinity. With these two fields computed, we shall determine c by requiring the continuity of $\partial\varphi/\partial n$ at the trailing edge. An equation for the upper surface of a NACA0012 (this is a classical wing profile in aerodynamics; the rear of the wing is called the trailing edge) is:

$$y = 0.17735\sqrt{x} - 0.075597x - 0.212836x^2 + 0.17363x^3 - 0.06254x^4. \tag{9.8}$$

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

$$-\Delta\varphi = 0 \qquad \text{in } \Omega, \qquad \varphi|_{\Gamma_1} = y - 0.1x, \quad \varphi|_{\Gamma_2} = c, \tag{9.9}$$

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

$$-\Delta\varphi_0 = 0 \text{ in } \Omega, \qquad \varphi_0|_{\Gamma_1} = y - 0.1x, \quad \varphi_0|_{\Gamma_2} = 0, \tag{9.10}$$

$$-\Delta\varphi_1 = 0 \text{ in } \Omega, \qquad \varphi_1|_{\Gamma_1} = 0, \quad \varphi_1|_{\Gamma_2} = 1. \tag{9.11}$$

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

$$c = -\frac{\varphi_0|_{P^+} + \varphi_0|_{P^-}}{(\varphi_1|_{P^+} + \varphi_1|_{P^-} - 2)}, \tag{9.12}$$

which can be programmed as:

$$c = -\frac{\varphi_0(0.99, 0.01) + \varphi_0(0.99, -0.01)}{(\varphi_1(0.99, 0.01) + \varphi_1(0.99, -0.01) - 2)}. \tag{9.13}$$

Example 9.3

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

border a(t=0,2*pi) { x=5*cos(t);  y=5*sin(t); };    //    approximates infinity
```

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

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

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

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

plot(psi1);

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


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

9.1.4 Error estimation

There are famous estimation between the numerical result u_h and the exact solution u of the problem 2.1 and 2.2: If triangulations $\{\mathcal{T}_h\}_{h\downarrow 0}$ is regular (see Section 5.4), then we have the estimates

$$|\nabla u - \nabla u_h|_{0,\Omega} \leq C_1 h \tag{9.14}$$

$$\|u - u_h\|_{0,\Omega} \leq C_2 h^2 \tag{9.15}$$

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

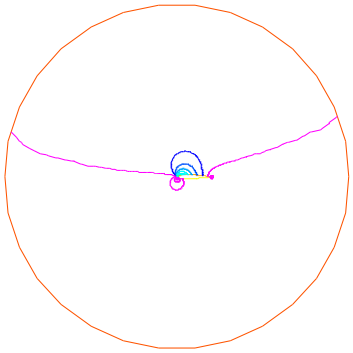


Figure 9.5: isovalue of $cp = -(\partial_x\psi)^2 - (\partial_y\psi)^2$

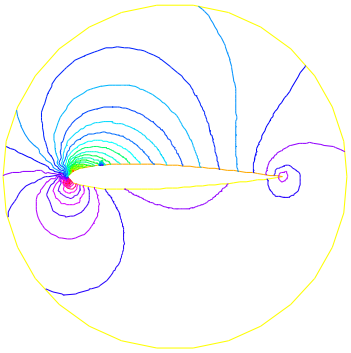


Figure 9.6: Zooming of cp

In this section we check (9.14) and (9.15). We will pick up numericall error if we use the numerical derivative, so we will use the following for (9.14).

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

The constants C_1, C_2 are depend on $\mathcal{T}_h$ and f, so we will find them by `FreeFem++` . In general, we cannot get the solution u as a elementary functions (see Section 4.7) even if spetical functions are added. Instead of the exact solution, here we use the approximate solution u_0 in $V_h(\mathcal{T}_h, P_2)$, $h \sim 0$.

Example 9.4

```
 1 : mesh Th0 = square(100,100);
 2 : fespace V0h(Th0,P2);
 3 : V0h u0,v0;
 4 : func f = x*y;                                    //    sin(pi*x)*cos(pi*y);
 5 :
 6 : solve Poisson0(u0,v0) =
 7 :     int2d(Th0)( dx(u0)*dx(v0) + dy(u0)*dy(v0) )      //    bilinear form
 8 :   - int2d(Th0)( f*v0 )                                 //    linear form
 9 :   + on(1,2,3,4,u0=0) ;                            //    boundary condition
10 :
11 : plot(u0);
12 :
13 : real[int] errL2(10), errH1(10);
14 :
15 : for (int i=1; i<=10; i++) {
16 :    mesh Th = square(5+i*3,5+i*3);
17 :    fespace Vh(Th,P1);
18 :    fespace Ph(Th,P0);
19 :    Ph h = hTriangle;                    //    get the size of all triangles
20 :    Vh u,v;
21 :    solve Poisson(u,v) =
```

```
22 :          int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v) )          //    bilinear form
23 :          - int2d(Th)( f*v )                                //    linear form
24 :          + on(1,2,3,4,u=0) ;                          //    boundary condition
25 :     V0h uu = u;
26 :     errL2[i-1] = sqrt( int2d(Th0)((uu - u0)^2) )/h[].max^2;
27 :     errH1[i-1] = sqrt( int2d(Th0)( f*(u0-2*uu+uu) ) )/h[].max;
28 : }
29 : cout << "C1 = " << errL2.max <<"("<<errL2.min<<")"<< endl;
30 : cout << "C2 = " << errH1.max <<"("<<errH1.min<<")"<< endl;
```

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

9.1.5 Periodic

We now solve the Poisson equation

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

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

Example 9.5 (periodic.edp)

```
mesh Th=square(10,10,[2*x*pi,2*y*pi]);
                              //    defined the fespacewith periodic condition
      //    label :  2 and 4 are left and right side with y the curve 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;                                          //    unknown and test function.
 func f=sin(x+pi/4.)*cos(y+pi/4.);                    //    right hand side function

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

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

The periodic condition does not necessarily require parallel to the axis. Example 9.6 give such example.

Example 9.6 (periodic4.edp)

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

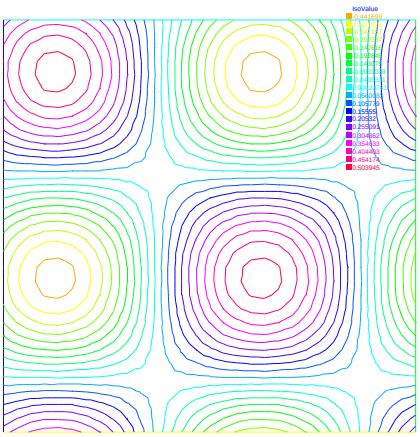


Figure 9.7: The isovalue of solution u with periodic boundary condition

```
border e(t=0,2*pi){ x=r*cos(t); y=-r*sin(t);label=0;};
int n = 10;
mesh Th= buildmesh(a(n)+b(n)+c(n)+d(n)+e(n));
plot(Th,wait=1);
real r2=1.732;
func abs=sqrt(x^2+y^2);
//    warning for periodic condition:
//    side a and c
//    on side a (label 1) x∈[0,1] or x-y∈[-1,1]
//    on side c (label 3) x∈[-1,0] or x-y∈[-1,1]
                    //    so the common abscissa can be respectively x and x+1
                      //    or you can 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");
```

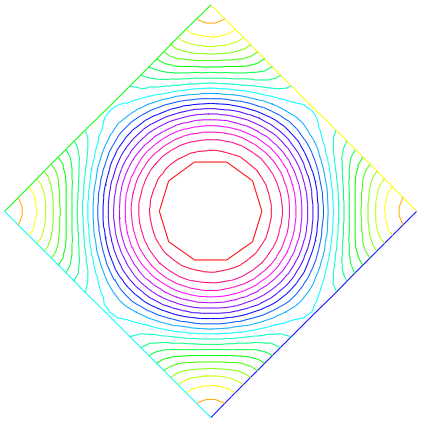


Figure 9.8: The isovalue of solution u for $\Delta u = ((y+x)^2+1)((y-x)^2+1) - k$, in Ω and $\partial_n u = 0$ on hole,and with two periodic boundary condition on external border

9.1.6 Poisson with mixed boundary condition

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

$$\begin{aligned} -\Delta u &= f \qquad \text{in } \Omega \\ u &= g \quad \text{on } \Gamma_D, \quad \partial u/\partial n = 0 \quad \text{on } \Gamma_N \end{aligned} \tag{9.16}$$

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

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

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

Example 9.7 *Assume that* $f = -2 \times 30(x^2+y^2)$ *and* $g = u_e = 10(x^2+y^2)^{1/4}\sin\left([\tan^{-1}(y/x)]/2\right) + 30(x^2y^2)$*, where* u_eS *is the exact solution.*

```
 1 : border N(t=0,1) { x=-1+t; y=0; label=1; };
 2 : border D1(t=0,1){ x=t;  y=0; label=2;};
 3 : border D2(t=0,1){ x=1; y=t; label=2; };
 4 : border D3(t=0,2){ x=1-t; y=1; label=2;};
 5 : border D4(t=0,1) { x=-1; y=1-t; label=2; };
 6 :
 7 : mesh T0h = buildmesh(N(10)+D1(10)+D2(10)+D3(20)+D4(10));
 8 : plot(T0h,wait=true);
 9 : fespace V0h(T0h,P1);
10 : V0h u0, v0;
11 :
12 : func f=-2*30*(x^2+y^2);                    //    given function
```

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

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

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

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

9.1.7 Poisson with mixte finite element

Here we consider the Poisson equation with mixed boundary value problems: For given functions f , g_d, g_n, find p such that

$$\begin{aligned} -\Delta p &= 1 \quad \text{in } \Omega \\ p &= g_d \quad \text{on } \Gamma_D, \quad \partial p/\partial n = g_n \quad \text{on } \Gamma_N \end{aligned} \tag{9.17}$$

where Γ_D is a part of the boundary Γ and $\Gamma_N = \Gamma \setminus \overline{\Gamma_D}$.
The mixte formulation is: find p and $\mathbf{u}$ such that

$$\begin{aligned} \nabla p + \mathbf{u} &= \mathbf{0} \quad \text{in } \Omega \\ \nabla.\mathbf{u} &= f \quad \text{in } \Omega \\ p &= g_d \quad \text{on } \Gamma_D, \quad \partial u.n = \mathbf{g}_n.n \quad \text{on } \Gamma_N \end{aligned} \tag{9.18}$$

where $\mathbf{g}_n$ is a vector such that $\mathbf{g}_n.n = g_n$.
The variationnal formulation is,

$$\begin{aligned} \forall \mathbf{v} \in \mathbb{V}_0, \quad \int_\Omega p\nabla.v + \mathbf{v}\mathbf{v} &= \int_{\Gamma_d} g_d \mathbf{v}.n \\ \forall q \in \mathbb{P} \quad \int_\Omega q\nabla.u &= \int_\Omega qf \\ \partial u.n &= \mathbf{g}_n.n \quad \text{on } \Gamma_N \end{aligned} \tag{9.19}$$

where the functionnal space are:

$$\mathbb{P} = L^2(\Omega), \qquad \mathbb{V} = H(div) = \{\mathbf{v} \in L^2(\Omega)^2, \nabla.\mathbf{v} \in L^2(\Omega)\}$$

and

$$\mathbb{V}_0 = \{\mathbf{v} \in \mathbb{V}; \quad \mathbf{v}.n = 0 \quad \text{on } \Gamma_N\}.$$

To write, the `FreeFem++` example, we have just to choose the finites elements spaces. here $\mathbb{V}$ space is discretize with Raviart-Thomas finite element `RT0` and $\mathbb{P}$ is discretize by constant finite element `P0`.

Example 9.8 (LaplaceRT.edp)

```
mesh Th=square(10,10);
fespace Vh(Th,RT0);
fespace Ph(Th,P0);
func gd = 1.;
func g1n = 1.;
func g2n = 1.;

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

problem laplaceMixte([u1,u2,p],[v1,v2,q],
                     solver=GMRES,eps=1.0e-10,
                     tgv=1e30,dimKrylov=150)
    =
    int2d(Th)( p*q*1e-15                    //    this term is here to be sur
```

```
                    //    that all sub matrix are inversible (LU requirement)
             + u1*v1 + u2*v2 + p*(dx(v1)+dy(v2)) + (dx(u1)+dy(u2))*q )
  + int2d(Th) ( q)
  - int1d(Th,1,2,3)( gd*(v1*N.x +v2*N.y))                          //    on ΓD
  + on(4,u1=g1n,u2=g2n);                                           //    on ΓN

 laplaceMixte;

 plot([u1,u2],coef=0.1,wait=1,ps="lapRTuv.eps",value=true);
 plot(p,fill=1,wait=1,ps="laRTp.eps",value=true);
```

9.1.8 Metric Adaptation and residual error indicator

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

Example 9.9 (adaptindicatorP2.edp) *First, we solve the same problem as in a previous example.*

```
 1 : border ba(t=0,1.0){x=t;   y=0;  label=1;};              //    see Fig,5.13
 2 : border bb(t=0,0.5){x=1;   y=t;  label=2;};
 3 : border bc(t=0,0.5){x=1-t; y=0.5;label=3;};
 4 : border bd(t=0.5,1){x=0.5; y=t;  label=4;};
 5 : border be(t=0.5,1){x=1-t; y=1;  label=5;};
 6 : border bf(t=0.0,1){x=0;   y=1-t;label=6;};
 7 : mesh Th = buildmesh (ba(6) + bb(4) + bc(4) +bd(4) + be(4) + bf(6));
 8 : savemesh(Th,"th.msh");
 9 : fespace Vh(Th,P2);
10 : fespace Nh(Th,P0);
11 : Vh u,v;
12 : Nh rho;
13 : real[int] viso(21);
14 : for (int i=0;i<viso.n;i++)
15 :   viso[i]=10.^(+(i-16.)/2.);
16 : real error=0.01;
17 : func f=(x-y);
18 : problem Probem1(u,v,solver=CG,eps=1.0e-6) =
19 :     int2d(Th,qforder=5)( u*v*1.0e-10+  dx(u)*dx(v) + dy(u)*dy(v))
20 :   + int2d(Th,qforder=5)( -f*v);
21 : /*************
```

Now, the local error indicator η_T is:

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

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

```
29 : *************/
30 :
31 : varf indicator2(uu,chiK) =
32 :      intalledges(Th)(chiK*lenEdge*square(jump(N.x*dx(u)+N.y*dy(u))))
33 :     +int2d(Th)(chiK*square(hTriangle*(f+dxx(u)+dyy(u))) );
34 : for (int i=0;i< 4;i++)
35 : {
36 :   Probem1;
37 :    cout << u[].min << " " << u[].max << endl;
38 :    plot(u,wait=1);
39 :    cout << " indicator2 " << endl;
40 :
41 :    rho[] = indicator2(0,Nh);
42 :    rho=sqrt(rho);
43 :    cout << "rho =   min " << rho[].min << " max=" << rho[].max << endl;
44 :    plot(rho,fill=1,wait=1,cmm="indicator density ",ps="rhoP2.eps",
                                        value=1,viso=viso,nbiso=viso.n);
45 :    plot(Th,wait=1,cmm="Mesh ",ps="ThrhoP2.eps");
46 :    Th=adaptmesh(Th,[dx(u),dy(u)],err=error,anisomax=1);
47 :    plot(Th,wait=1);
48 :    u=u;
49 :    rho=rho;
50 :   error = error/2;
51 : } ;
```

If the method is correct, we expect to look the graphics by an almost constant function η on your computer as in Fig. 9.9.

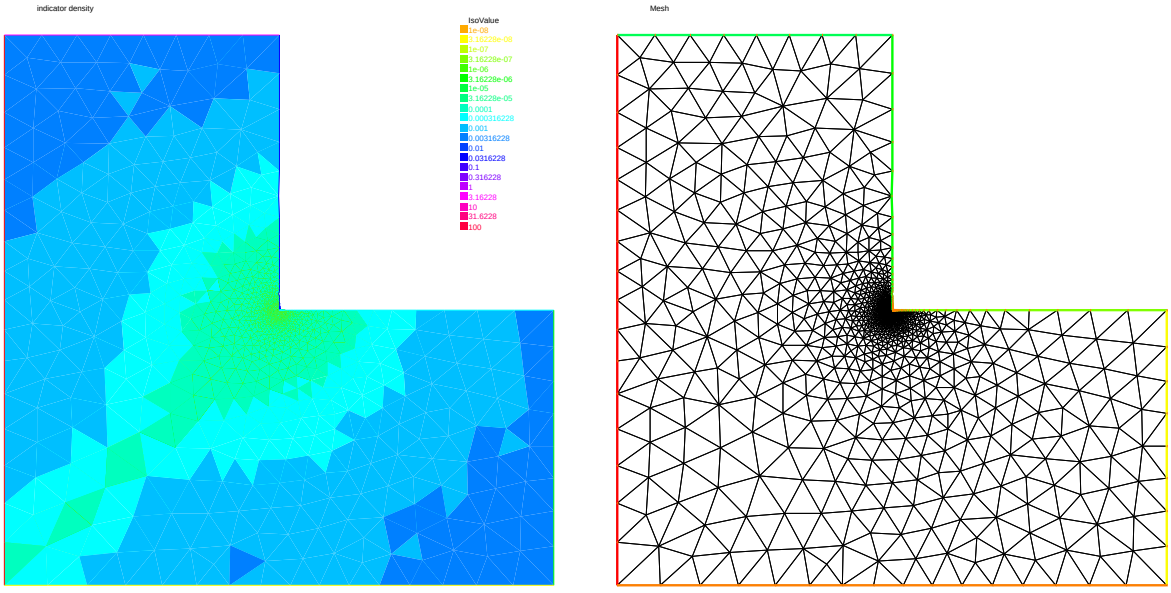


Figure 9.9: Density of the error indicator with isotropic P_2 metric

9.1.9 Adaptation using residual error indicator

In the previous example we compute the error indicator, now we use it, to adapt the mesh. The new mesh size is given by the following formulae:

$$h_{n+1}(x) = \frac{h_n(x)}{f_n(\eta_K(x))}$$

where $\eta_n(x)$ is the level of error at point x given by the local error indicator, h_n is the previous "mesh size" field, and f_n is a user function define by $f_n = min(3, max(1/3, \eta_n/\eta_n^*))$ where $\eta_n^* = mean(\eta_n)c$, and c is an user coefficient generally close to one.

Example 9.10 (AdaptResidualErrorIndicator.edp)
First a macro `MeshSizecomputation` *to get a P_1 mesh size as the average of edge lenght.*

```
//    macro the get the current mesh size
//    parameter
//    in:  Th the mesh
//    Vh P1 fespace on Th
//    out :
//    h:  the Vh finite element finite set to the current mesh size
macro  MeshSizecomputation(Th,Vh,h)
{  /* Th mesh   Vh  P1 finite element space
 h   the P1 mesh size value */
real[int]  count(Th.nv);
/* mesh size  (lenEdge =  integral(e) 1 ds)  */
varf vmeshsizen(u,v)=intalledges(Th,qfnbpE=1)(v);
/* number of edge / par vertex */
varf vedgecount(u,v)=intalledges(Th,qfnbpE=1)(v/lenEdge);
   /*
  computation of the mesh size
  ---------------------------- */
count=vedgecount(0,Vh);
h[]=0.;
h[]=vmeshsizen(0,Vh);
cout << " count min = "<< count.min << " " << count.max << endl;
h[]=h[]./count;
    cout << " -- bound meshsize = " <<h[].min << " " << h[].max << endl;
} //    end of macro MeshSizecomputation
```

A second macro to remesh according to the new mesh size.

```
//    macro to remesh according the de residual indicator
//    in:
//    Th the mesh
//    Ph P0 fespace on Th
//    Vh P1 fespace on Th
//    vindicator the varf of to evaluate the indicator to ^2
//    coef on etameam ..
//    ------

macro  ReMeshIndicator(Th,Ph,Vh,vindicator,coef)
{
Vh h=0;
/*evalutate the mesh size  */
```

```
MeshSizecomputation(Th,Vh,h);
Ph etak;
etak[]=vindicator(0,Ph);
etak[]=sqrt(etak[]);
real etastar= coef*(etak[].sum/etak[].n);
cout << " etastar = " << etastar << " sum=" << etak[].sum << " " << endl;

/* here etaK is discontinous
   we use the P1 L2 projection with mass lumping . */

Vh fn,sigma;
varf veta(unused,v)=int2d(Th)(etak*v);
varf vun(unused,v)=int2d(Th)(1*v);
fn[]  = veta(0,Vh);
sigma[]= vun(0,Vh);
fn[]= fn[]./ sigma[];
fn =  max(min(fn/etastar,3.),0.3333) ;

/* new mesh size */
h = h / fn ;
/* plot(h,wait=1); */
/*  build  the new mesh */
Th=adaptmesh(Th,IsMetric=1,h,splitpbedge=1,nbvx=10000);
}
```

We skip the mesh construction, see the previous example,

```
                                                //    FE space definition ---
fespace Vh(Th,P1);                          //    for the mesh size and solution
fespace Ph(Th,P0);                           //    for the error indicator

real hinit=0.2;                                          //    initial mesh size
Vh   h=hinit;                              //    the FE function for the mesh size
                        //    to build a mesh with a given mesh size :  meshsize
Th=adaptmesh(Th,h,IsMetric=1,splitpbedge=1,nbvx=10000);
plot(Th,wait=1,ps="RRI-Th-init.eps");
Vh u,v;

func f=(x-y);

problem Poisson(u,v) =
    int2d(Th,qforder=5)( u*v*1.0e-10+  dx(u)*dx(v) + dy(u)*dy(v))
  - int2d(Th,qforder=5)( f*v);

 varf indicator2(unused,chiK) =
     intalledges(Th)(chiK*lenEdge*square(jump(N.x*dx(u)+N.y*dy(u))))
    +int2d(Th)(chiK*square(hTriangle*(f+dxx(u)+dyy(u))) );

for (int i=0;i< 10;i++)
{
u=u;
Poisson;
plot(Th,u,wait=1);
real cc=0.8;
if(i>5) cc=1;
ReMeshIndicator(Th,Ph,Vh,indicator2,cc);
```

```
plot(Th,wait=1);
}
```

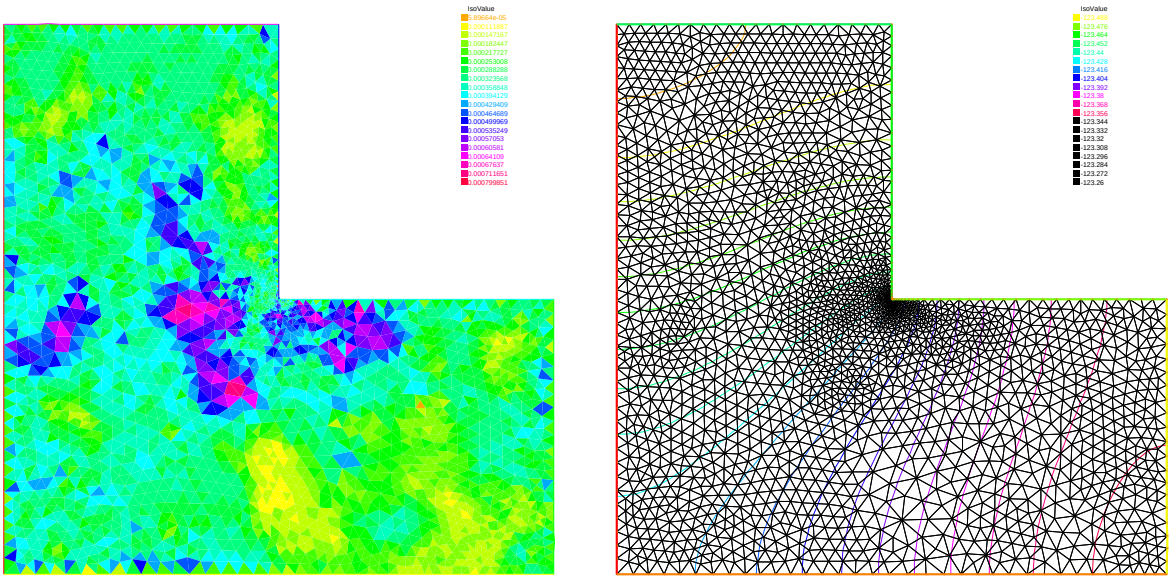


Figure 9.10: the error indicator with isotropic P_1 , the mesh and isovalue of the solution

9.2 Elasticity

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

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

then we have (see e.g. [15, p.32])

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

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

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

and the following is called *small* strain tensor

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

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

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

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

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

with the symmetry $c_{ijkl} = c_{jikl}, c_{ijkl} = c_{ijlk}, c_{ijkl} = c_{klij}$.
If Hooke's tensor $c_{ijkl}(\mathbf{x})$ do not depend on the choice of coordinate system, the material is called *isotropic* at $\mathbf{x}$. If c_{ijkl} is constant, the material is called *homogeneous*. In homogeneous isotropic case, there is *Lamé constants* λ, μ (see e.g. [15, p.43]) satisfying

$$\sigma_{ij} = \lambda\delta_{ij}\mathrm{div}u + 2\mu\varepsilon_{ij} \tag{9.20}$$

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

$$-\partial_j\sigma_{ij} = f_i \text{ in } \Omega\times]-h, h[, \quad i = 1, 2, 3 \tag{9.21}$$

$$\sigma_{ij}n_j = g_i \text{ on } \Gamma_N\times]-h, h[, \quad u_i = 0 \text{ on } \Gamma_D\times]-h, h[, \quad i = 1, 2, 3 \tag{9.22}$$

We now explain the plain elasticity.

