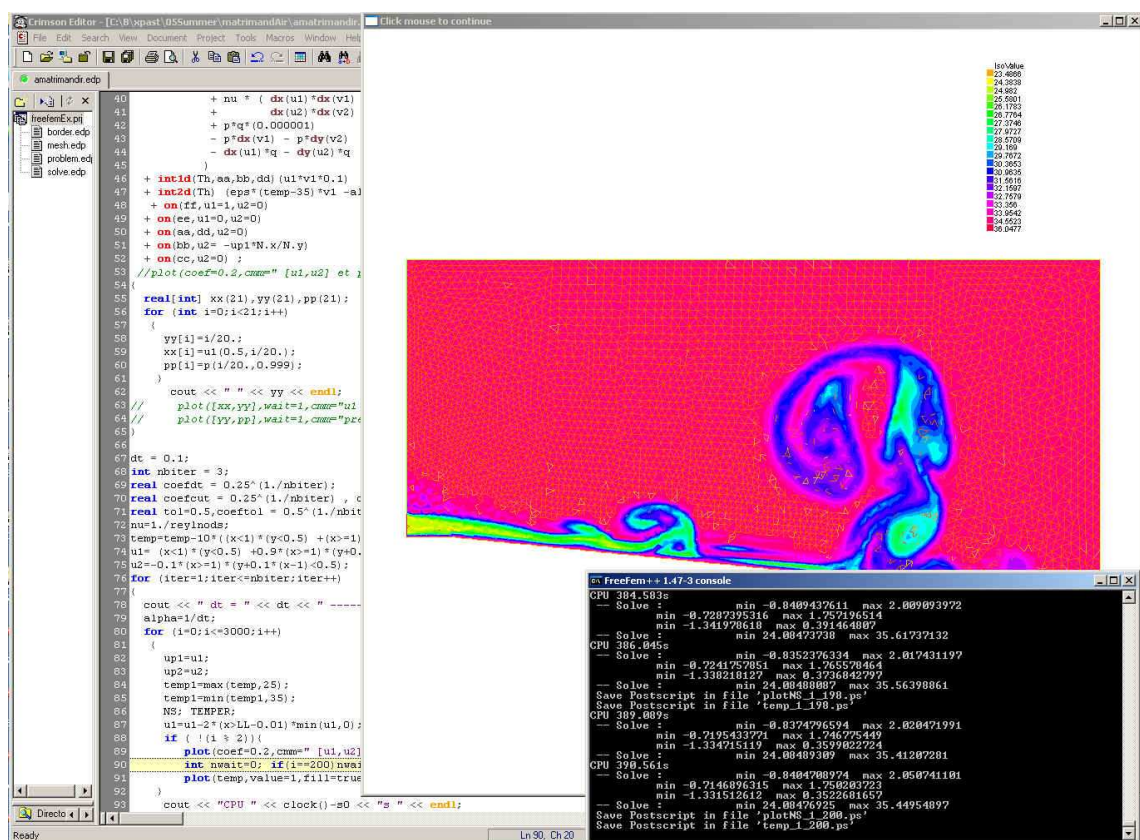


# Freefem++

Third Edition, Version 3.8

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

F. Hecht



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



# FreeFem++

Third Edition, Version 3.8

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

Frédéric Hecht<sup>1,4</sup>

<mailto:frederic.hecht@upmc.fr>

<http://www.ann.jussieu.fr/~hecht>

The main participants of the documentation and of the developement are:

- **Olivier Pironneau**, <mailto:olivier.pironneau@upmc.fr>, <http://www.ann.jussieu.fr/pironneau> 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
- **Jacques Morice**, <mailto:morice@ann.jussieu.fr>. Jacaues Morice is a Post-Doct at LJLL. His doing is Thesis in University of Bordeaux I on fast multipole method (FMM). On this version, he do all three dimension mesh generation and coupling with medit software.
- **Antoine Le Hyaric**, <mailto:lehyaric@ann.jussieu.fr>, <http://www.ann.jussieu.fr/~lehyaric/> 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.
- **Kohji Ohtsuka**, <mailto:ohtsuka@hkg.ac.jp>, <http://www.comfos.org/> Kohji 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, modeling and computing.



**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>) Référence : ANR-07-CIS7-002-01.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Installation . . . . .	2
1.1.1	Installation from sources . . . . .	2
1.1.2	Windows binaries install . . . . .	4
1.1.3	MacOS X binaries install . . . . .	5
1.2	How to use FreeFem++ . . . . .	5
1.3	Environment variables, and the init file . . . . .	7
1.4	History . . . . .	8
<b>2</b>	<b>Getting Started</b>	<b>11</b>
2.0.1	FEM by FreeFem++ : how does it work? . . . . .	12
2.0.2	Some Features of FreeFem++ . . . . .	16
2.1	The Development Cycle: Edit–Run/Visualize–Revise . . . . .	16
<b>3</b>	<b>Learning by Examples</b>	<b>19</b>
3.1	Membranes . . . . .	19
3.2	Heat Exchanger . . . . .	24
3.3	Acoustics . . . . .	26
3.4	Thermal Conduction . . . . .	28
3.4.1	Axisymmetry: 3D Rod with circular section . . . . .	29
3.4.2	A Nonlinear Problem : Radiation . . . . .	30
3.5	Irrotational Fan Blade Flow and Thermal effects . . . . .	30
3.5.1	Heat Convection around the airfoil . . . . .	32
3.6	Pure Convection : The Rotating Hill . . . . .	33
3.7	A Projection Algorithm for the Navier-Stokes equations . . . . .	37
3.8	The System of elasticity . . . . .	39
3.9	The System of Stokes for Fluids . . . . .	41
3.10	A Large Fluid Problem . . . . .	43
3.11	An Example with Complex Numbers . . . . .	46
3.12	Optimal Control . . . . .	47
3.13	A Flow with Shocks . . . . .	49
3.14	Classification of the equations . . . . .	51
<b>4</b>	<b>Syntax</b>	<b>55</b>
4.1	Data Types . . . . .	55
4.2	List of major types . . . . .	56
4.3	Global Variables . . . . .	57

4.4	System Commands . . . . .	58
4.5	Arithmetic . . . . .	58
4.6	One Variable Functions . . . . .	61
4.7	Functions of Two Variables . . . . .	63
4.7.1	Formula . . . . .	63
4.7.2	FE-function . . . . .	63
4.8	Arrays . . . . .	64
4.8.1	Arrays with two integer indices versus matrix . . . . .	69
4.8.2	Matrix construction and setting . . . . .	71
4.8.3	Matrix Operations . . . . .	72
4.8.4	Other arrays . . . . .	76
4.9	Loops . . . . .	77
4.10	Input/Output . . . . .	78
4.11	Exception handling . . . . .	79
<b>5</b>	<b>Mesh Generation</b>	<b>83</b>
5.1	Commands for Mesh Generation . . . . .	83
5.1.1	Square . . . . .	83
5.1.2	Border . . . . .	84
5.1.3	Data Structure and Read/Write Statements for a Mesh . . . . .	86
5.1.4	Mesh Connectivity . . . . .	88
5.1.5	The keyword "triangulate" . . . . .	91
5.2	Boundary FEM Spaces Built as Empty Meshes . . . . .	92
5.3	Remeshing . . . . .	93
5.3.1	Movemesh . . . . .	93
5.4	Regular Triangulation: hTriangle . . . . .	95
5.5	Adaptmesh . . . . .	95
5.6	Trunc . . . . .	100
5.7	Splitmesh . . . . .	101
5.8	Meshing Examples . . . . .	101
5.9	How to change the label of elements and border elements of a mesh in FreeFem++ ?	107
5.10	Mesh in three dimension . . . . .	108
5.10.1	Read/Write Statements for a Mesh in 3D . . . . .	108
5.10.2	TeGen: A tetrahedral mesh generator . . . . .	108
5.10.3	Reconstruct/Refine a three dimensional mesh with TetGen . . . . .	112
5.10.4	Moving mesh in three dimension . . . . .	114
5.10.5	Layer mesh . . . . .	114
5.11	Meshing examples . . . . .	119
5.11.1	Build a 3d mesh of a cube with a ballon incrustation . . . . .	121
5.12	Write solution at the format .sol and .solb . . . . .	122
5.13	Call medit with the keyword medit . . . . .	124
<b>6</b>	<b>Finite Elements</b>	<b>127</b>
6.1	Usage of two dimensional finite element spaces . . . . .	129
6.2	Usage of thee dimensional finite element spaces . . . . .	130
6.3	Lagrange finite element . . . . .	130
6.3.1	P0-element . . . . .	130

6.3.2	P1-element	131
6.3.3	P2-element	131
6.4	P1 Nonconforming Element	133
6.5	Other FE-space	133
6.6	Vector valued FE-function	134
6.6.1	Raviart-Thomas element	134
6.7	A Fast Finite Element Interpolator	136
6.8	Keywords: Problem and Solve	139
6.8.1	Weak form and Boundary Condition	139
6.9	Parameters affecting solve and problem	140
6.10	Problem definition	141
6.11	Numerical Integration	143
6.12	Variational Form, Sparse Matrix, PDE Data Vector	147
6.13	Interpolation matrix	151
6.14	Finite elements connectivity	152
<b>7</b>	<b>Visualization</b>	<b>153</b>
7.1	Plot	153
7.2	link with gnuplot	157
7.3	link with medit	157
<b>8</b>	<b>Algorithms</b>	<b>159</b>
8.1	conjugate Gradient/GMRES	159
8.2	Optimization	161
<b>9</b>	<b>Mathematical Models</b>	<b>163</b>
9.1	Static Problems	163
9.1.1	Soap Film	163
9.1.2	Electrostatics	165
9.1.3	Aerodynamics	166
9.1.4	Error estimation	168
9.1.5	Periodic	170
9.1.6	Poisson with mixed boundary condition	173
9.1.7	Poisson with mixte finite element	175
9.1.8	Metric Adaptation and residual error indicator	177
9.1.9	Adaptation using residual error indicator	178
9.2	Elasticity	180
9.2.1	Fracture Mechanics	184
9.3	Nonlinear Static Problems	187
9.3.1	Newton-Raphson algorithm	188
9.4	Eigenvalue Problems	190
9.5	Evolution Problems	194
9.5.1	Mathematical Theory on Time Difference Approximations.	195
9.5.2	Convection	197
9.5.3	2D Black-Scholes equation for an European Put option	199
9.6	Navier-Stokes Equation	201
9.6.1	Stokes and Navier-Stokes	201

9.6.2	Uzawa Conjugate Gradient . . . . .	205
9.6.3	NSUzawaCahouetChabart.edp . . . . .	207
9.7	Variational inequality . . . . .	208
9.8	Domain decomposition . . . . .	211
9.8.1	Schwarz Overlap Scheme . . . . .	211
9.8.2	Schwarz non Overlap Scheme . . . . .	213
9.8.3	Schwarz-gc.edp . . . . .	214
9.9	Fluid/Structures Coupled Problem . . . . .	216
9.10	Transmission Problem . . . . .	219
9.11	Free Boundary Problem . . . . .	221
9.12	nolinear-elas.edp . . . . .	224
9.13	Compressible Neo-Hookean Materials: Computational Solutions . . . . .	228
9.13.1	Notation . . . . .	228
9.13.2	A Neo-Hookean Compressible Material . . . . .	228
9.13.3	An Approach to Implementation in FreeFem++ . . . . .	229
<b>10</b>	<b>MPI Parallel version</b>	<b>231</b>
10.1	MPI keywords . . . . .	231
10.2	MPI constants . . . . .	231
10.3	MPI Constructor . . . . .	232
10.4	MPI functions . . . . .	232
10.5	MPI communicator operator . . . . .	233
10.6	Schwarz example in parallel . . . . .	234
<b>11</b>	<b>Parallel sparse solvers</b>	<b>237</b>
11.1	Using parallel sparse solvers in Freefem++ . . . . .	237
11.2	Sparse direct solver . . . . .	240
11.2.1	MUMPS solver . . . . .	240
11.2.2	SuperLU distributed solver . . . . .	243
11.2.3	Pastix solver . . . . .	245
11.3	Parallel sparse iterative solver . . . . .	246
11.3.1	pARMS solver . . . . .	247
11.3.2	Interfacing with HIPS . . . . .	249
11.3.3	Interfacing with HYPRE . . . . .	255
11.3.4	Conclusion . . . . .	256
11.4	Domain decomposition . . . . .	259
11.4.1	Communicators and groups . . . . .	259
11.4.2	Process . . . . .	260
11.4.3	Points to Points communicators . . . . .	260
11.4.4	Global operations . . . . .	260
<b>12</b>	<b>Mesh Files</b>	<b>265</b>
12.1	File mesh data structure . . . . .	265
12.2	bb File type for Store Solutions . . . . .	266
12.3	BB File Type for Store Solutions . . . . .	266
12.4	Metric File . . . . .	267
12.5	List of AM_FMT, AMDBA Meshes . . . . .	267



<b>13 Add new finite element</b>	<b>271</b>
13.1 Some notation . . . . .	271
13.2 Which class of add . . . . .	272
<b>A Table of Notations</b>	<b>277</b>
A.1 Generalities . . . . .	277
A.2 Sets, Mappings, Matrices, Vectors . . . . .	277
A.3 Numbers . . . . .	278
A.4 Differential Calculus . . . . .	278
A.5 Meshes . . . . .	279
A.6 Finite Element Spaces . . . . .	279
<b>B Grammar</b>	<b>281</b>
B.1 The bison grammar . . . . .	281
B.2 The Types of the languages, and cast . . . . .	285
B.3 All the operators . . . . .	285
<b>C Dynamical link</b>	<b>291</b>
C.1 A first example myfunction.cpp . . . . .	291
C.2 Example Discrete Fast Fourier Transform . . . . .	294
C.3 Load Module for Dervieux' P0-P1 Finite Volume Method . . . . .	296
C.4 Add a new finite element . . . . .	299
C.5 Add a new sparse solver . . . . .	302
<b>D Keywords</b>	<b>313</b>



# 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 solving 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, SuperLU . 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 bi-dimensional 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 [9].

- 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, ...
- A parallel version using `mpi`

## 1.1 Installation

First open the following web page

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

And choose your platform: Linux, Windows, MacOS X, or go to the end of the page to get the full list of downloads.

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  $\text{\LaTeX}$  and the `cygwin` environment from

<http://www.cygwin.com>

and under MacOS X we have used the apple Developer Tools Xcode,  $\text{\LaTeX}$  from <http://www.ctan.org/system/mac/texmac>.

FreeFem++ must be compiled and installed from the source archive. This archive is available from:

<http://www.freefem.org/ff++/index.htm>

To extract files from the compressed archive `freefem++-(VERSION).tar.gz` to 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 GLUT/OpenGL (use `ffglut` visualizator) or not just add `-nw` parameter.
2. `ffglut` the visualisator through a pipe of `freefem++` (remark: *if `ffglut` is not in the system path, you have no plot*)
3. FreeFem++-nw, postscript plot output only (batch version, no graphics windows via `ffglut` )
4. FreeFem++-mpi, parallel version, postscript output only
5. Sorry, the integrated development environment is reconstruction with the new architecture
6. /Applications/FreeFem++.app, Drag and Drop CoCoa MacOS Application
7. `bamg` , the `bamg` mesh generator
8. `cvms2` , a mesh file convertor
9. `drawbdmesh` , a mesh file viewer
10. `ffmedit` the `freefem++` version of `medit` software (thank to P. Frey)

Remark, in most cases you can set the level of output (verbosity) to value `nn` by adding the parameters `-v nn` on 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
```

If you are using `nedit` as your text editor, do one time `nedit -import edp.nedit` to have coloring syntax for your `.edp` files.

The syntax of tools FreeFem++,FreeFem++-nw on the command-line are

- `FreeFem++ [-?] [-vnn] [-fglut file1] [-glut file2] [-f] edppath` where the
- or `FreeFem++-nw -? [-vnn] [-fglut file1] [-glut file2] [-f] edppath` where the

- ? show the usage.
  - ffglut filename to store all the data for graphic in file filename, and to replay do `ffglut filename`.
  - glut ffglutprogram to change the visualisator program's.
  - nw no call to ffglut
  - v nn 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 on windows systeme.

where the part in [] is optional.

### 1.1.2 Windows binaries install

First download the windows installation file, then execute the download file to install FreeFem++. In Select Additional Task windows, check once the "Add application directory to your system path your system path.." boxes, because the screen on-the-fly plot is done with programm `ffglut.exe` and it can be not found.

After that you have two new icons on your desktop:

- FreeFem++ (VERSION) .exe the classical FreeFem++ application.
- FreeFem++ (VERSION) Examples a link to the FreeFem++ 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 applications: `FreeFem++-nw.exe`, `ffglut.exe`, ... the FreeFem++ application without graphic windows.

The syntax of tools on the command-line are same

#### 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 advanced features such as syntax coloring; a macro is available on [www.freefem.org](http://www.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 volunteer to program the color syntax. Download it from

<http://www.texniccenter.org/>

It is also an editor for TeX/LaTeX. It has a "tool" menu which can be configured to launch FreeFem++ programs as in:

- Select the Tools/Customize item which will bring up 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 binary version file, extract all the files 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 access to FreeFem++ just copy the file `FreeFem++` in a directory of your `$PATH` shell environment variable.

If you want to automatically launch the `FreeFem++.app`, double click on a `.edp` file icon. Under the finder pick a `.edp` in directory `examples++-tutorial` for example, select menu `File -> Get Info` and change `Open with:` (choose `FreeFem++.app`) and click on button `change All....`

## 1.2 How to use FreeFem++

**Under MacOS X with Graphic Interfaces** For testing or running an `.edp` file, just drag and drop the file icon on the MacOS application `FreeFem++.app` icon. You can also use the menu: `File -> Open` after launching the application.

One of the best ways on MacOS is to use the text editor `mi.app`

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

and to use the `edp` mode stored in `mode-mi-edp.zip`. After downloading and installing the `mi` editor, unzip `mode-mi-edp.zip` and put the created folder in the folder opened with the `mi.app` menu `Option->Open Mode Folder` menu and set `mi` as the default application for all the `.edp` files.

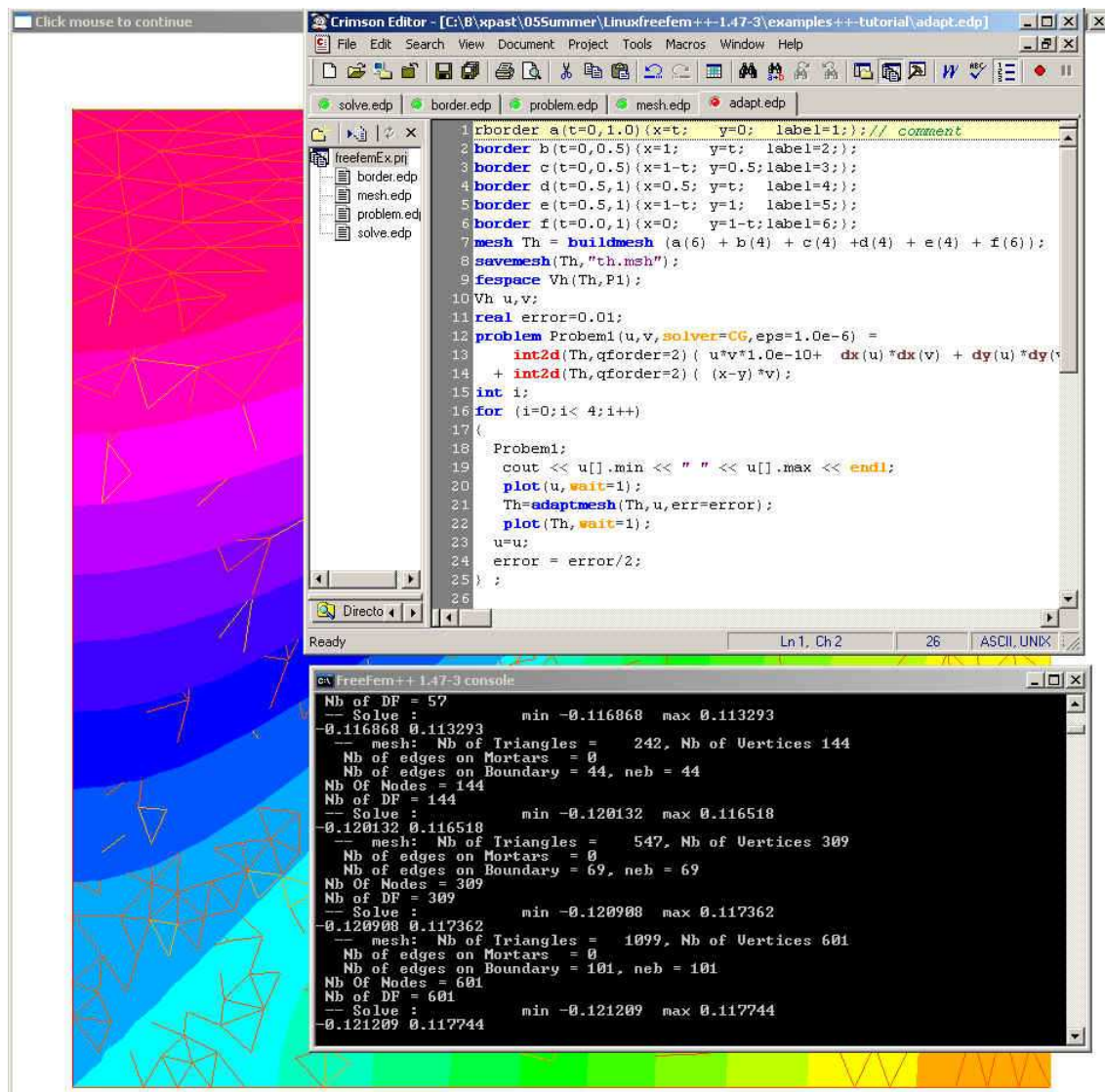


Figure 1.1: The 3 panels of the integrated environment built with the Crimson Editor with FreeFem++ in action. The Tools menu has an item to launch FreeFem++ by a Ctrl+1 command.

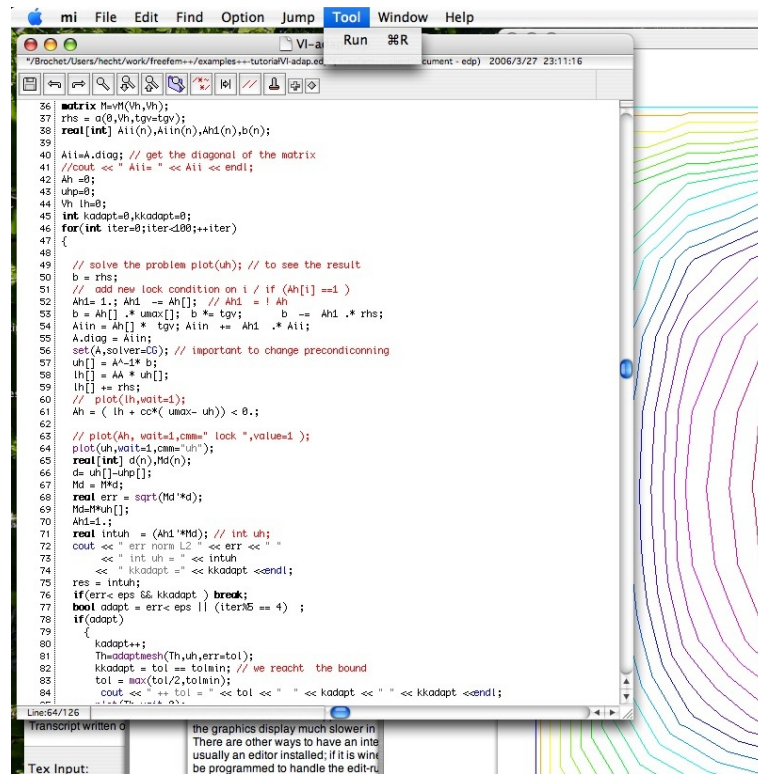


Figure 1.2: Screen of edp with mi text editor

**Under terminal** First choose the type of application from FreeFem++, FreeFem++-nw, FreeFem++-mpi, ... depending on your pleasure, system, etc. ... Next you enter, for example

FreeFem++ your-edp-file-path

### 1.3 Environment variables, and the init file

FreeFem++ reads a user's init file named `freefem++.pref` to initialize global variables: `verbosity`, `includepath`, `loadpath`.

Remark: the variable `verbosity` changes the level of internal printing (0, nothing (except mistake), 1 few, 10 lots, etc. ...), the default value is 2. The include files are searched from the `includepath` list and the load files are searched from `loadpath` list.

The syntax 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 before the init files.

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

Remark: the separator between directories must be ";" and not ":" because ":" is used under Windows.

Remark, to show the list of init of `freefem++`, do

```
Brochet$ export FF_VERBOSITY=100; ./FreeFem++-nw
-- verbosity is set to 100
insert init-files /etc/freefem++.pref $
...
```

## 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 form 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[18] 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 [26], templates and `bison` for its implementation; more detail can be found in [12]. 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*

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

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

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  $\Omega$  is the unit disk. The boundary  $C = \partial\Omega$  is

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

Note that in FreeFem++ the domain  $\Omega$  is assumed to be 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).

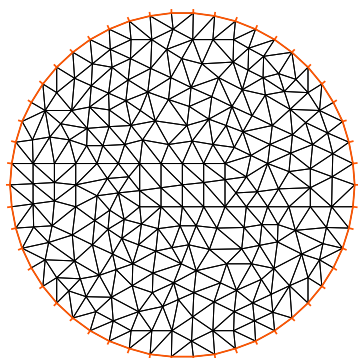


Figure 2.1: mesh Th by build(C(50))

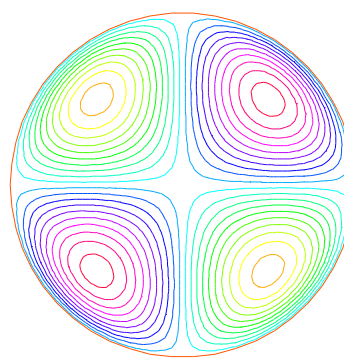


Figure 2.2: isovalue by plot(u)

### Example 2.1

```
// defining the boundary
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 in 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. Line 8 and 9 correspond to the mathematical variational equation

$$\int_{T_h} \left( \frac{\partial u}{\partial x} \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \frac{\partial v}{\partial y} \right) dx dy = \int_{T_h} f v dx dy$$

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  $\Gamma$  is described analytically by a parametric equation for  $x$  and for  $y$ . When  $\Gamma = \sum_{j=0}^J \Gamma_j$  then each curve  $\Gamma_j$ , must be specified and crossings of  $\Gamma_j$  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  $\Omega$  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  $\Gamma$ , 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  $\{T_k\}_{k=1,\dots,n_t}$  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  $\Omega$  is not a polygonal domain, a “skin” remains between the exact domain  $\Omega$  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  $\Gamma$ .

**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  $\mathcal{T}_h$ . 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*  $\phi_k$  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<sup>1</sup>. Then

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

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 in 2d, (see section 6 for the full description)

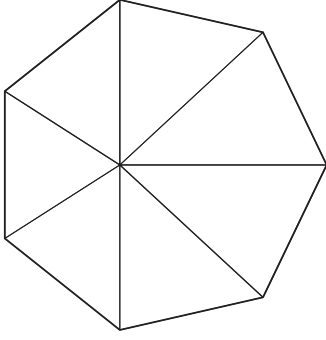
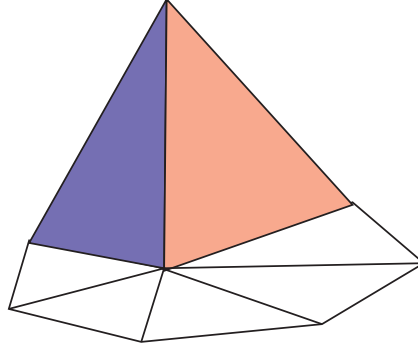
- P0 piecewise constant,
- P1 continuous piecewise linear,
- P2 continuous piecewise quadratic,
- RT0 Raviart-Thomas piecewise constant,
- P1nc piecewise linear non-conforming,
- P1dc piecewise linear discontinuous,
- P2dc piecewise quadratic discontinuous,
- P1b piecewise linear continuous plus bubble,

---

<sup>1</sup> The easiest way to define  $\phi_k$  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

$$\sum_i \lambda_i = 1, \quad \sum_i \lambda_i q^i = q$$

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

Figure 2.3: mesh  $\mathcal{T}_h$ Figure 2.4: Graph of  $\phi_1$  (left hand side) and  $\phi_6$ 

P2b piecewise quadratic continuous plus bubble.

...

Currently FreeFem++ implements the following elements in 3d, (see section 6 for the full description)

P03d piecewise constant,

P13d continuous piecewise linear,

P23d continuous piecewise quadratic,

RT03d Raviart-Thomas piecewise constant,

Edge03d The Nedelec Edge element

P1b3d piecewise linear 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:**  $\forall h \ u, v$  declares that  $u$  and  $v$  are approximated as above, namely

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

**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  $\Omega$ :

$$-\int_{\Omega} v \Delta u \, dx dy = \int_{\Omega} v f \, dx dy$$

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

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

$$\text{with } a(u, v) = \int_{\Omega} \nabla u \cdot \nabla v \, dx dy, \quad \ell(f, v) = \int_{\Omega} f v \, dx dy \quad (2.6)$$

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  $\mathbf{dx}(u) = \partial u / \partial x$ ,  $\mathbf{dy}(u) = \partial u / \partial y$  and

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

$$\int_{\Omega} f v \, dx dy \longrightarrow \text{int2d}(\text{Th}) ( f * v ) \quad (\text{notice here, } u \text{ is unused})$$

In FreeFem++ there is no need to distinguish the bilinear form  $a$  from the linear form  $\ell$ , 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 define the problem and then we solve it, write :

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

#### Step4: Solution and visualization

**6th line:** The current time in seconds 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

$$\sum_{j=0}^{M-1} A_{ij} u_j - F_i = 0, \quad i = 0, \dots, M-1; \quad F_i = \int_{\Omega} f \phi_i \, dx dy \quad (2.7)$$

which is found by using (2.4) and replacing  $v$  by  $\phi_i$  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 `tg`*vg* (*très grande valeur*) and it is generally possible to change this value , see the index item `solve!tg`*vg*.

The matrix  $A = (A_{ij})$  is called *stiffness matrix* .

If the user wants to access  $A$  directly he can do so by using (see section 6.12 page 147 for details)

```
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,Vh); // 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

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

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

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 FEM beginner, you also may want to read a book on variational formulations.

The development cycle will have the following steps:

**Modeling:** From strong forms of PDE to weak forms, one must know the variational formulation 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++ may depend on 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;
border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
border b(t=0,2*pi){ x=0.8+0.3*cos(t); y=0.3*sin(t);label=2;}
plot(a(50)+b(-30),wait=debug);           // plot the borders to see the intersection
                                           // (so change (0.8 in 0.3 in b) then needs a mouse click
mesh Th = buildmesh(a(50)+b(-30));
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 exists
    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 PDEs 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  $\Omega$  is attached to a planar rigid support  $\Gamma$ , and a force  $f(x)dx$  is exerted on each surface element  $dx = dx_1 dx_2$ . 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  $\Gamma_2$  of the membrane border  $\Gamma$  is not fixed to the support but is left hanging, then due to the 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  $\Delta$  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  $\varphi$  is the state of least energy, i.e.

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

and where  $V$  is the subspace of the Sobolev space  $H^1(\Omega)$  of functions which have zero trace on  $\Gamma_1$ . 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)

$$\int_{\Omega} \nabla \varphi \cdot \nabla w = \int_{\Omega} f w \quad \forall w \in V$$

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 from 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

$$\int_{\Omega} (A(u, v) w_1 + B(u, v) w_2) = 0 \quad \forall w_1, w_2 \dots$$

**Example** Let an ellipse have the length of the semimajor axis  $a = 2$ , 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");
```



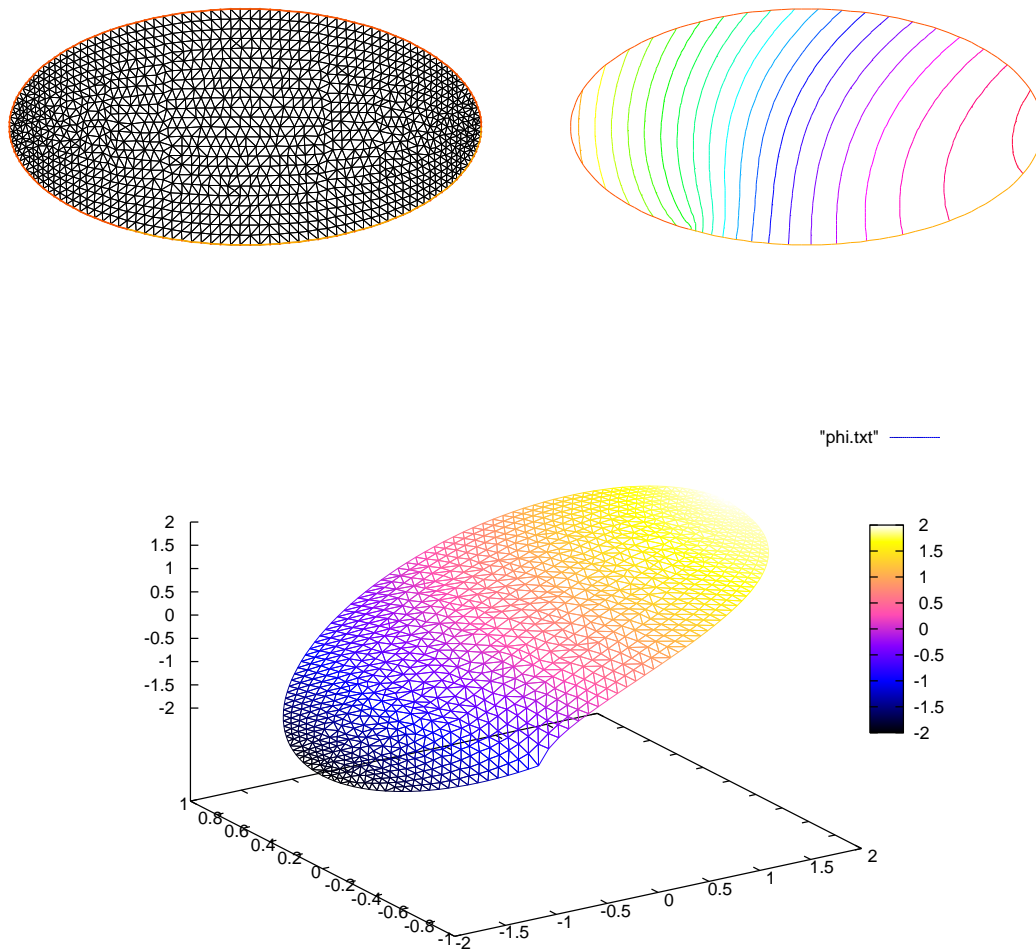


Figure 3.1: Mesh and level lines of the membrane deformation. Below: the same in 3D drawn by gnuplot from a file generated by FreeFem++ .

A triangulation is built by the keyword `buildmesh`. This 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 the number of 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  $\varphi$  (remove `Th` if  $\varphi$  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

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

except that on  $\Gamma_2$   $\partial_n \varphi = 2$  instead of zero. So we will consider a non-homogeneous Neumann condition and solve

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

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,P2);
  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 = 0.00462991
L2error 1 = 0.00117128
convergence rate = 1.9829
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"
  }
}

```

We use the finite element numbering, where  $Vh(i, j)$  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  $\kappa_2$  5 times larger than the one of  $C_0$ . We assume that the border of enclosure  $C_0$  is held at temperature  $20^\circ\text{C}$  and that we have waited long enough for thermal equilibrium.

In order to know  $u(x)$  at any point  $x$  of the domain  $\Omega$ , we must solve

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

where  $\Omega$  is the interior of  $C_0$  minus the conductors  $C_1$  and  $\Gamma$  is the boundary of  $\Omega$ , 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:

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

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  $\Gamma$ .

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

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 temperature  $u_1 = 60^\circ\text{C}$ .

### 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)+...);
```

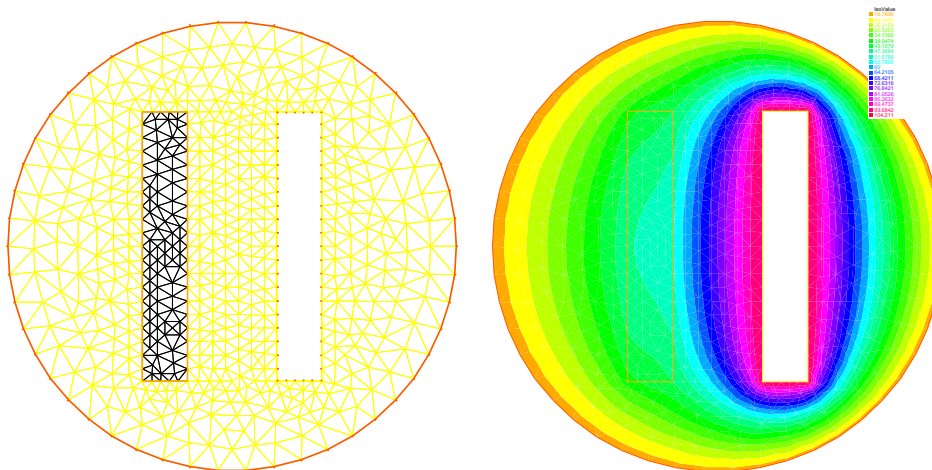


Figure 3.2: The heat exchanger

**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  $u(x, t) = \text{Re}(v(x)e^{ikt})$  where  $v$  is a solution of Helmholtz's equation:

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

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  $u_e \neq 0$  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");
```

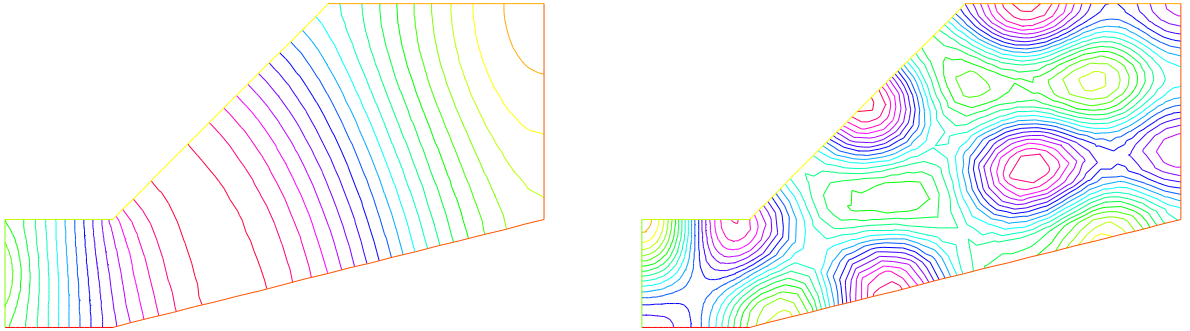


Figure 3.3: Left:Amplitude of an acoustic signal coming from the left vertical wall. Right: first eigen state ( $\lambda = (k/c)^2 = 19.4256$ ) close to 20 of eigenvalue problem :  $-\Delta\varphi = \lambda\varphi$  and  $\frac{\partial\varphi}{\partial n} = 0$  on  $\Gamma$

### 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  $\Omega$  in a time interval  $(0, T)$ .

$$\begin{aligned} \partial_t u - \nabla \cdot (\kappa \nabla u) &= 0 \text{ in } \Omega \times (0, T), \\ u(x, y, 0) &= u_0 + xu_1 \\ \kappa \frac{\partial u}{\partial n} + \alpha(u - u_e) &= 0 \text{ on } \Gamma \times (0, T). \end{aligned} \quad (3.2)$$

Here the diffusion  $\kappa$  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  $w \in H^1(\Omega)$ :

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

```
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.