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

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

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

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

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

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

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

In what follows we omit the overlines of $\overline{u}, \overline{f}, \overline{g}, \overline{\varepsilon}_{ij}$ and $\overline{\varepsilon}_{ij}$. Then we obtain similar equation of equilibrium given in (9.21) replacing $\Omega\times]-h, h[$ with Ω and changing $i = 1, 2$. In the case of plane stress, $\sigma_{ij} = \lambda^*\delta_{ij}\mathrm{div}u + 2\mu\varepsilon_{ij}$, $\lambda^* = (2\lambda\mu)/(\lambda + \mu)$.
The equations of elasticity are naturally written in variational form for the displacement vector $u(x) \in V$ as

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

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

Example 9.11 (Beam.edp) *Consider elastic plate with the undeformed rectangle shape* $[0, 10] \times [0, 2]$. *The body force is the gravity force* $\boldsymbol{f}$ *and the boundary force* $\boldsymbol{g}$ *is zero on lower and upper side. On the two vertical sides of the beam are fixed.*

```
                                                //    a weighting beam sitting on a

int bottombeam = 2;
border a(t=2,0)  { x=0; y=t ;label=1;};                          //    left beam
border b(t=0,10) { x=t; y=0 ;label=bottombeam;};            //    bottom of beam
border c(t=0,2)  { x=10; y=t ;label=1;};                        //    rigth beam
border d(t=0,10) { x=10-t; y=2; label=3;};                        //    top beam
real E = 21.5;
real sigma = 0.29;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -0.05;
mesh th = buildmesh( b(20)+c(5)+d(20)+a(5));
fespace Vh(th,[P1,P1]);
Vh [uu,vv], [w,s];
cout << "lambda,mu,gravity ="<<lambda<< " " << mu << " " << gravity << endl;
                                //    deformation of a beam under its own weight
real sqrt2=sqrt(2.);                                //    see lame.edp example 3.9
macro epsilon(u1,u2)  [dx(u1),dy(u2),(dy(u1)+dx(u2))/sqrt2]          //    EOM
macro div(u,v)  ( dx(u)+dy(v) )                                      //    EOM

solve bb([uu,vv],[w,s])=
        int2d(th)(
                  lambda*div(w,s)*div(uu,vv)
                  +2.*mu*( epsilon(w,s)'*epsilon(uu,vv) )
                 )
  + 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);
```

9.2.1 Fracture Mechanics

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

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

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

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

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

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

If the elasticity is homogeneous isotropic, then the displacement $\boldsymbol{u}(x)$ is decomposed in an open neighborhood U_k of γ_k as in (see e.g. [16])

$$\boldsymbol{u}(x) = \sum_{l=1}^{2} K_l(\gamma_k) r_k^{1/2} S_{kl}^C(\theta_k) + \boldsymbol{u}_{k,R}(x) \quad \text{for } x \in \Omega_\Sigma \cap U_k,\ k = 1, 2 \tag{9.23}$$

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

$$\begin{aligned} S_{k1}^C(\theta_k) &= \frac{1}{4\mu}\frac{1}{(2\pi)^{1/2}} \begin{bmatrix} [2\kappa - 1]\cos(\theta_k/2) - \cos(3\theta_k/2) \\ -[2\kappa + 1]\sin(\theta_k/2) + \sin(3\theta_k/2) \end{bmatrix}, \\ S_{k2}^C(\theta_k) &= \frac{1}{4\mu}\frac{1}{(2\pi)^{1/2}} \begin{bmatrix} -[2\kappa - 1]\sin(\theta_k/2) + 3\sin(3\theta_k/2) \\ -[2\kappa + 1]\cos(\theta_k/2) + \cos(3\theta_k/2) \end{bmatrix}. \end{aligned} \tag{9.24}$$

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

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

with only one crack tip $\gamma = (0, 0)$. Unfortunately, `FreeFem++` cannot treat crack, so we use the modification of the domain with U-shape channel (see Fig. 5.28) with $d = 0.0001$. The undeformed crack Σ is approximated by

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

and $\Gamma_D =$ `R` in Fig. 5.28. In this example, we use three technique:

- Fast Finite Element Interpolator from the mesh `Th` to `Zoom` for the scale-up of near γ.
- After obtaining the displacement vector $\boldsymbol{u} = (u, v)$, we shall watch the deformation of the crack near γ as follows,

```
mesh Plate = movemesh(Zoom,[x+u,y+v]);
plot(Plate);
```

- Important technique is adaptive mesh, because the large singularity occur at γ as shown in (9.23).

First example create mode I deformation by the opposed surface force on `B` and `T` in the vertical direction of Σ, and the displacement is fixed on `R`.
In a laboratory, fracture engineer use photoelasticity to make stress field visible, which shows the principal stress difference

$$\sigma_1 - \sigma_2 = \sqrt{(\sigma_{11} - \sigma_{22})^2 + 4\sigma_{12}^2} \tag{9.25}$$

where σ_1 and σ_2 are the principal stresses. In opening mode, the photoelasticity make symmetric pattern concentrated at γ.

Example 9.12 (Crack Opening, $K_2(\gamma) = 0$) `{CrackOpen.edp}`

```
real d = 0.0001;
int n = 5;
real cb=1, ca=1, tip=0.0;
border L1(t=0,ca-d) { x=-cb; y=-d-t; }
border L2(t=0,ca-d) { x=-cb; y=ca-t; }
border B(t=0,2) { x=cb*(t-1); y=-ca; }
border C1(t=0,1) { x=-ca*(1-t)+(tip-10*d)*t; y=d; }
border C21(t=0,1) { x=(tip-10*d)*(1-t)+tip*t; y=d*(1-t); }
border C22(t=0,1) { x=(tip-10*d)*t+tip*(1-t); y=-d*t; }
border C3(t=0,1) { x=(tip-10*d)*(1-t)-ca*t; y=-d; }
border C4(t=0,2*d) { x=-ca; y=-d+t; }
border R(t=0,2) { x=cb; y=cb*(t-1); }
border T(t=0,2) { x=cb*(1-t); y=ca; }
mesh Th = buildmesh (L1(n/2)+L2(n/2)+B(n)
                     +C1(n)+C21(3)+C22(3)+C3(n)+R(n)+T(n));
cb=0.1; ca=0.1;
plot(Th,wait=1);
mesh Zoom = buildmesh (L1(n/2)+L2(n/2)+B(n)+C1(n)
                       +C21(3)+C22(3)+C3(n)+R(n)+T(n));
plot(Zoom,wait=1);
real E = 21.5;
real sigma = 0.29;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
fespace Vh(Th,[P2,P2]);
fespace zVh(Zoom,P2);
Vh [u,v], [w,s];
solve  Problem([u,v],[w,s])  =
    int2d(Th)(
              2*mu*(dx(u)*dx(w)+ ((dx(v)+dy(u))*(dx(s)+dy(w)))/4 )
              + lambda*(dx(u)+dy(v))*(dx(w)+dy(s))/2
              )
    -int1d(Th,T)(0.1*(4-x)*s)+int1d(Th,B)(0.1*(4-x)*s)
    +on(R,u=0)+on(R,v=0);                                    //    fixed
;

zVh Sx, Sy, Sxy, N;
for (int i=1; i<=5; i++)
{
  mesh Plate = movemesh(Zoom,[x+u,y+v]);              //    deformation near γ
  Sx = lambda*(dx(u)+dy(v)) + 2*mu*dx(u);
  Sy = lambda*(dx(u)+dy(v)) + 2*mu*dy(v);
  Sxy = mu*(dy(u) + dx(v));
  N = 0.1*1*sqrt((Sx-Sy)^2+4*Sxy^2);          //   principal stress difference
```

```
  if (i==1) {
     plot(Plate,ps="1stCOD.eps",bw=1);                    //    Fig. 9.11
     plot(N,ps="1stPhoto.eps",bw=1);                      //    Fig. 9.11
  } else if (i==5) {
     plot(Plate,ps="LastCOD.eps",bw=1);                   //    Fig. 9.12
     plot(N,ps="LastPhoto.eps",bw=1);                     //    Fig. 9.12
     break;
  }
  Th=adaptmesh(Th,[u,v]);
  Problem;
}
```

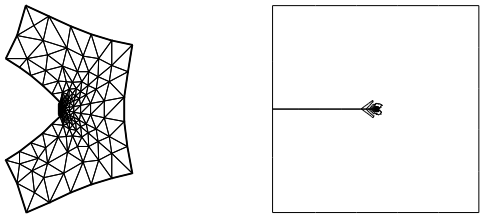


Figure 9.11: Crack open displacement (COD) and Principal stress difference in the first mesh

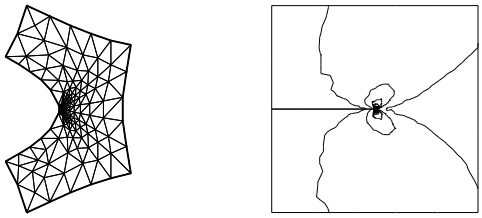


Figure 9.12: COD and Principal stress difference in the last adaptive mesh

It is difficult to create mode II deformation by the opposed shear force on `B` and `T` that is observed in a laboratory. So we use the body shear force along Σ, that is, the x-component f_1 of the body force $\boldsymbol{f}$ is given by

$$f_1(x,y) = H(y-0.001) * H(0.1-y) - H(-y-0.001) * H(y+0.1)$$

where $H(t) = 1$ if $t > 0$; $= 0$ if $t < 0$.

Example 9.13 (Crack Sliding, $K_2(\gamma) = 0$) `(use the same mesh Th)`

```
cb=0.01; ca=0.01;
mesh Zoom = buildmesh (L1(n/2)+L2(n/2)+B(n)+C1(n)
                          +C21(3)+C22(3)+C3(n)+R(n)+T(n));
(use same FE-space Vh and elastic modulus)
fespace Vh1(Th,P1);
Vh1 fx = ((y>0.001)*(y<0.1))-((y<-0.001)*(y>-0.1)) ;

solve  Problem([u,v],[w,s])  =
    int2d(Th)(
              2*mu*(dx(u)*dx(w)+ ((dx(v)+dy(u))*(dx(s)+dy(w)))/4 )
              + lambda*(dx(u)+dy(v))*(dx(w)+dy(s))/2
              )
    -int2d(Th)(fx*w)
    +on(R,u=0)+on(R,v=0);                                  //    fixed
;

for (int i=1; i<=3; i++)
```

```
{
  mesh Plate = movemesh(Zoom,[x+u,y+v]);              //    deformation near γ
  Sx = lambda*(dx(u)+dy(v)) + 2*mu*dx(u);
  Sy = lambda*(dx(u)+dy(v)) + 2*mu*dy(v);
  Sxy = mu*(dy(u) + dx(v));
  N = 0.1*1*sqrt((Sx-Sy)^2+4*Sxy^2);          //   principal stress difference
  if (i==1) {
     plot(Plate,ps="1stCOD2.eps",bw=1);                    //    Fig. 9.14
     plot(N,ps="1stPhoto2.eps",bw=1);                      //    Fig. 9.13
  } else if (i==3) {
     plot(Plate,ps="LastCOD2.eps",bw=1);                   //    Fig. 9.14
     plot(N,ps="LastPhoto2.eps",bw=1);                     //    Fig. 9.14
     break;
  }
  Th=adaptmesh(Th,[u,v]);
  Problem;
}
```

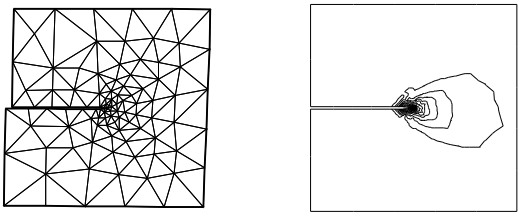


Figure 9.13: (COD) and Principal stress difference in the first mesh

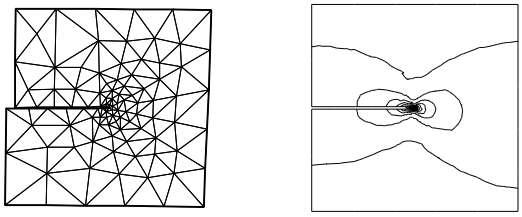


Figure 9.14: COD and Principal stress difference in the last adaptive mesh

9.3 Nonlinear Static Problems

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

$$J(u) = \int_\Omega \frac{1}{2} f(|\nabla u|^2) - u * b$$

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

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

9.3.1 Newton-Raphson algorithm

Now, we solve the Euler problem $\nabla J(u) = 0$ with Newton-Raphson algorithm, that is,

$$u^{n+1} = u^n - (\nabla^2 J(u^n))^{-1} * \nabla J(u^n)$$

First we introduice the two variational form `vdJ` and `vhJ` to compute respectively ∇J and $\nabla^2 J$

```
//    method of Newton-Raphson to solve dJ(u)=0;
                                                                          //
```

$$u^{n+1} = u^n - (\frac{\partial dJ}{\partial u_i})^{-1} * dJ(u^n)$$

```
//    ------------------------------------------
  Ph dalpha ;                                  //   to store 2f''(|∇u|^2) optimisation


  //    the variational form of evaluate dJ = ∇J
  //    ----------------------------------------
  //    dJ = f'()*( dx(u)*dx(vh) + dy(u)*dy(vh)
  varf vdJ(uh,vh) =  int2d(Th)( alpha*( dx(u)*dx(vh) + dy(u)*dy(vh) ) - b*vh)
  + on(1,2,3,4, uh=0);


  //    the variational form of evaluate ddJ = ∇^2 J
  //    hJ(uh,vh) = f'()*( dx(uh)*dx(vh) + dy(uh)*dy(vh)
              //    + 2*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 = 2*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-Raphson");
```

Remark: This example is in `Newton.edp` file of `examples++-tutorial` directory.

9.4 Eigenvalue Problems

This section depend on your FreeFem++ compilation process (see README_arpack), of this tools. This tools is available in FreeFem++ if the word "eigenvalue" appear in line "Load:", like:

```
-- FreeFem++ v1.28 (date Thu Dec 26 10:56:34 CET 2002)
 file : LapEigenValue.edp
 Load: lg_fem lg_mesh eigenvalue
```

This tools is based on the arpack++ [1] the object-oriented version of ARPACK eigenvalue package [1].
The function EigenValue compute the generalized eigenvalue of $Au = \lambda Bu$ where sigma $=\sigma$ is the shift of the method. The matrix OP is defined with $A - \sigma B$. The return value is the number of converged eigenvalue (can be greater than the number of eigen value nev=)

```
int k=EigenValue(OP,B,nev= , sigma= );
```

where the matrix $OP = A - \sigma B$ with a solver and boundary condition, and the matrix B.

Note 9.1 *Boundary condition and Eigenvalue Problems*
The lock (Dirichlet) boundary condition is make with exact penalization so we put 1e30=tgv on the diagonal term of the lock degree of freedom (see equation (6.17)). So take Dirichlet boundary condition just on A and not on B. because we solve $w = OP^{-}1 * B * v$.
If you put lock (Dirichlet) boundary condition on B matrix (with key work **on***) you get small spurious modes* (10^{-30})*, du to boundary condition, but if you forget the lock boundary condition on B matrix (no key work "on") you get huge spurious* (10^{30}) *modes associated to boundary conditon. We compute only small mode, so we get the good one in this case.*

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 FE function array to store the eigenvectors

rawvector= an array of type real[int,int] to store eigenvectors by column. (up to version 2-17).

For real nonsymmetric problems, complex eigenvectors are given as two consecutive vectors, so if eigenvalue k and $k+1$ are complex conjugate eigenvalues, the kth vector will contain the real part and the $k+1$th vector the imaginary part of the corresponding complex conjugate eigenvectors.

tol= the relative accuracy to which eigenvalues are to be determined;

sigma= the shift value;

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

maxit= the maximum number of iterations allowed;

ncv= the number of Arnoldi vectors generated at each iteration of ARPACK.

Example 9.14 (lapEignenValue.edp) *In the first example, we compute the eigenvalue and the eigenvector of the Dirichlet problem on square* $\Omega =]0,\pi[^2$.
The problem is find: λ, *and* ∇u_λ *in* $\mathbb{R}\times H_0^1(\Omega)$

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

The exact eigenvalues are $\lambda_{n,m} = (n^2 + m^2), (n,m) \in \mathbb{N}_*{}^2$ *with the associated eigenvectors are* $u_{m,n} = sin(nx) * sin(my)$.
We use the generalized inverse shift mode of the `arpack++` *library, to find 20 eigenvalue and eigenvector close to the shift value* $\sigma = 20$.

```
//    Computation of the eigen value and eigen vector of the
//    Dirichlet problem on square ]0,π[^2
//    ------------------------------------
//    we use the inverse shift mode
//    the shift is given with the real sigma
//    -------------------------------------
//    find λ and u_λ ∈ H_0^1(Ω) such that:
//                                  ∫_Ω ∇u_λ∇v = λ ∫_Ω u_λ v, ∀v ∈ H_0^1(Ω)
verbosity=10;
mesh Th=square(20,20,[pi*x,pi*y]);
fespace Vh(Th,P2);
Vh u1,u2;


real sigma = 20;                                          //    value of the shift

                                    //    OP = A - sigma B ; // the shifted matrix
varf  op(u1,u2)= int2d(Th)(  dx(u1)*dx(u2) + dy(u1)*dy(u2) - sigma* u1*u2 )
                     +  on(1,2,3,4,u1=0) ;                //    Boundary condition

varf b([u1],[u2]) = int2d(Th)(  u1*u2 );    //   no Boundary condition see note
9.1
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 penalization:
      //    we put 1e30=tgv on the diagonal term of the lock degree of freedom.
          //    So take Dirichlet boundary condition just on a variational form
                                          //    and not on b variational form.
                                           //    because we solve w = OP^-1 * B * v

int nev=20;                 //    number of computed eigen value 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 tolerance
//    maxit= the maximum iteration see arpack doc.
//    ncv see arpack doc.  http://www.caam.rice.edu/software/ARPACK/
//    the return value is number of converged eigen value.

for (int i=0;i<k;i++)
{
  u1=eV[i];
  real gg = int2d(Th)(dx(u1)*dx(u1) + dy(u1)*dy(u1));
  real mm= int2d(Th)(u1*u1) ;
  cout << " ---- " <<  i<< " " << ev[i]<< " err= "
       <<int2d(Th)(dx(u1)*dx(u1) + dy(u1)*dy(u1) - (ev[i])*u1*u1) << " --- "<<endl;
  plot(eV[i],cmm="Eigen  Vector "+i+" valeur =" + ev[i]  ,wait=1,value=1);
}
```

The output of this example is:

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


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

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

Eigenvalues:
  lambda[1]: 5.0002
  lambda[2]: 8.00074
  lambda[3]: 10.0011
  lambda[4]: 10.0011
  lambda[5]: 13.002
  lambda[6]: 13.0039
  lambda[7]: 17.0046
  lambda[8]: 17.0048
  lambda[9]: 18.0083
  lambda[10]: 20.0096
  lambda[11]: 20.0096
  lambda[12]: 25.014
  lambda[13]: 25.0283
  lambda[14]: 26.0159
```

```
 lambda[15]: 26.0159
 lambda[16]: 29.0258
 lambda[17]: 29.0273
 lambda[18]: 32.0449
 lambda[19]: 34.049
 lambda[20]: 34.0492

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

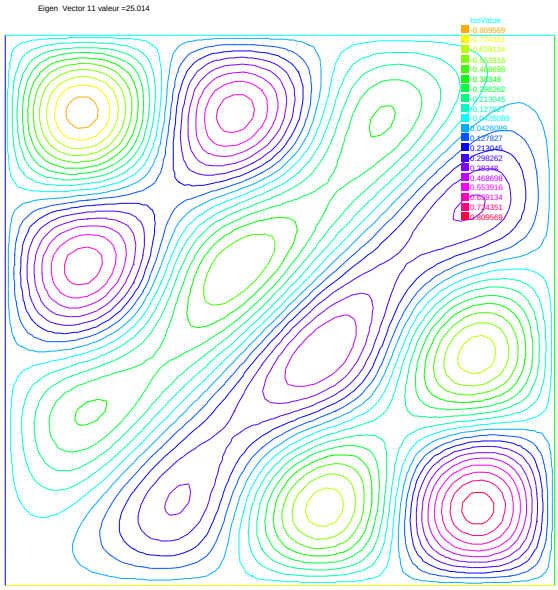


Figure 9.15: Isovalue of 11th eigenvector $u_{4,3} - u_{3,4}$

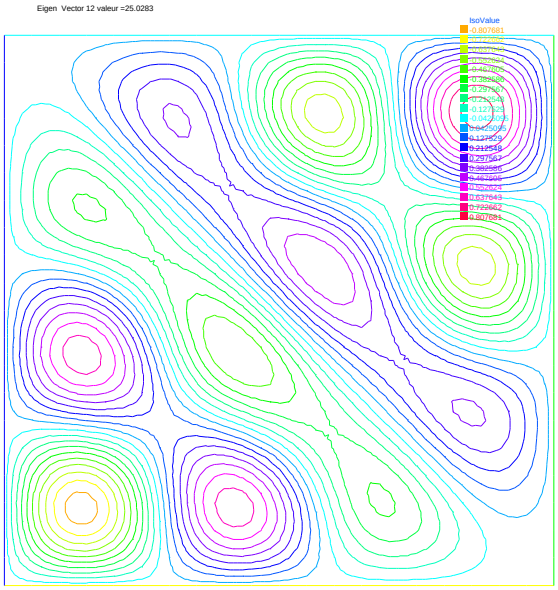


Figure 9.16: Isovalue of 12th eigenvector $u_{4,3} + u_{3,4}$

9.5 Evolution Problems

`FreeFem++` also solve evolution problems such as the heat problem

$$\begin{aligned} &\frac{\partial u}{\partial t} - \mu\Delta u = f \quad \text{in } \Omega\times]0,T[, \\ &u(\boldsymbol{x},0) = u_0(\boldsymbol{x}) \quad \text{in } \Omega; \qquad (\partial u/\partial n)(\boldsymbol{x},t) = 0 \quad \text{on } \partial\Omega\times]0,T[. \end{aligned} \tag{9.26}$$

with a positive viscosity coefficient μ and homogeneous Neumann boundary conditions. We solve (9.26) by FEM in space and finite differences in time. We use the definition of the partial derivative of the solution in the time derivative,

$$\frac{\partial u}{\partial t}(x,y,t) = \lim_{\tau\to 0}\frac{u(x,y,t) - u(x,y,t-\tau)}{\tau}$$

which indicate that $u^m(x,y) = u(x,y,m\tau)$ imply

$$\frac{\partial u}{\partial t}(x,y,m\tau) \simeq \frac{u^m(x,y) - u^{m-1}(x,y)}{\tau}$$

The time discretization of heat equation (9.27) is as follows:

$$\begin{aligned} &\frac{u^{m+1}-u^m}{\tau} - \mu\Delta u^{m+1} = f^{m+1} \quad \text{in } \Omega \\ &u^0(\boldsymbol{x}) = u_0(\boldsymbol{x}) \quad \text{in } \Omega; \qquad \partial u^{m+1}/\partial n(\boldsymbol{x}) = 0 \quad \text{on } \partial\Omega, \quad \text{for all } m = 0,\cdots,[T/\tau], \end{aligned} \tag{9.27}$$

which is so-called *backward Euler method* for (9.27). Multiplying the test function v both sides of the formula just above, we have

$$\int_\Omega \{u^{m+1}v - \tau\Delta u^{m+1}v\} = \int_\Omega \{u^m + \tau f^{m+1}\}v\,.$$

By the divergence theorem, we have

$$\int_\Omega \{u^{m+1}v + \tau\nabla u^{m+1}\cdot\nabla v\} - \int_{\partial\Omega}\tau\left(\partial u^{m+1}/\partial n\right)v = \int_\Omega \{u^m v + \tau f^{m+1}v\}.$$

By the boundary condition $\partial u^{m+1}/\partial n = 0$, it follows that

$$\int_\Omega \{u^{m+1}v + \tau\nabla u^{m+1}\cdot\nabla v\} - \int_\Omega \{u^m v + \tau f^{m+1}v\} = 0. \tag{9.28}$$

Using the identity just above, we can calculate the finite element approximation u_h^m of u^m in a step-by-step manner with respect to t.

Example 9.15 *We now solve the following example with the exact solution* $u(x,y,t) = tx^4$.

$$\begin{aligned} &\frac{\partial u}{\partial t} - \mu\Delta u = x^4 - \mu 12tx^2 \text{ in } \Omega\times]0,3[,\ \Omega =]0,1[^2 \\ &u(x,y,0) = 0 \quad \text{on } \Omega, \qquad u|_{\partial\Omega} = t * x^4 \end{aligned}$$

```
//     heat equation ∂_t u = −μΔu = x^4 − μ12tx^2
mesh Th=square(16,16);
fespace Vh(Th,P1);

Vh u,v,uu,f,g;
real dt = 0.1, mu = 0.01;
problem dHeat(u,v) =
    int2d(Th)( u*v + dt*mu*(dx(u)*dx(v) + dy(u)*dy(v)))
    + int2d(Th) (- uu*v - dt*f*v )
    + on(1,2,3,4,u=g);

real t = 0;                                          //    start from t=0
uu = 0;                                              //    u(x,y,0)=0
for (int m=0;m<=3/dt;m++)
{
   t=t+dt;
   f = x^4-mu*t*12*x^2;
   g = t*x^4;
   dHeat;
   plot(u,wait=true);
   uu = u;
   cout <<"t="<<t<<"L^2-Error="<<sqrt( int2d(Th)((u-t*x^4)^2) ) << endl;
}
```

In the last statement, the L^2-error $\left(\int_\Omega |u - tx^4|^2\right)^{1/2}$ is calculated at $t = m\tau, \tau = 0.1$. At $t = 0.1$, the error is 0.000213269. The errors increase with m and 0.00628589 at $t = 3$.
*The iteration of the backward Euler (9.28) is made by **for loop** (see Section 4.9).*

Note 9.2 *The stiffness matrix in loop is used over and over again.* `FreeFem++` *support reuses of stiffness matrix.*

9.5.1 Mathematical Theory on Time Difference Approximations.

In this section, we show the advantage of implicit schemes. Let V, H be separable Hilbert space and V is dense in H. Let a be a continuous bilinear form over $V \times V$ with coercivity and symmetry. Then $\sqrt{a(v,v)}$ become equivalent to the norm $\|v\|$ of V.
Problem Ev(f, Ω): For a given $f \in L^2(0,T;V')$, $u^0 \in H$

$$\begin{aligned} \frac{d}{dt}(u(t),v) + a(u(t),v) &= (f(t),v) \qquad \forall v \in V, , \quad a.e.\, t \in [0,T] \\ u(0) &= u^0 \end{aligned} \tag{9.29}$$

where V' is the dual space of V. Then, there is an unique solution $u \in L^\infty(0,T;H) \cap L^2(0,T;V)$. Let us denote the time step by $\tau > 0$, $N_T = [T/\tau]$. For the discretization, we put $u^n = u(n\tau)$ and consider the time difference for each $\theta \in [0,1]$

$$\begin{aligned} \frac{1}{\tau}\left(u_h^{n+1} - u_h^n, \phi_i\right) + a\left(u_h^{n+\theta}, \phi_i\right) &= \langle f^{n+\theta}, \phi_i \rangle \\ i = 1, \cdots, m, \quad n = 0, \cdots, N_T & \\ u_h^{n+\theta} = \theta u_h^{n+1} + (1-\theta)u_h^n, \quad f^{n+\theta} = \theta f^{n+1} + (1-\theta) f^n & \end{aligned} \tag{9.30}$$

Formula (9.30) is the *forward Euler scheme* if $\theta = 0$, *Crank-Nicolson scheme* if $\theta = 1/2$, the *backward Euler scheme* if $\theta = 1$.
Unknown vectors $u^n = (u_h^1, \cdots, u_h^M)^T$ in

$$u_h^n(x) = u_1^n \phi_1(x) + \cdots + u_m^n \phi_m(x), \quad u_1^n, \cdots, u_m^n \in \mathbb{R}$$

are obtained from solving the matrix

$$\begin{aligned} (M + \theta\tau A)u^{n+1} &= \{M - (1-\theta)\tau A\}u^n + \tau\left\{\theta f^{n+1} + (1-\theta)f^n\right\} \\ M = (m_{ij}), \quad m_{ij} &= (\phi_j, \phi_i), \qquad A = (a_{ij}), \quad a_{ij} = a(\phi_j, \phi_i) \end{aligned} \tag{9.31}$$

Refer [21, pp.70–75] for solvability of (9.31). The stability of (9.31) is in [21, Theorem 2.13]:

> Let $\{\mathcal{T}_h\}_{h\downarrow 0}$ be regular triangulations (see Section 5.4). Then there is a number $c_0 > 0$ independent of h such that,
>
> $$|u_h^n|^2 \le \begin{cases} \frac{1}{\delta}\left\{|u_h^0|^2 + \tau\sum_{k=0}^{n-1}\|f^{k+\theta}\|_{V_h'}^2\right\} & \theta \in [0, 1/2) \\ |u_h^0|^2 + \tau\sum_{k=0}^{n-1}\|f^{k+\theta}\|_{V_h'}^2 & \theta \in [1/2, 1] \end{cases} \tag{9.32}$$
>
> if the following are satisfied:
>
> 1. When $\theta \in [0, 1/2)$, then we can take a time step τ in such a way that
>
> $$\tau < \frac{2(1-\delta)}{(1-2\theta)c_0^2}h^2 \tag{9.33}$$
>
> for arbitrary $\delta \in (0, 1)$.
> 2. When $1/2 \le \theta \le 1$, we can take τ arbitrary.