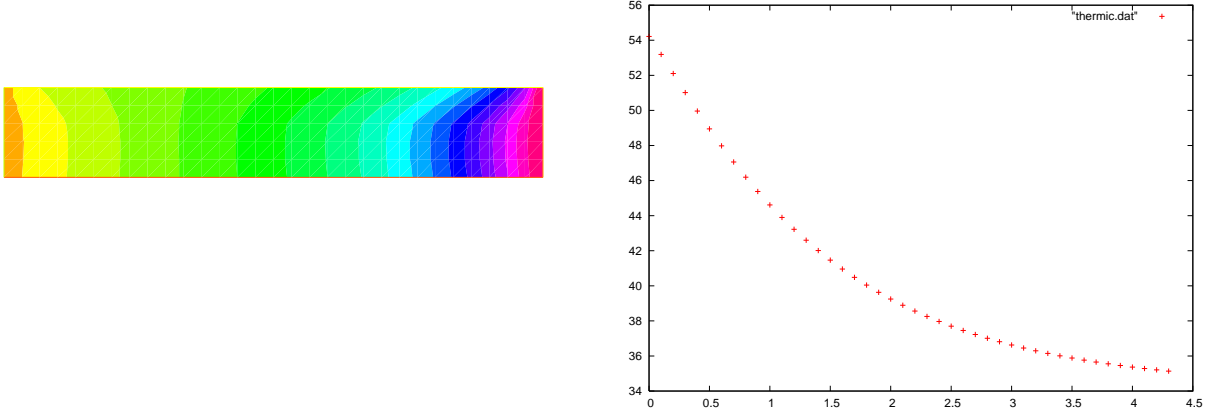


Figure 3.4: Temperature at  $T=4.9$ . Right: decay of temperature versus time at  $x=3, y=0.5$

### 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,  $\theta$  polar angle in a fixed plane perpendicular to the axis):

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

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

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

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  $\Gamma_1$ .

### 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);
real rad=1e-8, uek=ue+273;
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:

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

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  $\psi$  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  $\psi$ .

**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

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

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.

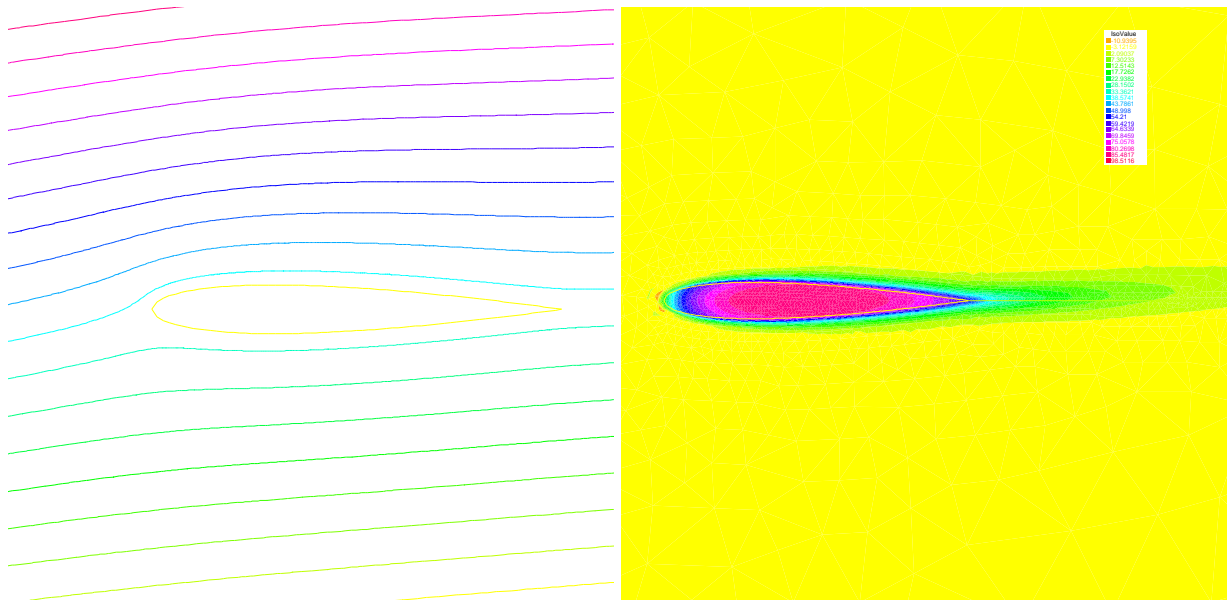


Figure 3.5: Zoom around the NACA0012 airfoil showing the streamlines (curve  $\psi = \text{constant}$ ). To obtain such a plot use the interactive graphic command: “+” and p. Right: temperature distribution at time  $T=25$  (now the maximum is at 90 instead of 120). Note that an incidence angle has been 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  $\kappa$  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  $\Omega$ ;
- 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  $\Omega$  be the unit disk centered at 0; consider the rotation vector field

$$\mathbf{u} = [u_1, u_2], \quad u_1 = y, \quad u_2 = -x.$$

Pure convection by  $\mathbf{u}$  is

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

The exact solution  $c(x_t, t)$  at time  $t$  en point  $x_t$  is given by

$$c(x_t, t) = c^0(x, 0)$$

where  $x_t$  is the particle path in the flow starting at point  $x$  at time 0. So  $x_t$  are solutions of

$$\dot{x}_t = \mathbf{u}(x_t), \quad , \quad x_{t=0} = x, \quad \text{where} \quad \dot{x}_t = \frac{d(t \mapsto x_t)}{dt}$$

The ODE are reversible and we want the solution at point  $x$  at time  $t$  ( not at point  $x_t$ ) the initial point is  $x_{-t}$ , and we have

$$c(x, t) = c^0(x_{-t}, 0)$$

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 compute  $c \circ X$  with  $X$  is the convect field defined by  $X(x) = x_{dt}$  and where  $x_\tau$  is particule path in the steady state velocity field  $\mathbf{u} = [u1, u2]$  starting at point  $x$  at time  $\tau = 0$ , so  $x_\tau$  is solution of the following ODE:

$$\dot{x}_\tau = u(x_\tau), \quad x_{\tau=0} = x.$$

When  $\mathbf{u}$  is piecewise constant; this is possible because  $x_\tau$  is then a polygonal curve which can be computed exactly and the solution exists always when  $u$  is divergence free; `convect` returns  $c(x_{df}) = C \circ X$ .

### 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 many formulations possible. We shall implement here the so-called dual- $P_1^{DC}$  formulation (see Ern[11]):

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

where  $E$  is the set of inner edges and  $E_T^-$  is the set of boundary edges where  $\mathbf{u} \cdot \mathbf{n} < 0$  (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)
int2d(Th)((cc/dt+(v1*dx(cc)+v2*dy(cc)))*w) // problem Aduel(cc,w) =
```

```

+ intalldges(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; Aduel;
  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, `intalldges` to integrate on all edges of all triangles

$$\text{intalldges}(\text{Th}) \equiv \sum_{T \in \text{Th}} \int_{\partial T} \quad (3.5)$$

(so all internal edges are seen two times), `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.

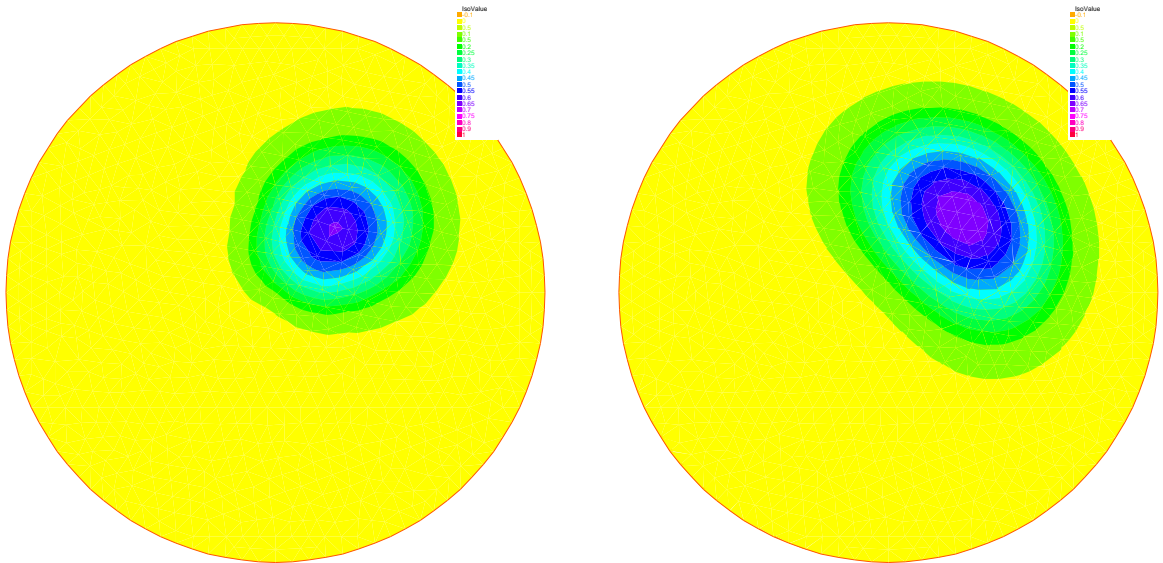


Figure 3.6: The rotated hill after one revolution, left with Characteristics-Galerkin, on the right with Discontinuous  $P_1$  Galerkin FEM.

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)
    + intalldedges(Th)((1-nTonEdge)*w*(a1*abs(n)-n/2)*jump(cc));
varf bbidual(ccold,w) = - int2d(Th)(ccold*w/dt);
matrix AA= aadual(Vh,Vh);
matrix BB = bbidual(Vh,Vh);
set (AA,init=t,solver=sparsesolver);
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 tell the solver that the matrix has not changed (`init=true`), and the name parameter are the same that in problem definition (see. 6.9) .

**Finite Volume Methods** can also be handled with FreeFem++ but it requires programming. For instance the  $P_0 - P_1$  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  $\sigma_i$  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 triangulations. We show here an example of a complex algorithm and our 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

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

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

where  $u \circ X(x) = u(x - u(x)\delta t)$  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  $\Omega$ )

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

and define

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

where  $\tilde{\mathbf{u}}$  is the  $(\mathbf{u}^{m+1}, \mathbf{v}^{m+1})$  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)

// file 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;
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;
}

```

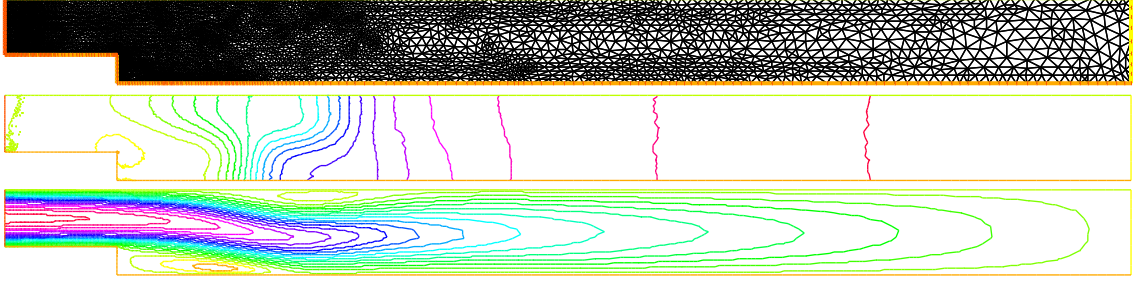


Figure 3.7: Rannacher's projection algorithm: result on an adapted mesh (top) showing the pressure (middle) and the horizontal velocity  $u$  at Reynolds 400.

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  $\epsilon(u) = \epsilon_{ij}(u)$

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

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

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

and where  $\lambda, \mu$  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  $\nu$ , 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  $\Omega$ . The components along  $x$  and  $y$  of the strain  $\mathbf{u}(x)$  in a section  $\Omega$  subject to forces  $\mathbf{f}$  perpendicular to the axis are governed by

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

where  $\lambda, \mu$  are the Lamé coefficients introduced above.

Remark, we do not use this equation because the associated variational form does not give the right boundary condition, we simply use

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

where the corresponding variational form is:

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

where  $:$  denote the tensor scalar product, i.e.  $a : b = \sum_{i,j} a_{ij} b_{ij}$ .

So the variational form can be written as :

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

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

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

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) == epsilon(u):epsilon(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

```

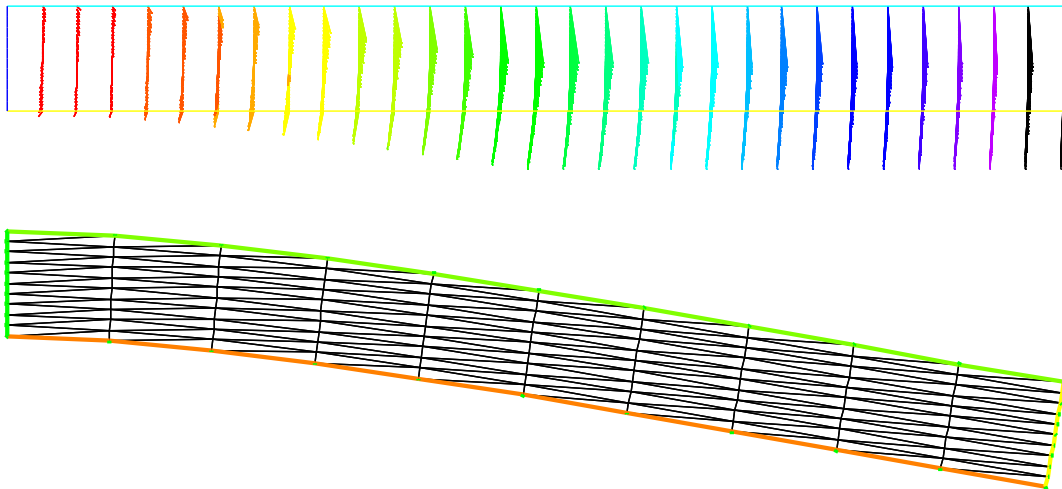


Figure 3.8: Solution of Lamé's equations for elasticity for a 2D beam deflected by its own weight and clamped by its left vertical side; result are shown with a amplification factor equal to 100. *Remark: the size of the arrow is automatically bound, but the color gives the real length*

## 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,

$$\begin{aligned} -\Delta \mathbf{u} + \nabla p &= 0 \\ \nabla \cdot \mathbf{u} &= 0 \end{aligned}$$

where  $\mathbf{u} = (u_1, u_2)$  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 finite element spaces for the LBB conditions to be satisfied:

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

```

// 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))
    - 1e-10*p*pp)
  + on(1,2,4,u=0,v=0) + on(3,u=1,v=0);
plot([u,v],p,wait=1);

```

Remark, we add a stabilization term  $-10e-10*p*pp$  to fix the constant part of the pressure.

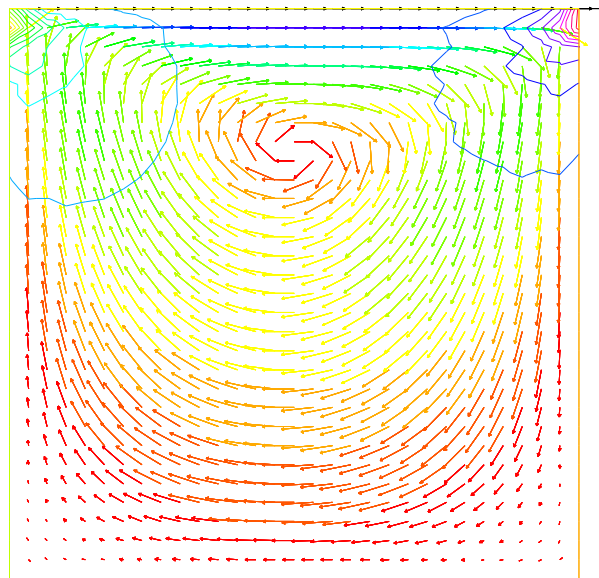


Figure 3.9: Solution of Stokes' equations for the driven cavity problem, showing the velocity field and the pressure level lines.

Results are shown on figure 3.9

### 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

$$\begin{aligned}
 \partial_t \theta + u \nabla \theta - \nabla \cdot (\kappa_T^m \nabla \theta) &= 0 \\
 \partial_t u + u \nabla u - \nabla \cdot (\mu_T \nabla u) + \nabla p + e(\theta - \theta_0) \mathbf{e}_2, \quad \nabla \cdot u &= 0 \\
 \mu_T = c_\mu \frac{k^2}{\epsilon}, \quad \kappa_T = \kappa \mu_T \\
 \partial_t k + u \nabla k + \epsilon - \nabla \cdot (\mu_T \nabla k) &= \frac{\mu_T}{2} |\nabla u + \nabla u^T|^2 \\
 \partial_t \epsilon + u \nabla \epsilon + c_2 \frac{\epsilon^2}{k} - \frac{c_\epsilon}{c_\mu} \nabla \cdot (\mu_T \nabla \epsilon) &= \frac{c_1}{2} k |\nabla u + \nabla u^T|^2 = 0
 \end{aligned} \tag{3.6}$$

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

$$\begin{aligned}
 \frac{1}{\delta t} (\theta^{m+1} - \theta^m \circ X^m) - \nabla \cdot (\kappa_T^m \nabla \theta^{m+1}) &= 0 \\
 \frac{1}{\delta t} (u^{m+1} - u^m \circ X^m) - \nabla \cdot (\mu_T^m \nabla u^{m+1}) + \nabla p^{m+1} + e(\theta^{m+1} - \theta_0) \mathbf{e}_2, \quad \nabla \cdot u^{m+1} &= 0 \\
 \frac{1}{\delta t} (k^{m+1} - k^m \circ X^m) + k^{m+1} \frac{\epsilon^m}{k^m} - \nabla \cdot (\mu_T^m \nabla k^{m+1}) &= \frac{\mu_T^m}{2} |\nabla u^m + \nabla u^{mT}|^2 \\
 \frac{1}{\delta t} (\epsilon^{m+1} - \epsilon^m \circ X^m) + c_2 \epsilon^{m+1} \frac{\epsilon^m}{k^m} - \frac{c_\epsilon}{c_\mu} \nabla \cdot (\mu_T^m \nabla \epsilon^{m+1}) &= \frac{c_1}{2} k^m |\nabla u^m + \nabla u^{mT}|^2 \\
 \mu_T^{m+1} = c_\mu \frac{k^{m+12}}{\epsilon^{m+1}}, \quad \kappa_T^{m+1} = \kappa \mu_T^{m+1}
 \end{aligned} \tag{3.7}$$

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 reynods=500;
// cout << " Enter the reynolds number :"; cin >> reynods;
assert(reynods>1 && reynods < 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./reynolds;

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,
    absserror=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  $\Gamma_1$  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.

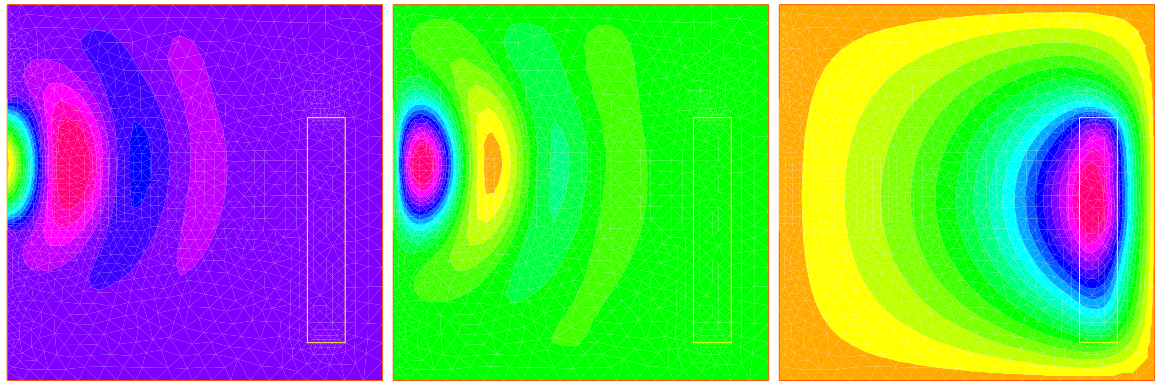


Figure 3.10: A microwave oven: real (left) and imaginary (middle) parts of wave and temperature (right).

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)

```
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;}
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;}
```

// file muwave.edp

```

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

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

where the desired state  $u_d$ , the boundary data  $u_{\Gamma}$  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  $\Omega$ .

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

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

Obviously  $J'_b$  is equal to  $\delta J$  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,uuh,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,u)
= 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  $\Omega$  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 that an *adjoint state* could have been used. Define  $p$  by

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

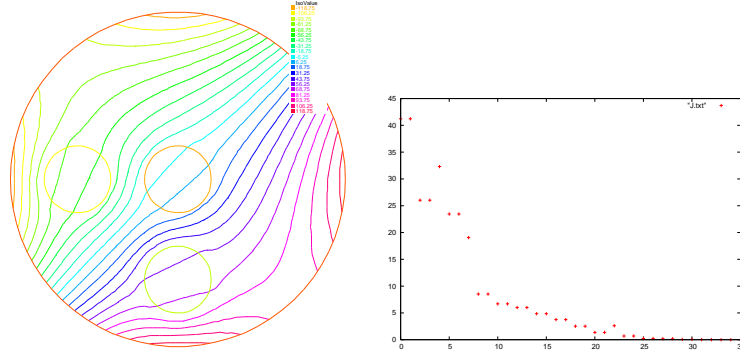


Figure 3.11: On the left the level lines of  $u$ . On the right the successive evaluations of  $J$  by BFGS (5 values above 500 have been removed for readability)

Consequently

$$\begin{aligned}\delta J &= - \int_{\Omega} (\nabla \cdot (\kappa \nabla p)) \delta u \\ &= \int_{\Omega} (\kappa \nabla p \cdot \nabla \delta u) = - \int_{\Omega} (\delta \kappa \nabla p \cdot \nabla u)\end{aligned}\quad (3.8)$$

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 characteristics provided that the mean values  $\bar{f} = \frac{1}{2}(f^+ + f^-)$  are used at shocks in the scheme.

$$\begin{aligned}\partial_t \rho + \bar{u} \nabla \rho + \bar{\rho} \nabla \cdot u &= 0 \\ \bar{\rho} (\partial_t u + \frac{\bar{\rho} u}{\bar{\rho}} \nabla u + \nabla p) &= 0 \\ \partial_t p + \bar{u} \nabla p + (\gamma - 1) \bar{p} \nabla \cdot u &= 0\end{aligned}\quad (3.9)$$

One possibility is to couple  $u, p$  and then update  $\rho$ , i.e.

$$\frac{1}{(\gamma - 1) \delta t \bar{p}^m} (p^{m+1} - p^m \circ X^m) + \nabla \cdot u^{m+1} = 0$$

$$\begin{aligned} \frac{\bar{\rho}^m}{\delta t}(u^{m+1} - u^m \circ \tilde{X}^m) + \nabla p^{m+1} &= 0 \\ \rho^{m+1} &= \rho^m \circ X^m + \frac{\bar{\rho}^m}{(\gamma - 1)\bar{p}^m}(p^{m+1} - p^m \circ X^m) \end{aligned} \quad (3.10)$$

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

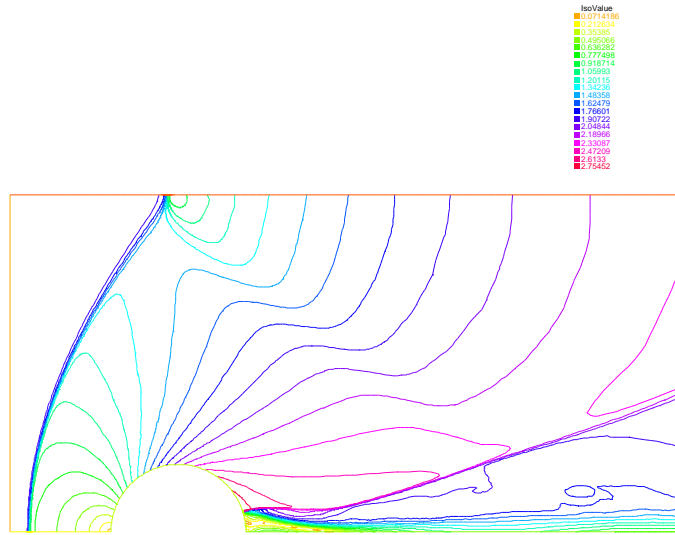


Figure 3.12: Pressure for a Euler flow around a disk at Mach 2 computed by (3.10)

```

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 aaal(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) + aaal(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);
fespace Vh(Th,P1);
Wh u,v,u1,v1,u1,v1;
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,u1,v1,rh)
= int2d(Th)( (u*u1+v*v1+r*rh)/dt
+ ((dx(r)*u1+ dy(r)*v1) - (dx(rh)*u + dy(rh)*v))
)
+ int2d(Th)(-(rh*convect([u1,v1],-dt,r1) + u1*convect([u1,v1],-dt,u1)
+ v1*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,
    absserror=1,nbjacoby=2, omega=1.8,ratio=1.8, nbsmooth=3,
    splitbedge=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.

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

The range of  $x$  over which the equation is taken, here  $\Omega$ , is called the *domain* of the PDE. The highest derivation index, here  $m$ , is called the *order*. If  $F$  and  $\varphi$  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  $\varphi$ . 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  $\sum_{i,j=1}^d a_{ij} b_{ij}$ .

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

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  $\varphi$  by 1,  $\partial \varphi / \partial x_i$  by  $z_i$  and  $\partial^2 \varphi / \partial x_i \partial x_j$  by  $z_i z_j$ , where  $z$  is a vector in  $\mathcal{R}^d$ :

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

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

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

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

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

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:

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

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:

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

**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  $\Omega$ , or on part of  $\Gamma$  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  $\Omega$  (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  $\varphi$  is a difficult art.

In general, an elliptic equation is well posed (*i.e.*  $\varphi$  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

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

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 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

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

**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).

**volume** give the volume of the current tetrahedra (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). (**istreamvalue**).

**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 realvalue approximation value of  $\pi$ .

## 4.4 System Commands

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

```
dumpable(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  $\max(a, b)$  or  $\min(a, b)$ . 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  $a + ib$  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  $(r, \phi)$ , So  $z$  is 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  $\phi$  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*  $x^y = \text{pow}(x,y) = x^y$ ; the *exponent function*  $\exp(x) (= e^x)$ ; the *logarithmic function*  $\log(x) (= \ln x)$  or  $\log_{10}(x) (= \log_{10} x)$ ; the *trigonometric functions*  $\sin(x)$ ,  $\cos(x)$ ,  $\tan(x)$  depending on angles measured by *radian*; the inverse of  $\sin x$ ,  $\cos x$ ,  $\tan x$  called *circular function* or also call *inverse trigonometric function*  $\text{asin}(x) (= \arcsin x)$ ,  $\text{acos}(x) (= \arccos x)$ ,  $\text{atan}(x) (= \arctan x)$ ; the  $\text{atan2}(x,y)$  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*,

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

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

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

The real function to round to integer are  $\text{floor}(x)$  round to largest integral value not greater than  $x$ ,  $\text{ceil}(x)$  round to smallest integral value not less than  $x$ ,  $\text{rint}(x)$  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 operations 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 is 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

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

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

## 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]); //      ] - \pi, \pi[2
fespace Vh(Th,P2);
func z=x+y*1i; //      z = x + iy
func f=imag(sqrt(z)); //      f = \Im \sqrt{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  $\text{fh}$  ( $=f_h$ ) is explained in Section 6.

**Note 4.1** The command `plot` is valid only for real or complex FE-functions (2d or 3d).

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]); // ] - pi, pi[ ^2
fespace Vh(Th,P2);
func z=x+y*1i; // z = x + iy
func f=imag(sqrt(z)); // f = \Im \sqrt{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

```

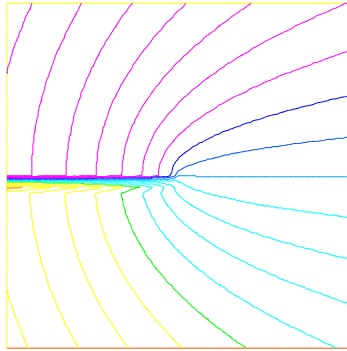


Figure 4.1:  $\Im \sqrt{z}$  has branch

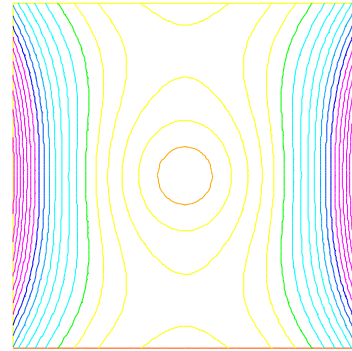


Figure 4.2:  $|\sin(z/10) \exp(z^2/10)|$

## 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 known 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

```

```

tab(10:11)=3.14;
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)
tab.sort ; // sort the array tab (version 2.18)
cout << " tab (after sort) " << tab << endl;
int[int] ii(0:d.n-1); // set array ii to 0,1, ..., d.n-1 (v3.2)
d=-1:-5; // set d to -1,-2, .. -5 (v3.2)

sort(d,ii); // sort array d and ii in parallele
cout << " d " << d << "\n ii = " << ii << endl;

```

produce the output

```

2.15 1.03 size of tab = 10 min: 1.03 max:2.15 sum : 11.42
resize tab: 12
    1.03    2.15    1.03    1.03    1.03
    1.03    1.03    1.03    1.03    1.03
    3.14    3.14
0  1.5
d = ( a ? b : c ) is 5
    3    3    2    3    3
tab (after sort) 12
1.03 1.03 1.03 1.03 1.03
1.03 1.03 1.03 1.03 2.15
3.14 3.14
d 5
-5 -4 -3 -2 -1

ii = 5
 4  3  2  1  0

```

You can set array like in matlab or scilab with operator `::`, the array generator of `a:c` is equivalent to `a:1:c`, and the array set by `a:b:c` is set to size  $\lfloor (b-a)/c \rfloor + 1$  and the value  $i$  is set by  $a + i(b-a)/c$ .

So you have of `int`, `real`, `complex` array with two operator (`.in`, `.re`) two get the real and imaginary array of complex array (without copy) :

```

// version 3.2 mai 2009
// like matlab. and scilab

{
int[int] tt(2:10); // 2,3,4,5,6,7,8,9,10
int[int] t1(2:3:10); // 2,5,8,

```

```

cout << " tt(2:10)= " << tt << endl;
cout << " t1(2:3:10)= " << t1 << endl;
tt=1:2:5;
cout << " 1.:2:5 => " << tt << endl;
}

{
real[int] tt(2:10); // 2,3,4,5,6,7,8,9,10
real[int] t1(2.:3:10.); // 2,5,8,
cout << " tt(2:10)= " << tt << endl;
cout << " t1(2:3:10)= " << t1 << endl;
tt=1.:0.5:3.999;
cout << " 1.:0.5:3.999 => " << tt << endl;
}
{
complex[int] tt(2.+0i:10.+0i); // 2,3,4,5,6,7,8,9,10
complex[int] t1(2.:3.:10.); // 2,5,8,
cout << " tt(2.+0i:10.+0i)= " << tt << endl;
cout << " t1(2.:3.:10.)= " << t1 << endl;
cout << " tt.re real part array " << tt.re << endl ;
// the real part array of the complex array
cout << " tt.im imag part array " << tt.im << endl ;
// the imag part array of the complex array
}

```

The output is :

```

tt(2:10)= 9
 2  3  4  5  6
 7  8  9 10
t1(2:3:10)= 3
 2  5  8
1.:2:5 => 3
 1  3  5
tt(2:10) = = 9
 2  3  4  5  6
 7  8  9 10
t1(2.:3:10.)= 3
 2  5  8
1.:0.5:3.999 => 6
 1 1.5  2 2.5  3
3.5
tt(2.+0i:10.+0i)= 9
(2,0) (3,0) (4,0) (5,0) (6,0)
(7,0) (8,0) (9,0) (10,0)
t1(2.:3.:10.);= 3
(2,0) (5,0) (8,0)

tt.re real part array  9
 2  3  4  5  6
 7  8  9 10
tt.im imag part array  9
 0  0  0  0  0
 0  0  0  0

```

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;
cout << "b*=a; b =" << b << endl;
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;
*/

cout << " ||a||_1 = " << a.l1 << endl;
cout << " ||a||_2 = " << a.l2 << endl;
cout << " ||a||_inf = " << a.linfty << endl;
cout << " sum a_i = " << a.sum << endl;
cout << " max a_i = " << a.max << endl;
cout << " min a_i = " << a.min << endl;
```

// same b = b .\* a

// same b = b ./ a

// ---- the methods --

```

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 statistical 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;
A(2,:) = 4;
A(:,1) = 5;
A(0:N-1,2) = 2;
A(1,0:2) = 3;

cout << " A = " << A << endl;

C = cb*cc';
C += 3*cb*cc';
C -= 5i*cb*cc';
cout << " C = " << C << endl;

matrix B;
B = A;
B=A(I,J);
B=A(I^-1,J^-1);

A = 2.*b*c';
cout << " A = " << A << endl;
B = b*c';
B = b*c';
B = (2*b*c')(I,J);
B = (3.*b*c')(I^-1,J^-1);
cout << "B = (3.*b*c')(I^-1,J^-1) = " << B << endl;

```

*// set the all matrix  
 // the full line 2  
 // the full column 1  
 // set the column 2  
 // set the line 1 from 0 to 2*

*// outer product*

*// the way to transform a array to a sparse matrix*

*// B(i,j)= A(I(i),J(j))  
 // B(I(i),J(j))= A(i,j)*

*// outer product*

*// outer product B(i,j) = b(i)\*c(j)  
 // outer product B(i,j) = b(i)\*c(j)  
 // outer product B(i,j) = b(I(i))\*c(J(j))  
 // outer product B(I(i),J(j)) = b(i)\*c(j)*

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 construction and setting

- To change the solver associated to a matrix do

```
set(M,solver=sparsesolver);
```

The default solver is GMRES.

- from a variationnal form: (see section 6.12 page 147 for details)

```

varf vDD(u,v) = int2d(Thm)(u*v*1e-10);
matrix DD=vDD(Lh,Lh);

```

- To set from a constant matrix

```

matrix A =
      [[ 0, 1, 0, 10],
       [ 0, 0, 2, 0],
       [ 0, 0, 0, 3],
       [ 4,0 , 0, 0]];

```

- To set from a block matrix

```

matrix M=[
      [ Asd[0] ,0      ,0      ,0      ,Csd[0] ],
      [ 0      ,Asd[1] ,0      ,0      ,Csd[1] ],
      [ 0      ,0      ,Asd[2] ,0      ,Csd[2] ],
      [ 0      ,0      ,0      ,Asd[3] ,Csd[3] ],
      [ Csd[0]' ,Csd[1]' ,Csd[2]' ,Csd[3]' ,DD      ]
];

```

```

//      to now to pack the right hand side
real[int] bb =[rhssd[0][], rhssd[1][],rhssd[2][],rhssd[3][],rhs1[] ];
set(M,solver=sparsesolver);
xx = M^-1 * bb;
[usd[0][],usd[1][],usd[2][],usd[3][],lh[]] = xx;
//      to dispatch
//      the solution on each part.

```

where here Asd and Csd are array of matrix (from example mortar-DN-4.edp of examples++-tutori

- 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

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

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 automatically 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.

- matrix renumbering

```
int[int] I(15),J(15);                                // two array for renumbering
                                                    //
                                                    // the way to transform a matrix to a sparse matrix

matrix B;
B = A;                                                // copie matrix 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)
B.resize(10,20);                                     // resize the sparse matrix and remove outbound term
```

where A is a given matrix.