Example 9.16

```
mesh Th=square(12,12);
fespace Vh(Th,P1);
fespace Ph(Th,P0);

Ph h = hTriangle;                              //    mesh sizes for each triangle
real tau = 0.1, theta=0.;
func real f(real t) {
   return x^2*(x-1)^2 + t*(-2 + 12*x - 11*x^2 - 2*x^3 + x^4);
}
ofstream out("err02.csv");                     //    file to store calculations
out << "mesh size = "<<h[].max<<", time step = "<<tau<<endl;
for (int n=0;n<5/tau;n++) \\
   out<<n*tau<<",";
out << endl;
Vh u,v,oldU;
Vh f1, f0;
problem aTau(u,v) =
  int2d(Th)( u*v + theta*tau*(dx(u)*dx(v) + dy(u)*dy(v) + u*v))
  - int2d(Th)(oldU*v - (1-theta)*tau*(dx(oldU)*dx(v)+dy(oldU)*dy(v)+oldU*v))
  - int2d(Th)(tau*( theta*f1+(1-theta)*f0 )*v );
```

```
while (theta <= 1.0) {
  real t = 0, T=3;                                          //    from t=0 to T
  oldU = 0;                                                 //    u(x,y,0)=0
  out <<theta<<",";
  for (int n=0;n<T/tau;n++) {
      t = t+tau;
      f0 = f(n*tau); f1 = f((n+1)*tau);
      aTau;
      oldU = u;
      plot(u);
      Vh uex = t*x^2*(1-x)^2;                          //    exact sol.= tx^2(1-x)^2
      Vh err = u - uex;                                //    err =FE-sol - exact
      out<< abs(err[].max)/abs(uex[].max) <<",";       //    ||err||_{L^∞(Ω)}/||u_ex||_{L^∞(Ω)}
  }
  out << endl;
  theta = theta + 0.1;
}
```

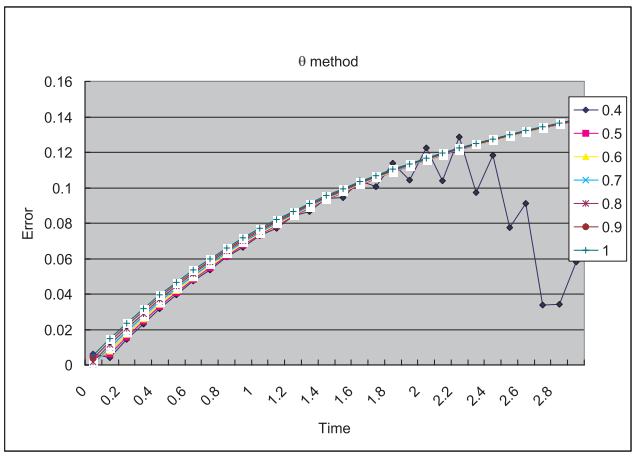


Figure 9.17: $\max_{x\in\Omega} |u_h^n(\theta) - u_{ex}(n\tau)| / \max_{x\in\Omega} |u_{ex}(n\tau)|$ at $n = 0, 1, \cdots, 29$

We can see in Fig. 9.17 that $u_h^n(\theta)$ become unstable at $\theta = 0.4$, and figures are omitted in the case $\theta < 0.4$.

9.5.2 Convection

The hyperbolic equation

$$\partial_t u - \boldsymbol{\alpha} \cdot \nabla u = f; \quad \text{for a vector-valued function } \boldsymbol{\alpha}, \ \partial_t = \frac{\partial}{\partial t}, \tag{9.34}$$

appear frequently in scientific problems, for example, Navier-Stokes equation, Convection-Diffusion equation, etc.
In the case of 1-dimensional space, we can easily find the general solution $(x,t) \mapsto u(x,t) = u^0(x - \alpha t)$ of the following equation, if α is constant,

$$\partial_t u + \alpha \partial_x u = 0, \qquad u(x,0) = u^0(x), \tag{9.35}$$

because $\partial_t u + \alpha \partial_x u = -\alpha \dot{u}^0 + a\dot{u}^0 = 0$, where $\dot{u}^0 = du^0(x)/dx$. Even if α is not constant construction, the principle is similar. One begins the ordinary differential equation (with convention which α is prolonged by zero apart from $(0, L) \times (0, T)$):

$$\dot{X}(\tau) = -\alpha(X(\tau), \tau), \quad \tau \in (0, t) \quad X(t) = x$$

In this equation τ is the variable and x, t is parameters, and we denote the solution by $X_{x,t}(\tau)$. Then it is noticed that $(x, t) \to v(X(\tau), \tau)$ in $\tau = t$ satisfy the equation

$$\partial_t v + \alpha \partial_x v = \partial_t X \dot{v} + a \partial_x X \dot{v} = 0$$

and by the definition $\partial_t X = \dot{X} = -\alpha$ and $\partial_x X = \partial_x x$ in $\tau = t$, because if $\tau = t$ we have $X(\tau) = x$. The general solution of (9.35) is thus the value of the boundary condition in $X_{x,t}(0)$, it is has to say $u(x, t) = u^0(X_{x,t}(0))$ if $X_{x,t}(0)$ is on the x axis, $u(x, t) = u^0(X_{x,t}(0))$ if $X_{x,t}(0)$ is on the axis of t.
In higher dimension $\Omega \subset R^d$, $d = 2, 3$, the equation of the convection is written

$$\partial_t u + \boldsymbol{\alpha} \cdot \nabla u = 0 \text{ in } \Omega \times (0, T)$$

where $\boldsymbol{a}(x, t) \in R^d$. `FreeFem++` implements the Characteristic-Galerkin method for convection operators. Recall that the equation (9.34) can be discretized as

$$\frac{Du}{Dt} = f \text{ i.e. } \frac{du}{dt}(X(t), t) = f(X(t), t) \text{ where } \frac{dX}{dt}(t) = \boldsymbol{\alpha}(X(t), t)$$

where D is the total derivative operator. So a good scheme is one step of backward convection by the method of Characteristics-Galerkin

$$\frac{1}{\tau}\left(u^{m+1}(x) - u^m(X^m(x))\right) = f^m(x) \tag{9.36}$$

where $X^m(x)$ is an approximation of the solution at $t = m\tau$ of the ordinary differential equation

$$\frac{d\boldsymbol{X}}{dt}(t) = \boldsymbol{\alpha}^m(\boldsymbol{X}(t)),\ \boldsymbol{X}((m+1)\tau) = x.$$

where $\boldsymbol{\alpha}^m(x) = (\alpha_1(x, m\tau), \alpha_2(x, m\tau))$. Because, by Taylor's expansion, we have

$$\begin{aligned} u^m(\boldsymbol{X}(m\tau)) &= u^m(\boldsymbol{X}((m+1)\tau)) - \tau \sum_{i=1}^{d} \frac{\partial u^m}{\partial x_i}(\boldsymbol{X}((m+1)\tau)) \frac{\partial X_i}{\partial t}((m+1)\tau) + o(\tau) \\ &= u^m(x) - \tau \boldsymbol{\alpha}^m(x) \cdot \nabla u^m(x) + o(\tau) \end{aligned} \tag{9.37}$$

where $X_i(t)$ are the i-th component of $\boldsymbol{X}(t)$, $u^m(x) = u(x, m\tau)$ and we used the chain rule and $x = \boldsymbol{X}((m+1)\tau)$. From (9.37), it follows that

$$u^m(X^m(x)) = u^m(x) - \tau \boldsymbol{\alpha}^m(x) \cdot \nabla u^m(x) + o(\tau). \tag{9.38}$$

Also we apply Taylor's expansion for $t \mapsto u^m(x - \boldsymbol{\alpha}^m(x)t)$, $0 \le t \le \tau$, then

$$u^m(x - \boldsymbol{\alpha}\tau) = u^m(x) - \tau \boldsymbol{\alpha}^m(x) \cdot \nabla u^m(x) + o(\tau).$$

Putting

$$\texttt{convect}\,(\boldsymbol{\alpha}, \tau, u^m) = u^m\left(x - \boldsymbol{\alpha}^m \tau\right),$$

we can get the approximation

$$u^m(X^m(x)) \approx \texttt{convect}([a_1^m, a_2^m], \tau, u^m) \quad \text{by } x = X((m+1)\tau).$$

A classical convection problem is that of the "rotating bell" (quoted from [13][p.16]). Let Ω be the unit disk centered at 0, with its center rotating with speed $\alpha_1 = y$, $\alpha_2 = -x$ We consider the problem (9.34) with $f = 0$ and the initial condition $u(x,0) = u^0(x)$, that is, from (9.36)

$$u^{m+1}(x) = u^m(X^m(x)) \approx \texttt{convect}(\boldsymbol{\alpha}, \tau, u^m).$$

The exact solution is $u(x,t) = u(\boldsymbol{X}(t))$ where $\boldsymbol{X}$ equals x rotated around the origin by an angle $\theta = -t$ (rotate in clockwise). So, if u^0 in a 3D perspective looks like a bell, then u will have exactly the same shape, but rotated by the same amount. The program consists in solving the equation until $T = 2\pi$, that is for a full revolution and to compare the final solution with the initial one; they should be equal.

Example 9.17 (convect.edp)

```
border C(t=0, 2*pi) { x=cos(t);  y=sin(t); };        // the unit circle
mesh Th = buildmesh(C(70));                          //    triangulates the disk
fespace Vh(Th,P1);
Vh u0 = exp(-10*((x-0.3)^2 +(y-0.3)^2));                    //    give u^0

real dt = 0.17,t=0;                                        //    time step
Vh a1 = -y, a2 = x;                                  //    rotation velocity
Vh u;                                                      //    u^{m+1}
for (int m=0; m<2*pi/dt ; m++) {
    t += dt;
    u=convect([a1,a2],dt,u0);                        //    u^{m+1} = u^m(X^m(x))
    u0=u;                                                  //    m++
    plot(u,cmm=" t="+t + ", min=" + u[].min + ", max=" +  u[].max,wait=0);
};
```

Note 9.3 *The scheme* `convect` *is unconditionally stable, then the bell become lower and lower (the maximum of* u^{37} *is* 0.406 *as shown in Fig. 9.19).*

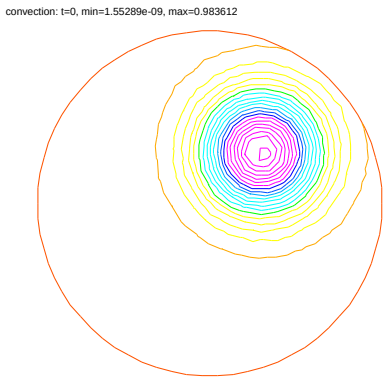


Figure 9.18: $u^0 = e^{-10((x-0.3)^2+(y-0.3)^2)}$

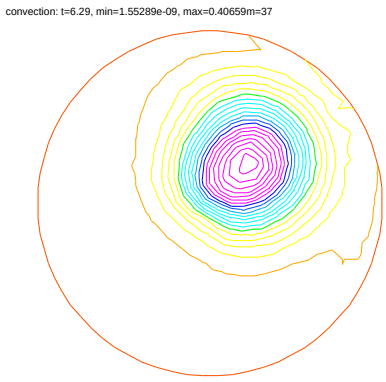


Figure 9.19: The bell at $t = 6.29$

9.5.3 2D Black-Scholes equation for an European Put option

In mathematical finance, an option on two assets is modeled by a Black-Scholes equations in two space variables, (see for example Wilmott et al[36] or Achdou et al [3]).

$$\begin{aligned}\partial_t u + \frac{(\sigma_1 x)^2}{2}\frac{\partial^2 u}{\partial x^2} + \frac{(\sigma_2 y)^2}{2}\frac{\partial^2 u}{\partial y^2} \\ + \rho xy\frac{\partial^2 u}{\partial x \partial y} + rS_1\frac{\partial u}{\partial x} + rS_2\frac{\partial u}{\partial y} - rP = 0\end{aligned} \tag{9.39}$$

which is to be integrated in $(0,T)\times\mathbb{R}^+\times\mathbb{R}^+$ subject to, in the case of a put

$$u(x,y,T) = (K - \max(x,y))^+ . \tag{9.40}$$

Boundary conditions for this problem may not be so easy to device. As in the one dimensional case the PDE contains boundary conditions on the axis $x_1 = 0$ and on the axis $x_2 = 0$, namely two one dimensional Black-Scholes equations driven respectively by the data $u(0,+\infty,T)$ and $u(+\infty,0,T)$. These will be automatically accounted for because they are embedded in the PDE. So if we do nothing in the variational form (i.e. if we take a Neumann boundary condition at these two axis in the strong form) there will be no disturbance to these. At infinity in one of the variable, as in 1D, it makes sense to impose $u = 0$. We take

$$\sigma_1 = 0.3,\quad \sigma_2 = 0.3,\quad \rho = 0.3,\quad r = 0.05,\quad K = 40,\quad T = 0.5 \tag{9.41}$$

An implicit Euler scheme is used and a mesh adaptation is done every 10 time steps. To have an unconditionally stable scheme, the first order terms are treated by the Characteristic Galerkin method, which, roughly, approximates

$$\frac{\partial u}{\partial t} + a_1\frac{\partial u}{\partial x} + a_2\frac{\partial u}{\partial y} \approx \frac{1}{\tau}\left(u^{n+1}(x) - u^n(x - \alpha\tau)\right) \tag{9.42}$$

Example 9.18 *[BlackSchol.edp]*

```
                                                //    file BlackScholes2D.edp
int m=30,L=80,LL=80, j=100;
real sigx=0.3, sigy=0.3, rho=0.3, r=0.05, K=40, dt=0.01;
mesh th=square(m,m,[L*x,LL*y]);
fespace Vh(th,P1);

Vh u=max(K-max(x,y),0.);
Vh xveloc, yveloc, v,uold;

for (int n=0; n*dt <= 1.0; n++)
{
  if(j>20)  { th = adaptmesh(th,u,verbosity=1,abserror=1,nbjacoby=2,
              err=0.001, nbvx=5000, omega=1.8, ratio=1.8, nbsmooth=3,
              splitpbedge=1, maxsubdiv=5,rescaling=1) ;
     j=0;
     xveloc = -x*r+x*sigx^2+x*rho*sigx*sigy/2;
     yveloc = -y*r+y*sigy^2+y*rho*sigx*sigy/2;
     u=u;
```

```
    };
  uold=u;
  solve eq1(u,v,init=j,solver=LU) = int2d(th)( u*v*(r+1/dt)
        + dx(u)*dx(v)*(x*sigx)^2/2 + dy(u)*dy(v)*(y*sigy)^2/2
        + (dy(u)*dx(v) + dx(u)*dy(v))*rho*sigx*sigy*x*y/2)
        - int2d(th)( v*convect([xveloc,yveloc],dt,w)/dt) + on(2,3,u=0);

  j=j+1;
};
plot(u,wait=1,value=1);
```

Results are shown on Fig. 9.18).

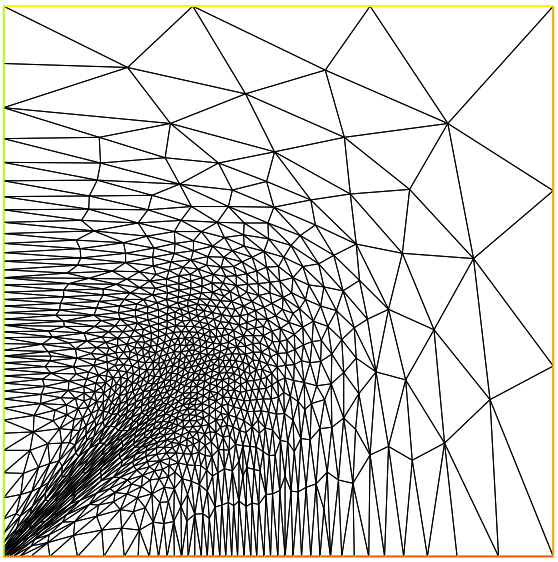


Figure 9.20: The adapted triangulation

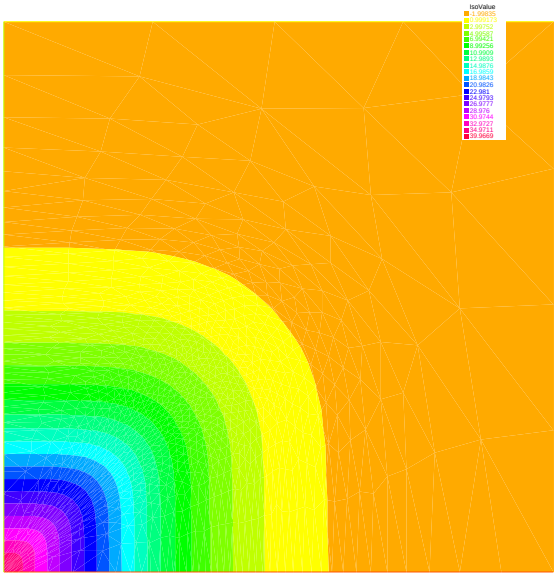


Figure 9.21: The level line of the European basquet put option

9.6 Navier-Stokes Equation

9.6.1 Stokes and Navier-Stokes

The Stokes equations are: for a given $\boldsymbol{f} \in L^2(\Omega)^2$,

$$\left.\begin{array}{rl} -\Delta \boldsymbol{u} + \nabla p & = \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} & = 0 \end{array}\right\} \quad \text{in } \Omega \tag{9.43}$$

where $\boldsymbol{u} = (u_1, u_2)$ is the velocity vector and p the pressure. For simplicity, let us choose Dirichlet boundary conditions on the velocity, $\boldsymbol{u} = \boldsymbol{u}_\Gamma$ on Γ.

In Temam [Theorem 2.2], there ia a weak form of (9.43): Find $\boldsymbol{v} = (v_1, v_2) \in \boldsymbol{V}(\Omega)$

$$\boldsymbol{V}(\Omega) = \{\boldsymbol{w} \in H_0^1(\Omega)^2 |\ \mathrm{div}\boldsymbol{w} = 0\}$$

which satisfy

$$\sum_{i=1}^{2} \int_\Omega \nabla u_i \cdot \nabla v_i = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{w} \quad \text{for all } v \in V$$

Here it is used the existence $p \in H^1(\Omega)$ such that $\boldsymbol{u} = \nabla p$, if

$$\int_\Omega \boldsymbol{u} \cdot \boldsymbol{v} = 0 \quad \text{for all } \boldsymbol{v} \in V$$

Another weak form is derived as follows: We put

$$\boldsymbol{V} = H_0^1(\Omega)^2; \quad W = \left\{ q \in L^2(\Omega) \,\middle|\, \int_\Omega q = 0 \right\}$$

By multiplying the first equation in (9.43) with $v \in V$ and the second with $q \in W$, subsequent integration over Ω, and an application of Green's formula, we have

$$\begin{aligned} \int_\Omega \nabla \boldsymbol{u} \cdot \nabla \boldsymbol{v} - \int_\Omega \mathrm{div}\boldsymbol{v}\, p &= \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} \\ \int_\Omega \mathrm{div}\boldsymbol{u}\, q &= 0 \end{aligned}$$

This yields the weak form of (9.43): Find $(\boldsymbol{u}, p) \in \boldsymbol{V} \times W$ such that

$$a(\boldsymbol{u}, \boldsymbol{v}) + b(\boldsymbol{v}, p) = (\boldsymbol{f}, \boldsymbol{v}) \tag{9.44}$$

$$b(\boldsymbol{u}, q) = 0 \tag{9.45}$$

for all $(\boldsymbol{v}, q) \in V \times W$, where

$$a(\boldsymbol{u}, \boldsymbol{v}) = \int_\Omega \nabla \boldsymbol{u} \cdot \nabla \boldsymbol{v} = \sum_{i=1}^{2} \int_\Omega \nabla u_i \cdot \nabla v_i \tag{9.46}$$

$$b(\boldsymbol{u}, q) = -\int_\Omega \mathrm{div}\boldsymbol{u}\, q \tag{9.47}$$

Now, we consider finite element spaces $\boldsymbol{V}_h \subset \boldsymbol{V}$ and $W_h \subset W$, and we assume the following basis functions

$$\begin{aligned} &\boldsymbol{V}_h = V_h \times V_h, \quad V_h = \{v_h |\ v_h = v_1\phi_1 + \cdots + v_{M_V}\phi_{M_V}\}, \\ &W_h = \{q_h |\ q_h = q_1\varphi_1 + \cdots + q_{M_W}\varphi_{M_W}\} \end{aligned}$$

The discrete weak form is: Find $(\boldsymbol{u}_h, p_h) \in \boldsymbol{V}_h \times W_h$ such that

$$\begin{aligned} a(\boldsymbol{u}_h, \boldsymbol{v}_h) + b(\boldsymbol{v}_h, p) &= (\boldsymbol{f}, \boldsymbol{v}_h), \quad \forall \boldsymbol{v}_h \in \boldsymbol{V}_h \\ b(\boldsymbol{u}_h, q_h) &= 0, \quad \forall q_h \in W_h \end{aligned} \tag{9.48}$$

Note 9.4 *Assume that:*

1. *There is a constant* $\alpha_h > 0$ *such that*

$$a(\boldsymbol{v}_h, \boldsymbol{v}_h) \geq \alpha \|\boldsymbol{v}_h\|^2_{1,\Omega} \quad \textit{for all } \boldsymbol{v}_h \in Z_h$$

where

$$Z_h = \{\boldsymbol{v}_h \in \boldsymbol{V}_h |\ b(\boldsymbol{w}_h, q_h) = 0 \quad \textit{for all } q_h \in W_h\}$$

2. *There is a constant* $\beta_h > 0$ *such that*

$$\sup_{\boldsymbol{v}_h \in \boldsymbol{V}_h} \frac{b(\boldsymbol{v}_h, q_h)}{\|\boldsymbol{v}_h\|_{1,\Omega}} \geq \beta_h \|q_h\|_{0,\Omega} \quad \textit{for all } q_h \in W_h$$

Then we have an unique solution $(\boldsymbol{u}_h, p_h)$ *of (9.48) satisfying*

$$\|\boldsymbol{u} - \boldsymbol{u}_h\|_{1,\Omega} + \|p - p_h\|_{0,\Omega} \leq C \left(\inf_{\boldsymbol{v}_h \in \boldsymbol{V}_h} \|u - v_h\|_{1,\Omega} + \inf_{q_h \in W_h} \|p - q_h\|_{0,\Omega} \right)$$

with a constant $C > 0$ *(see e.g. [19, Theorem 10.4]).*

Let us denote that

$$\begin{aligned} A &= (A_{ij}),\ A_{ij} = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \qquad i, j = 1, \cdots, M_{\boldsymbol{V}} \\ \boldsymbol{B} &= (Bx_{ij}, By_{ij}),\ Bx_{ij} = -\int_\Omega \partial\phi_j/\partial x\, \varphi_i \qquad By_{ij} = -\int_\Omega \partial\phi_j/\partial y\, \varphi_i \\ &\qquad i = 1, \cdots, M_W;\ j = 1, \cdots, M_V \end{aligned} \tag{9.49}$$

then (9.48) is written by

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B}^* \\ \boldsymbol{B} & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{U}_h \\ \{p_h\} \end{pmatrix} = \begin{pmatrix} \boldsymbol{F}_h \\ 0 \end{pmatrix} \tag{9.50}$$

where

$$\boldsymbol{A} = \begin{pmatrix} A & 0 \\ 0 & A \end{pmatrix} \qquad \boldsymbol{B}^* = \begin{Bmatrix} Bx^T \\ By^T \end{Bmatrix} \qquad \boldsymbol{U}_h = \begin{Bmatrix} \{u_{1,h}\} \\ \{u_{2,h}\} \end{Bmatrix} \qquad \boldsymbol{F}_h = \begin{Bmatrix} \{\int_\Omega f_1 \phi_i\} \\ \{\int_\Omega f_2 \phi_i\} \end{Bmatrix}$$

Penalty method: This method consists of replacing (9.48) by a more regular problem: Find $(\boldsymbol{v}_h^\epsilon, p_h^\epsilon) \in \boldsymbol{V}_h \times \tilde{W}_h$ satisfying

$$\begin{aligned} a(\boldsymbol{u}_h^\epsilon, \boldsymbol{v}_h) + b(\boldsymbol{v}_h, p_h^\epsilon) &= (\boldsymbol{f}, \boldsymbol{v}_h), \quad \forall \boldsymbol{v}_h \in \boldsymbol{V}_h \\ b(\boldsymbol{u}_h^\epsilon, q_h) - \epsilon(p_h^\epsilon, q_h) &= 0, \quad \forall q_h \in \tilde{W}_h \end{aligned} \tag{9.51}$$

where $\tilde{W}_h \subset L^2(\Omega)$. Formally, we have

$$\mathrm{div}\boldsymbol{u}_h^\epsilon = \epsilon p_h^\epsilon$$

and the corresponding algebraic problem

$$\begin{pmatrix} \boldsymbol{A} & B^* \\ B & -\epsilon I \end{pmatrix} \begin{pmatrix} \boldsymbol{U}_h^\epsilon \\ \{p_h^\epsilon\} \end{pmatrix} = \begin{pmatrix} \boldsymbol{F}_h \\ 0 \end{pmatrix}$$

Note 9.5 *We can eliminate* $p_h^\epsilon = (1/\epsilon)BU_h^\epsilon$ *to obtain*

$$(A + (1/\epsilon)B^*B)\boldsymbol{U}_h^\epsilon = \boldsymbol{F}_h^\epsilon \tag{9.52}$$

Since the matrix $A + (1/\epsilon)B^*B$ *is symmetric, positive-definite, and sparse, (9.52) can be solved by known technique. There is a constant* $C > 0$ *independent of* ϵ *such that*

$$\|\boldsymbol{u}_h - \boldsymbol{u}_h^\epsilon\|_{1,\Omega} + \|p_h - p_h^\epsilon\|_{0,\Omega} \leq C\epsilon$$

(see e.g. [19, 17.2])

Example 9.19 (Cavity.edp) *The driven cavity flow problem is solved first at zero Reynolds number (Stokes flow) and then at Reynolds 100. The velocity pressure formulation is used first and then the calculation is repeated with the stream function vorticity formulation.*
We solve the driven cavity problem by the penalty method (9.51) where $\boldsymbol{u}_\Gamma \cdot \boldsymbol{n} = 0$ *and* $\boldsymbol{u}_\Gamma \cdot \boldsymbol{s} = 1$ *on the top boundary and zero elsewhere (* $\boldsymbol{n}$ *is the unit normal to* Γ*, and* $\boldsymbol{s}$ *the unit tangent to* Γ*). The mesh is constructed by*

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

We use a classical Taylor-Hood element technic to solve the problem:

The velocity is approximated with the P_2 *FE (* X_h *space), and the the pressure is approximated with the* P_1 *FE (* M_h *space),*

where

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

and

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

The FE spaces and functions are constructed by

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

The Stokes operator is implemented as a system-solve for the velocity $(u1, u2)$ *and the pressure* p*. The test function for the velocity is* $(v1, v2)$ *and* q *for the pressure, so the variational form (9.48) in freefem language is:*

```
solve Stokes (u1,u2,p,v1,v2,q,solver=Crout) =
    int2d(Th)( ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
             +  dx(u2)*dx(v2) + dy(u2)*dy(v2) )
             - p*q*(0.000001)
             - p*dx(v1) - p*dy(v2)
             - dx(u1)*q - dy(u2)*q
            )
  + on(3,u1=1,u2=0)
```

```
  + on(1,2,4,u1=0,u2=0);                //    see Section 5.1.1 for labels 1,2,3,4
```

Each unknown has its own boundary conditions.