### 4.8.3 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`)

#### 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;

```

*You can also resize a sparse matrix A*

```
A.resize(10,100);
```

*remark, the new size can be greater or lesser than the previous size, all new term are put to zero.*

On the triangulation of Figure 2.4 this produce the following:

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

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 = # Sparse Matrix (Morse)
# first line: n m (is symmetric) 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 1.0000000000000000199e+30
2      2 1.0000000000000000199e+30
3      3 1.0000000000000000199e+30
4      4 1.0000000000000000199e+30
5      5 1.0000000000000000199e+30
6      6 1.0000000000000000199e+30

```

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

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

But in examples++-load/lapack.edp you can inverse full matrix using lapack labrary and this small dynamic link interface (see for more detail section C page 291).

```
load "lapack"
load "fflapack"
int n=5;
real[int,int] A(n,n),A1(n,n),B(n,n);
for(int i=0;i<n;++i)
for(int j=0;j<n;++j)
  A(i,j)= (i==j) ? n+1 : 1;
cout << A << endl;
A1=A-1; // def in load "lapack"
cout << A1 << endl;

B=0;
for(int i=0;i<n;++i)
  for(int j=0;j<n;++j)
    for(int k=0;k<n;++k)
      B(i,j) +=A(i,k)*A1(k,j);
cout << B << endl; // A1+A-1; attention ne marche pas

inv(A1); // def in load "fflapack"
cout << A1 << endl;
```

and the output is:

```
5 5
 6  1  1  1  1
 1  6  1  1  1
 1  1  6  1  1
 1  1  1  6  1
 1  1  1  1  6

error: dgesv_ 0
5 5
0.18 -0.02 -0.02 -0.02 -0.02
-0.02 0.18 -0.02 -0.02 -0.02
-0.02 -0.02 0.18 -0.02 -0.02
-0.02 -0.02 -0.02 0.18 -0.02
-0.02 -0.02 -0.02 -0.02 0.18

5 5
 1 -1.387778781e-17 -1.040834086e-17 3.469446952e-17  0
-1.040834086e-17  1 -1.040834086e-17 -2.081668171e-17  0
3.469446952e-18 -5.551115123e-17  1 -2.081668171e-17 -2.775557562e-17
1.387778781e-17 -4.510281038e-17 -4.857225733e-17  1 -2.775557562e-17
-1.387778781e-17 -9.714451465e-17 -5.551115123e-17 -4.163336342e-17  1

5 5
 6  1  1  1  1
 1  6  1  1  1
```

```

1   1   6   1   1
1   1   1   6   1
1   1   1   1   6

```

to compile `lapack.cpp` or `fblas.cpp` you must have the library `lapack` on your system and try in directory `examples++-load`

```

ff-c++ lapack.cpp -llapack
ff-c++ fblas.cpp -llapack

```

#### 4.8.4 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<sup>1</sup>[26] 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; // 'tab tab
cout << tab << endl; cout << tab[1] << " "

```

<sup>1</sup>Standard template Library, now part of standard C++



```

<< tab[9] << endl; real[string] map; // a dynamic array
for (i=0;i<10;i=i+1)
{
    tab[i] = i*i;
    cout << i << " " << tab[i] << "\n";
};

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

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 opening 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 except 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 handling 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  $4 \times 5$  grid in the unit square  $[0, 1]^2$  whose labels are shown in Fig. 5.1. If you want to

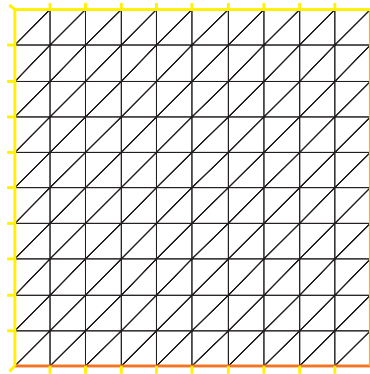


Figure 5.1: Boundary labels of the mesh by `square(10,10)`

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 named parameter `flags=icase` with `icase`:

- 0** will produce a mesh where all quad are slip with diagonal  $x - y = cte$
- 1** will produce Union Jack flag type of mesh.
- 2** will produce a mesh where all quad are slip with diagonal  $x + y = cte$  (v 3.8)
- 3** same as case 0 except in two corner such that no triangle with 3 vertices on boundary (v 3.8)
- 4** same as case 2 except in two corner such that no triangle with 3 vertices on boundary (v 3.8)

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

Adding the named parameter `label=labs`, you can change the 4 defaults label number to `labs[i-1]` where can be for for example `int[int] labs=[10,11,12,13,14]`, and adding the named parameter `region=10` to change the region number from 0 to 10 for example (v 3.8). To see all this fags, you can try the example `examples++/square-mesh.edp`:

```
for (int i=0;i<5;++i)
{
    int[int] labs=[11,12,13,14];
    mesh Th=square(3,3,flags=i,label=labs,region=10);
    plot(Th,wait=1,cmm=" square flags = "+i );
}
```

### 5.1.2 Border

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

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

We can easily check the orientation by drawing the curve  $t \mapsto (\varphi_x(t), \varphi_y(t))$ ,  $t_0 \leq t \leq t_1$ . 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  $\Gamma_j$  can only intersect at their end points.

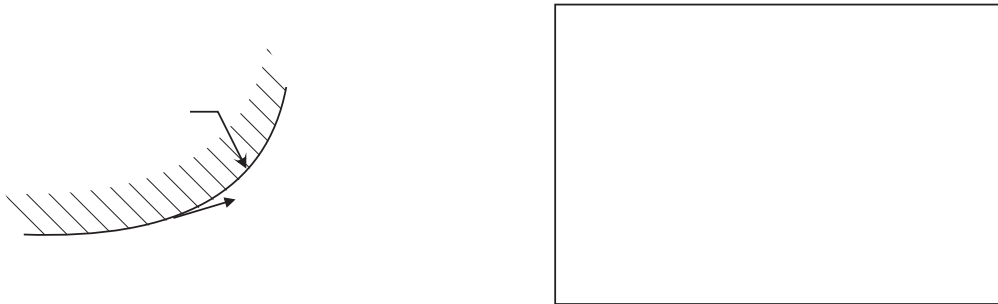


Figure 5.2: Orientation of the boundary defined by  $(\phi_x(t), \phi_y(t))$

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



**mesh** Mesh\_Name = **buildmesh**( $\Gamma_1(m_1) + \dots + \Gamma_J(m_J)$  OptionalParameter);

where  $m_j$  are positive or negative numbers to indicate how many points should be put on  $\Gamma_j$ ,  $\Gamma = \bigcup_{j=1}^J \Gamma_j$ , and the optional parameter (separated with comma) can be

**nbvx**=<int value> , to set the maximal number of vertices in the mesh.

**fixeborder**=<bool value> , to say if the mesh generator can change the boundary mesh or not (by default the boundary mesh can change and in case of periodic boundary problem (see. 6), it can be dangerous).

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 assigns “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 Thwithoutohole= buildmesh(a(50)+b(+30));
5: mesh Thwithhole = buildmesh(a(50)+b(-30));
6: plot(Thwithoutohole,wait=1,ps="Thwithoutohole.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.

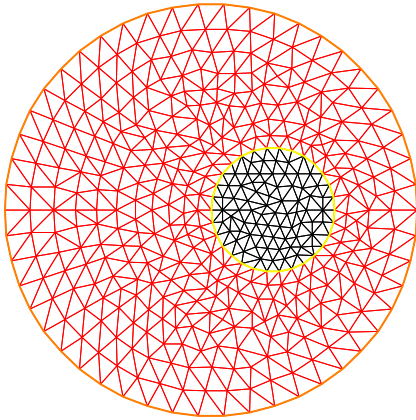


Figure 5.3: mesh without hole

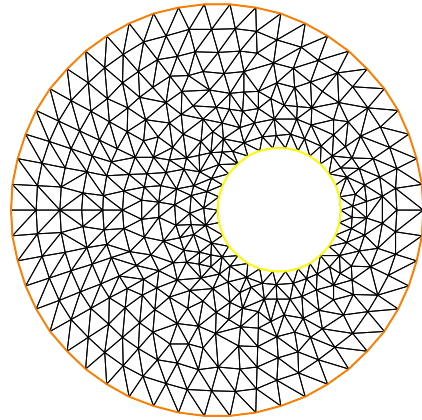


Figure 5.4: mesh with hole

**Note 5.3** The border is evaluated only at the time **plot** or **buildmesh** is called so the global variables are defined at this time and the following code can not work:

```
real r=1; border a(t=0,2*pi){ x=r*cos(t); y=r*sin(t);label=1;}
r=0.3 ; border b(t=0,2*pi){ x=r*cos(t); y=r*sin(t);label=1;}
mesh Thwithhole = buildmesh(a(50)+b(-30)); // bug (a trap) because
// the two circles have the same radius = 0.3
```

### 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  $q^i$ ,  $i = 1, \dots, n_v$ , we denote by  $(q_x^i, q_y^i)$  the  $x$ -coordinate and  $y$ -coordinate.

Each triangle  $T_k, k = 1, \dots, 10$  has three vertices  $q^{k_1}, q^{k_2}, q^{k_3}$  that are oriented counterclockwise.

The boundary consists of 10 lines  $L_i, i = 1, \dots, 10$  whose end points are  $q^{i_1}, q^{i_2}$ .

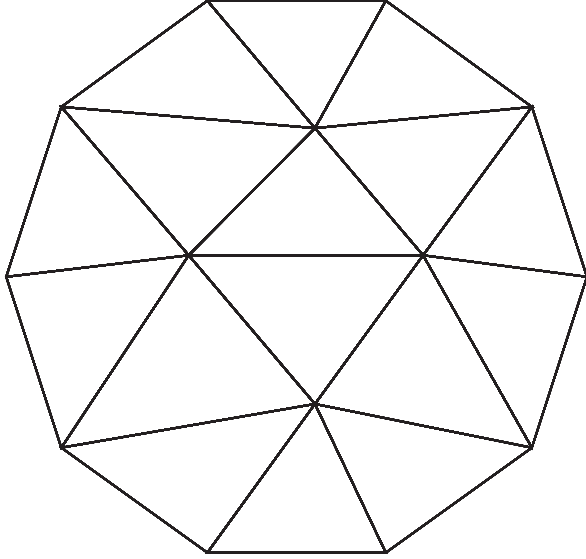


Figure 5.5: mesh by buildmesh(C(10))

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  $q^{10}, q^6$ .

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 bidimensional 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;};
```

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
... .. ...	
-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
...	
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
...	
10 6 1	$10_1$ $10_2$ boundary label=1

Table 5.1: The structure of “mesh\_sample.msh”

```

border ceiling(t=1,0){ x=t; y=1; label=5;};
border left(t=1,0){ x=0; y=t; label=5;};
int n=10;
mesh th= buildmesh(floor(n)+right(n)+ceiling(n)+left(n));
savemesh(th,"toto.am_fmt");           // "formatted Marrocco" format
savemesh(th,"toto.Th");               // "bamg"-type mesh
savemesh(th,"toto.msh");              // freefem format
savemesh(th,"toto.nopo");             // modulef format see [10]
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 "formatted Marrocco"
savemesh(th,"toto.Th");               // format database db mesh "bamg"
savemesh(th,"toto.msh");              // format freefem
savemesh(th,"toto.nopo");             // modulef format see [10]
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
//  $u + \Delta u = g$  in  $\Omega$  ,
//  $u = 0$  on  $\Gamma_1$  and  $\frac{\partial u}{\partial n} = g$  on  $\Gamma_2$ 
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) //  $\frac{\partial u}{\partial n} = g$  on  $\Gamma_2$ 
+ 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 vertex 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 containing 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;

//      new method to get boundary inforamtion and mesh adjacent

int k=0,l=1,e=1;
Th.nbe ; //      return the number of boundary element
Th.be(k); //      return the boundary element  $k \in \{0, \dots, Th.nbe - 1\}$ 
Th.be(k)[1]; //      return the vertices  $l \in \{0,1\}$  of boundary elmt k
Th.be(k).Element ; //      return the triangle contening the boundary elmt k
Th.be(k).whoinElement ; //      return the edge number of triangle contening
                                           //      the boundary elmt k
Th[k].adj(e) ; //      return adjacent triangle to k by edge e, and change
               //      the value of e to the corresponding edge in the adjacent triangle
Th[k] == Th[k].adj(e) //      non adjacent triangle return the same
Th[k] != Th[k].adj(e) //      true adjacent triangle

cout << " print mesh connectivity " << endl;
int nbelement = Th.nt;
for (int k=0;k<nbelement;++k)
    cout << k << " : " << int(Th[k][0]) << " " << int(Th[k][1])
        << " " << int(Th[k][2])
        << " , label " << Th[k].label << endl;
//

for (int k=0;k<nbelement;++k)
    for (int e=0,ee;e<3;++e)
        //      remark FH hack: set ee to e, and ee is change by method adj,
        //      in () to make difference with named parameters.
        cout << k << " " << e << " <=> " << int(Th[k].adj((ee=e))) << " " << ee
            << " adj: " << ( Th[k].adj((ee=e)) != Th[k]) << endl;
        //      note : if k == int(Th[k].adj(ee=e)) not adjacent element

int nbboundaryelement = Th.nbe;

for (int k=0;k<nbboundaryelement;++k)
    cout << k << " : " << Th.be(k)[0] << " " << Th.be(k)[1] << " , label "
        << Th.be(k).label << " tria " << int(Th.be(k).Element)
        << " " << Th.be(k).whoinElement << 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

print mesh connectivity
0 : 0 1 4 , label 0
1 : 0 4 3 , label 0
...
6 : 4 5 8 , label 0
7 : 4 8 7 , label 0
0 0 <=> 3 1 adj: 1
0 1 <=> 1 2 adj: 1
0 2 <=> 0 2 adj: 0
...
6 2 <=> 3 0 adj: 1
7 0 <=> 7 0 adj: 0
7 1 <=> 4 0 adj: 1
7 2 <=> 6 1 adj: 1
0 : 0 1 , label 1 tria 0 2
1 : 1 2 , label 1 tria 2 2
...
6 : 0 3 , label 4 tria 1 1
7 : 3 6 , label 4 tria 5 1
```

### 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 from a table function. The coordinates of the points and the value of the table function are defined separately with rows of the form:  $x \ y \ f(x,y)$  in a file such as:

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

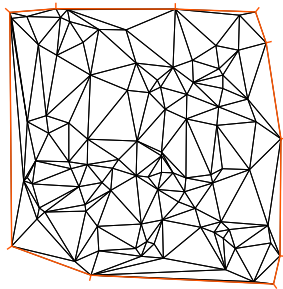


Figure 5.6: Delaunay mesh of the convex hull of point set in file xyf

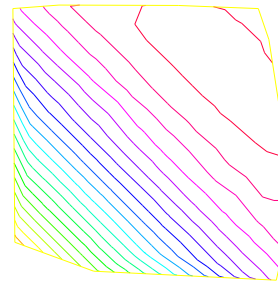


Figure 5.7: Isovalue of table function

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

The following example shows how to make a mesh from the file "xyf" with the format stated just above. The command `triangulate` command use only use 1st and 2nd rows.

```
mesh Thxy=triangulate("xyf");           // build the Delaunay mesh of the convex hull
// points are defined by the first 2 columns of file xyf
plot(Thxy,ps="Thxyf.ps");               // (see figure 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][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 defined 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 Multiplier 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 Multiplier space
// build a mesh without internal point
// of pseudo sub domain
// -----

assert(version>=1.40);
mesh Th=square(10,10);
int[int] ssd(Th.nt);
for(int i=0;i<ssd.n;i++) // build the pseudo region numbering
{ int iq=i/2; // because 2 triangle per quad
int ix=iq%10; //
int iy=iq/10; //
ssd[i]= 1 + (ix>=5) + (iy>=5)*2;
}
Th=emptymesh(Th,ssd); // build empty with
// all edge  $e = T1 \cap T2$  and  $ssd[T1] \neq ssd[T2]$ 

plot(Th,wait=1,ps="emptymesh-2.eps"); // see figure 5.9
savemesh(Th,"emptymesh-2.msh");
}
```



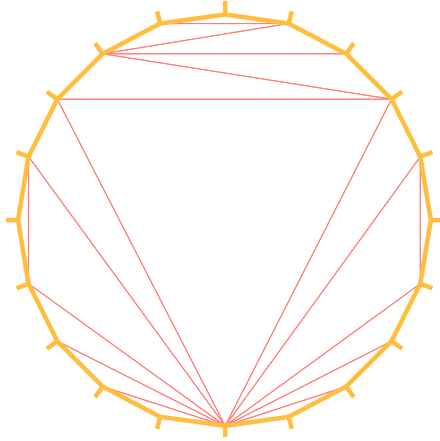


Figure 5.8: The empty mesh with boundary

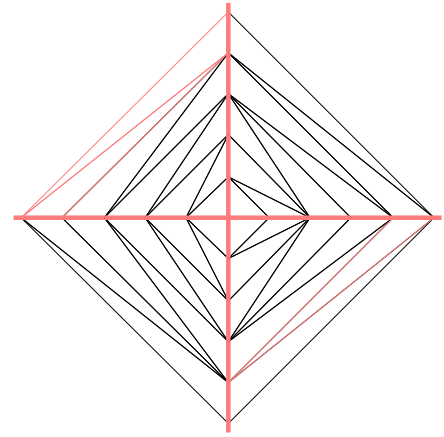


Figure 5.9: An empty mesh defined from a pseudo region numbering of triangle

## 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  $\Phi(x, y) = (\Phi_1(x, y), \Phi_2(x, y))$  of shape. It is also useful to handle free boundary problems or optimal shape problems.

If  $\Omega$  is triangulated as  $T_h(\Omega)$ , and  $\Phi$  is a displacement vector then  $\Phi(T_h)$  is obtained by

```
mesh Th=movemesh(Th,[Phi1,Phi2]);
```

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

```

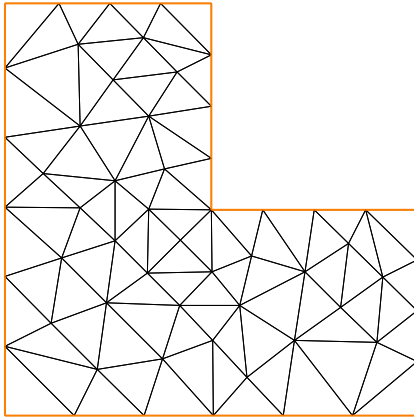


Figure 5.10: L-shape

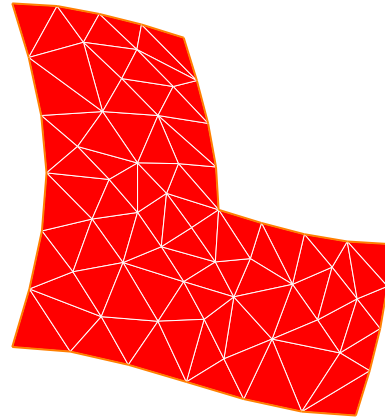


Figure 5.11: moved L-shape

**Note 5.4** Consider a function  $u$  defined on a mesh  $Th$ . A statement like  $Th=movemesh(Th\dots)$  does not change  $u$  and so the old mesh still exists. It will be destroyed when no function use it. A statement like  $u = u$  redefines  $u$  on the new mesh  $Th$  with interpolation and therefore destroys the old  $Th$  if  $u$  was the only function using it.

**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

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[];
u=0;
u[]=tmp;
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

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

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

1.

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

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

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

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

We put  $h(\mathcal{T}_h) = \max\{\text{diam}(T_k) | T_k \in \mathcal{T}_h\}$ , 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

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

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;
    plot(Th, fh, wait=1);
}

```

*// old mesh is deleted*

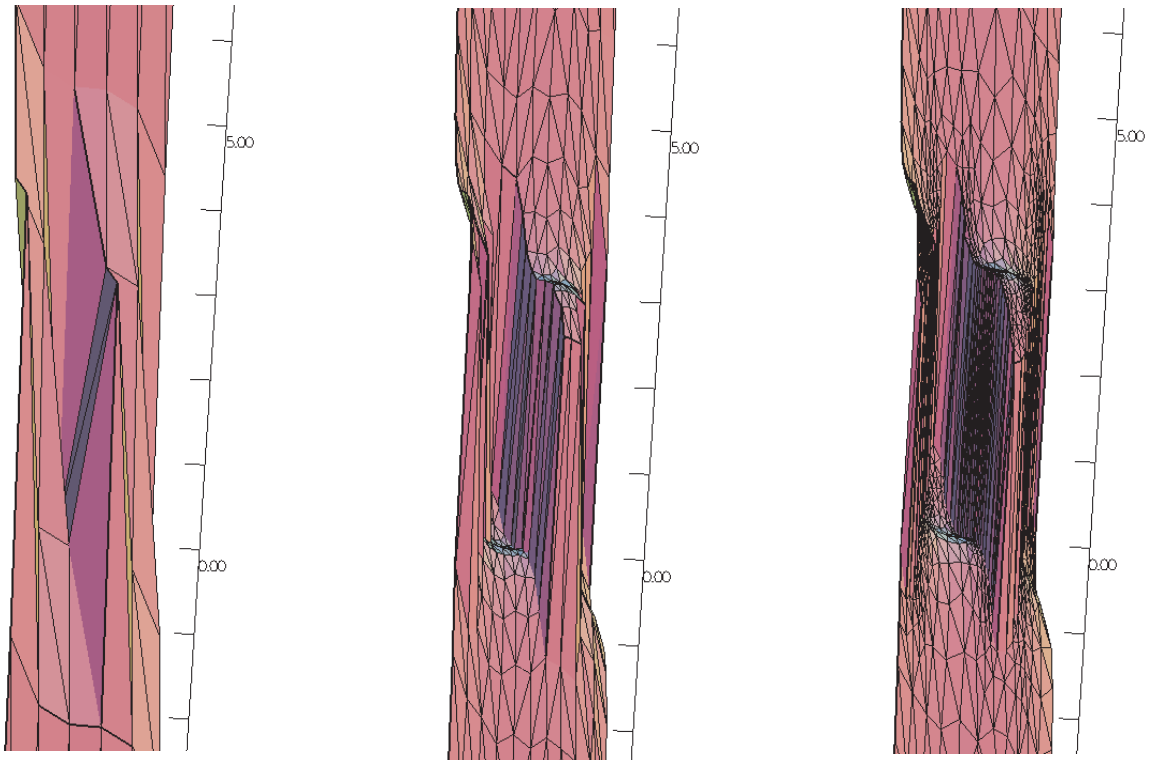


Figure 5.12: 3D graphs for the initial mesh and 1st and 2nd mesh adaptation

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

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

create the new mesh ATh by the Hessian

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

of a function (formula or FE-function). Mesh adaptation is a very powerful tool when the solution of a problem vary locally and sharply.

Here we solve the problem (2.1)-(2.2), when  $f = 1$  and  $\Omega$  is a L-shape domain.

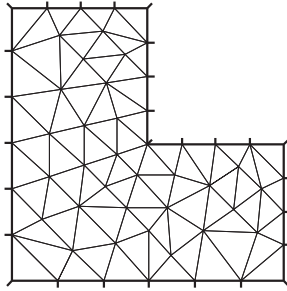


Figure 5.13: L-shape domain and its boundary name

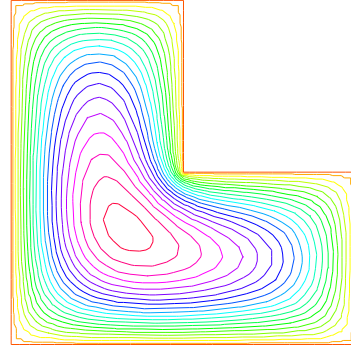


Figure 5.14: Final solution after 4-times adaptation

**Example 5.5 (Adapt.edp)** *The solution has the singularity  $r^{3/2}$ ,  $r = |x - \gamma|$  at the point  $\gamma$  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) ;
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 [9]. It has a number of default parameters which can be modified :

Si  $f1, f2$  sont des fonctions et  $thold$ ,  $Thnew$  des maillages.

```
Thnew = adaptmesh(Thold, f1 ... );
Thnew = adaptmesh(Thold, f1,f2 ... );
Thnew = adaptmesh(Thold, [f1,f2] ... );
```

Les paramters additionnels de adaptmesh représente par les "..."

**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).

**nbjacob=** 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-repartition of relative error (false is the default). In this case the metric is defined by

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

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

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

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

**verbosity=** informational messages level (can be chosen between 0 and  $\infty$ ). 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  $f_{xx} (= \partial^2 f / \partial x^2)$ ,  $f_{xy} (= \partial^2 f / \partial x \partial y)$ ,  $f_{yy} (= \partial^2 f / \partial y^2)$  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 constant mesh size.

So to build 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");

```

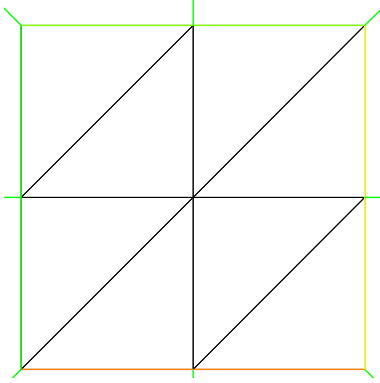


Figure 5.15: Initial mesh

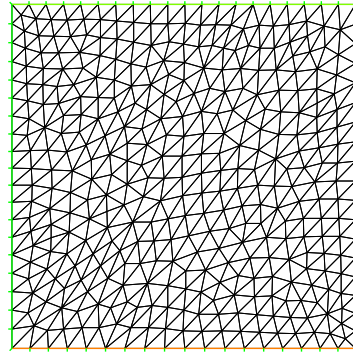


Figure 5.16: first iteration

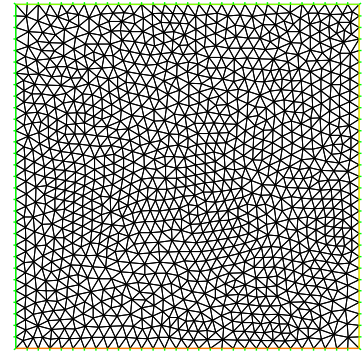


Figure 5.17: last iteration

## 5.6 Trunc

Two operators have been introduced 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 all triangles of a mesh `Th` are splitted in  $3 \times 3$  , just write:

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

The `truncmesh.edp` example constructs all "trunc" mesh to the support of the basic function of the space  $V_h$  (cf.  $\text{abs}(u) > 0$ ), split all the triangles in  $5 \times 5$ , 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");
u[][i]=0;
}

```

*// plot the mesh of  
 // the function's support  
 // reset*

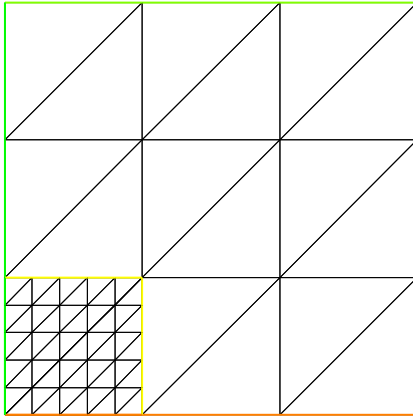


Figure 5.18: mesh of support the function P1 number 0, splitted in 5×5

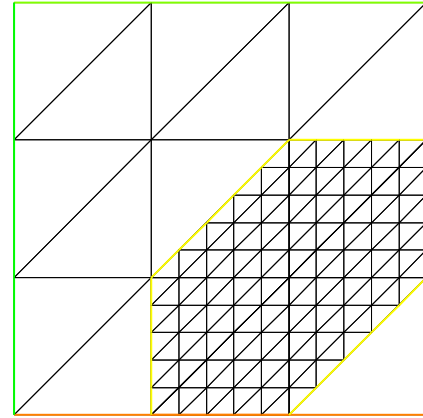


Figure 5.19: mesh of support the function P1 number 6, splitted in 5×5

## 5.7 Splitmesh

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

```

{
    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");
    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");
}

```

*// new stuff 2004 splitmesh (version 1.37)*  
*// see figure 5.20*  
*// see figure 5.21*

## 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;};

```

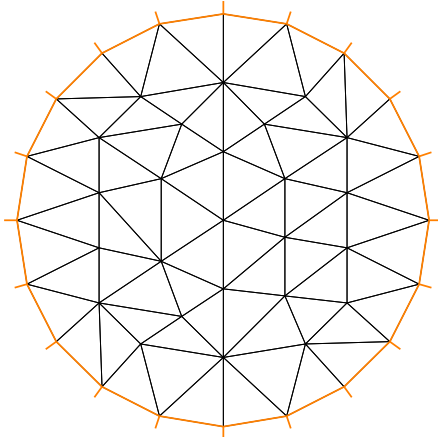
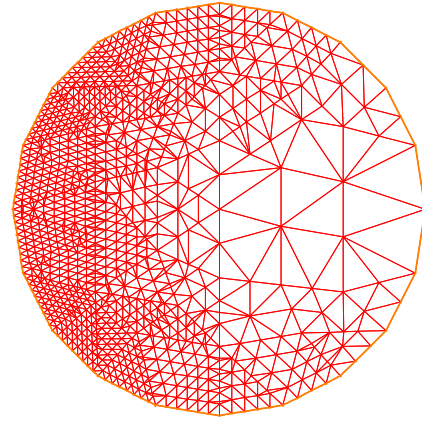


Figure 5.20: initial mesh

Figure 5.21: all left mesh triangle is split conformally in  $\text{int}(1+5*(\text{square}(x-0.5)+y*y)^2)$  triangles.

```

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

**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

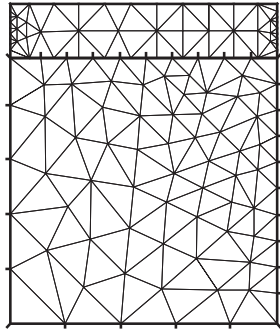


Figure 5.22: Two rectangles touching by a side

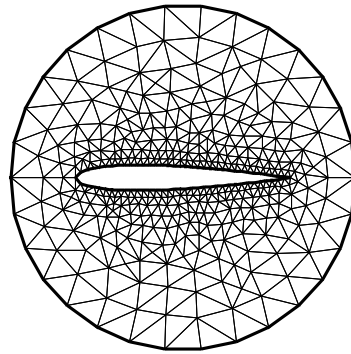


Figure 5.23: NACA0012 Airfoil

**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

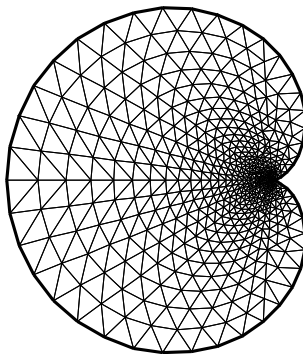


Figure 5.24: Domain with Cardioid curve boundary

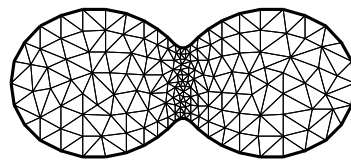


Figure 5.25: Domain with Cassini Egg curve boundary

**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

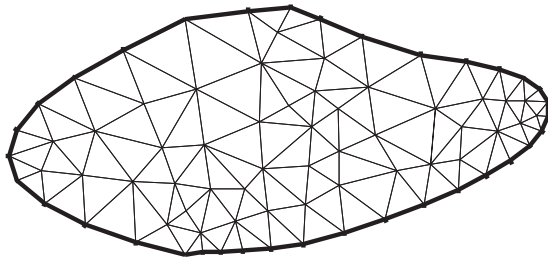


Figure 5.26: Boundary drawn by Bezier curves

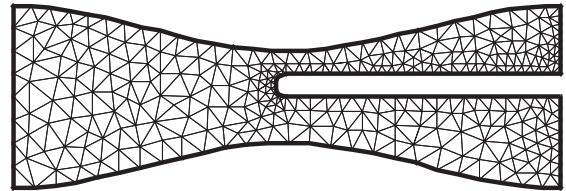


Figure 5.27: Section of Engine

**Example 5.13 (Domain with U-shape channel)**

```

real d = 0.1;
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; }

```

// width of U-shape

```

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;
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);

```

// angle of V-shape  
Fig. 5.29

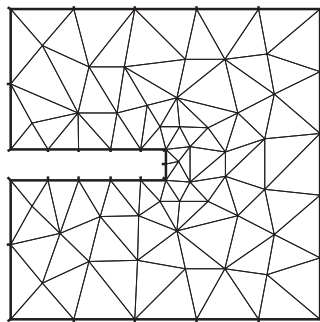


Figure 5.28: Domain with U-shape channel changed by d

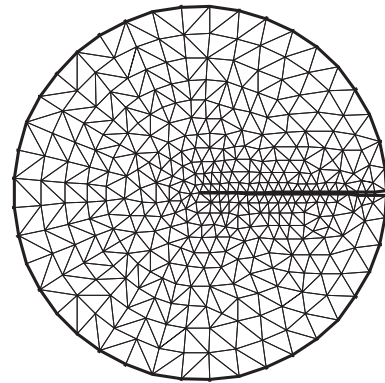


Figure 5.29: Domain with V-shape cut changed by dAg

#### 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

```

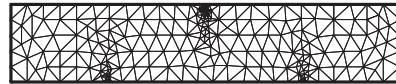
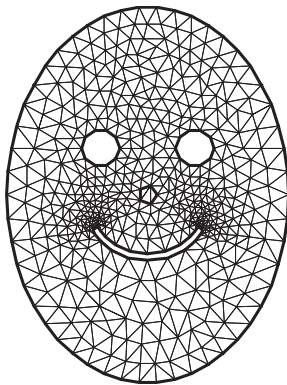


Figure 5.31: Domain for three-point bending test

Figure 5.30: Smiling face (Mouth is changeable)

## 5.9 How to change the label of elements and border elements of a mesh in FreeFem++ ?

Changing the label of elements and border elements will be done using the keyword **change**. The parameters for this command line are for a two dimensional and dimensional case:

**label** = is a vector of integer that contains successive pair of the old label number to the new label number .

**region** = is a vector of integer that contains successive pair of the old region number to new region number.

These vectors are composed of  $n_l$  successive pair of number  $O, N$  where  $n_l$  is the number (label or region) that we want to change. For example, we have

$$\text{label} = [O_1, N_1, \dots, O_{n_l}, N_{n_l}] \quad (5.3)$$

$$\text{region} = [O_1, N_1, \dots, O_{n_l}, N_{n_l}] \quad (5.4)$$

where  $O_i$  is the  $i^{\text{nd}}$  old number (label or region) to change in new number  $N_i$ .

An example of using this function is given in "glumesh2D.edp":

### Example 5.17 (glumesh2D.edp)

```

1:
2: mesh Th1=square(10,10);
3: mesh Th2=square(20,10,[x+1,y]);
4: verbosity=3;
5: int[int] r1=[2,0], r2=[4,0];
6: plot(Th1,wait=1);
7: Th1=change(Th1,label=r1); // Change the label of Edges of Th1 with label 2 in 0.
8: plot(Th1,wait=1);
9: Th2=change(Th2,label=r2); // Change the label of Edges of Th2 with label 4 in 0.
10: mesh Th=Th1+Th2; // "gluing together" of meshes Th1 and Th2
11: cout << " nb lab = " << int1d(Th1,1,3,4)(1./lenEdge)+int1d(Th2,1,2,3)(1./lenEdge)
12:      << " == " << int1d(Th,1,2,3,4)(1./lenEdge) <<" == " << ((10+20)+10)*2 << endl;
13: plot(Th,wait=1);
14: fespace Vh(Th,P1);
15: macro Grad(u) [dx(u),dy(u)]; // definition of a macro
16: Vh u,v;
17: solve P(u,v)=int2d(Th)(Grad(u)'*Grad(v))-int2d(Th)(v)+on(1,3,u=0);
18: plot(u,wait=1);

```

**“gluing” different mesh** In line 10 of previous file, the method to “gluing” different mesh of the same dimension in FreeFem++ is using. This function is just call using the symbol addition “+” between meshes. The method implemented need that the point in adjacent mesh are the same.

## 5.10 Mesh in three dimension

### 5.10.1 Read/Write Statements for a Mesh in 3D

In three dimension, the file mesh format supported for input and output files by FreeFem++ are the extension .msh and .mesh. These format are described in Chapter Mesh Files in two dimension.

**extension file .msh** The structure of the extension .msh in three dimensional are given in Table 5.2. In this structure,  $n_v$  denotes the number of vertices,  $n_{tet}$  the number of tetrahedra and  $n_{tri}$  the number of triangles. For each vertex  $q^i$ ,  $i = 1, \dots, n_v$ , we denote by  $(q_x^i, q_y^i, q_z^i)$  the  $x$ -coordinate, the  $y$ -coordinate and the  $z$ -coordinate. Each tetrahedra  $T_k$ ,  $k = 1, \dots, n_{tet}$  has four vertices  $q^{k_1}, q^{k_2}, q^{k_3}, q^{k_4}$ . The boundary consists of an union of triangles. Each triangle  $be_j$ ,  $j = 1, \dots, n_{tri}$  has three vertices  $q^{j_1}, q^{j_2}, q^{j_3}$ .

$n_v$	$n_{tet}$	$n_{tri}$		
$q_x^1$	$q_y^1$	$q_z^1$	Vertex label	
$q_x^2$	$q_y^2$	$q_z^2$	Vertex label	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$q_x^{n_v}$	$q_y^{n_v}$	$q_z^{n_v}$	Vertex label	
1 <sub>1</sub>	1 <sub>2</sub>	1 <sub>3</sub>	1 <sub>4</sub>	region label
2 <sub>1</sub>	2 <sub>2</sub>	2 <sub>3</sub>	2 <sub>4</sub>	region label
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(n_{tet})_1$	$(n_{tet})_2$	$(n_{tet})_3$	$(n_{tet})_4$	region label
1 <sub>1</sub>	1 <sub>2</sub>	1 <sub>3</sub>	boundary label	
2 <sub>1</sub>	2 <sub>2</sub>	2 <sub>3</sub>	boundary label	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$(n_{tri})_1$	$(n_{tri})_2$	$(n_{tri})_3$	boundary label	

Table 5.2: The structure of mesh file format “.msh” in three dimension.

**extension file .mesh** The data structure for a three dimensional mesh is composed of the data structure presented in Section 12.1 and a data structure for tetrahedrons. The tetrahedrons of a three dimensional mesh are refereed using the following field:

- Tetrahedra  
 $(I) \text{ NbOfTetrahedrons}$   
 $((@Vertex_i^j, j=1,4), (I) Ref\phi_i^{tet}, i=1, \text{NbOfTetrahedrons})$

This field is express with the notation of Section 12.1.

### 5.10.2 TeGen: A tetrahedral mesh generator

**TetGen** TetGen is a software developed by Hang Si of Weierstrass Institute for Applied Analysis and Stochastics of Berlin in Germany [36]. TetGen is a free for research and non-commercial uses. For any commercial licence utilization, a commercial licence is available upon request of Hang Si.

This software is a tetrahedral mesh generator of a three dimensional domain defined by its boundary. The input domain take into account a polyhedral or a piecewise linear complex. This tetrahelization is a constrained Delaunay tetrahelization.



The method used in TetGen to control the quality of the mesh is a Delaunay refinement from Shewchuk [37]. The quality measure of this algorithm is the Radius-Edge Ratio (see Section 1.3.1 [36] for more details). A theoretical bounds of this ratio of the algorithm of Shewchuk is obtained for a given complex of vertices, constrained segments and facets of surface mesh, with no input angle less than 90 degree. This theoretical bounds is 2.0.

The call of Tetgen is done with the keyword **tetg**. The parameters of this command line is:

**label** = is a vector of integer that contains the old labels number at index  $2i$  and the new labels number at index  $2i + 1$  of Triangles. This parameters is initialized as label for the keyword change (5.3).

**switch** = A string expression. This string corresponds to the command line switch of Tetgen see Section 3.2 of [36].

**nbofholes**= Number of holes.

**holelist** = This array correspond to **holelist** of tetgenio data structure [36]. A real vector of size  $3 \times \text{nbofholes}$ . In TetGen, each hole is associated with a point inside this domain. This vector is  $x_1^h, y_1^h, z_1^h, x_2^h, y_2^h, z_2^h, \dots$ , where  $x_i^h, y_i^h, z_i^h$  is the associated point with the  $i^{\text{th}}$  hole.

**nbofregions** = Number of regions.

**regionlist** = This array correspond to **regionlist** of tetgenio data structure [36]. The attribute and the volume constraint of region are given in this real vector of size  $5 \times \text{nbofregions}$ . The  $i^{\text{th}}$  region is described by five elements:  $x$ -coordinate,  $y$ -coordinate and  $z$ -coordinate of a point inside this domain ( $x_i, y_i, z_i$ ); the attribute ( $at_i$ ) and the maximum volume for tetrahedrons ( $mvol_i$ ) for this region. The **regionlist** vector is:  $x_1, y_1, z_1, at_1, mvol_1, x_2, y_2, z_2, at_2, mvol_2, \dots$ .

**nboffacetcl**= Number of facets constraints.

**facetcl**= This array correspond to **facetconstraintlist** of tetgenio data structure [36]. The  $i^{\text{th}}$  facet constraint is defined by the facet marker  $Ref_i^{fc}$  and the maximum area for faces  $marea_i^{fc}$ . The **facetcl** array is:  $Ref_1^{fc}, marea_1^{fc}, Ref_2^{fc}, marea_2^{fc}, \dots$ . This parameters has no effect if switch q is not selected.