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

$$\begin{aligned} \tfrac{1}{\tau}(u^{n+1} - u^n \circ X^n) - \nu\Delta u^{n+1} + \nabla p^{n+1} &= 0, \\ \nabla \cdot u^{n+1} &= 0 \end{aligned} \tag{9.53}$$

The term $u^n \circ X^n(x) \approx u^n(x - u^n(x)\tau)$ will be computed by the operator "convect" , so we obtain

```
int i=0;
real  nu=1./100.;
real dt=0.1;
real alpha=1/dt;

Xh up1,up2;

problem  NS (u1,u2,p,v1,v2,q,solver=Crout,init=i) =
    int2d(Th)(
              alpha*( u1*v1 + u2*v2)
             + nu * ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
             +  dx(u2)*dx(v2) + dy(u2)*dy(v2) )
             - p*q*(0.000001)
             - p*dx(v1) - p*dy(v2)
             - dx(u1)*q - dy(u2)*q
            )
  + int2d(Th) ( -alpha*
       convect([up1,up2],-dt,up1)*v1 -alpha*convect([up1,up2],-dt,up2)*v2 )
  + on(3,u1=1,u2=0)
  + on(1,2,4,u1=0,u2=0)
;
```

```
for (i=0;i<=10;i++)
 {
   up1=u1;
   up2=u2;
   NS;
   if ( !(i % 10))                                    //    plot every 10 iteration
    plot(coef=0.2,cmm=" [u1,u2] and p  ",p,[u1,u2]);
 } ;
```

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

9.6.2 Uzawa Conjugate Gradient

We solve Stokes problem without penalty. The classical iterative method of Uzawa is described by the algorithm (see e.g.[19, 17.3], [28, 13] or [29, 13]):

Initialize: Let p_h^0 be an arbitrary chosen element of $L^2(\Omega)$.

Calculate u_h: Once p_h^n is known, $\boldsymbol{v}_h^n$ is the solution of

$$\boldsymbol{u}_h^n = A^{-1}(\boldsymbol{f}_h - \boldsymbol{B}^* p_h^n)$$

Advance p_h: Let p_h^{n+1} be defined by

$$p_h^{n+1} = p_h^n + \rho_n \boldsymbol{B}\boldsymbol{u}_h^n$$

There is a constant $\alpha > 0$ such that $\alpha \le \rho_n \le 2$ for each n, then $\boldsymbol{u}_h^n$ converges to the solution $\boldsymbol{u}_h$, and then $B\boldsymbol{v}_h^n \to 0$ as $n \to \infty$ from the *Advance* p_h. This method in general converges quite slowly. First we define mesh, and the Taylor-Hood approximation. So X_h is the velocity space, and M_h is the pressure space.

Example 9.20 (StokesUzawa.edp)

```
mesh Th=square(10,10);
fespace Xh(Th,P2),Mh(Th,P1);
Xh u1,u2,v1,v2;
Mh p,q,ppp;                                        //    ppp is a working pressure


varf bx(u1,q) = int2d(Th)( -(dx(u1)*q));
varf by(u1,q) = int2d(Th)( -(dy(u1)*q));
varf a(u1,u2)= int2d(Th)(  dx(u1)*dx(u2) + dy(u1)*dy(u2) )
                    +  on(3,u1=1)  +  on(1,2,4,u1=0) ;
                      //    remark:  put the  on(3,u1=1) before  on(1,2,4,u1=0)
                                    //    because we want zero on intersection %

matrix A= a(Xh,Xh,solver=CG);
matrix Bx= bx(Xh,Mh);                                          //     B = (Bx  By)
matrix By= by(Xh,Mh);

Xh bc1; bc1[] = a(0,Xh);            //    boundary condition contribution on u1
Xh bc2; bc2   = 0 ;              //    no boundary condition contribution on u2
Xh b;
```

$p_h^n \to \boldsymbol{B}A^{-1}(-\boldsymbol{B}^* p_h^n) = -div\boldsymbol{u}_h$ *is realized as the function* `divup`.

```
func real[int] divup(real[int] & pp)
{
                                                                    //    compute u1(pp)
   b[]  = Bx'*pp; b[] *=-1; b[] += bc1[] ;    u1[] = A^-1*b[];
                                                                    //    compute u2(pp)
   b[]  = By'*pp; b[] *=-1; b[] += bc2[] ;    u2[] = A^-1*b[];
   //     u^n = A^{-1}(Bx^T p^n   By^T p^n)^T
   ppp[] =   Bx*u1[];                                                //     ppp = Bxu_1
   ppp[] +=  By*u2[];                                                  //       +Byu_2
   return ppp[] ;
};
```

Call now the conjugate gradient algorithm:

```
p=0;q=0;                                                                  //     p_h^0 = 0
LinearCG(divup,p[],eps=1.e-6,nbiter=50);                          //     p_h^{n+1} = p_h^n + Bu_h^n
//    if n > 50 or |p_h^{n+1} - p_h^n| ≤ 10^{-6}, then the loop end.
divup(p[]);                                          //    compute the final solution

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

9.6.3 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 (see [30] for all the details).
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 $\nabla.((\alpha Id+\nu\Delta)^{-1}\nabla$, where Id is the identity operator. So the preconditioer suggested is $\alpha\Delta^{-1} + \nu Id$.

To implement this, we reuse the previous example, by including a file. Then we define the time step Δt, viscosity, and new variational form and matrix.

Example 9.21 (NSUzawaCahouetChabart.edp)

```
include "StokesUzawa.edp"                          //    include the Stokes part
real dt=0.05, alpha=1/dt;                                                 //    Δt

cout << " alpha = " << alpha;
real xnu=1./400;                                //    viscosity ν = Reynolds number^{-1}

                                          //    the new variational form with mass term
varf at(u1,u2)= int2d(Th)( xnu*dx(u1)*dx(u2)
                         + xnu*dy(u1)*dy(u2) + u1*u2*alpha  )
                         +  on(1,2,4,u1=0)  + on(3,u1=1) ;

A = at(Xh,Xh,solver=CG);                                    //    change the matrix

                                        //    set the 2 convect variational form
varf  vfconv1(uu,vv)  = int2d(Th,qforder=5) (convect([u1,u2],-dt,u1)*vv*alpha);
varf  vfconv2(v2,v1)  = int2d(Th,qforder=5) (convect([u1,u2],-dt,u2)*v1*alpha);
```

```
int idt;                                                    //    index of time set
real temps=0;                                               //    current time

Mh pprec,prhs;
varf vfMass(p,q) = int2d(Th)(p*q);
matrix MassMh=vfMass(Mh,Mh,solver=CG);

varf vfLap(p,q)  = int2d(Th)(dx(pprec)*dx(q)+dy(pprec)*dy(q) + pprec*q*1e-10);
matrix LapMh= vfLap(Mh,Mh,solver=Cholesky);
```

The function to define the preconditioner

```
func real[int]  CahouetChabart(real[int] & xx)
{                                                           //    xx = ∫(divu)w_i
                                                            //    αLapMh^{-1} + νMassMh^{-1}
   pprec[]= LapMh^-1* xx;
   prhs[] =  MassMh^-1*xx;
   pprec[] = alpha*pprec[]+xnu* prhs[];
   return pprec[];
};
```

The loop in time. Warning with the stop test of the conjugate gradient, because we start from the previous solution and the end the previous solution is close to the final solution, don't take a relative stop test to the first residual, take an 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 Conjugate Gradient with preconditioner '
                             //    warning eps < 0 => absolue stop test
   LinearCG(divup,p[],eps=-1.e-6,nbiter=50,precon=CahouetChabart);
   divup(p[]);                                              //    computed the velocity

   plot([u1,u2],p,wait=!(idt%10),value= 1,coef=0.1);
 }
```

9.7 Variational inequality

We present, a classical examples of variational inequality.
Let us denote $C = \{u \in H_0^1(\Omega), u \leq g\}$
The problem is :

$$u = arg \min_{u \in C} J(u) = \frac{1}{2} \int_\Omega \nabla u . \nabla u - \int_\Omega f u$$

where f and g are given function.

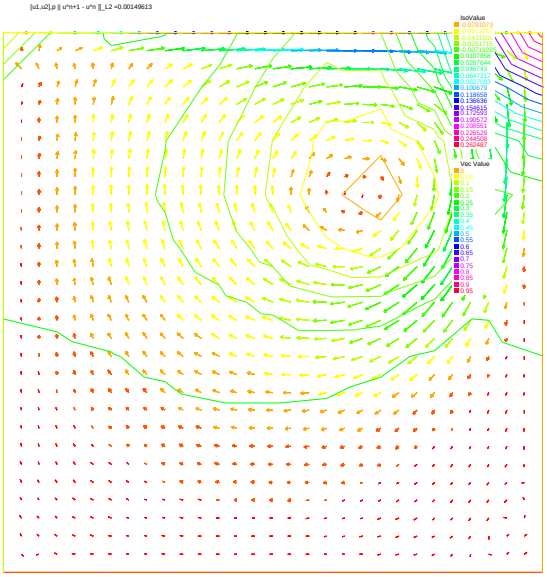


Figure 9.22: Solution of the cavity driven problem at Reynolds number 400 with the Cahouet-Chabart algorithm.

The solution is a projection on the convex C of $f^\star$ for the scalar product $((v,w)) = \int_\Omega \nabla v.\nabla w$ of $H_0^1(\Omega)$ where $f^\star$ is solution of $((f^\star, v)) = \int_\Omega fv, \forall v \in H_0^1(\Omega)$. The projection on a convex satisfy clearly $\forall v \in C, \quad ((u-v, u-\tilde{f})) \leq 0$, and after expanding, we get the classical inequality

$$\forall v \in C, \quad \int_\Omega \nabla(u-v)\nabla u \leq \int_\Omega (u-v)f.$$

We can also rewrite the problem as a saddle point problem
Find λ, u such that:

$$\max_{\lambda \in L^2(\Omega), \lambda \geq 0} \min_{u \in H_0^1(\Omega)} \mathcal{L}(u,\lambda) = \frac{1}{2}\int_\Omega \nabla u.\nabla u - \int_\Omega fu + \int_\Omega \lambda(u-g)^+$$

where $((u-g)^+ = max(0, u-g)$
This saddle point problem is equivalent to find u, λ such that:

$$\begin{cases} \int_\Omega \nabla u.\nabla v + \lambda v^+ \, d\omega = \int_\Omega fu, & \forall v \in H_0^1(\Omega) \\ \int_\Omega \mu(u-g)^+ = 0, & \forall \mu \in L^2(\Omega), \mu \geq 0, \lambda \geq 0, \end{cases} \tag{9.54}$$

A algorithm to solve the previous problem is:

1. k=0, and choose, λ_0 belong $H^{-1}(\Omega)$
2. loop on $k = 0, ...$
   (a) set $\mathcal{I}_k = \{x \in \Omega / \lambda_k + c * (u_{k+1} - g) \leq 0\}$
   (b) $V_{g,k+1} = \{v \in H_0^1(\Omega) / v = g \text{ on } I_k\}$,

(c) $V_{0,k+1} = \{v \in H_0^1(\Omega)/v = 0 \text{ on } I_k\}$,

(d) Find $u_{k+1} \in V_{g,k+1}$ and $\lambda_{k+1} \in H^{-1}(\Omega)$ such that

$$\begin{cases} \displaystyle\int_\Omega \nabla u_{k+1}.\nabla v_{k+1}\, d\omega = \int_\Omega f v_{k+1}, \quad \forall v_{k+1} \in V_{0,k+1} \\ < \lambda_{k+1}, v >= \displaystyle\int_\Omega \nabla u_{k+1}.\nabla v - f v\, d\omega \end{cases}$$

where $<,>$ is the duality bracket between $H_0^1(\Omega)$ and $H^{-1}(\Omega)$, and c is a penalty constant (large enough).

You can find all the mathematic about this algorithm in [32].
Now how to do that in `FreeFem++`
The full example is:

Example 9.22 (VI.edp)

```
mesh Th=square(20,20);
real eps=1e-5;
fespace Vh(Th,P1);                                          //    P1 FE space
int n = Vh.ndof;                                      //    number of Degree of freedom
Vh uh,uhp;                                             //    solution and previous one
Vh Ik;                                 //    to def the set where the containt is reached.
real[int] rhs(n);                        //    to store the right and side of the equation
real c=1000;                             //    the penalty parameter of the algoritm
func f=1;                                          //    right hand side function
func fd=0;                               //    Dirichlet boundary condition function
Vh g=0.05;                                            //    the discret function g

real[int] Aii(n),Aiin(n);   //    to store the diagonal of the matrix 2 version

real tgv = 1e30;                          //    a huge value for exact penalization
                                                      //    of boundary condition
//    the variatonal form of the problem:
varf a(uh,vh) =                                     //    definition of the problem
    int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) )                 //    bilinear form
  - int2d(Th)( f*vh )                                            //    linear form
  + on(1,2,3,4,uh=fd) ;                                //    boundary condition form


//    two version of the matrix of the problem
matrix A=a(Vh,Vh,tgv=tgv,solver=CG);                          //    one changing
matrix AA=a(Vh,Vh,solver:GC);                      //    one for computing residual

 //    the mass Matrix construction:
varf vM(uh,vh) = int2d(Th)(uh*vh);
matrix M=vM(Vh,Vh);                //    to do a fast computing of L^2 norm :  sqrt(
u'*(w=M*u))

Aii=A.diag;     //    get the diagonal of the matrix (appear in version 1.46-1)

rhs = a(0,Vh,tgv=tgv);
Ik =0;
uhp=-tgv;                                                 //    previous value is
```

```
Vh lambda=0;
for(int iter=0;iter<100;++iter)
{
  real[int] b(n) ; b=rhs;                    //    get a copy of the Right hand side
  real[int] Ak(n);                  //    the complementary of Ik ( !Ik = (Ik-1))
                    //    Today the operator Ik- 1.  is not implement so we do:
  Ak= 1.; Ak  -= Ik[];                                    //    build Ak = !  Ik
      //    adding new locking condition on b and on the diagonal if (Ik ==1 )
  b = Ik[] .* g[];       b *= tgv;      b  -=  Ak .* rhs;
  Aiin = Ik[] *  tgv;       Aiin  +=  Ak  .* Aii;          //   set Aii= tgv i ∈ Ik
  A.diag = Aiin;       //    set the matrix diagonal (appear in version 1.46-1)
  set(A,solver=CG);       //    important to change preconditioning for solving
  uh[] = A^-1* b;           //    solve the problem with more locking condition
  lambda[] = AA * uh[];          //    compute the residual ( fast with matrix)
  lambda[] += rhs;                                    //    remark rhs = -∫ fv

  Ik = ( lambda + c*( g- uh)) < 0.;              //    the new of locking value

   plot(Ik, wait=1,cmm=" lock set ",value=1,ps="VI-lock.eps",fill=1 );
   plot(uh,wait=1,cmm="uh",ps="VI-uh.eps");
                 //    trick to compute L^2 norm of the variation (fast method)
      real[int] diff(n),Mdiff(n);
      diff= uh[]-uhp[];
      Mdiff = M*diff;
      real err = sqrt(Mdiff'*diff);
  cout << "  || u_{k=1} - u_{k} ||_2 " << err << endl;
  if(err< eps) break;                                          //    stop test
  uhp[]=uh[] ;                                  //    set the previous solution
}
savemesh(Th,"mm",[x,y,uh*10]);                    //    for medit plotting
```

Remark, as you can see on this example, some vector , or matrix operator are not implemented so a way is to skip the expression and we use operator +=, −= to merge the result.

9.8 Domain decomposition

We present, three classic 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.

9.8.1 Schwarz Overlap Scheme

To solve

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

the Schwarz algorithm runs like this

$$\begin{aligned} -\Delta u_1^{n+1} &= f \text{ in } \Omega_1 \quad u_1^{n+1}|_{\Gamma_1} = u_2^n \\ -\Delta u_2^{n+1} &= f \text{ in } \Omega_2 \quad u_2^{n+1}|_{\Gamma_2} = u_1^n \end{aligned}$$

where Γ_i is the boundary of Ω_i and on the condition that $\Omega_1 \cap \Omega_2 \neq \emptyset$ and that u_i are zero at iteration 1.

Here we take Ω_1 to be a quadrangle, Ω_2 a disk and we apply the algorithm starting from zero.

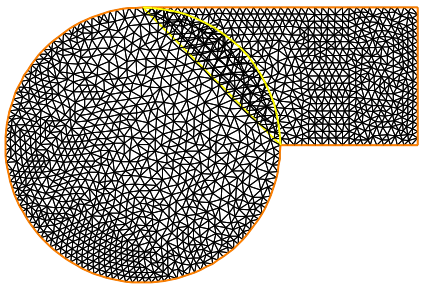


Figure 9.23: The 2 overlapping mesh `TH` and `th`

Example 9.23 (Schwarz-overlap.edp)

```
int inside = 2;                                          //    inside boundary
int outside = 1;                                         //    outside boundary
border a(t=1,2){x=t;y=0;label=outside;};
border b(t=0,1){x=2;y=t;label=outside;};
border c(t=2,0){x=t ;y=1;label=outside;};
border d(t=1,0){x = 1-t; y = t;label=inside;};
border e(t=0, pi/2){ x= cos(t); y = sin(t);label=inside;};
border e1(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;
mesh th = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
mesh TH = buildmesh( e(5*n) + e1(25*n) );
plot(th,TH,wait=1);                                      //    to see the 2 meshes
```

The space and problem definition is :

```
fespace vh(th,P1);
fespace VH(TH,P1);
vh u=0,v; VH U,V;
int i=0;

problem PB(U,V,init=i,solver=Cholesky) =
    int2d(TH)( dx(U)*dx(V)+dy(U)*dy(V) )
  + int2d(TH)( -V) + on(inside,U = u)  + on(outside,U= 0 ) ;
problem pb(u,v,init=i,solver=Cholesky) =
    int2d(th)( dx(u)*dx(v)+dy(u)*dy(v) )
  + int2d(th)( -v) + on(inside ,u = U) + on(outside,u = 0 ) ;
```

The calculation loop:

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

```
    pb;
    plot(U,u,wait=true);
};
```

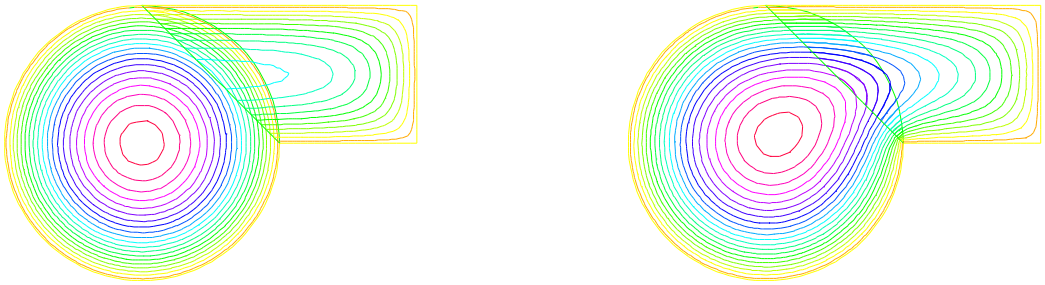


Figure 9.24: Isovalues of the solution at iteration 0 and iteration 9

9.8.2 Schwarz non Overlap Scheme

To solve

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

the Schwarz algorithm for domain decomposition without overlapping runs like this

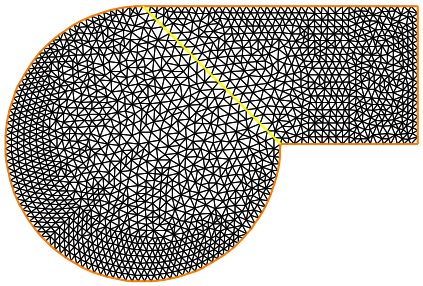


Figure 9.25: The two none overlapping mesh `TH` and `th`

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 upgradingλ with

$$\lambda = \lambda \pm \frac{(u_1 - u_2)}{2}$$

where the sign + or − of ± is choose to have convergence.

Example 9.24 (Schwarz-no-overlap.edp)

```
                                                //    schwarz1 without overlapping
int inside = 2;
int outside = 1;
border a(t=1,2){x=t;y=0;label=outside;};
border b(t=0,1){x=2;y=t;label=outside;};
border c(t=2,0){x=t ;y=1;label=outside;};
border d(t=1,0){x = 1-t; y = t;label=inside;};
border e(t=0, 1){ x= 1-t; y = t;label=inside;};
border e1(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;
mesh th = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
mesh TH = buildmesh ( e(5*n) + e1(25*n) );
plot(th,TH,wait=1,ps="schwarz-no-u.eps");
fespace vh(th,P1);
fespace VH(TH,P1);
vh u=0,v; VH U,V;
vh lambda=0;
int i=0;

problem PB(U,V,init=i,solver=Cholesky) =
    int2d(TH)( dx(U)*dx(V)+dy(U)*dy(V) )
  + int2d(TH)( -V)
  + int1d(TH,inside)(-lambda*V) +    on(outside,U= 0 ) ;
problem pb(u,v,init=i,solver=Cholesky) =
    int2d(th)( dx(u)*dx(v)+dy(u)*dy(v) )
  + int2d(th)( -v)
  + int1d(th,inside)(+lambda*v) +    on(outside,u = 0 ) ;


for ( i=0 ;i< 10; i++)
{
   PB;
   pb;
   lambda = lambda - (u-U)/2;
   plot(U,u,wait=true);
};

plot(U,u,ps="schwarz-no-u.eps");
```

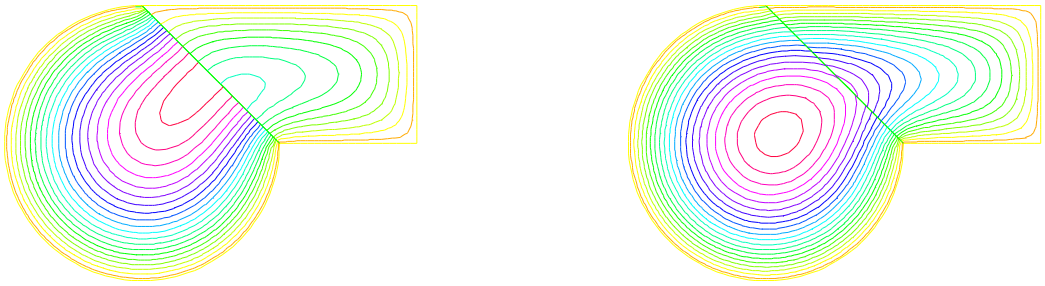


Figure 9.26: Isovalues of the solution at iteration 0 and iteration 9 without overlapping

9.8.3 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 componant. The border problem is solve with conjugate gradient.
First, we construct the two domain

Example 9.25 (Schwarz-gc.edp)

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

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

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

border GammaArc(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;
                                                //    build the mesh of Ω1 and Ω2
mesh Th1 = buildmesh( Gamma1(5*n) + Gamma2(5*n) + GammaInside(5*n) + Gamma3(5*n));
mesh Th2 = buildmesh ( GammaInside(-5*n) + GammaArc(25*n) );
plot(Th1,Th2);

                                                //    defined the 2 FE space
fespace Vh1(Th1,P1),      Vh2(Th2,P1);
```

Note 9.6 *It is impossible to define a function just on a part of boundary, so the lambda function must be defined on the all domain Ω_1 such as*

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

The two Poisson problem:

```
Vh1 u1,v1;              Vh2 u2,v2;
int i=0;                                        //    for factorization optimization
problem Pb2(u2,v2,init=i,solver=Cholesky) =
    int2d(Th2)( dx(u2)*dx(v2)+dy(u2)*dy(v2) )
  + int2d(Th2)( -v2)
  + int1d(Th2,inside)(-lambda*v2) +    on(outside,u2= 0 ) ;
problem Pb1(u1,v1,init=i,solver=Cholesky) =
    int2d(Th1)( dx(u1)*dx(v1)+dy(u1)*dy(v1) )
  + int2d(Th1)( -v1)
  + int1d(Th1,inside)(+lambda*v1) +    on(outside,u1 = 0 ) ;
```

or, we define a border matrix , because the lambda function is none zero inside the domain Ω_1:

```
varf b(u2,v2,solver=CG) =int1d(Th1,inside)(u2*v2);
matrix B= b(Vh1,Vh1,solver=CG);
```

The boundary problem function,

$$\lambda \longrightarrow \int_{\Gamma_i} (u_1 - u_2)v_1$$

```
func real[int] BoundaryProblem(real[int] &l)
{
   lambda[]=l;                                          //    make FE function form l
   Pb1;     Pb2;
   i++;                                                 //    no refactorization i !=0
   v1=-(u1-u2);
   lambda[]=B*v1[];
   return lambda[] ;
};
```

Note 9.7 *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 Conjugate 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
```

9.9 Fluid/Structures Coupled Problem

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

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

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

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

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

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

where δ is the Kronecker symbol and where λ, μ are two constants describing the material mechanical properties in terms of the modulus of elasticity, and Young's modulus.
The equations of elasticity are naturally written in variational form for the displacement vector $v(x) \in V$ as

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

The data are the gravity force $\boldsymbol{g}$ and the boundary stress $\boldsymbol{h}$.

Example 9.26 (fluidStruct.edp) *In our example the Lamé system and the Stokes system are coupled by a common boundary on which the fluid stress creates a displacement of the boundary and hence changes the shape of the domain where the Stokes problem is integrated. The geometry is that of a vertical driven cavity with an elastic lid. The lid is a beam with weight so it will be deformed by its own weight and by the normal stress due to the fluid reaction. The cavity is the* 10×10 *square and the lid is a rectangle of height* $l = 2$.

A beam sits on a box full of fluid rotating because the left vertical side has velocity one. The beam is bent by its own weight, but the pressure of the fluid modifies the bending.
The bending displacement of the beam is given by (uu,vv) whose solution is given as follows.

```
//    Fluid-structure interaction for a weighting beam sitting on a
                                //    square cavity filled with a fluid.

int bottombeam = 2;                                    //    label of bottombeam
border a(t=2,0)  { x=0; y=t ;label=1;};                    //    left beam
border b(t=0,10) { x=t; y=0 ;label=bottombeam;};          //    bottom of beam
border c(t=0,2)  { x=10; y=t ;label=1;};                   //    rigth beam
border d(t=0,10) { x=10-t; y=2; label=3;};                   //    top beam
real E = 21.5;
real sigma = 0.29;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -0.05;
```

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

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

Then Stokes equation for fluids ast low speed are solved in the box below the beam, but the beam has deformed the box (see border h):

```
                    //   Stokes on square b,e,f,g driven cavite on left side g
border e(t=0,10) { x=t; y=-10; label= 1; };                          //    bottom
border f(t=0,10) { x=10; y=-10+t ; label= 1; };                       //    right
border g(t=0,10) { x=0; y=-t ;label= 2;};                             //    left
border h(t=0,10) { x=t; y=vv(t,0)*( t>=0.001 )*(t <= 9.999);
                   label=3;};                      //    top of cavity deformed

mesh sh = buildmesh(h(-20)+f(10)+e(10)+g(10));
plot(sh,wait=1);
```

We use the Uzawa conjugate gradient to solve the Stokes problem like in example Section 9.6.2

```
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[] ;
};
```

do a loop on the two problem

```
for(step=0;step<2;++step)
 {
   p=0;q=0;u1=0;v1=0;

   LinearCG(divup,p[],eps=1.e-3,nbiter=50);
   divup(p[]);
```

Now the beam will feel the stress constraint from the fluid:

```
  Vh sigma11,sigma22,sigma12;
  Vh uu1=uu,vv1=vv;

  sigma11([x+uu,y+vv]) = (2*dx(u1)-p);
  sigma22([x+uu,y+vv]) = (2*dy(u2)-p);
  sigma12([x+uu,y+vv]) = (dx(u1)+dy(u2));
```

which comes as a boundary condition to the PDE of the beam:

```
  varf fluidf([uu,vv],[w,s]) fluidforce =
  solve  bbst([uu,vv],[w,s],init=i)  =
     int2d(th)(
                  lambda*div(w,s)*div(uu,vv)
                  +2.*mu*( epsilon(w,s)'*epsilon(uu,vv) )
              )
  + 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);
 }                                                    //    end of loop
```

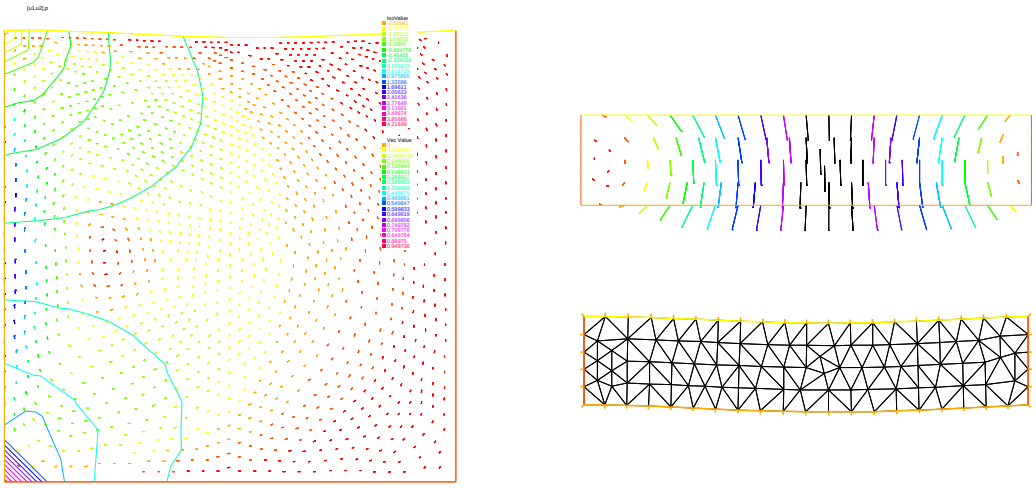


Figure 9.27: Fluid velocity and pressure (left) and displacement vector (center) of the structure and displaced geometry (right) in the fluid-structure interaction of a soft side and a driven cavity

9.10 Transmission Problem

Consider an elastic plate whose displacement change vertically, which is made up of three plates of different materials, welded on each other. Let Ω_i, $i = 1, 2, 3$ be the domain occupied by i-th material with tension μ_i (see Section 9.1.1). The computational domain Ω is the interior of $\overline{\Omega_1} \cup \overline{\Omega_2} \cup \overline{\Omega_3}$. The vertical displacement $u(x, y)$ is obtained from

$$-\mu_i \Delta u = f \text{ in } \Omega_i \tag{9.55}$$

$$\mu_i \partial_n u|_{\Gamma_i} = -\mu_j \partial_n u|_{\Gamma_j} \quad \text{on } \overline{\Omega_i} \cap \overline{\Omega_j} \qquad \text{if } 1 \le i < j \le 3 \tag{9.56}$$

where $\partial_n u|_{\Gamma_i}$ denotes the value of the normal derivative $\partial_n u$ on the boundary Γ_i of the domain Ω_i. By introducing the characteristic function χ_i of Ω_i, that is,

$$\chi_i(x) = 1 \quad \text{if } x \in \Omega_i; \qquad \chi_i(x) = 0 \quad \text{if } x \notin \Omega_i \tag{9.57}$$

we can easily rewrite (9.55) and (9.56) to the weak form. Here we assume that $u = 0$ on $\Gamma = \partial\Omega$.
problem Transmission: For a given function f, find u such that

$$a(u, v) = \ell(f, v) \quad \text{for all } v \in H_0^1(\Omega) \tag{9.58}$$

$$a(u, v) = \int_\Omega \mu \nabla u \cdot \nabla v, \quad \ell(f, v) = \int_\Omega f v$$

where $\mu = \mu_1\chi_1 + \mu_2\chi_2 + \mu_3\chi_3$. Here we notice that μ become the discontinuous function.
With dissipation, and at the thermal equilibrium, the temperature equation is:
This example explains the definition and manipulation of *region*, i.e. subdomains of the whole domain.
Consider this L-shaped domain with 3 diagonals as internal boundaries, defining 4 subdomains:

```
//    example using region keyword
```

```
                                //    construct a mesh with 4 regions (sub-domains)
border a(t=0,1){x=t;y=0;};
border b(t=0,0.5){x=1;y=t;};
border c(t=0,0.5){x=1-t;y=0.5;};
border d(t=0.5,1){x=0.5;y=t;};
border e(t=0.5,1){x=1-t;y=1;};
border f(t=0,1){x=0;y=1-t;};
                                                              //    internal boundary
border i1(t=0,0.5){x=t;y=1-t;};
border i2(t=0,0.5){x=t;y=t;};
border i3(t=0,0.5){x=1-t;y=t;};

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

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

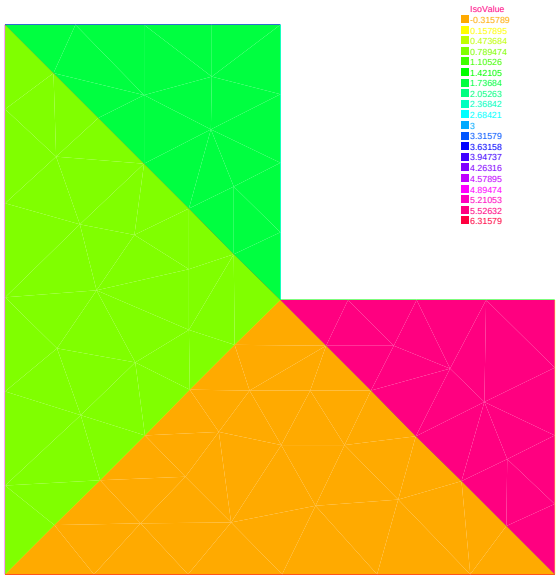


Figure 9.28: the function `reg`

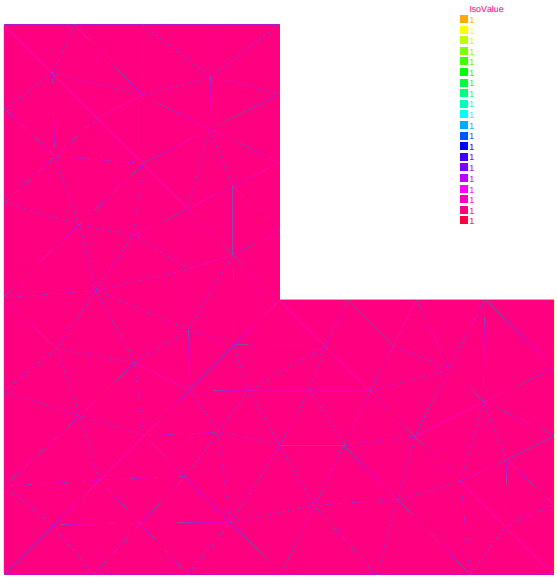


Figure 9.29: the function `nu`

`region` is a keyword of `FreeFem++` which is in fact a variable depending of the current position (is not a function today, use `Ph reg=region;` to set a function). This variable value returned is the number of the subdomain of the current position. This number is defined by ”buildmesh” which scans while building the mesh all its connected component. So to get the number of a region containing a particular point one does:

```
int nupper=reg(0.4,0.9);          //    get the region number of point (0.4,0.9)
int nlower=reg(0.9,0.1);          //    get the region number of point (0.4,0.1)
cout << " nlower " <<  nlower << ", nupper = " << nupper<< endl;
```

```
          //    defined the characteristics functions of upper and lower region
Ph nu=1+5*(region==nlower) + 10*(region==nupper);
plot(nu,fill=1,wait=1);
```

This is particularly useful to define discontinuous functions such as might occur when one part of the domain is copper and the other one is iron, for example.
We this in mind we proceed to solve a Laplace equation with discontinuous coefficients (ν is 1, 6 and 11 below).

```
Ph nu=1+5*(region==nlower) + 10*(region==nupper);
plot(nu,fill=1,wait=1);
problem lap(u,v) =   int2d(th)( nu*( dx(u)*dx(v)*dy(u)*dy(v) ))
                   + int2d(-1*v) + on(a,b,c,d,e,f,u=0);
plot(u);
```

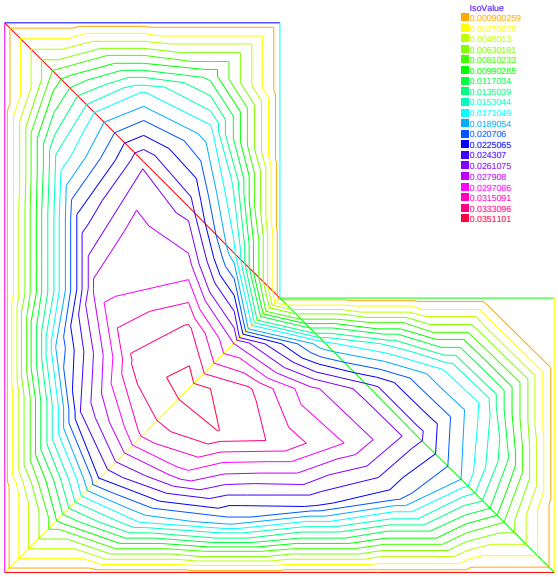


Figure 9.30: the isovalue of the solution u

9.11 Free Boundary Problem

The domain Ω is defined with:

```
real L=10;                                          //   longueur du domaine
real h=2.1;                                       //    hauteur du bord gauche
real h1=0.35;                                     //    hauteur du bord droite

                                                  //    maillage d'un tapeze
border a(t=0,L){x=t;y=0;};                        //    bottom:  Γa
border b(t=0,h1){x=L;y=t;};                       //    right:  Γb
border f(t=L,0){x=t;y=t*(h1-h)/L+h;};             //    free surface:  Γf
border d(t=h,0){x=0;y=t;};                        //    left:  Γd

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

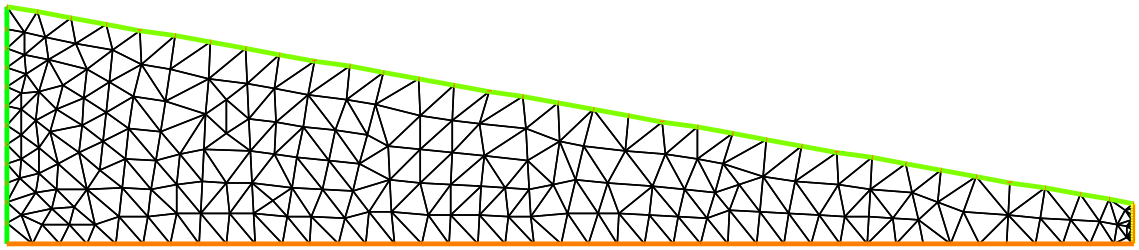


Figure 9.31: The mesh of the domain Ω

The free boundary problem is:
Find u and Ω such that:

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega \\ u = y & \text{on } \Gamma_b \\ \dfrac{\partial u}{\partial n} = 0 & \text{on } \Gamma_d \cup \Gamma_a \\ \dfrac{\partial u}{\partial n} = \dfrac{q}{K} n_x \text{ and } u = y & \text{on } \Gamma_f \end{cases}$$

We use a fixed point method; $\Omega^0 = \Omega$
in two step, fist we solve the classical following problem:

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega^n \\ u = y & \text{on } \Gamma_b^n \\ \dfrac{\partial u}{\partial n} = 0 & \text{on } \Gamma_d^n \cup \Gamma_a^n \\ u = y & \text{on } \Gamma_f^n \end{cases}$$

The variational 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 construct a domain deformation $\mathcal{F}(x,y) = [x, y - v(x,y)]$
where v is solution of the following problem:

$$\begin{cases} -\Delta v &= 0 & \text{in } \Omega^n \\ v &= 0 & \text{on } \Gamma_a^n \\ \dfrac{\partial v}{\partial n} &= 0 & \text{on } \Gamma_b^n \cup \Gamma_d^n \\ \dfrac{\partial v}{\partial n} &= \dfrac{\partial u}{\partial n} - \dfrac{q}{K} n_x & \text{on } \Gamma_f^n \end{cases}$$

The variational formulation is:
find v on V, such than $v = 0$ on Γ_a^n

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

Finally the new domain $\Omega^{n+1} = \mathcal{F}(\Omega^n)$

Example 9.27 (freeboundary.edp) *The FreeFem++ :implementation is:*

```
real q=0.02;                                                    //   flux entrant
real K=0.5;                                                     //   permeabilité

fespace Vh(Th,P1);
int j=0;

Vh u,v,uu,vv;

problem Pu(u,uu,solver=CG) = int2d(Th)( dx(u)*dx(uu)+dy(u)*dy(uu))
  + on(b,f,u=y) ;

problem Pv(v,vv,solver=CG) = int2d(Th)( dx(v)*dx(vv)+dy(v)*dy(vv))
  +  on (a, v=0) + int1d(Th,f)(vv*((q/K)*N.y- (dx(u)*N.x+dy(u)*N.y)));


real errv=1;
real erradap=0.001;
verbosity=1;
while(errv>1e-6)
{
  j++;
  Pu;
  Pv;
  plot(Th,u,v ,wait=0);
  errv=int1d(Th,f)(v*v);
   real coef=1;

                                                                                //
  real mintcc = checkmovemesh(Th,[x,y])/5.;
  real mint = checkmovemesh(Th,[x,y-v*coef]);

  if (mint<mintcc ||  j%10==0) {                //    mesh to bad => remeshing
    Th=adaptmesh(Th,u,err=erradap ) ;
    mintcc = checkmovemesh(Th,[x,y])/5.;
  }
```

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

    if (mint>mintcc) break;

    cout << " min |T]  " << mint << endl;
    coef /= 1.5;
  }

  Th=movemesh(Th,[x,y-coef*v]);                    //    calcul de la deformation
  cout << "\n\n"<<j <<"------------ errv = " << errv << "\n\n";

}
plot(Th,ps="d_Thf.eps");
plot(u,wait=1,ps="d_u.eps");
```

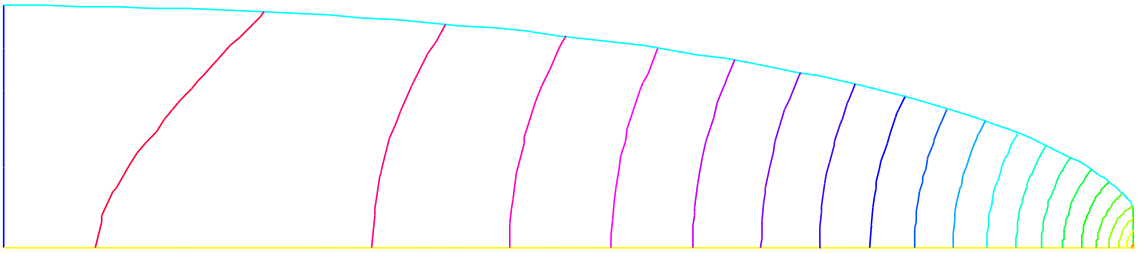


Figure 9.32: The final solution on the new domain Ω^{72}

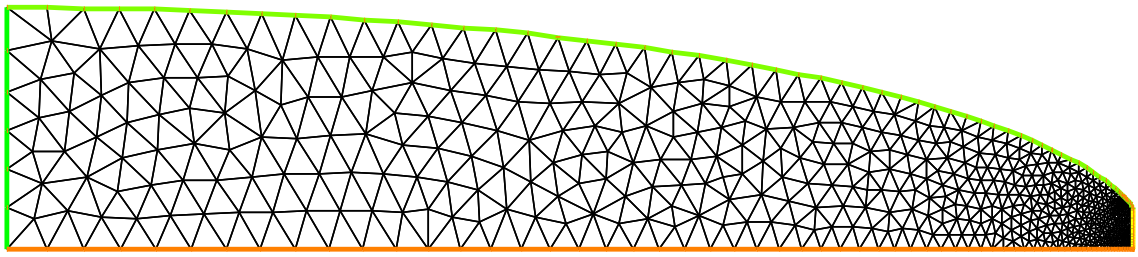


Figure 9.33: The adapted mesh of the domain Ω^{72}

9.12 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:

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

denote $\mathbf{u} = (u_1, u_2)$, $\mathbf{v} = (v_1, v_2)$, $\mathbf{w} = (w_1, w_2)$.
So, the differential of J is

$$DJ(\boldsymbol{u})(\boldsymbol{v}) = \int DF2(\boldsymbol{u})(\boldsymbol{v})\ f'(F2(\boldsymbol{u}))) - \int_{\Gamma_p} P_a v_2$$

where $DF2(\boldsymbol{u})(\boldsymbol{v}) = 2\ A(\ DE[\mathbf{u}](\mathbf{v})\ ,\ E[\mathbf{u}]\ )$ and DE is the first differential of E.
The second order differential is

$$\begin{aligned} D^2J(\mathbf{u})((\mathbf{v}),(\mathbf{w})) &= \int DF2(\boldsymbol{u})(\boldsymbol{v})\ DF2(\boldsymbol{u})(\boldsymbol{w})\ f''(F2(\mathbf{u}))) \\ &+ \int D^2F2(\boldsymbol{u})(\boldsymbol{v},\boldsymbol{w})\ f'(F2(\mathbf{u}))) \end{aligned}$$

where

$$D^2F2(\boldsymbol{u})(\boldsymbol{v},\boldsymbol{w}) = 2\ A(\ D^2E[\mathbf{u}](\boldsymbol{v},\boldsymbol{w})\ ,\ E[\boldsymbol{u}]\ ) + 2\ A(\ DE[\boldsymbol{u}](\boldsymbol{v})\ , DE[\boldsymbol{u}](\boldsymbol{w})\ ).$$

and D^2E is the second differential of E.

So all notation can be define with `macro` like:

```
macro EL(u,v) [dx(u),(dx(v)+dy(u)),dy(v)]                    //    is [ε11,2ε12,ε22]

macro ENL(u,v) [
(dx(u)*dx(u)+dx(v)*dx(v))*0.5,
(dx(u)*dy(u)+dx(v)*dy(v))     ,
(dy(u)*dy(u)+dy(v)*dy(v))*0.5 ]                                  //    EOM ENL

macro dENL(u,v,uu,vv) [(dx(u)*dx(uu)+dx(v)*dx(vv)),
  (dx(u)*dy(uu)+dx(v)*dy(vv)+dx(uu)*dy(u)+dx(vv)*dy(v)),
  (dy(u)*dy(uu)+dy(v)*dy(vv)) ]                                    //


macro E(u,v) (EL(u,v)+ENL(u,v))                              //    is [E11,2E12,E22]
macro dE(u,v,uu,vv) (EL(uu,vv)+dENL(u,v,uu,vv))                  //
macro ddE(u,v,uu,vv,uuu,vvv) dENL(uuu,vvv,uu,vv)                 //
macro F2(u,v) (E(u,v)'*A*E(u,v))                                 //
macro dF2(u,v,uu,vv)  (E(u,v)'*A*dE(u,v,uu,vv)*2. )              //
macro ddF2(u,v,uu,vv,uuu,vvv) (
            (dE(u,v,uu,vv)'*A*dE(u,v,uuu,vvv))*2.
          + (E(u,v)'*A*ddE(u,v,uu,vv,uuu,vvv))*2.  )                 //    EOM
```

The Newton Method is
choose $n = 0$,and u_O, v_O the initial displacement

- loop:

- find (du, dv) : solution of

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

- $un = un - du, \quad vn = vn - dv$

- until (du, dv) small is enough

The way to implement this algorithm in FreeFem++ is use a macro tool to implement A and $F2$, f, f',f''.
A macro is like is ccp preprocessor of C++ , but this begin by macro and the end of the macro definition is the begin of the comment //. In this case the macro is very useful because the type of parameter can be change. And it is easy to make automatic differentiation.

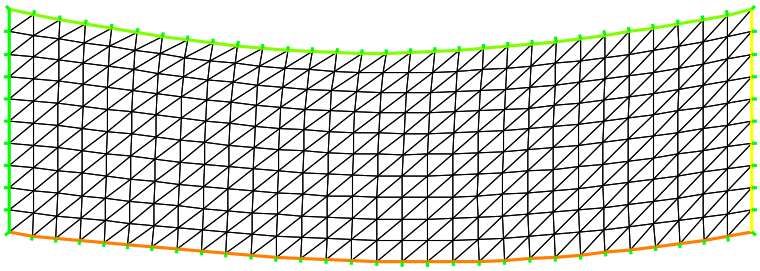


Figure 9.34: The deformed domain

```
                                                    //    non linear elasticity model

                                                   //    for hyper elasticity problem
                                                  //    ----------------------------
macro f(u) (u)                                                    //    end of macro
macro df(u) (1)                                                   //    end of macro
macro ddf(u) (0)                                                  //    end of macro

                                 //    -- du caouchouc --- CF cours de Herve Le Dret.
                                                  //    -------------------------------
real mu = 0.012e5;                                                //    kg/cm^2
real lambda =  0.4e5;                                             //    kg/cm^2
//
//     σ = 2μE + λtr(E)Id
//     A(u,v) = σ(u) : E(v)
//
//     ( a b )
//     ( b c )
//
//     tr*Id :  (a,b,c) -> (a+c,0,a+c)
//     so the associed matrix is:
//     ( 1 0 1 )
//     ( 0 0 0 )
//     ( 1 0 1 )
                                                          //    ------------------v
real a11= 2*mu +  lambda  ;
real a22= mu ;                                           //    because [0,2*t12,0]'A[0,2*s12,0] =
                                                //    = 2*mu*(t12*s12+t21*s21) = 4*mu*t12*s12
real a33= 2*mu +   lambda ;
real a12= 0 ;
real a13= lambda ;
real a23= 0 ;
                                                                  //    symetric part
```

```
real a21= a12 ;
real a31= a13 ;
real a32= a23 ;

                                                                    //    the matrix A.
func A = [ [ a11,a12,a13],[ a21,a22,a23],[ a31,a32,a33] ];

real Pa=1e2;                                                 //    a pressure of 100 Pa
//    -----------------

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

plot(Th);



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




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

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

varf vmass([uu,vv],[w,s],solver=CG) =  int2d(Th)( uu*w + vv*s );
matrix M=vmass(Vh,Vh);
problem NonLin([uu,vv],[w,s],solver=LU)=
 int2d(Th,qforder=1)(                                              //    (D^2J(un)) part
                       dF2(un,vn,uu,vv)*dF2(un,vn,w,s)*ddfe2
                    +  ddF2(un,vn,w,s,uu,vv)*dfe2
             )
   - int1d(Th,3)(Pa*s)
   - int2d(Th,qforder=1)(                                          //    (DJ(un)) part
           dF2(un,vn,w,s)*dfe2    )
   + on(right,left,uu=0,vv=0);
;
                                                                  //    Newton's method
                                                                  //    ---------------
Sh u1,v1;
for (int i=0;i<10;i++)
{
  cout << "Loop " << i << endl;
  e2 = F2(un,vn);
  dfe2 = df(e2) ;
  ddfe2 = ddf(e2);
  cout << "  e2 max " <<e2[].max << " , min" << e2[].min << endl;
  cout << " de2 max "<< dfe2[].max << " , min" << dfe2[].min << endl;
```