Principal switch parameters in TetGen:

- p Tetrahedralization of boundary.
- q Quality mesh generation. The bound of Radius-Edge Ratio will be given after the option q. By default, this value is 2.0.
- a Construct with the volumes constraints on tetrahedrons. These volumes constraints are defined with the bound of the previous switch q or in the parameter **regionlist**.
- A Attributes reference to region given in the **regionlist**. The other regions have label 0. The option AA gives a different label at each region. This switch work with the option 'p'. If option 'r' is used, this switch has no effect.
- r Reconstructs and Refines a previously generated mesh. This character is only used with the command line **tetgreconstruction**.
- Y This switch allow to preserve the mesh on the exterior boundary. This switch must be used to ensure conformal mesh between two adjacents mesh.

- YY This switch allow to preserve the mesh on the exterior and interior boundary.
- C The consistency of the result's mesh is testing by TetGen.
- CC The consistency of the result's mesh is testing by TetGen and also checks constrained delaunay mesh (if 'p' switch is selected) or the consistency of Conformal Delaunay (if 'q' switch is selected).
- V Give information of the work of TetGen. More information can be obtained in specified 'VV' or 'VVV'.
- Q Quiet: No terminal output except errors
- M The coplanar facets are not merging.
- T Set a tolerance for coplanar test. The default value is  $1e - 8$ .
- d Intersections of facets are detected.

To obtain a tetrahedral mesh generator with tetgen, we need the surface mesh of three dimensional domain. We give now the command line in FreeFem++ to construct these meshes.

**keyword: “movemesh23”** A simple method to construct a surface is to place a two dimensional domain in a three dimensional space. This corresponding to move the domain by a displacement vector of this form  $\Phi(x, y) = (\Phi_1(x, y), \Phi_2(x, y), \Phi_3(x, y))$ . The result of moving a two dimensional mesh Th2 by this three dimensional displacement is obtained using:

```
mesh3 Th3 = movemesh23(Th2, transfo=[ $\Phi_1, \Phi_2, \Phi_3$ ]);
```

The parameters of this command line are:

**transfo** = [ $\Phi_1, \Phi_2, \Phi_3$ ] set the displacement vector of transformation  $\Phi(x, y) = [\Phi_1(x, y), \Phi_2(x, y), \Phi_3(x, y)]$ .

**label** = set integer label of triangles

**orientation=** set integer orientation of mesh.

**ptmerge** = A real expression. When you transform a mesh, some points can be merged. This parameters is the criteria to define two merging points. By default, we use

$$ptmerge = 1e - 7 \text{ Vol}(B),$$

where  $B$  is the smallest axis parallel boxes containing the discretize domain of  $\Omega$  and  $\text{Vol}(B)$  is the volume of this box.

We can “gluing” surface meshes using the process given in Section 5.9. An example of obtaining a three dimensional mesh using the command line tetg and movemesh23 is given in the file tetgencube.edp.

#### Example 5.18 (tetgencube.edp)

```

// file tetgencube.edp
load "msh3"
load "tetgen"

real x0,x1,y0,y1;
x0=1.; x1=2.; y0=0.; y1=2*pi;
mesh Thsq1 = square(5,35,[x0+(x1-x0)*x,y0+(y1-y0)*y]);
```

```

func ZZ1min = 0;
func ZZ1max = 1.5;
func XX1 = x;
func YY1 = y;

mesh3 Th31h = movemesh23(Thsq1,transfo=[XX1,YY1,ZZ1max]);
mesh3 Th31b = movemesh23(Thsq1,transfo=[XX1,YY1,ZZ1min]);

// //////////////////////////////////////

x0=1.; x1=2.; y0=0.; y1=1.5;
mesh Thsq2 = square(5,8,[x0+(x1-x0)*x,y0+(y1-y0)*y]);

func ZZ2 = y;
func XX2 = x;
func YY2min = 0.;
func YY2max = 2*pi;

mesh3 Th32h = movemesh23(Thsq2,transfo=[XX2,YY2max,ZZ2]);
mesh3 Th32b = movemesh23(Thsq2,transfo=[XX2,YY2min,ZZ2]);

// //////////////////////////////////////

x0=0.; x1=2*pi; y0=0.; y1=1.5;
mesh Thsq3 = square(35,8,[x0+(x1-x0)*x,y0+(y1-y0)*y]);
func XX3min = 1.;
func XX3max = 2.;
func YY3 = x;
func ZZ3 = y;

mesh3 Th33h = movemesh23(Thsq3,transfo=[XX3max,YY3,ZZ3]);
mesh3 Th33b = movemesh23(Thsq3,transfo=[XX3min,YY3,ZZ3]);

// //////////////////////////////////////
mesh3 Th33 = Th31h+Th31b+Th32h+Th32b+Th33h+Th33b; // "gluing" surface meshes to obtain
the surface of cube
savemesh(Th33,"Th33.mesh");

// build a mesh of a axis parallel box with TetGen
real[int] domain =[1.5,pi,0.75,145,0.0025];
mesh3 Thfinal = tetg(Th33,switch="paAAQY",nbofregions=1,regionlist=domain); //
Tetrahelize the interior of the cube with tetgen
savemesh(Thfinal,"Thfinal.mesh");

// build a mesh of a half cylindrical shell of interior radius 1. and exterior radius
2 and heigh 1.5
func mv2x = x*cos(y);
func mv2y = x*sin(y);
func mv2z = z;
mesh3 Thmv2 = movemesh3(Thfinal, transfo=[mv2x,mv2y,mv2z]);
savemesh(Thmv2,"halfcylindricalshell.mesh")

```

The command **movemesh** is describe in the following section.

**The keyword “tetgtransfo”** This keyword correspond to a composition of command line `tetg` and `movemesh23`:

`tetgtransfo( Th2, transfo= [Φ1, Φ2, Φ3] ), ... ) = tetg( Th3surf, ... )`,

where `Th3surf = movemesh23( Th2,transfo=[Φ1, Φ2, Φ3] )` and `Th2` is the input two dimensional mesh of `tetgtransfo`.

The parameters of this command line are on the one hand the parameters:

`label, switch, regionlist nboffacetcl facetcl`

of keyword `tetg` and on the other hand the parameter `ptmerge` of keyword `movemesh23`.

**Remark:** To use `tetgtransfo`, the result’s mesh of `movemesh23` must be an enclosed surface and defined one region. Therefore, the parameter `regionlist` is defined for one region.

An example of this keyword can be found in line of file “`buildlayers.edp`”

**The keyword “tetgconvexhull”** FreeFem++ , using `tetgen`, is able to build a tetrahelization from a set of points. This tetrahelization is a Delaunay mesh of the convex hull of the set of points.

The coordinates of the points can be initializes in two way. The first is a file that contains the coordinate of points  $X_i = (x_i, y_i, z_i)$ . This files is organized as follow:

$$\begin{array}{ccc} n_v & & \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_{n_v} & y_{n_v} & z_{n_v} \end{array}$$

The second way is to give three arrays that corresponding respectively to  $x$ -coordinates,  $y$ -coordinates and  $z$ -coordinates.

The parameters of this command line are

**switch** = A string expression. This string corresponds to the command line *switch* of TetGen see Section 3.2 of [36].

**reftet** = An integer expression. set the label of tetrahedrons.

**label** = An integer expression. set the label of triangles.

In the string `switch`, we can’t used the option ‘p’ and ‘q’ of `tetgen`.

### 5.10.3 Reconstruct/Refine a three dimensional mesh with TetGen

Meshes in three dimension can be refined using TetGen with the command line `tetgreconstruction`.

The parameter of this keyword are

**reftet**= an integer array that allow to change the label of tetrahedrons. This array is defined as the parameter `reftet` in the keyword `change`.

**label**= an integer array that allow to change the label of boundary triangles. This array is defined as the parameter `label` in the keyword `change`.

**sizevolume**= a function. This function allows to constraint volume size of tetrahedrons in the domain.

The parameter switch `nbofregions`, `regionlist`, `nboffacetcl` and `facetcl` of the command line which call TetGen (`tetg`) is used for `tetgrefine`.

In the parameter switch=, the character 'r' should be used without the character 'p'. For instance, see the manual of TetGen [36] for effect of 'r' to other character.

The parameter `regionlist` allow to define a new volume constraint in the region. The label in the `regionlist` will be the previous label of region. This parameter and `nbofregions` can't be used with parameter `sizevolume`.

Example:

#### Example 5.19 (refinesphere.edp)

// file refinesphere.edp

```
load "msh3"
load "tetgen"
load "medit"

mesh Th=square(10,20,[x*pi-pi/2,2*y*pi]); // ] $\frac{\pi}{2}$ ,frac-pi2[ $\times$ ]0,2 $\pi$ [
// a parametrization of a sphere

func f1 =cos(x)*cos(y);
func f2 =cos(x)*sin(y);
func f3 = sin(x);
// partiel derivative of the parametrization DF

func f1x=sin(x)*cos(y);
func f1y=-cos(x)*sin(y);
func f2x=-sin(x)*sin(y);
func f2y=cos(x)*cos(y);
func f3x=cos(x);
func f3y=0;
// M = DF'DF

func m11=f1x^2+f2x^2+f3x^2;
func m21=f1x*f1y+f2x*f2y+f3x*f3y;
func m22=f1y^2+f2y^2+f3y^2;

func perio=[[4,y],[2,y],[1,x],[3,x]];
real hh=0.1;
real vv= 1/square(hh);
verbosity=2;
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
plot(Th,wait=1);

verbosity=2;

// construction of the surface of spheres

real Rmin = 1.;
func f1min = Rmin*f1;
func f2min = Rmin*f2;
func f3min = Rmin*f3;

mesh3 Th3=movemesh23(Th,transfo=[f1min,f2min,f3min]);

real[int] domain = [0.,0.,0.,145,0.01];
mesh3 Th3sph=tetg(Th3,switch="paAAQYY",nbofregions=1,regionlist=domain);

int[int] newlabel = [145,18];
```

```

real[int] domainrefine = [0.,0.,0.,145,0.0001];
mesh3 Th3sphrefine=tetgreconstruction(Th3sph,switch="raAQ",reftet=newlabel,
nbofregions=1,regionlist=domainrefine);

int[int] newlabel2 = [145,53];
func fsize = 0.01/(( 1 + 5*sqrt( (x-0.5)^2+(y-0.5)^2+(z-0.5)^2 ) ^3);
mesh3 Th3sphrefine2=tetgreconstruction(Th3sph,switch="raAQ",reftet=newlabel2,
sizeofvolume=fsize);

medit('sphere',Th3sph);
medit('isotroperefine',Th3sphrefine);
medit('anisotroperefine',Th3sphrefine2);

```

### 5.10.4 Moving mesh in three dimension

Meshes in three dimension can be translated rotated and deformed using the command line `movemesh` as in the 2D case (see section `movemesh` in chapter 5). If  $\Omega$  is tetrahelized as  $T_h(\Omega)$ , and  $\Phi(x,y,z) = (\Phi_1(x,y,z), \Phi_2(x,y,z), \Phi_3(x,y,z))$  is a displacement vector then  $\Phi(T_h)$  is obtained by

```
mesh3 Th = movemesh( Th, transfo=[ $\Phi_1$ ,  $\Phi_2$ ,  $\Phi_3$ ], ... );
```

The parameters of `movemesh` in three dimension are

**transfo** = [ $\Phi_1, \Phi_2, \Phi_3$ ] set the displacement vector of 3D transformation [ $\Phi_1(x,y,z), \Phi_2(x,y,z), \Phi_3(x,y,z)$ ].

**region** = set integer label of tetrahedrons. 0 by default.

**label** = set the label of faces of border. This parameters is initialized as label for the keyword change (5.3).

**facemerge** = An integer expression. When you transform a mesh, some faces can be merged. This parameters equals to one if merge's faces is considered. Otherwise equals to zero. By default, this parameter is equals to 1.

**ptmerge** = A real expression. When you transform a mesh, some points can be merged. This parameters is the criteria to define two merging points. By default, we use

$$ptmerge = 1e - 7 \text{ Vol}(B),$$

where  $B$  is the smallest axis parallel boxes containing the discretion domain of  $\Omega$  and  $\text{Vol}(B)$  is the volume of this box.

An example of this command can be found in the file "Poisson3d.edp" situated in the directory `examples++-3d`.

### 5.10.5 Layer mesh

In this section, we present the command line to obtain a Layer mesh: `buildlayermesh`. This mesh is obtaining in extending a two dimensional mesh in the  $z$ -axis.

The domain  $\Omega_{3d}$  defined by the layer mesh is equals to  $\Omega_{3d} = \Omega_{2d} \times [zmin, zmax]$  where  $\Omega_{2d}$  is the domain define by the two dimensional mesh,  $zmin$  and  $zmax$  are function of  $\Omega_{2d}$  in  $R$  that defines respectively the lower surface and upper surface of  $\Omega_{3d}$ .

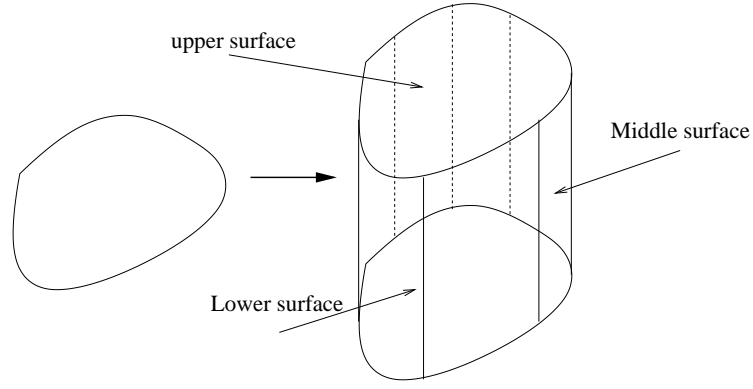


Figure 5.32: Example of Layer mesh in three dimension.

For a vertex of two dimensional mesh  $V_i^{2d} = (x_i, y_i)$ , we introduce the number of associated vertices in the  $z$ -axis  $M_i + 1$ . We denote by  $M$  the maximum of  $M_i$  over the vertices of the two dimensional mesh. This value are called the number of layers (if  $\forall i, M_i = M$  then there are  $M$  layers in the mesh of  $\Omega_{3d}$ ).  $V_i^{2d}$  generated  $M + 1$  vertices which are defined by

$$\forall j = 0, \dots, M, \quad V_{i,j}^{3d} = (x_i, y_i, \theta_i(z_{i,j})),$$

where  $(z_{i,j})_{j=0,\dots,M}$  are the  $M + 1$  equidistant points on the interval  $[zmin(V_i^{2d}), zmax(V_i^{2d})]$ :

$$z_{i,j} = j \delta\alpha + zmin(V_i^{2d}), \quad \delta\alpha = \frac{zmax(V_i^{2d}) - zmin(V_i^{2d})}{M}.$$

The function  $\theta_i$ , defined on  $[zmin(V_i^{2d}), zmax(V_i^{2d})]$ , is given by

$$\theta_i(z) = \begin{cases} \theta_{i,0} & \text{if } z = zmin(V_i^{2d}), \\ \theta_{i,j} & \text{if } z \in ]\theta_{i,j-1}, \theta_{i,j}], \end{cases}$$

with  $(\theta_{i,j})_{j=0,\dots,M_i}$  are the  $M_i + 1$  equidistant points on the interval  $[zmin(V_i^{2d}), zmax(V_i^{2d})]$ .

Set a triangle  $K = (V_{i1}^{2d}, V_{i2}^{2d}, V_{i3}^{2d})$  of the two dimensional mesh.  $K$  is associated with a triangle on the upper surface (resp. on the lower surface) of layer mesh:  $(V_{i1,M}^{3d}, V_{i2,M}^{3d}, V_{i3,M}^{3d})$  (resp.  $(V_{i1,0}^{3d}, V_{i2,0}^{3d}, V_{i3,0}^{3d})$ ). Also  $K$  is associated with  $M$  volume prismatic elements which are defined by

$$\forall j = 0, \dots, M, \quad H_j = (V_{i1,j}^{3d}, V_{i2,j}^{3d}, V_{i3,j}^{3d}, V_{i1,j+1}^{3d}, V_{i2,j+1}^{3d}, V_{i3,j+1}^{3d}).$$

Theses volume elements can be have some merged point:

- 0 merged point : prism
- 1 merged points : pyramid
- 2 merged points : tetrahedra
- 3 merged points : no elements

The volume with merged points are called degenerate elements. To obtain a meshing with tetrahedrons, we decompose the pyramid into two tetrahedrons and the prism into three tetrahedrons. These tetrahedrons are obtaining in cut the quadrilateral face of pyramid and prism with the diagonal with have the vertex with the maximum index (see [8] for explication of this choice).

The triangles on the middle surface obtained with the decomposition of the volume prismatic elements are the triangles generated by the edges on the border of the two dimensional mesh. The label of triangles on the border elements and tetrahedrons are defined with the label of these associated elements.

The arguments of `buildlayermesh` is a two dimensional mesh and the number of layers  $M$ . The parameters of this command are:

**zbound** = `[zmin,zmax]` where `zmin` and `zmax` are functions expression. These functions define the lower surface mesh and upper mesh of surface mesh.

**coef** = A function expression between `[0,1]`. This parameter is used to introduce degenerate element in mesh. The number of associated points of vertex  $V_i^{2d}$  is the integer part of  $coef(V_i^{2d})M$ .

**region** = This vector is used to initialize the region of tetrahedrons. This vector contains successive pair of the 2d region number at index  $2i$  and the corresponding 3d region number at index  $2i + 1$ , like (5.3). become the

**labelmid** = This vector is used to initialize the 3d labels number of the vertical face or mid face from the 2d label number. This vector contains successive pair of the 2d label number at index  $2i$  and the corresponding 3d label number at index  $2i + 1$ , like (5.3).

**labelup** = This vector is used to initialize the 3d labels number of the upper/top face from the 2d region number. This vector contains successive pair of the 2d region number at index  $2i$  and the corresponding 3d label number at index  $2i + 1$ , like (5.3).

**labeldown** = Same as previous case but just for the lower/down face label .

Moreover, we also add post processing parameters that allow to moving the mesh. These parameters correspond to parameters `transfo`, `facemerge` and `ptmerge` of the command line `movemesh`.

The vector `region`, `labelmid`, `labelup` and `labeldown` These vectors are composed of  $n_l$  successive pair of number  $O_i, N_l$  where  $n_l$  is the number (label or region) that we want to get.

An example of this command line is given in `buildlayermesh.edp`.

### Example 5.20 (cube.idp)

```
load "medit"
load "msh3"
func mesh3 Cube(int[int] & NN,real[int,int] &BB ,int[int,int] & L)
{
    //      first build the 6 faces of the hex.

    real x0=BB(0,0),x1=BB(0,1);
    real y0=BB(1,0),y1=BB(1,1);
    real z0=BB(2,0),z1=BB(2,1);

    int nx=NN[0],ny=NN[1],nz=NN[2];
    mesh Thx = square(nx,ny,[x0+(x1-x0)*x,y0+(y1-y0)*y]);

    int[int] rup=[0,L(2,1)], rdown=[0,L(2,0)],
    rmid=[1,L(1,0), 2,L(0,1), 3, L(1,1), 4, L(0,0) ];
    mesh3 Th=buildlayers(Thx,nz, zbound=[z0,z1],
    labelmid=rmid, labelup = rup,
    labeldown = rdown);

    return Th;
}
```



The unite cube example:

```
include "Cube.idp"
int [int] NN=[10,10,10];           // the number of step in each direction
real [int,int] BB=[[0,1],[0,1],[0,1]]; // bounding box
int [int,int] L=[[1,2],[3,4],[5,6]]; // the label of the 6 face left,right,
                                     // front, back, down, right

mesh3 Th=Cube(NN,BB,L);
medit("Th",Th);                    // see figure 5.33
```

The cone example (an axisymtric mesh on a triangle with degenerateness).