```
  cout << "dde2 max "<< ddfe2[].max << " , min" << ddfe2[].min << endl;
  NonLin;                                        //    compute [uu, vv] = (D^2 J(un))^-1 (DJ(un))

  w[]   = M*uu[];
  real res = sqrt(w[]' * uu[]);                                  //    norme L^2 of [uu, vv]
  u1 = uu;
  v1 = vv;
  cout << " L^2 residual = " << res << endl;
  cout << " u1 min =" <<u1[].min << ", u1 max= " << u1[].max << endl;
  cout << " v1 min =" <<v1[].min << ", v2 max= " << v1[].max << endl;
  plot([uu,vv],wait=1,cmm=" uu, vv " );
  un[] -= uu[];
  plot([un,vn],wait=1,cmm=" deplacement " );
  if (res<1e-5) break;
}

plot([un,vn],wait=1);
mesh th1 = movemesh(Th, [x+un, y+vn]);
plot(th1,wait=1);                                                //    see figure 9.34
```

9.13 Compressible Neo-Hookean Materials: Computational Solutions

Author : Alex Sadovsky mailsashas@gmail.com

9.13.1 Notation

In what follows, the symbols $\boldsymbol{u}, \mathbf{F}, \mathbf{B}, \mathbf{C}, \underline{\sigma}$ denote, respectively, the displacement field, the deformation gradient, the left Cauchy-Green strain tensor $\mathbf{B} = \mathbf{F}\mathbf{F}^T$, the right Cauchy-Green strain tensor $\mathbf{C} = \mathbf{F}^T\mathbf{F}$, and the Cauchy stress tensor. We also introduce the symbols $I_1 := \mathbf{tr}\ \mathbf{C}$ and $J := \det \mathbf{F}$. Use will be made of the identity

$$\frac{\partial J}{\partial \mathbf{C}} = J\mathbf{C}^{-1} \tag{9.59}$$

The symbol $\mathbf{I}$ denotes the identity tensor. The symbol Ω_0 denotes the reference configuration of the body to be deformed. The unit volume in the reference (resp., deformed) configuration is denoted dV (resp., dV_0); these two are related by

$$dV = JdV_0,$$

which allows an integral over Ω involving the Cauchy stress $\mathbf{T}$ to be rewritten as an integral of the Kirchhoff stress $\kappa = J\mathbf{T}$ over Ω_0.

Recommended References

For an exposition of nonlinear elasticity and of the underlying linear- and tensor algebra, see [33]. For an advanced mathematical analysis of the Finite Element Method, see [34]. An explanation of the Finite Element formulation of a nonlinear elastostatic boundary value problem, see http://www.engin.brown.edu/courses/en222/Notes/FEMfinitestrain/FEMfinitestrain.htm.

9.13.2 A Neo-Hookean Compressible Material

Constitutive Theory and Tangent Stress Measures The strain energy density function is given by

$$W = \frac{\mu}{2}(I_1 - \mathbf{tr}\ \mathbf{I} - 2\ln J) \tag{9.60}$$

(see [31], formula (12)).
The corresponding 2nd Piola-Kirchoff stress tensor is given by

$$\mathbf{S}_n := \frac{\partial W}{\partial \mathbf{E}}(\mathbf{F}_n) = \mu(\mathbf{I} - \mathbf{C}^{-1}) \tag{9.61}$$

The Kirchhoff stress, then, is

$$\kappa = \mathbf{F}\mathbf{S}\mathbf{F}^T = \mu(\mathbf{B} - \mathbf{I}) \tag{9.62}$$

The tangent Kirchhoff stress tensor at $\mathbf{F}_n$ acting on $\delta\mathbf{F}_{n+1}$ is, consequently,

$$\frac{\partial \kappa}{\partial \mathbf{F}}(\mathbf{F}_n)\delta\mathbf{F}_{n+1} = \mu\left[\mathbf{F}_n(\delta\mathbf{F}_{n+1})^T + \delta\mathbf{F}_{n+1}(\mathbf{F}_n)^T\right] \tag{9.63}$$

The Weak Form of the BVP in the Absence of Body (External) Forces The Ω_0 we are considering is an elliptical annulus, whose boundary consists of two concentric ellipses (each allowed to be a circle as a special case), with the major axes parallel. Let P denote the dead stress load (traction) on a portion $\partial\Omega_0^t$ (= the inner ellipse) of the boundary $\partial\Omega_0$. On the rest of the boundary, we prescribe zero displacement.
The weak formulation of the boundary value problem is

$$\left.\begin{array}{rcl} 0 & = & \int_{\Omega_0} \kappa[\mathbf{F}] \ : \ \left\{(\boldsymbol{\nabla}\otimes \boldsymbol{w})(\mathbf{F})^{-1}\right\} \\ & - & \int_{\partial\Omega_0^t} P\cdot\hat{N}_0 \end{array}\right\}$$

For brevity, in the rest of this section we assume $P = 0$. The provided FreeFem++ code, however, does not rely on this assumption and allows for a general value and direction of P.
Given a Newton approximation $\boldsymbol{u}_n$ of the displacement field $\boldsymbol{u}$ satisfying the BVP, we seek the correction $\delta\boldsymbol{u}_{n+1}$ to obtain a better approximation

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \delta\boldsymbol{u}_{n+1}$$

by solving the weak formulation

$$\left.\begin{array}{rcl} 0 & = & \int_{\Omega_0} \kappa[\mathbf{F}_n + \delta\mathbf{F}_{n+1}] \ : \ \left\{(\boldsymbol{\nabla}\otimes \boldsymbol{w})(\mathbf{F}_n + \delta\mathbf{F}_{n+1})^{-1}\right\} - \int_{\partial\Omega_0} P\cdot\hat{N}_0 \\ & = & \int_{\Omega_0} \left\{\kappa[\mathbf{F}_n] + \frac{\partial\kappa}{\partial\mathbf{F}}[\mathbf{F}_n]\delta\mathbf{F}_{n+1}\right\} \ : \ \left\{(\boldsymbol{\nabla}\otimes \boldsymbol{w})(\mathbf{F}_n + \delta\mathbf{F}_{n+1})^{-1}\right\} \\ & = & \int_{\Omega_0} \left\{\kappa[\mathbf{F}_n] + \frac{\partial\kappa}{\partial\mathbf{F}}[\mathbf{F}_n]\delta\mathbf{F}_{n+1}\right\} \ : \ \left\{(\boldsymbol{\nabla}\otimes \boldsymbol{w})(\mathbf{F}_n^{-1} + \mathbf{F}_n^{-2}\delta\mathbf{F}_{n+1})\right\} \\ & = & \int_{\Omega_0} \kappa[\mathbf{F}_n] \ : \ \left\{(\boldsymbol{\nabla}\otimes \boldsymbol{w})\mathbf{F}_n^{-1}\right\} \\ & - & \int_{\Omega_0} \kappa[\mathbf{F}_n] \ : \ \left\{(\boldsymbol{\nabla}\otimes \boldsymbol{w})(\mathbf{F}_n^{-2}\delta\mathbf{F}_{n+1})\right\} \\ & + & \int_{\Omega_0} \left\{\frac{\partial\kappa}{\partial\mathbf{F}}[\mathbf{F}_n]\delta\mathbf{F}_{n+1}\right\} \ : \ \left\{(\boldsymbol{\nabla}\otimes \boldsymbol{w})\mathbf{F}_n^{-1}\right\} \end{array}\right\} \quad \text{for all test functions } \boldsymbol{w}, \tag{9.64}$$

where we have taken

$$\delta\mathbf{F}_{n+1} = \boldsymbol{\nabla}\otimes\delta\boldsymbol{u}_{n+1}$$

Note: Contrary to standard notational use, the symbol δ here bears no variational context. By δ we mean simply an increment in the sense of Newton's Method. The role of a variational virtual displacement here is played by $\boldsymbol{w}$.

9.13.3 An Approach to Implementation in `FreeFem++`

The associated file is `examples++-tutorial/nl-elast-neo-Hookean.edp`.
Introducing the code-like notation, where a string in <>'s is to be read as one symbol, the individual components of the tensor

$$< TanK >:= \frac{\partial \kappa}{\partial \mathbf{F}}[\mathbf{F}_n]\delta\mathbf{F}_{n+1} \tag{9.65}$$

will be implemented as the macros $< TanK11 >, < TanK12 >, \ldots$.
The individual components of the tensor quantities

$$\mathbf{D}_1 := \mathbf{F}_n(\delta\mathbf{F}_{n+1})^T + \delta\mathbf{F}_{n+1}(\mathbf{F}_n)^T,$$

$$\mathbf{D}_2 := \mathbf{F}_n^{-T}\delta\mathbf{F}_{n+1},$$

$$\mathbf{D}_3 := (\boldsymbol{\nabla} \otimes \boldsymbol{w})\mathbf{F}_n^{-2}\delta\mathbf{F}_{n+1},$$

and

$$\mathbf{D}_4 := (\boldsymbol{\nabla} \otimes \boldsymbol{w})\mathbf{F}_n^{-1},$$

will be implemented as the macros

$$\left.\begin{array}{llll} < d1Aux11 >, < d1Aux12 >, & \ldots & , < d1Aux22 >, \\ < d2Aux11 >, < d2Aux12 >, & \ldots & , < d2Aux22 > \\ < d3Aux11 >, < d3Aux12 >, & \ldots & , < d3Aux22 > \\ < d4Aux11 >, < d4Aux12 >, & \ldots & , < d4Aux22 > \end{array}\right\}, \tag{9.66}$$

respectively.
In the above notation, the tangent Kirchhoff stress term becomes

$$\frac{\partial \kappa}{\partial \mathbf{F}}(\mathbf{F}_n)\ \delta\mathbf{F}_{n+1} = \mu\, \mathbf{D}_1 \tag{9.67}$$

while the weak BVP formulation acquires the form

$$\left.\begin{array}{rcl} 0 & = & \int_{\Omega_0} \kappa[\mathbf{F}_n]\ :\ \mathbf{D}_4 \\ & - & \int_{\Omega_0} \kappa[\mathbf{F}_n]\ :\ \mathbf{D}_3 \\ & + & \int_{\Omega_0} \left\{\frac{\partial \kappa}{\partial \mathbf{F}}[\mathbf{F}_n]\delta\mathbf{F}_{n+1}\right\}\ :\ \mathbf{D}_4 \end{array}\right\} \quad \text{for all test functions } \boldsymbol{w} \tag{9.68}$$

Chapter 10

Parallel version experimental

A first attempt of parallelization of `FreeFem++` is made here with **`mpi`**. We add 4 words in the language:

`mpisize` The total number of processes

`mpirank` the number of my current process in $\{0, ..., mpisize - 1\}$.

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

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

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

10.1 Schwarz in parallel

This example is just the rewritting of example `schwarz-overlap` in section 9.8.1.

```
[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 version 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 11

Graphical User Interface

`FreeFem++-cs` is part of the standard `FreeFem++` package. It runs on Linux, MacOS X and Windows.
"-cs" stands for "client/server". The executable program named `FreeFem++-cs` contains code for running a graphical user interface client and a computational server. The server is automatically started every time every time the user asks for a script to be run. To run `FreeFem++-cs`, just type its name in, optionally followed by a script name. It will open a window corresponding to fig. 11.1.

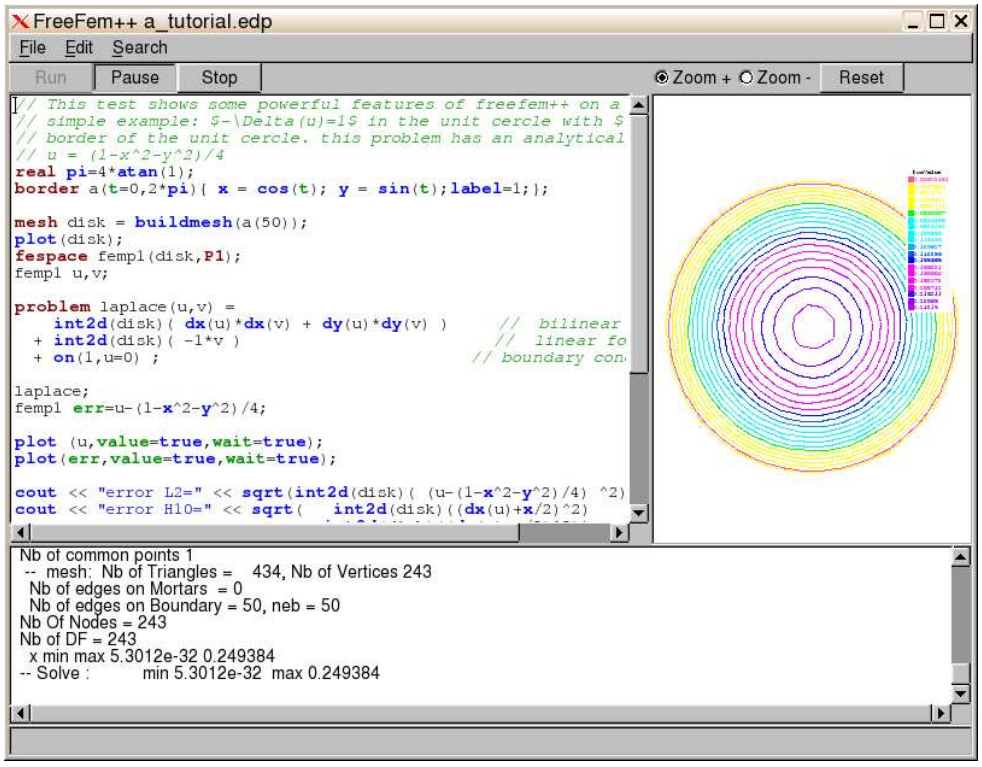


Figure 11.1: `FreeFem++-cs` main window

The main characteristics of `FreeFem++-cs` are :

- A main window composed of three panels : editor with syntax highlighting (right), `FreeFem++` messages (bottom) and graphics (left).
- Panel sizes can be changed by dragging their borders with the mouse. Any of the three panels can be made to fill the whole window.
- The edited script can be run at any time by clicking on the "Run" button.
- Dragging a `FreeFem++` script file icon from a file manager into the editor window makes `FreeFem++-cs` edit that script.
- Graphics can be examined (e.g. zoomed) while `FreeFem++` is running.
- A running `FreeFem++` computation can be paused or stopped at any time.

All commands should be self-explanatory. Here are just a few useful hints :

- There is no need to save a `FreeFem++` script to run it. It is run exactly as displayed in the editor window, and the corresponding file is not touched.
- The current directory is updated every time a script is loaded or saved. All `include` directives are therefore relative to the directory where the main script is located.
- Specifying `wait=1` in a `plot` command is exactly equivalent to clicking on the "Pause" button when the plot is displayed.
- In zoom mode, if your mouse has more than one button, a middle-click resets any zooming coefficient, and a right-click zooms in the opposite way of the left-click.

Chapter 12

Mesh Files

12.1 File mesh data structure

The mesh data structure, output of a mesh generation algorithm, refers to the geometric data structure and in some case to another mesh data structure.
In this case, the fields are

- `MeshVersionFormatted 0`

- `Dimension (I)` dim

- `Vertices (I)` NbOfVertices
  $\big(\big($ `(R)` x_i^j , j=1,dim $\big)$, `(I)` $Ref\phi_i^v$, i=1 , NbOfVertices $\big)$

- `Edges (I)` NbOfEdges
  $\big($ `@@Vertex`$_i^1$, `@@Vertex`$_i^2$, `(I)` $Ref\phi_i^e$, i=1 , NbOfEdges $\big)$

- `Triangles (I)` NbOfTriangles
  $\big(\big($ `@@Vertex`$_i^j$, j=1,3 $\big)$, `(I)` $Ref\phi_i^t$, i=1 , NbOfTriangles $\big)$

- `Quadrilaterals (I)` NbOfQuadrilaterals
  $\big(\big($ `@@Vertex`$_i^j$, j=1,4 $\big)$, `(I)` $Ref\phi_i^t$, i=1 , NbOfQuadrilaterals $\big)$

- `Geometry`
  `(C*)` FileNameOfGeometricSupport

  - `VertexOnGeometricVertex`
    `(I)` NbOfVertexOnGeometricVertex
    $\big($ `@@Vertex`$_i$, `@@Vertex`$_i^{geo}$, i=1,NbOfVertexOnGeometricVertex $\big)$
  - `EdgeOnGeometricEdge`
    `(I)` NbOfEdgeOnGeometricEdge
    $\big($ `@@Edge`$_i$, `@@Edge`$_i^{geo}$, i=1,NbOfEdgeOnGeometricEdge $\big)$

- `CrackedEdges (I)` NbOfCrackedEdges
  $\big($ `@@Edge`$_i^1$, `@@Edge`$_i^2$, i=1 , NbOfCrackedEdges $\big)$

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

- `MeshSupportOfVertices`
  `(C*)` FileNameOfMeshSupport

  - `VertexOnSupportVertex`
    `(I)` NbOfVertexOnSupportVertex
    $\left(\texttt{@@Vertex}_i, \texttt{@@Vertex}_i^{supp}\right.$, i=1,NbOfVertexOnSupportVertex$\left.\right)$
  - `VertexOnSupportEdge`
    `(I)` NbOfVertexOnSupportEdge
    $\left(\texttt{@@Vertex}_i, \texttt{@@Edge}_i^{supp}, \texttt{(R)}\ u_i^{supp}\right.$, i=1,NbOfVertexOnSupportEdge$\left.\right)$
  - `VertexOnSupportTriangle`
    `(I)` NbOfVertexOnSupportTriangle
    $\left(\texttt{@@Vertex}_i, \texttt{@@Tria}_i^{supp}, \texttt{(R)}\ u_i^{supp}, \texttt{(R)}\ v_i^{supp}\right.$,
    i=1 , NbOfVertexOnSupportTriangle$\left.\right)$
  - `VertexOnSupportQuadrilaterals`
    `(I)` NbOfVertexOnSupportQuadrilaterals
    $\left(\texttt{@@Vertex}_i, \texttt{@@Quad}_i^{supp}, \texttt{(R)}\ u_i^{supp}, \texttt{(R)}\ v_i^{supp}\right.$,
    i=1 , NbOfVertexOnSupportQuadrilaterals$\left.\right)$

12.2 bb File type for Store Solutions

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

- `nbsol` is a integer equal to the number of solutions.
- `nbv` is a integer equal to the number of vertex .
- $\texttt{U}_{ij}$ is a real equal the value of the i solution at vertex j on the associated mesh background if read file, generated if write file.

12.3 BB File Type for Store Solutions

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

- `n` is a integer equal to the number of solutions
- `typesol`$^{\texttt{k}}$, type of the solution number k, is

  - typesol[k] = 1 the solution k is scalar (1 value per vertex)
  - typesol[k] = 2 the solution k is vectorial (2 values per unknown)
  - typesol[k] = 3 the solution k is a 2×2 symmetric matrix (3 values per vertex)
  - typesol[k] = 4 the solution k is a 2×2 matrix (4 values per vertex)

- nbv is a integer equal to the number of vertices
- U_{ij}^k is a real equal to 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.

12.4 Metric File

A metric file can be of two types, isotropic or anisotropic.
the isotropic file is such that
nbv 1
$h_i \quad \forall i \in \{1, ..., \mathbf{nbv}\}$
where

- nbv is a integer equal to the number of vertices.
- h_i is the wanted mesh size near the vertex i on background mesh, the metric is $\mathcal{M}_i = h_i^{-2} Id$, where Id is the identity matrix.

The metric anisotrope
nbv 3
$a11_i, a21_i, a22_i \quad \forall i \in \{1, ..., \mathbf{nbv}\}$
where

- nbv is a integer equal to the number of vertices,
- $a11_i$, $a12_i$, $a22_i$ is metric $\mathcal{M}_i = \begin{pmatrix} a11_i & a12_i \\ a12_i & a22_i \end{pmatrix}$ which define the wanted mesh size in a vicinity of the vertex i such that h in direction $u \in \mathbb{R}^2$ is equal to $|u| / \sqrt{u \cdot \mathcal{M}_i \, u}$, where $\cdot$ is the dot product in $\mathbb{R}^2$, and $|\cdot|$ is the classical norm.

12.5 List of AM_FMT, AMDBA Meshes

The mesh is only composed of triangles and can be defined with the help of the following two integers and four arrays:

nbt is the number of triangles.

nbv is the number of vertices.

nu(1:3,1:nbt) is an integer array giving the three vertex numbers counterclockwise for each triangle.

`c(1:2,nbv)` is a real array giving the two coordinates of each vertex.

`refs(nbv)` is an integer array giving the reference numbers of the vertices.

`reft(nbv)` is an integer array giving the reference numbers of the triangles.

AM_FMT Files In fortran the `am_fmt` files are read as follows:

```
open(1,file='xxx.am_fmt',form='formatted',status='old')
  read (1,*) nbv,nbt
  read (1,*)  ((nu(i,j),i=1,3),j=1,nbt)
  read (1,*)  ((c(i,j),i=1,2),j=1,nbv)
  read (1,*)  ( reft(i),i=1,nbt)
  read (1,*)  ( refs(i),i=1,nbv)
close(1)
```

AM Files In fortran the `am` files are read as follows:

```
open(1,file='xxx.am',form='unformatted',status='old')
  read (1,*) nbv,nbt
  read (1)  ((nu(i,j),i=1,3),j=1,nbt),
&   ((c(i,j),i=1,2),j=1,nbv),
&   ( reft(i),i=1,nbt),
&   ( refs(i),i=1,nbv)
close(1)
```

AMDBA Files In fortran the `amdba` files are read as follows:

```
open(1,file='xxx.amdba',form='formatted',status='old')
  read (1,*) nbv,nbt
  read (1,*) (k,(c(i,k),i=1,2),refs(k),j=1,nbv)
  read (1,*) (k,(nu(i,k),i=1,3),reft(k),j=1,nbt)
close(1)
```

msh Files First, we add the notions of boundary edges

`nbbe` is the number of boundary edge.

`nube(1:2,1:nbbe)` is an integer array giving the two vertex numbers

`refbe(1:nbbe)` is an integer array giving the two vertex numbers

In fortran the `msh` files are read as follows:

```
open(1,file='xxx.msh',form='formatted',status='old')
  read (1,*) nbv,nbt,nbbe
  read (1,*) ((c(i,k),i=1,2),refs(k),j=1,nbv)
  read (1,*) ((nu(i,k),i=1,3),reft(k),j=1,nbt)
  read (1,*) ((ne(i,k),i=1,2), refbe(k),j=1,nbbe)
close(1)
```

ftq Files In fortran the `ftq` files are read as follows:

```
open(1,file='xxx.ftq',form='formatted',status='old')
 read (1,*) nbv,nbe,nbt,nbq
 read (1,*) (k(j),(nu(i,j),i=1,k(j)),reft(j),j=1,nbe)
 read (1,*) ((c(i,k),i=1,2),refs(k),j=1,nbv)
close(1)
```

where if `k(j) = 3` then the element *j* is a triangle and if `k = 4` the the element *j* is a quadrilateral.

Chapter 13

Add new finite element

13.1 Some notation

For a function $\boldsymbol{f}$ taking value in $\mathbb{R}^N$, $N = 1, 2, \cdots$, we define the finite element approximation $\Pi_h \boldsymbol{f}$ of $\boldsymbol{f}$. Let us denote the number of the degrees of freedom of the finite element by $NbDoF$. Then the i-th base $\boldsymbol{\omega}_i^K$ ($i = 0, \cdots, NbDoF - 1$) of the finite element space has the j-th component ω_{ij}^K for $j = 0, \cdots, N - 1$.
The operator Π_h is called the interpolator of the finite element. We have the identity $\boldsymbol{\omega}_i^K = \Pi_h \boldsymbol{\omega}_i^K$.
Formally, the interpolator Π_h is constructed by the following formula:

$$\Pi_h \boldsymbol{f} = \sum_{k=0}^{\texttt{kPi}-1} \alpha_k \boldsymbol{f}_{j_k}(P_{p_k}) \boldsymbol{\omega}_{i_k}^K \tag{13.1}$$

where P_p is a set of $npPi$ points,
In the formula (13.1), the list p_k, j_k, i_k depend just on the type of finite element (not on the element), but the coefficient α_k can be depending on the element.

Example 1: classical scalar Lagrange finite element, first we have `kPi = npPi = NbOfNode` and

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

Example 2: The Raviart-Thomas finite element:

$$RT0_h = \{\mathbf{v} \in H(div)/\forall K \in \mathcal{T}_h \quad \mathbf{v}_{|K}(x, y) = \begin{vmatrix} \alpha_K \\ \beta_K \end{vmatrix} + \gamma_K \begin{vmatrix} x \\ y \end{vmatrix} \} \tag{13.2}$$

The degree of freedom are the flux throw an edge e of the mesh, where the flux of the function $\mathbf{f} : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$ is $\int_e \mathbf{f}.n_e$, n_e is the unit normal of edge e (this implies a orientation of all the edges of the mesh, for example 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 formula with one point, the middle point of the edge. Consider a triangle T with three vertices $(\mathbf{a}, \mathbf{b}, \mathbf{c})$. Let denote the vertices numbers by i_a, i_b, i_c, and define the three edge vectors $\mathbf{e}^0, \mathbf{e}^1, \mathbf{e}^2$ by $sgn(i_b - i_c)(\mathbf{b} - \mathbf{c})$, $sgn(i_c - i_a)(\mathbf{c} - \mathbf{a})$, $sgn(i_a - i_b)(\mathbf{a} - \mathbf{b})$,
The three basis functions are:

$$\boldsymbol{\omega}_0^K = \frac{sgn(i_b - i_c)}{2|T|}(x - a), \quad \boldsymbol{\omega}_1^K = \frac{sgn(i_c - i_a)}{2|T|}(x - b), \quad \boldsymbol{\omega}_2^K = \frac{sgn(i_a - i_b)}{2|T|}(x - c), \tag{13.3}$$

where $|T|$ is the area of the triangle T.
So we have $N = 2$, `kPi = 6`; `npPi = 3`; and:

- $P_p = \left\{ \frac{\mathbf{b+c}}{2}, \frac{\mathbf{a+c}}{2}, \frac{\mathbf{b+a}}{2} \right\}$
- $\alpha_0 = -\mathbf{e}_2^0, \alpha_1 = \mathbf{e}_1^0, \alpha_2 = -\mathbf{e}_2^1, \alpha_3 = \mathbf{e}_1^1, \alpha_4 = -\mathbf{e}_2^2, \alpha_5 = \mathbf{e}_1^2$ (effectively, the vector $(-\mathbf{e}_2^m, \mathbf{e}_1^m)$ is orthogonal to the edge $\mathbf{e}^m = (e_1^m, e_2^m)$ with a length equal to the side of the edge or equal to $\int_{e^m} 1$).
- $i_k = \{0, 0, 1, 1, 2, 2\}$,
- $p_k = \{0, 0, 1, 1, 2, 2\}$, $j_k = \{0, 1, 0, 1, 0, 1, 0, 1\}$.

13.2 Which class of add

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

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

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 (13.1)
       3,      //    number npPi of integration point to build interpolator
       0       //    an array to store the coef α_k to build interpolator
               //    here this array is no constant so we have
               //    to rebuilt for each element.
       )
  {
    const R2 Pt[] = { R2(0.5,0.5), R2(0.0,0.5), R2(0.5,0.0) };
                                                  //    the set of Point in K̂
    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 (13.1)

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

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

} ;
```

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

This array is:

```
int TypeOfFE_RTortho::Data[]={

              //    for each df 0,1,3 :
        3,4,5,//    the support of the node of the df
        0,0,0,//    the number of the df on the node
        0,1,2,//    the node of the df
        0,0,0,//    the df come from which FE (generally 0)
        0,1,2,//    which are de df on sub FE
        0,0 }; //    for each component j=0,N-1 it give the sub FE associated
```

where the support is a number $0,1,2$ for vertex support, $3,4,5$ for edge support, and finaly 6 for element support.
The function to defined the function 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 $f(P), dx(f)(P), dy(f)(P), \ldots$ $i \in [0,$ `last_operatortype`$[$. Remark for optimization, this value is computed only if $whatd[k]$ is true, and the numbering is defined with

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

The shape function :

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

                                                          //    ------------
  if (whatd[op_id])                                 //    value of the function
```

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

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

      f1[2] =  (P.x-C.x)*a2;
      f0[2] = -(P.y-C.y)*a2;
     }
                                                     //    ----------------
     if (whatd[op_dx])                          //    value of the dx of function
     {
      assert(val.K()>op_dx);
      val(0,1,op_dx) =  a0;
      val(1,1,op_dx) =  a1;
      val(2,1,op_dx) =  a2;
      }
     if (whatd[op_dy])
     {
      assert(val.K()>op_dy);
      val(0,0,op_dy) =  -a0;
      val(1,0,op_dy) =  -a1;
      val(2,0,op_dy) =  -a2;
     }

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

}
```

The function to defined the coefficient α_k:

```
void TypeOfFE_RT::Pi_h_alpha(const baseFElement & K,KN_<double> & v) const
{
  const Triangle & T(K.T);

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

Now , we just need to add a new key work in `FreeFem++`, Two way, with static or dynamic link so at the end of the file, we add :

With dynamic link is very simple (see section C of appendix), just add before the end of `FEM2d namespace` add:

```
static TypeOfFE_RTortho The_TypeOfFE_RTortho;                                   //
static AddNewFE("RT0Ortho", The_TypeOfFE_RTortho);
}                                                                    //    FEM2d namespace
```

Try with "./load.link" command in `examples++-load/` and see `BernardiRaugel.cpp` or `Morley.cpp` new finite element examples.

Otherwise with static link (for expert only), add

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

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

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

```
               //    correct Problem 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 line 1
   init_FE_ADD();                                                 //    new line 2
}
```

and now you have to change the makefile.
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 1.6 2005-12-27 16:32:24 hecht Exp $

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

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

and do in the `freefem++` root directory

```
  autoreconf
./reconfigure
make
```

For codewarrior compilation add the file in the project an remove the flag in panal PPC linker FreeFEm++ Setting Dead-strip Static Initializition Code Flag.

Appendix A

Table of Notations

Here mathematical expressions and corresponding `FreeFem++` commands are noted.

A.1 Generalities

δ_{ij} Kronecker delta (0 if $i \neq j$, 1 if $i = j$ for integers i, j)

$\forall$ for all

$\exists$ there exist

i.e. that is

PDE partial differential equation (with boundary conditions)

$\emptyset$ the empty set

$\mathbb{N}$ the set of integers ($a \in \mathbb{N} \Leftrightarrow$ `int a`); "`int`" means *long integer* inside `FreeFem++`

$\mathbb{R}$ the set of real numbers ($a \in \mathbb{R} \Leftrightarrow$ `real a`) ;*double* inside `FreeFem++`

$\mathbb{C}$ the set of complex numbers ($a \in \mathbb{C} \Leftrightarrow$ `complex a`); *complex¡double¿*

$\mathbb{R}^d$ d-dimensional Euclidean space

A.2 Sets, Mappings, Matrices, Vectors

Let E, F, G be three sets and A subset of E.

$\{x \in E | P\}$ the subset of E consisting of the elements possessing the property P

$E \cup F$ the set of elements belonging to E or F

$E \cap F$ the set of elements belonging to E and F

$E \setminus A$ the set $\{x \in E | x \notin A\}$

$E + F$ $E \cup F$ with $E \cap F = \emptyset$

$E \times F$ the cartesian product of E and F

E^n the n-th power of E ($E^2 = E \times E$, $E^n = E \times E^{n-1}$)

$f : E \to F$ the mapping form E into F, i.e., $E \ni x \mapsto f(x) \in F$

I_E **or** I the identity mapping in E,i.e., $I(x) = x \quad \forall x \in E$

$f \circ g$ for $f : F \to G$ and $g : E \to F$, $E \ni x \mapsto (f \circ g)(x) = f(g(x)) \in G$ (see Section 4.6)

$f|_A$ the restriction of $f : E \to F$ to the subset A of E

$\{a_k\}$ column vector with components a_k

(a_k) row vector with components a_k

$(a_k)^T$ denotes the transpose of a matrix (a_k), and is $\{a_k\}$

$\{a_{ij}\}$ matrix with components a_{ij}, and $(a_{ij})^T = (a_{ji})$

A.3 Numbers

For two real numbers a, b

$[a, b]$ is the interval $\{x \in \mathbb{R} |\ a \leq x \leq b\}$

$]a, b]$ is the interval $\{x \in \mathbb{R} |\ a < x \leq b\}$

$[a, b[$ is the interval $\{x \in \mathbb{R} |\ a \leq x < b\}$

$]a, b[$ is the interval $\{x \in \mathbb{R} |\ a < x < b\}$

A.4 Differential Calculus

$\partial f/\partial x$ the partial derivative of $f : \mathbb{R}^d \to \mathbb{R}$ with respect to x (`dx(f)`)

∇f the gradient of $f : \Omega \to \mathbb{R}$,i.e., $\nabla f = (\partial f/\partial x,\ \partial f/\partial y)$

div$\boldsymbol{f}$ **or** $\nabla.\boldsymbol{f}$ the divergence of $\boldsymbol{f} : \Omega \to \mathbb{R}^d$, i.e., div$\boldsymbol{f} = \partial f_1/\partial x + \partial f_2/\partial y$

Δf the Laplacian of $f : \Omega \to \mathbb{R}$, i.e., $\Delta f = \partial^2 f/\partial x^2 + \partial^2 f/\partial y^2$

A.5 Meshes

Ω usually denotes a domain on which PDE is defined

Γ denotes the boundary of Ω,i.e., $\Gamma = \partial\Omega$ (keyword **border**, see Section 5.1.2)

$\mathcal{T}_h$ the triangulation of Ω, i.e., the set of triangles T_k, where h stands for mesh size (keyword **mesh**, **buildmesh**, see Section 5)

n_t the number of triangles in $\mathcal{T}_h$ (get by `Th.nt`, see "mesh.edp")

Ω_h denotes the approximated domain $\Omega_h = \cup_{k=1}^{n_t} T_k$ of Ω. If Ω is polygonal domain, then it will be $\Omega = \Omega_h$

Γ_h the boundary of Ω_h

n_v the number of vertices in $\mathcal{T}_h$ (get by `Th.nv`)

$[q^i q^j\ ]$ the segment connecting q^i and q^j

$q^{k_1}, q^{k_2}, q^{k_3}$ the vertices of a triangle T_k with anti-clock direction (get the coordinate of q^{k_j} by (`Th[k-1][j-1].x, Th[k-1][j-1].y`))

I_Ω the set $\{i \in \mathbb{N} |\ q^i \notin \Gamma_h\}$

A.6 Finite Element Spaces

$L^2(\Omega)$ the set $\left\{ w(x,y) \,\middle|\, \int_\Omega |w(x,y)|^2 \mathrm{d}x\mathrm{d}y < \infty \right\}$

$$\text{norm: } \|w\|_{0,\Omega} = \left(\int_\Omega |w(x,y)|^2 \mathrm{d}x\mathrm{d}y \right)^{1/2}$$

$$\text{scalar product: } (v,w) = \int_\Omega vw$$

$H^1(\Omega)$ the set $\left\{ w \in L^2(\Omega) \,\middle|\, \int_\Omega \left(|\partial w/\partial x|^2 + |\partial w/\partial y|^2 \right) \mathrm{d}x\mathrm{d}y < \infty \right\}$

$$\text{norm: } \|w\|_{1,\Omega} = \left(\|w\|_{0,\Omega}^2 + \|\nabla u\|_{0.\Omega}^2 \right)^{1/2}$$

$H^m(\Omega)$ the set $\left\{ w \in L^2(\Omega) \,\middle|\, \int_\Omega \frac{\partial^{|\alpha|} w}{\partial x^{\alpha_1} \partial y^{\alpha_2}} \in L^2(\Omega) \quad \forall \alpha = (\alpha_1, \alpha_2) \in \mathbb{N}^2,\ |\alpha| = \alpha_1 + \alpha_2 \right\}$

$$\text{scalar product: } (v,w)_{1,\Omega} = \sum_{|\alpha| \le m} \int_\Omega D^\alpha v D^\alpha w$$

$H_0^1(\Omega)$ the set $\left\{w \in H^1(\Omega) \mid u = 0 \quad \text{on } \Gamma\right\}$

$L^2(\Omega)^2$ denotes $L^2(\Omega) \times L^2(\Omega)$, and also $H^1(\Omega)^2 = H^1(\Omega) \times H^1(\Omega)$

V_h denotes the finite element space created by " `fespace` Vh(Th,*)" in `FreeFem++` (see Section 6 for "*")

$\Pi_h f$ the projection of the function f into V_h (" `func f=x^2*y^3; Vh v = f;`" means `v` = Π_h `f`)

$\{v\}$ for FE-function v in V_h means the column vector $(v_1, \cdots, v_M)^T$ if $v = v_1\phi_1 + \cdots + v_M\phi_M$, which is shown by " `fespace Vh(Th,P2); Vh v; cout << v[] << endl;`"

Appendix B

Grammar

B.1 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:
  no_ternary_expr
| no_ternary_expr '?' no_set_expr ':' no_set_expr ;

no_ternary_expr:
          unary_expr
        | no_ternary_expr '*' no_ternary_expr
        | no_ternary_expr '.*' no_ternary_expr
        | no_ternary_expr './' no_ternary_expr
        | no_ternary_expr '/' no_ternary_expr
        | no_ternary_expr '%' no_ternary_expr
        | no_ternary_expr '+' no_ternary_expr
        | no_ternary_expr '-' no_ternary_expr
        | no_ternary_expr '<<' no_ternary_expr
        | no_ternary_expr '>>' no_ternary_expr
        | no_ternary_expr '&' no_ternary_expr
        | no_ternary_expr '&&' no_ternary_expr
        | no_ternary_expr '|' no_ternary_expr
        | no_ternary_expr '||' no_ternary_expr
        | no_ternary_expr '<' no_ternary_expr
        | no_ternary_expr '<=' no_ternary_expr
        | no_ternary_expr '>' no_ternary_expr
        | no_ternary_expr '>=' no_ternary_expr
```

```
        | no_ternary_expr '==' no_ternary_expr
        | no_ternary_expr '!=' no_ternary_expr ;

sub_script_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
        |   sub_script_expr
        |   parameters ',' FESPACE
        |   parameters ',' no_set_expr
        |   parameters ',' id '=' no_set_expr ;

array:   no_comma_expr
       | array ',' no_comma_expr ;


unary_expr:
    pow_expr
  | unop  pow_expr %prec UNARY ;

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

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

B.2 The Types of the languages, and cast

B.3 All the operators

```
- CG,  type :<TypeSolveMat>
- Cholesky,  type :<TypeSolveMat>
- Crout,  type :<TypeSolveMat>
- GMRES,  type :<TypeSolveMat>
- LU,  type :<TypeSolveMat>
- LinearCG,  type :<Polymorphic>   operator() :
 (    <long> :   <Polymorphic>, <KN<double> *>, <KN<double> *> )

- N,  type :<Fem2D::R3>
- NoUseOfWait,  type :<bool *>
- P,  type :<Fem2D::R3>
- P0,  type :<Fem2D::TypeOfFE>
- P1,  type :<Fem2D::TypeOfFE>
- P1nc,  type :<Fem2D::TypeOfFE>
- P2,  type :<Fem2D::TypeOfFE>
- RT0,  type :<Fem2D::TypeOfFE>
- RTmodif,  type :<Fem2D::TypeOfFE>
- abs,  type :<Polymorphic>  operator() :
 (    <double> :   <double> )

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

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

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

- append,  type :<std::ios_base::openmode>
- asin,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )

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

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

- atan2,  type :<Polymorphic>   operator() :
 (    <double> :   <double>, <double> )

- atanh,  type :<Polymorphic>   operator() :
 (    <double> :   <double> )
```

```
- buildmesh,  type :<Polymorphic>   operator() :
 (    <Fem2D::Mesh> :   <E_BorderN> )

- buildmeshborder,  type :<Polymorphic>   operator() :
 (    <Fem2D::Mesh> :   <E_BorderN> )

- cin,  type :<istream>
- clock,  type :<Polymorphic>
 (    <double> :   )

- conj,  type :<Polymorphic>   operator() :
 (    <complex> :   <complex> )

- convect,  type :<Polymorphic>   operator() :
 (    <double> :   <E_Array>, <double>, <double> )

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

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

- cout,  type :<ostream>
- dumptable,  type :<Polymorphic>   operator() :
 (    <ostream> :   <ostream> )

- dx,  type :<Polymorphic>   operator() :
 (    <LinearComb<MDroit, C_F0>> :   <LinearComb<MDroit, C_F0>> )
 (    <double> :   <std::pair<FEbase<double> *, int>> )
 (    <LinearComb<MGauche, C_F0>> :   <LinearComb<MGauche, C_F0>> )

- dy,  type :<Polymorphic>   operator() :
 (    <LinearComb<MDroit, C_F0>> :   <LinearComb<MDroit, C_F0>> )
 (    <double> :   <std::pair<FEbase<double> *, int>> )
 (    <LinearComb<MGauche, C_F0>> :   <LinearComb<MGauche, C_F0>> )

- endl,  type :<char>
- exec,  type :<Polymorphic>   operator() :
 (    <long> :   <string> )

- exit,  type :<Polymorphic>  operator() :
 (    <long> :   <long> )

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

```
- false,  type :<bool>
- imag,  type :<Polymorphic>   operator() :
 (    <double> :   <complex> )

- int1d,  type :<Polymorphic>   operator() :
 (    <CDomainOfIntegration> :   <Fem2D::Mesh>... )

- int2d,  type :<Polymorphic>   operator() :
 (    <CDomainOfIntegration> :   <Fem2D::Mesh>... )

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

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

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

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

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

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

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

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

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

- nuTriangle,  type :<long>
```

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

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

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

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

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

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

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

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

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

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

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

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

```
  (    <Fem2D::Mesh> :   <long>, <long>, <E_Array> )

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

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

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

Appendix C

Dynamical link

Now, it's possible to add built-in functionnalites in FreeFem++ under the three environnents Linux, Windows and MacOS X 10.3 or newer. It is agood idea to, first try the example load.edp in directory example++-load.
But unfortunately, you need to install a c++ compiler (generally g++/gcc compiler) to compile your function.

Windows Install the cygwin environnent or the mingw

MacOs Install the developer tools xcode on the apple DVD

Linux/Unix Install the correct compiler (gcc for instance)

Now, assume , you are in a shell window (a cygwin window under Windows) in the directory example++-load. Remark that in the sub directory include they are all the FreeFem++ include file to make the link with FreeFem++.

Note C.1 *If you try to load dynamically a file with command* `load "xxx"`

- *Under unix (Linux or MacOs), the file* `xxx.so` *to be loaded must be either first in the search directory of routine* `dlopen` *(see the environment variable* `$LD_LIBRARY_PATH` *or in the current directory, and the suffix* `".so"` *or the prefix* `"./"` *is automaticaly added.*
- *Under Windows, the file* `xxx.dll` *to be loaded must be in the* `loadLibary` *search directory which includes the directory of the application,*

The compilation of your module: the script `./load.link` compiles and makes the link with FreeFem++, but be careful, the script has no way to known if you try to compile for a pure Windows environment or for a cygwin environment so to build the load module under cygwin you must add the `-cygwin` parameter.

C.1 A first example myfunction.cpp

The following defines a new function call myfunction with no parameter, but using the x, y current value.

```
#include  <iostream>
```

```
#include  <cfloat>
using namespace std;
#include "error.hpp"
#include "AFunction.hpp"
#include "rgraph.hpp"
#include "RNM.hpp"
#include "fem.hpp"
#include "FESpace.hpp"
#include "MeshPoint.hpp"

using namespace Fem2D;
double myfunction(Stack stack)
{
                                                    //    to get FreeFem++ data
  MeshPoint &mp= *MeshPointStack(stack); //    the struct to get x,y, normal ,
value
  double x= mp.P.x;                                 //    get the current x value
  double y= mp.P.y;                                 //    get the current y value
                                 //    cout << "x = " << x << " y=" << y << endl;
  return sin(x)*cos(y);
}
```

Now the Problem is to build the link with `FreeFem++`, to do that we need two classes, one to call the function `myfunction`
All `FreeFem++` evaluable expression must be a struct/class `C++` which derive from `E_F0`. By default this expression does not depend of the mesh position, but if they derive from `E_F0mps` the expression depends of the mesh position, and for more details see [11].

```
                                    //    A class build the link with FreeFem++
                           //    generaly this class are already in AFunction.hpp
              //    but unfortunatly, I have no simple function with no parameter
                                      //    in FreeFem++ depending of the mesh,
template<class R>
class  OneOperator0s : public OneOperator {

                            //    the class to defined a evaluated a new function
                          //    It must devive from E_F0 if it is mesh independent
                                     //    or from E_F0mps if it is mesh dependent
  class E_F0_F :public  E_F0mps { public:
    typedef  R (*func)(Stack stack) ;
    func f;                           //    the pointeur to the fnction myfunction
    E_F0_F(func ff)  : f(ff) {}
                                     //    the operator evaluation in FreeFem++
    AnyType operator()(Stack stack)  const {return SetAny<R>( f(stack)) ;}

  };

  typedef  R (*func)(Stack ) ;
  func  f;
public:
                           //    the function which build the FreeFem++ byte code
  E_F0 * code(const basicAC_F0 & ) const { return  new E_F0_F(f);}
                //    the constructor to say ff is a function without parameter
                                                    //    and returning a R
  OneOperator0s(func  ff): OneOperator(map_type[typeid(R).name()]),f(ff){}
```

```
};
```

To finish we must add this new function in FreeFem++ table , to do that include :

```
class Init { public:  Init();  };

Init init;

Init::Init(){
  Global.Add("myfunction","(",new OneOperator0s<double>(myfunction));
}
```

It will be called automatically at load module time.

To compile and link, do

```
Brochet% ./load.link myfunction.cpp
g++ -c -g -Iinclude myfunction.cpp
g++ -bundle -undefined dynamic_lookup -g myfunction.o -o ./myfunction.dylib
```

To, try the simple example under Linux or MacOS, do

```
Brochet% FreeFem++-nw load.edp
-- FreeFem++ v 1.4800028 (date Tue Oct  4 11:56:46 CEST 2005)
 file : load.edp
 Load: lg_fem lg_mesh eigenvalue  UMFPACK
    1 :                                    //    Example of dynamic function load
    2 :                                    //    --------------------------------
    3 :                          //    Id : freefem++doc.tex,v1.642008/02/0414 : 07 : 38hechtExp
    4 :
    5 :  load "myfunction"
     lood: myfunction

load: dlopen(./myfunction) = 0xb01cc0

    6 :  mesh Th=square(5,5);
    7 :  fespace Vh(Th,P1);
    8 :  Vh uh=myfunction();                      //    warning do not forget ()
    9 :  cout << uh[].min << " " << uh[].max << endl;
   10 :  sizestack + 1024 =1240  ( 216 )

 -- square mesh : nb vertices  =36 ,  nb triangles = 50 ,  nb boundary edges 20
   Nb of edges on Mortars  = 0
   Nb of edges on Boundary = 20, neb = 20
 Nb Of Nodes = 36
 Nb of DF = 36
0 0.841471
times: compile 0.05s, execution -3.46945e-18s
 CodeAlloc : nb ptr  1394,  size :71524
Bien: On a fini Normalement
```

Under Windows, launch FreeFem++ with the mouse on the example.

C.2 Example Discrete Fast Fourier Transform

This will add FFT to `FreeFem++`,taken from `http://www.fftw.org/`. To download and install under `download/include` just go in `download/fftw` and try`make`.

The 1D dfft (fast discret fourier transform) for a simple array f of size n is defined by the following formula

$$\texttt{dfft}(f,\varepsilon)_k = \sum_{j=0}^{n-1} f_i e^{\varepsilon 2\pi i k j/n}$$

The 2D DFT for an array of size $N = n \times m$ is

$$\texttt{dfft}(f,m,\varepsilon)_{k+nl} = \sum_{j'=0}^{m-1}\sum_{j=0}^{n-1} f_{i+nj} e^{\varepsilon 2\pi i (kj/n+lj'/m)}$$

Remark: the value n is given by $size(f)/m$, and the numbering is row-major order.
So the classical discrete DFT is $\hat{f} = \texttt{dfft}(f,-1)/\sqrt{n}$ and the reverse dFT $f = \texttt{dfft}(\hat{f},1)/\sqrt{n}$

Remark: the 2D Laplace operator is

$$f(x,y) = 1/\sqrt{N}\sum_{j'=0}^{m-1}\sum_{j=0}^{n-1} \hat{f}_{i+nj} e^{\varepsilon 2\pi i (xj+yj')}$$

and we have

$$f_{k+nl} = f(k/n, l/m)$$

So

$$\widehat{\Delta f_{kl}} = -((2\pi)^2((\tilde{k})^2 + (\tilde{l})^2))\widehat{f_{kl}}$$

where $\tilde{k} = k$ if $k \le n/2$ else $\tilde{k} = k - n$ and $\tilde{l} = l$ if $l \le m/2$ else $\tilde{l} = l - m$.
And to get a real function we need all symetric modes around to zero, so n and m must be odd.
To compile and make a new library

```
% ./load.link  dfft.cpp ../download/install/lib/libfftw3.a -I../download/install/include
export MACOSX_DEPLOYMENT_TARGET=10.3
g++ -c -Iinclude -I../download/install/include dfft.cpp
g++ -bundle -undefined dynamic_lookup dfft.o -o ./dfft.dylib ../download/install/lib/libfftw3.a
```

To test ,

```
-- FreeFem++ v 1.4800028 (date Mon Oct 10 16:53:28 EEST 2005)
 file : dfft.edp
 Load: lg_fem cadna lg_mesh eigenvalue  UMFPACK
    1 :                                    //    Example of dynamic function load
    2 :                                    //    --------------------------------
    3 :                          //    Id : freefem++doc.tex, v1.642008/02/0414 : 07 : 38hechtExp
    4 :                                      //    Discret Fast Fourier Transform
    5 :                                     //    --------------------------------
    6 :  load "dfft" lood: init dfft

load: dlopen(dfft.dylib) = 0x2b0c700

    7 :
    8 : int nx=32,ny=16,N=nx*ny;
```

```
    9 :        //    warning the fourier space is not exactly the unite square
                                                //    due to periodic condition
   10 : mesh Th=square(nx-1,ny-1,[(nx-1)*x/nx,(ny-1)*y/ny]);
   11 :               //    warring the numbering is of the vertices (x,y) is
   12 :                                          //    given by i = x/nx + nx * y/ny
   13 :
   14 : fespace Vh(Th,P1);
   15 :
   16 : func f1 = cos(2*x*2*pi)*cos(3*y*2*pi);
   17 : Vh<complex> u=f1,v;
   18 : Vh w=f1;
   19 :
   20 :
   21 : Vh  ur,ui;
   22 :    //    in dfft the matrix n,m is in row-major order ann array n,m is
   23 :          //    store j + m* i  ( the transpose of the square numbering )
   24 :  v[]=dfft(u[],ny,-1);
   25 :  u[]=dfft(v[],ny,+1);
   26 :  u[] /= complex(N);
   27 :  v = f1-u;
   28 : cout << " diff = "<< v[].max << " " <<  v[].min << endl;
   29 : assert( norm(v[].max) < 1e-10 &&  norm(v[].min) < 1e-10) ;
   30 :  //    ------- a more hard example ----
   31 :  //    Lapacien en FFT
   32 :  //    -Δu = f  with biperiodic condition
   33 : func f = cos(3*2*pi*x)*cos(2*2*pi*y);                                  //
   34 : func ue =  +(1./(square(2*pi)*13.))*cos(3*2*pi*x)*cos(2*2*pi*y);     //

   35 : Vh<complex> ff = f;
   36 : Vh<complex> fhat;
   37 : fhat[] = dfft(ff[],ny,-1);
   38 :
   39 : Vh<complex> wij;
   40 :  //    warning in fact we take mode between -nx/2, nx/2 and -ny/2,ny/2
   41 :                                        //    thank to the operator ?:
   42 : wij = square(2.*pi)*(square(( x<0.5?x*nx:(x-1)*nx))
          + square((y<0.5?y*ny:(y-1)*ny)));
   43 : wij[][0] = 1e-5;                                 //    to remove div / 0
   44 : fhat[] = fhat[]./ wij[];                                              //
   45 : u[]=dfft(fhat[],ny,1);
   46 : u[] /= complex(N);
   47 : ur = real(u);                                              //    the solution
   48 : w = real(ue);                                        //    the exact solution
   49 : plot(w,ur,value=1 ,cmm=" ue   ", wait=1);
   50 : w[] -= ur[];                                                 //    array sub
   51 : real err= abs(w[].max)+abs(w[].min) ;
   52 : cout << " err = " << err << endl;
   53 : assert( err  < 1e-6);
   54 :  sizestack + 1024 =3544  ( 2520 )
----------CheckPtr:-----init execution ------ NbUndelPtr  2815  Alloc: 111320  NbPtr
6368

 -- square mesh : nb vertices  =512 ,  nb triangles = 930 ,  nb boundary edges 92
   Nb of edges on Mortars  = 0
   Nb of edges on Boundary = 92, neb = 92
 Nb Of Nodes = 512
```

```
 Nb of DF = 512
0x2d383d8 -1 16 512 n: 16 m:32
 dfft 0x402bc08 = 0x4028208 n = 16 32 sign = -1
 --- --- ---0x2d3ae08 1 16 512 n: 16 m:32
 dfft 0x4028208 = 0x402bc08 n = 16 32 sign = 1
 --- --- --- diff = (8.88178e-16,3.5651e-16) (-6.66134e-16,-3.38216e-16)
0x2d3cfb8 -1 16 512 n: 16 m:32
 dfft 0x402de08 = 0x402bc08 n = 16 32 sign = -1
 --- --- ---0x2d37ff8 1 16 512 n: 16 m:32
 dfft 0x4028208 = 0x402de08 n = 16 32 sign = 1
 --- --- --- err = 3.6104e-12
times: compile 0.13s, execution 2.05s
----------CheckPtr:-----end execution -- ------ NbUndelPtr  2815  Alloc: 111320
NbPtr 26950
 CodeAlloc : nb ptr  1693,  size :76084
Bien: On a fini Normalement
               CheckPtr:Nb of undelete pointer is 2748 last 114
               CheckPtr:Max Memory used    228.531 kbytes  Memory undelete 105020
```

C.3 Load Module for Dervieux' P0-P1 Finite Volume Method

the associed edp file is `examples++-load/convect_dervieux.edp`

```
                                          //    Implementation of P1-P0 FVM-FEM
   //    ------------------------------------------------------------------------
                                 //     Id : freefem + +doc.tex, v1.642008/02/0414 : 07 : 38hechtExp
  //    compile and link with ./load.link mat_dervieux.cpp (i.e.  the file name
without .cpp)
#include  <iostream>
#include  <cfloat>
#include  <cmath>
using namespace std;
#include "error.hpp"
#include "AFunction.hpp"
#include "rgraph.hpp"
#include "RNM.hpp"
                                                  //    remove problem of include
#undef  HAVE_LIBUMFPACK
#undef HAVE_CADNA
#include "MatriceCreuse_tpl.hpp"
#include "MeshPoint.hpp"
#include "lgfem.hpp"
#include "lgsolver.hpp"
#include "problem.hpp"


class MatrixUpWind0 :  public E_F0mps { public:
  typedef Matrice_Creuse<R> * Result;
  Expression emat,expTh,expc,expu1,expu2;
  MatrixUpWind0(const basicAC_F0 & args)
  {

    args.SetNameParam();
    emat =args[0];                                 //    the matrix expression
```

```
    expTh= to<pmesh>(args[1]);          //    a the expression to get the mesh
    expc = CastTo<double>(args[2]);    //    the expression to get c (must be a
double)
                                                //    a array expression [ a, b]
    const E_Array * a= dynamic_cast<const E_Array*>((Expression) args[3]);
    if (a->size() != 2) CompileError("syntax:  MatrixUpWind0(Th,rhi,[u1,u2])");
    int err =0;
    expu1= CastTo<double>((*a)[0]);     //    fist exp of the array (must be a
double)
    expu2= CastTo<double>((*a)[1]);    //    second exp of the array (must be a
double)

  }

  ~MatrixUpWind0()
  {
  }

  static ArrayOfaType  typeargs()
  { return  ArrayOfaType(atype<Matrice_Creuse<R>*>(),
    atype<pmesh>(),atype<double>(),atype<E_Array>());}
  static  E_F0 * f(const basicAC_F0 & args){ return new MatrixUpWind0(args);}
  AnyType operator()(Stack s) const ;

};

int   fvmP1P0(double q[3][2], double u[2],double c[3], double a[3][3], double where[3]
)
{                                  //    computes matrix a on a triangle for the
Dervieux FVM
  for(int i=0;i<3;i++) for(int j=0;j<3;j++) a[i][j]=0;

    for(int i=0;i<3;i++){
        int ip = (i+1)%3, ipp =(ip+1)%3;
        double unL =-((q[ip][1]+q[i][1]-2*q[ipp][1])*u[0]
                -(q[ip][0]+q[i][0]-2*q[ipp][0])*u[1])/6;
        if(unL>0) { a[i][i] += unL; a[ip][i]-=unL;}
            else{ a[i][ip] += unL; a[ip][ip]-=unL;}
        if(where[i]&&where[ip]){                  //    this is a boundary edge
            unL=((q[ip][1]-q[i][1])*u[0] -(q[ip][0]-q[i][0])*u[1])/2;
            if(unL>0) { a[i][i]+=unL; a[ip][ip]+=unL;}
        }
    }
  return 1;
}

                                                  //    the evaluation routine
AnyType MatrixUpWind0::operator()(Stack stack) const
{
  Matrice_Creuse<R> * sparse_mat =GetAny<Matrice_Creuse<R>* >((*emat)(stack));
  MatriceMorse<R> * amorse =0;
  MeshPoint *mp(MeshPointStack(stack)) , mps=*mp;
  Mesh * pTh = GetAny<pmesh>((*expTh)(stack));
  ffassert(pTh);
  Mesh & Th (*pTh);
  {
```

```
    map< pair<int,int>, R> Aij;
    KN<double> cc(Th.nv);
    double infini=DBL_MAX;
    cc=infini;
    for (int it=0;it<Th.nt;it++)
      for (int iv=0;iv<3;iv++)
    {
      int i=Th(it,iv);
      if ( cc[i]==infini) {                                  //    if nuset the set
        mp->setP(&Th,it,iv);
        cc[i]=GetAny<double>((*expc)(stack));
      }
    }

    for (int k=0;k<Th.nt;k++)
      {
    const Triangle & K(Th[k]);
    const Vertex & A(K[0]), &B(K[1]),&C(K[2]);
    R2 Pt(1./3.,1./3.);
    R u[2];
    MeshPointStack(stack)->set(Th,K(Pt),Pt,K,K.lab);
    u[0] = GetAny< R>( (*expu1)(stack) ) ;
    u[1] = GetAny< R>( (*expu2)(stack) ) ;

    int ii[3] ={  Th(A), Th(B),Th(C)};
    double q[3][2]= { { A.x,A.y} ,{B.x,B.y},{C.x,C.y} } ;     //    coordinates
of 3 vertices (input)
    double c[3]={cc[ii[0]],cc[ii[1]],cc[ii[2]]};
    double a[3][3], where[3]={A.lab,B.lab,C.lab};
    if (fvmP1P0(q,u,c,a,where) )
      {
        for (int i=0;i<3;i++)
          for (int j=0;j<3;j++)
        if (fabs(a[i][j]) >= 1e-30)
          { Aij[make_pair(ii[i],ii[j])]+=a[i][j];
            }
      }
      }
    amorse=  new MatriceMorse<R>(Th.nv,Th.nv,Aij,false);
  }
  sparse_mat->pUh=0;
  sparse_mat->pVh=0;
  sparse_mat->A.master(amorse);
  sparse_mat->typemat=(amorse->n == amorse->m) ? TypeSolveMat(TypeSolveMat::GMRES)
: TypeSolveMat(TypeSolveMat::NONESQUARE);     //    none square matrice (morse)
  *mp=mps;

  if(verbosity>3) { cout << "  End Build MatrixUpWind : " << endl;}

  return sparse_mat;
}


class Init { public:
  Init();
};
```

```
Init init;
Init::Init()
  {
    cout << " lood: init Mat Chacon " << endl;
    Global.Add("MatUpWind0","(", new OneOperatorCode<MatrixUpWind0 >( ));
  }
```

C.4 Add a new finite element

First read the section 13 of the appendix, we add two new finite elements examples in the directory `examples++-load`.

Bernardi Raugel The Bernardi-Raugel finite element is make to solve Navier Stokes equation in u, p formulation, and the velocity space P_K^{br} is minimal to prove the inf-sup condition with piecewise contante pressure by triangle.
the finite element space V_h is

$$V_h = \{u \in H^1(\Omega)^2; \quad \forall K \in T_h, u_{|K} \in P_K^{br}\}$$

where

$$P_K^{br} = span\{\lambda_i^K e_k\}_{i=1,2,3,k=1,2} \cup \{\lambda_i^K \lambda_{i+1}^K n_{i+2}^K\}_{i=1,2,3}$$

with notation $4 = 1, 5 = 2$ and where λ_i^K are the barycentric coordonnate of the triangle K, $(e_k)_{k=1,2}$ the canonical basis of $\mathbb{R}^2$ and n_k^K the outer normal of triangle K opposite to vertex k.

```
//    The P2BR finite element :  the Bernadi Raugel Finite Element
//    F. Hecht, decembre 2005
//    -------------
 //    See Bernardi, C., Raugel, G.:  Analysis of some finite elements for the
Stokes problem.  Math.  Comp.  44, 71-79 (1985).
//    It is a 2d coupled FE
//    the Polynomial space is P1^2 + 3 normals bubbles edges function (P2)
       //    the degre of freedom is 6 values at of the 2 componantes at the 3
vertices
//    and the 3 flux on the 3 edges
//    So 9 degrees of freedom and N= 2.

//    ---------------------- related files:
//    to check and validate :  testFE.edp
//    to get a real example :  NSP2BRP0.edp
//    ------------------------------------------------------------

                                                          //    ----------------------
#include "error.hpp"
#include "AFunction.hpp"
#include "rgraph.hpp"
using namespace std;
#include "RNM.hpp"
#include "fem.hpp"
#include "FESpace.hpp"
#include "AddNewFE.h"
```

```
namespace  Fem2D {

  class TypeOfFE_P2BRLagrange : public  TypeOfFE { public:
    static int Data[];

    TypeOfFE_P2BRLagrange(): TypeOfFE(6+3+0,
      2,
      Data,
      4,
      1,
      6+3*(2+2),                              //    nb coef to build interpolation
      9,                                     //    np point to build interpolation
      0)
    {
 ....                                                 //    to long see the source
     }
    void FB(const bool * whatd, const Mesh & Th,const Triangle & K,const R2 &P,
RNMK_ & val) const;
    void TypeOfFE_P2BRLagrange::Pi_h_alpha(const baseFElement & K,KN_<double> &
v) const;
  } ;
                                                //    on what nu df on node node of df
 int TypeOfFE_P2BRLagrange::Data[]={
   0,0, 1,1, 2,2,  3,4,5,
   0,1, 0,1, 0,1,  0,0,0,
   0,0, 1,1, 2,2,  3,4,5,
   0,0, 0,0, 0,0,  0,0,0,
   0,1, 2,3, 4,5,  6,7,8,
   0,0
};

void TypeOfFE_P2BRLagrange::Pi_h_alpha(const baseFElement & K,KN_<double> & v) const
  {
    const Triangle & T(K.T);
    int k=0;
                             //    coef pour les 3 sommets fois le 2 composantes
    for (int i=0;i<6;i++)
      v[k++]=1;
                                                //    integration sur les aretes
    for (int i=0;i<3;i++)
      {

        R2 N(T.Edge(i).perp());
N  *= T.EdgeOrientation(i)*0.5 ;
        v[k++]= N.x;
        v[k++]= N.y;
        v[k++]= N.x;
        v[k++]= N.y;
      }
  }

  void TypeOfFE_P2BRLagrange::FB(const bool * whatd,const Mesh & ,const Triangle
& K,const R2 & P,RNMK_ & val) const
  {
 ....                                                 //    to long see the source
  }
```

```
        //    ---- cooking to add the finite elemet to freefem table --------
                                  //    a static variable to def the finite element
  static TypeOfFE_P2BRLagrange P2LagrangeP2BR;
                                                 //    now adding FE in FreeFEm++ table
  static AddNewFE P2BR("P2BR",&P2LagrangeP2BR);
                                                                  //    --- end cooking
}                                                              //    end FEM2d namespace
```

A way to check the finite element

```
load "BernadiRaugel"
                                    //    a macro the compute numerical derivative
macro DD(f,hx,hy) ( (f(x1+hx,y1+hy)-f(x1-hx,y1-hy))/(2*(hx+hy)))          //
mesh Th=square(1,1,[10*(x+y/3),10*(y-x/3)]);

real x1=0.7,y1=0.9, h=1e-7;
int it1=Th(x1,y1).nuTriangle;

fespace Vh(Th,P2BR);

Vh [a1,a2],[b1,b2],[c1,c2];


for (int i=0;i<Vh.ndofK;++i)
cout << i << " " << Vh(0,i) << endl;
for (int i=0;i<Vh.ndofK;++i)
{
  a1[]=0;
  int j=Vh(it1,i);
  a1[][j]=1;                                                  //    a bascis functions
  plot([a1,a2], wait=1);

  [b1,b2]=[a1,a2];                                          //    do the interpolation

  c1[] = a1[] - b1[];

  cout << " ---------" << i << " " << c1[].max << " " << c1[].min << endl;
  cout << " a = " << a1[] <<endl;
  cout << " b = " << b1[] <<endl;
  assert(c1[].max < 1e-9 && c1[].min > -1e-9);                //    check if the
interpolation is correct

                              //    check the derivative and numerical derivative

  cout << " dx(a1)(x1,y1) = " << dx(a1)(x1,y1) << " == " << DD(a1,h,0) << endl;
  assert( abs(dx(a1)(x1,y1)-DD(a1,h,0) ) < 1e-5);
  assert( abs(dx(a2)(x1,y1)-DD(a2,h,0) ) < 1e-5);
  assert( abs(dy(a1)(x1,y1)-DD(a1,0,h) ) < 1e-5);
  assert( abs(dy(a2)(x1,y1)-DD(a2,0,h) ) < 1e-5);

}
```

A real example using this finite element, just a small modification of the `NSP2P1.edp` examples, just the begenning is change to

```
load "BernadiRaugel"

real s0=clock();
mesh Th=square(10,10);
fespace Vh2(Th,P2BR);
fespace Vh(Th,P0);
Vh2 [u1,u2],[up1,up2];
Vh2 [v1,v2];
```

And the plot instruction is also change because the pressure is constant, and we cannot plot isovalues.

Morley See the example.

C.5 Add a new sparse solver

I will show the sketch of the code, see the full code from `SuperLU.cpp` or `NewSolve.cpp`. First the include files:

```
#include  <iostream>
using namespace std;

#include "rgraph.hpp"
#include "error.hpp"
#include "AFunction.hpp"

                                                    //   #include "lex.hpp"
#include "MatriceCreuse_tpl.hpp"
#include "slu_ddefs.h"
#include "slu_zdefs.h"
```

A small template driver to unified the `double` and `Complex` version.

```
template <class R> struct SuperLUDriver
{

};


template <> struct SuperLUDriver<double>
{
  ....  double version
};

template <> struct SuperLUDriver<Complex>
{
....  Complex version
};
```

To get Matrix value, we have just to remark that the Morse Matrice the storage, is the SLU_NR format is the compressed row storage, this is the transpose of the compressed column storage. So if AA is a MatriceMorse you have with SuperLU notatiopn.

```
n=AA.n;
m=AA.m;
nnz=AA.nbcoef;
a=AA.a;
asub=AA.cl;
xa=AA.lg;
options.Trans = TRANS;

Dtype_t R_SLU = SuperLUDriver<R>::R_SLU_T();
Create_CompCol_Matrix(&A, m, n, nnz, a, asub, xa, SLU_NC, R_SLU, SLU_GE);
```

To get vector infomation, to solver the linear solver $x = A^{-1}b$

```
  void Solver(const MatriceMorse<R> &AA,KN_<R> &x,const KN_<R> &b) const
 {
 ....
    Create_Dense_Matrix(&B, m, 1, b, m, SLU_DN, R_SLU, SLU_GE);
    Create_Dense_Matrix(&X, m, 1, x, m, SLU_DN, R_SLU, SLU_GE);
 ....
 }
```

The two BuildSolverSuperLU function, to change the default sparse solver variable DefSparseSolver<double>::solver

```
MatriceMorse<double>::VirtualSolver *
BuildSolverSuperLU(const MatriceMorse<double> *A,int strategy,double tgv, double
eps,
                   double  tol_pivot,double tol_pivot_sym,
              int NbSpace,int itmax ,const  void * precon, ?void * stack)
{
    if(verbosity>9)
       cout << " BuildSolverSuperLU<double>" << endl;
    return new SolveSuperLU<double>(*A,strategy,tgv,eps,tol_pivot,tol_pivot_sym);
}


MatriceMorse<Complex>::VirtualSolver *
BuildSolverSuperLU(const MatriceMorse<Complex> *A,int strategy,double tgv, double
eps,
           double tol_pivot,double tol_pivot_sym,
    int NbSpace,int itmax ,const  void * precon, void * stack)
{
    if(verbosity>9)
    cout << " BuildSolverSuperLU<Complex>" << endl;
    return new SolveSuperLU<Complex>(*A,strategy,tgv,eps,tol_pivot,tol_pivot_sym);
}
```

The link to FreeFem++

```
class Init { public:
    Init();
```

```
};


  To set the  2 default sparse solver double and complex:


DefSparseSolver<double>::SparseMatSolver SparseMatSolver_R ; ;
DefSparseSolver<Complex>::SparseMatSolver SparseMatSolver_C;
```

To save the default solver type

```
TypeSolveMat::TSolveMat  TypeSolveMatdefaultvalue=TypeSolveMat::defaultvalue;
```

To reset to the default solver, call this function:

```
bool SetDefault()
{
    if(verbosity>1)
  cout << " SetDefault sparse to default" << endl;
    DefSparseSolver<double>::solver =SparseMatSolver_R;
    DefSparseSolver<Complex>::solver =SparseMatSolver_C;
    TypeSolveMat::defaultvalue =TypeSolveMat::SparseSolver;
}
```

To set the default solver to superLU, call this function:

```
bool SetSuperLU()
{
    if(verbosity>1)
  cout << " SetDefault sparse solver to SuperLU" << endl;
    DefSparseSolver<double>::solver  =BuildSolverSuperLU;
    DefSparseSolver<Complex>::solver =BuildSolverSuperLU;
    TypeSolveMat::defaultvalue =TypeSolveMatdefaultvalue;
}
```

To add new function/name `defaultsolver`, `defaulttoSuperLU`in `freefem++`, and set the default solver to the new solver., just do:

```
Init init;
Init::Init()
{

  SparseMatSolver_R= DefSparseSolver<double>::solver;
  SparseMatSolver_C= DefSparseSolver<Complex>::solver;

  if(verbosity>1)
    cout << "\n Add: SuperLU,  defaultsolver defaultsolverSuperLU" << endl;
  TypeSolveMat::defaultvalue=TypeSolveMat::SparseSolver;
  DefSparseSolver<double>::solver =BuildSolverSuperLU;
  DefSparseSolver<Complex>::solver =BuildSolverSuperLU;
                      //    test if the name "defaultsolver" exist in freefem++
  if(! Global.Find("defaultsolver").NotNull() )
    Global.Add("defaultsolver","(",new OneOperator0<bool>(SetDefault));
  Global.Add("defaulttoSuperLU","(",new OneOperator0<bool>(SetSuperLU));
}
```

To compile superlu.cpp, just do:

1. download the SuperLu 3.0 package and do

```
curl   http://crd.lbl.gov/~xiaoye/SuperLU/superlu_3.0.tar.gz  -o supe
tar xvfz superlu_3.0.tar.gz
go SuperLU_3.0 directory
$EDITOR  make.inc
make
```

2. In directoy include do to have a correct version of `SuperLu` header due to mistake in case of inclusion of `double` and `Complex` version in the same file.

```
tar xvfz ../SuperLU_3.0-include-ff.tar.gz
```

I will give a correct one to compile with freefm++.

To compile the freefem++ load file of SuperLu with freefem do: some find like :

```
./load.link SuperLU.cpp  -L$HOME/work/LinearSolver/SuperLU_3.0/ -lsup
```

And to test the simple example:

A example:

```
load "SuperLU"
verbosity=2;
for(int i=0;i<3;++i)
{
//    if i == 0 then SuperLu solver
//    i == 1 then GMRES solver
//    i == 2 then Default solver
  {
    matrix A =
      [[ 0, 1, 0, 10],
       [ 0,  0,  2, 0],
       [ 0, 0, 0,  3],
       [ 4,0 , 0, 0]];
    real[int] xx = [ 4,1,2,3], x(4), b(4);
    b = A*xx;
    cout << b << " " << xx << endl;
    set(A,solver=sparsesolver);
    x = A^-1*b;
    cout << x << endl;
  }

  {
    matrix<complex> A =
      [[ 0, 1i, 0, 10],
       [ 0 ,  0,  2i, 0],
       [ 0, 0, 0,  3i],
       [ 4i,0 , 0, 0]];
    complex[int] xx = [ 4i,1i,2i,3i], x(4), b(4);
```

```
    b = A*xx;
    cout << b << " " << xx << endl;
    set(A,solver=sparsesolver);
    x = A^-1*b;
    cout << x << endl;
  }
  if(i==0)defaulttoGMRES();
  if(i==1)defaultsolver();
}
```

To Test do for exemple:

```
FreeFem++ SuperLu.edp
```

FreeFem++ LGPL License

This is The FreeFem++ software. Programs in it were maintained by

- Frédéric hecht <Frederic.Hecht@upmc.fr>
- Olivier Pironneau <Olivier.Pironneau@upmc.fr>
- Antoine Le Hyaric <lehyaric@ann.jussieu.fr>

All its programs except files the comming from ARPACK++ software (files in directory ARPACK/arpack++/include) COOOL sofware (files in directory src/Algo) and the file mt19937ar.cpp which may be redistributed under the terms of the GNU LESSER GENERAL PUBLIC LICENSE Version 2.1, February 1999
GNU LESSER GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION
0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called ”this License”). Each licensee is addressed as ”you”.
A ”library” means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables. The ”Library”, below, refers to any such software library or work which has been distributed under these terms. A ”work based on the Library” means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term ”modification”.)
”Source code” for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.
Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.
1. You may copy and distribute verbatim copies of the Library’s complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.
5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a ”work that uses the Library”. Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.
However, linking a ”work that uses the Library” with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a ”work that uses the library”. The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.
When a ”work that uses the Library” uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.
If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)
Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.
6. As an exception to the Sections above, you may also combine or link a ”work that uses the Library” with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer’s own use and reverse engineering for debugging such modifications.
You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:
a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable ”work that uses the Library”, as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user’s computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.
c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the ”work that uses the Library” must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.
It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.
This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and ”any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.
14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.
NO WARRANTY
15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY ”AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INAC-

CURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
END OF TERMS AND CONDITIONS

Appendix D

Keywords

Main Keywords

```
adaptmesh
Cmatrix
R3
bool
border
break
buildmesh
catch
cin
complex
continue
cout
element
else
end
fespace
for
func
if
ifstream
include
int
intalledge
load
macro
matrix
mesh
movemesh
ofstream
plot
problem
real
return
savemesh
solve
string
try
throw
vertex
varf
while
```

Second category of Keywords

```
int1d
int2d
on
square
```

Third category of Keywords

```
dx
dy
convect
jump
mean
```

Fourth category of Keywords

```
wait
ps
solver
CG
LU
UMFPACK
factorize
init
endl
```

Other Reserved Words

```
x, y, z, pi, i,
sin, cos, tan, atan, asin, acos,
cotan,sinh,cosh,tanh,cotanh,
exp, log, log10, sqrt
abs, max, min,
```

Bibliography

[1] R. B. LEHOUCQ, D. C. SORENSEN, AND C. YANG *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods* SIAM, ISBN 0-89871-407-9 // `http://www.caam.rice.edu/software/ARPACK/`

[2] BABBUŠKA, I: Error bounds for finite element method, Numer. Math. 16, 322-333.

[3] Y. ACHDOU AND O. PIRONNEAU: Computational Methods for Option Pricing. SIAM monograph (2005).

[4] D. BERNARDI, F.HECHT, K. OHTSUKA, O. PIRONNEAU: *freefem+ documentation*, on the web at ftp://www.freefem.org/freefemplus.

[5] D. BERNARDI, F.HECHT, O. PIRONNEAU, C. PRUD'HOMME: *freefem documentation*, on the web at http://www. freefem.fr/freefem

[6] DAVIS, T. A: Algorithm 8xx: UMFPACK V4.1, an unsymmetric-pattern multifrontal method TOMS, 2003 (under submission) `http://www.cise.ufl.edu/research/sparse/umfpack`

[7] GEORGE, P.L: *Automatic triangulation*, Wiley 1996.

[8] HECHT, F: The mesh adapting software: bamg. INRIA report 1998.

[9] Library Modulef , INRIA, `http://www.inria-rocq/modulef`

[10] A. ERN AND J.-L. GUERMOND, Discontinuous Galerkin methods for Friedrichs' symmetric systems and Second-order PDEs, SIAM J. Numer. Anal., (2005). See also: Theory and Practice of Finite Elements, vol. 159 of Applied Mathematical Sciences, Springer-Verlag, New York, NY, 2004.

[11] F. HECHT. C++ Tools to construct our user-level language. Vol 36, N°, 2002 pp 809-836, Modél. math et Anal Numér.

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

[13] B. LUCQUIN, O. PIRONNEAU: *Introduction to Scientific Computing* Wiley 1998.

[14] I. DANAILA, F. HECHT, AND O. PIRONNEAU. *Simulation numérique en C++*. Dunod, Paris, 2003.

[15] J. NEČAS AND L/ HLAVÁČEK, Mathematical theory of elastic and elasto-plastic bodies: An introduction, Elsevier, 1981.

[16] K. OHTSUKA, O. PIRONNEAU AND F. HECHT: Theoretical and Numerical analysis of energy release rate in 2D fracture, INFORMATION **3** (2000), 303–315.

[17] F. PREPARATA, M. SHAMOS *Computational Geometry* Springer series in Computer sciences, 1984.

[18] 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

[19] ROBERTS, J.E. AND THOMAS J.-M: Mixed and Hybrid Methods, Handbook of Numerical Anaysis, Vol.II, North-Holland, 1993

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

[21] TABATA, M: Numerical solutions of partial differential equations II (in Japanese), Iwanami Applied Math., 1994

[22] THOMASSET, F: Implementation of finite element methods of Navier-Stokes Equations, Springer-Verlag, 1981

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

[24] Bison documentation

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

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

[27] 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).

[28] R. GLOWINSKI AND O.PIRONNEAU, Numerical methods for the Stokes problem, Chapter 13 of Energy Methods in Finite Element Analysis, R.Glowinski, E.Y. Rodin, O.C. Zienkiewicz eds., J.Wiley & Sons, Chichester, UK, 1979, pp. 243-264.

[29] R. GLOWINSKI, Numerical Methods for Nonlinear Variational Problems, Springer-Verlag, New York, NY, 1984.

[30] R. GLOWINSKI, Finite Element Methods for Incompressible Viscous Flow. In Handbook of Numerical Analysis, Vol. IX, P.G. Ciarlet and J.L. Lions, eds., North-Holland, Amsterdam, 2003, pp.3-1176.

[31] Horgan, C; Saccomandi, G; "Constitutive Models for Compressible Nonlinearly Elastic Materials with Limiting Chain Extensibility," Journal of Elasticity, Volume 77, Number 2, November 2004, pp. 123-138(16).

[32] KAZUFUMI ITO, AND KARL KUNISCH, Semi smooth newton methods for variational inequalities of the first kind , M2AN, vol 37, N°, 2003, pp 41-62.

[33] Ogden, RW; Non-Linear Elastic Deformations, Dover, 1984.

[34] Raviart, P; Thomas J; Introduction à l'analyse numérique des équations aux dérivées partielles, Masson, 1983.

[35] M. A. TAYLOR, B. A. WINGATE , L. P. BOS, Several new quadrature formulas for polynomial integration in the triangle , Report-no: SAND2005-0034J, `http://xyz.lanl.gov/format/math.NA/0501496`

[36] Wilmott, Paul and Howison, Sam and Dewynne, Jeff : A student introduction to mathematical finance, Cambridge University Press (1995).

Index

Book Description

Fruit of a long maturing process freefem, in its last avatar, `FreeFem++`, is a high level integrated development environment (IDE) for partial differential equations (PDE). It is the ideal tool for teaching the finite element method but it is also perfect for research to quickly test new ideas or multi-physics and complex applications.

`FreeFem++` has an advanced automatic mesh generator, capable of a posteriori mesh adaptation; it has a general purpose elliptic solver interfaced with fast algorithms such as the multi-frontal method UMFPACK. Hyperbolic and parabolic problems are solved by iterative algorithms prescribed by the user with the high level language of `FreeFem++`. It has several triangular finite elements, including discontinuous elements. Finally everything is there in `FreeFem++` to prepare research quality reports: color display online with zooming and other features and postscript printouts.

This book is ideal for students at Master level, for researchers at any level and for engineers also in financial mathematics.

Editorial Reviews

"...Impossible to put the book down, suspense right up to the last page..."
A. TANH, Siam Chronicle.

"...The chapter on discontinuous fems is so hilarious ...."
B. GALERKINE, .

About the Authors Российской академии наук

Frédéric Hecht is a professor of numerical analysis at the university of Paris VI and does his research at the Laboratoire Jacques-Louis Lions (LJLL). He is also a member of the project gamma at INRIA. He is the author of several advanced software tools for automatic triangulations including the public domain EMC2.

Olivier Pironneau is a professor of numerical analysis at the university of Paris VI and at LJLL. His scientific contributions are in numerical methods for fluids. He is a member of the Institut Universitaire de France and of the French Academy of Sciences

Koji Ohtsuka is a professor at the Hiroshima Kokusai Gakuin University, Japan and chairman of the World Scientific and Engineering academy and Society, Japan chapter. His research is in fracture dynamics, modelling and computing.

Antoine Le Hyaric is a research engineer from the "Centre National de la Recherche Scientifique" (CNRS) at LJLL . He is an expert in software engineering for scientific applications. He has applied his skills mainly to electromagnetics simulation, parallel computing and three-dimensional visualization.