#### Example 5.21 (cone.edp) load "msh3"

```
load "medit"

// cone using buildlayers with a triangle

real RR=1,HH=1;
border Taxe(t=0,HH){x=t;y=0;label=0;};
border Hypo(t=1,0){x=HH*t;y=RR*t;label=1;};
border Vert(t=0,RR){x=HH;y=t;label=2;};
int nn=10; real h= 1./nn;
mesh Th2=buildmesh( Taxe(HH*nn)+ Hypo(sqrt(HH*HH+RR*RR)*nn) + Vert(RR*nn) );
plot(Th2,wait=1); // the 2d mesh

int MaxLayersT=(int(2*pi*RR/h)/4)*4; // number of layers
real zminT = 0, zmaxT = 2*pi; // height 2*pi
func fx= y*cos(z); func fy= y*sin(z); func fz= x;
int [int] r1T=[0,0], r2T=[0,0,2,2], r4T=[0,2];

// trick function:
func deg= max(.01,y/max(x/HH,0.4) /RR); // the function defined the proportion
// of number layer close to axis with reference MaxLayersT

mesh3 Th3T=buildlayers(Th2,coef= deg, MaxLayersT,
                      zbound=[zminT,zmaxT],transfo=[fx,fy,fz],
                      facemerge=0, region=r1T, labelmid=r2T);
medit("cone",Th3T); // see figure 5.34
```

#### Example 5.22 (buildlayermesh.edp)

```
// file buildlayermesh.edp

load "msh3"
load "tetgen"

// Test 1

int C1=99, C2=98; // could be anything
border C01(t=0,pi){ x=t; y=0; label=1;};
border C02(t=0,2*pi){ x=pi; y=t; label=1;};
border C03(t=0,pi){ x=pi-t; y=2*pi; label=1;};
border C04(t=0,2*pi){ x=0; y=2*pi-t; label=1;};
```

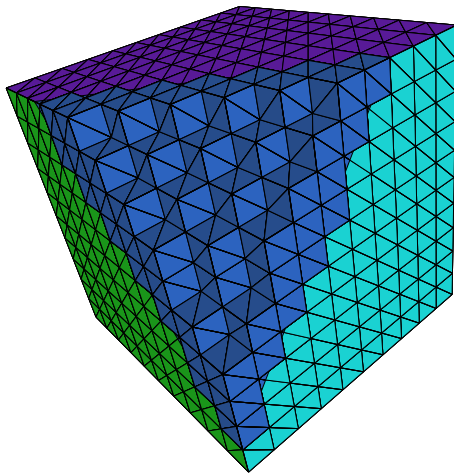


Figure 5.33: the mesh of cube make with cube.edp

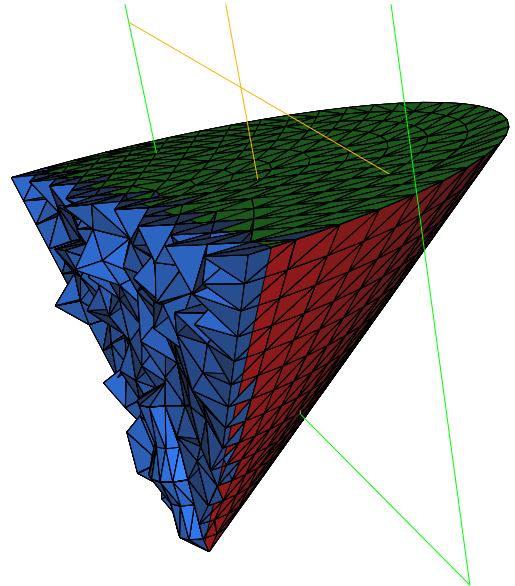


Figure 5.34: the mesh of cone make with cone.edp

```

border C11(t=0,0.7){ x=0.5+t; y=2.5; label=C1;}
border C12(t=0,2){ x=1.2; y=2.5+t; label=C1;}
border C13(t=0,0.7){ x=1.2-t; y=4.5; label=C1;}
border C14(t=0,2){ x=0.5; y=4.5-t; label=C1;}

border C21(t=0,0.7){ x= 2.3+t; y=2.5; label=C2;}
border C22(t=0,2){ x=3; y=2.5+t; label=C2;}
border C23(t=0,0.7){ x=3-t; y=4.5; label=C2;}
border C24(t=0,2){ x=2.3; y=4.5-t; label=C2;}

mesh Th=buildmesh( C01(10)+C02(10)+ C03(10)+C04(10)
                  + C11(5)+C12(5)+C13(5)+C14(5)
                  + C21(-5)+C22(-5)+C23(-5)+C24(-5));

mesh Ths=buildmesh( C01(10)+C02(10)+ C03(10)+C04(10)
                   + C11(5)+C12(5)+C13(5)+C14(5) );

// construction of a box with one hole and two regions

func zmin=0.;
func zmax=1.;
int MaxLayer=10;

func XX = x*cos(y);
func YY = x*sin(y);
func ZZ = z;

int[int] r1=[0,41], r2=[98,98, 99,99, 1,56];
int[int] r3=[4,12];
// The triangles of upper surface mesh

```

```

//      generated by the triangle in the 2D region of mesh Th of label 4 as label 12.
int[int] r4=[4,45];                                //      The triangles of lower surface mesh
//      generated by the triangle in the 2D region of mesh Th of label 4 as label 45.

mesh3 Th3=buildlayers( Th, MaxLayer, zbound=[zmin,zmax], region=r1,
                      labelmid=r2, labelup = r3, labeldown = r4 );
savemesh(Th3,"box2region1hole.mesh");

//      construction of a sphere with TetGen

func XX1 = cos(y)*sin(x);
func YY1 = sin(y)*sin(x);
func ZZ1 = cos(x);
string test="paACQ";
cout << "test=" << test << endl;
mesh3 Th3sph=tetgtransfo(Ths,transfo=[XX1,YY1,ZZ1],switch=test,nbofregions=1,
                        regionlist=domain);
savemesh(Th3sph,"sphere2region.mesh");

```

## 5.11 Meshing examples

**Example 5.23 (lac.edp)** //file "lac.edp"

```

load 'msh3'
int nn=5;
border cc(t=0,2*pi){x=cos(t);y=sin(t);label=1;}
mesh Th2 = buildmesh(cc(100));
fespace Vh2(Th2,P2);
Vh2 ux,uy,p2;
int[int] rup=[0,2], rdlow=[0,1], rmid=[1,1,2,1,3,1,4,1];
func zmin = 2-sqrt(4-(x*x+y*y));
func zmax = 2-sqrt(3.);

mesh3 Th = buildlayers(Th2,nn,
                      coeff = max((zmax-zmin)/zmax, 1./nn),
                      zbound=[zmin,zmax],
                      labelmid=rmid;
                      labelup=rup;
                      labeldown=rldow);
savemesh(Th,'Th.meshb');
exec('medit Th; Th.meshb');

```

**Example 5.24 (tetgenholeregion.edp)**

```

load "msh3"
load "tetgen"

```

```

mesh Th=square(10,20,[x*pi-pi/2,2*y*pi]);

//      ]-pi/2, -pi/2[ x]0, 2pi[
//      a parametrization of a sphere

func f1 =cos(x)*cos(y);
func f2 =cos(x)*sin(y);
func f3 = sin(x);

//      partiel derivative of the parametrization DF

```

```

func f1x=sin(x)*cos(y);
func f1y=-cos(x)*sin(y);
func f2x=-sin(x)*sin(y);
func f2y=cos(x)*cos(y);
func f3x=cos(x);
func f3y=0;

//       $M = DF'DF$ 

func m11=f1x^2+f2x^2+f3x^2;
func m21=f1x*f1y+f2x*f2y+f3x*f3y;
func m22=f1y^2+f2y^2+f3y^2;

func perio=[[4,y],[2,y],[1,x],[3,x]];
real hh=0.1;
real vv= 1/square(hh);
verbosity=2;
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
plot(Th,wait=1);

verbosity=2;

//      construction of the surface of spheres

real Rmin  = 1.;
func f1min = Rmin*f1;
func f2min = Rmin*f2;
func f3min = Rmin*f3;

mesh3 Th3sph = movemesh23(Th,transfo=[f1min,f2min,f3min]);

real Rmax  = 2.;
func f1max = Rmax*f1;
func f2max = Rmax*f2;
func f3max = Rmax*f3;

mesh3 Th3sph2 = movemesh23(Th,transfo=[f1max,f2max,f3max]);

cout << "addition" << endl;
mesh3 Th3 = Th3sph+Th3sph2;

real[int] domain2 = [1.5,0.,0.,145,0.001,0.5,0.,0.,18,0.001];
cout << "=====" << endl;
cout << " tetgen call without hole " << endl;
cout << "=====" << endl;
mesh3 Th3fin = tetg(Th3,switch="paAAQYY",nbofregions=2,regionlist=domain2);
cout << "=====" << endl;
cout << "finish tetgen call without hole" << endl;
cout << "=====" << endl;
savemesh(Th3fin,"spherewithtworegion.mesh");

real[int] hole = [0.,0.,0.];
real[int] domain = [1.5,0.,0.,53,0.001];
cout << "=====" << endl;
cout << " tetgen call with hole " << endl;
cout << "=====" << endl;
mesh3 Th3finhole=tetg(Th3,switch="paAAQYY",nbofholes=1,holelist=hole,
nbofregions=1,regionlist=domain);

```

```

cout << "=====" << endl;
cout << "finish tetgen call with hole " << endl;
cout << "=====" << endl;
savemesh(Th3finhole,"spherewithahole.mesh");

```

### 5.11.1 Build a 3d mesh of a cube with a ballon incrustation

First the MeshSurface.idp file to build boundary mesh of a Hexaedra and of a Sphere.

```

func mesh3 SurfaceHex(int[int] & N,real[int,int] &B ,int[int,int] & L,int orientation)
{
    real x0=B(0,0),x1=B(0,1);
    real y0=B(1,0),y1=B(1,1);
    real z0=B(2,0),z1=B(2,1);

    int nx=N[0],ny=N[1],nz=N[2];

    mesh Thx = square(ny,nz,[y0+(y1-y0)*x,z0+(z1-z0)*y]);
    mesh Thy = square(nx,nz,[x0+(x1-x0)*x,z0+(z1-z0)*y]);
    mesh Thz = square(nx,ny,[x0+(x1-x0)*x,y0+(y1-y0)*y]);

    int[int] refx=[0,L(0,0)],refX=[0,L(0,1)]; // Xmin, Ymax faces labels renumbering
    int[int] refy=[0,L(1,0)],refY=[0,L(1,1)]; // Ymin, Ymax faces labesl renumbering
    int[int] refz=[0,L(2,0)],refZ=[0,L(2,1)]; // Zmin, Zmax faces labels renumbering

    mesh3 Thx0 = movemesh23(Thx,transfo=[x0,x,y],orientation=-orientation,label=refx);
    mesh3 Thx1 = movemesh23(Thx,transfo=[x1,x,y],orientation=+orientation,label=refX);
    mesh3 Thy0 = movemesh23(Thy,transfo=[x,y0,y],orientation=+orientation,label=refy);
    mesh3 Thy1 = movemesh23(Thy,transfo=[x,y1,y],orientation=-orientation,label=refY);
    mesh3 Thz0 = movemesh23(Thz,transfo=[x,y,z0],orientation=-orientation,label=refz);
    mesh3 Thz1 = movemesh23(Thz,transfo=[x,y,z1],orientation=+orientation,label=refZ);
    mesh3 Th= Thx0+Thx1+Thy0+Thy1+Thz0+Thz1;
    return Th;
}

func mesh3 Sphere(real R,real h,int L,int orientation)
{
    mesh Th=square(10,20,[x*pi-pi/2,2*y*pi]); // ] $\frac{-\pi}{2}$ ,frac{-pi2[x]0,2\pi[
// a parametrization of a sphere

    func f1 =cos(x)*cos(y);
    func f2 =cos(x)*sin(y);
    func f3 = sin(x);

// partiel derivative

    func f1x=sin(x)*cos(y);
    func f1y=-cos(x)*sin(y);
    func f2x=-sin(x)*sin(y);
    func f2y=cos(x)*cos(y);
    func f3x=cos(x);
    func f3y=0;

// the metric on the sphere M = DF'DF

    func m11=f1x^2+f2x^2+f3x^2;
    func m21=f1x*f1y+f2x*f2y+f3x*f3y;
    func m22=f1y^2+f2y^2+f3y^2;

```

```

func perio=[4,y],[2,y],[1,x],[3,x]; // to store the periodic condition

real hh=h/R; // hh mesh size on unite sphere
real vv= 1/square(hh);
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
int[int] ref=[0,L];

mesh3 ThS= movemesh23(Th,transfo=[f1*R,f2*R,f3*R],orientation=orientation,refface=ref);
return ThS;
}

```

The test of the two functions and the call to tetgen mesh generator

```

load "tetgen"
include "MeshSurface.idp"
    real hs = 0.1; // mesh size on sphere
    int[int] N=[20,20,20];
    real [int,int] B=[[-1,1],[-1,1],[-1,1]];
    int [int,int] L=[[1,2],[3,4],[5,6]];
    mesh3 ThH = SurfaceHex(N,B,L,1);
    mesh3 ThS =Sphere(0.5,hs,7,1); // "gluing" surface meshes to tolat boundary meshes

    mesh3 ThHS=ThH+ThS;
    savemesh(ThHS,"Hex-Sphere.mesh");
    exec("ffmedit Hex-Sphere.mesh;rm Hex-Sphere.mesh"); // see 5.35

    real voltet=(hs^3)/6.;
    cout << " voltet = " << voltet << endl;
    real[int] domaine = [0,0,0,1,voltet,0,0,0.7,2,voltet];

    mesh3 Th = tetg(ThHS,switch="pqaAAYYQ",nbofregions=2,regionlist=domaine);
    medit("Cube-With-Ball",Th); // see 5.36

```

## 5.12 Write solution at the format .sol and .solb

With the keyword savesol, we can store a scalar functions, a scalar FE functions, a vector fields, a vector FE fields, a symmetric tensor and a symmetric FE tensor.. Such format is used in medit.

**extension file .sol** The first two lines of the file are

- MeshVersionFormatted 0
- Dimension (I) dim

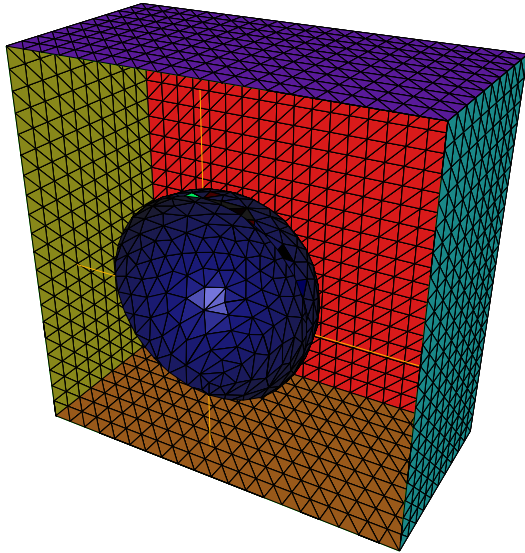


Figure 5.35: The surface mesh of the Hex with internal Sphere

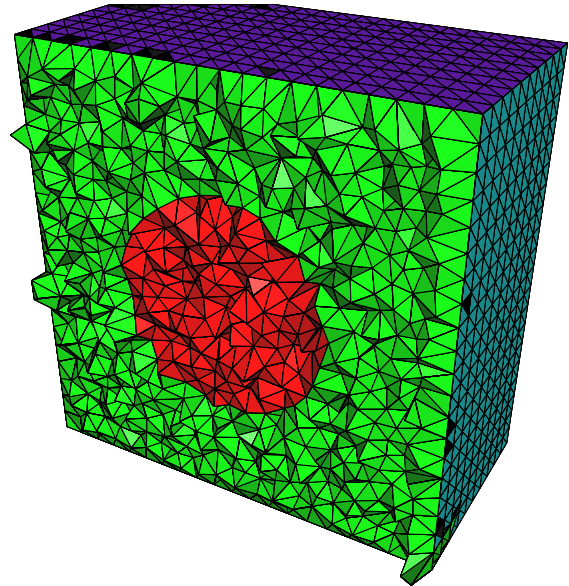


Figure 5.36: The tet mesh of the cube with internal ball

The following fields are begin with one of the following keyword: SolAtVertices, SolAtEdges, SolAtTriangles, SolAtQuadrilaterals, SolAtTetrahedra, SolAtPentahedra, SolAtHexahedra.

In each field, we give then in the next line the number of elements in solutions (SolAtVertices: number of vertices, SolAtTriangles: number of triangles, ...). In other lines, we give the number of solutions, the type of solution (1: scalar, 2: vector, 3: symmetric tensor). And finally, we give the solutions at the different elements.

The file must be finished with the keyword End.

The real element of symmetric tensor

$$ST^{3d} = \begin{pmatrix} ST_{xx}^{3d} & ST_{xy}^{3d} & ST_{xz}^{3d} \\ ST_{yx}^{3d} & ST_{yy}^{3d} & ST_{yz}^{3d} \\ ST_{zx}^{3d} & ST_{zy}^{3d} & ST_{zz}^{3d} \end{pmatrix} \quad ST^{2d} = \begin{pmatrix} ST_{xx}^{2d} & ST_{xy}^{2d} \\ ST_{yx}^{2d} & ST_{yy}^{2d} \end{pmatrix} \quad (5.5)$$

stored in the extension .sol are respectively  $ST_{xx}^{3d}, ST_{yx}^{3d}, ST_{yy}^{3d}, ST_{zx}^{3d}, ST_{zy}^{3d}, ST_{zz}^{3d}$  and  $ST_{xx}^{2d}, ST_{yx}^{2d}, ST_{yy}^{2d}$ . An example of field with the keyword SolAtTetrahedra:

- SolAtTetrahedra
  - (I) NbOfTetrahedrons
  - nbsol typesol<sup>1</sup> ... typesol<sup>n</sup>
  - (( $(U_{ij}^k, \forall i \in \{1, \dots, \text{nbreaksol}^k\})$ ,  $\forall k \in \{1, \dots, \text{nbsol}\}$ )  $\forall j \in \{1, \dots, \text{NbOfTetrahedrons}\}$ )

where

- nbsol is an integer equal to the number of solutions
- typesol<sup>k</sup>, type of the solution number k, is
  - typesol<sup>k</sup> = 1 the solution k is scalar.

- $\text{typesol}^k = 2$  the solution  $k$  is vectorial.
- $\text{typesol}^k = 3$  the solution  $k$  is a symmetric tensor or symmetric matrix.
- $\text{nbrealsol}^k$  number of real to describe solution number  $k$  is
  - $\text{nbrealsol}^k = 1$  the solution  $k$  is scalar.
  - $\text{nbrealsol}^k = \text{dim}$  the solution  $k$  is vectorial ( $\text{dim}$  is the dimension of the solution).
  - $\text{nbrealsol}^k = \text{dim} * (\text{dim} + 1)/2$  the solution  $k$  is a symmetric tensor or symmetric matrix.
- $U_{ij}^k$  is a real equal to the value of the component  $i$  of the solution  $k$  at tetrahedra  $j$  on the associated mesh.

This field is express with the notation of Section 12.1. The format `.solb` is the format `.sol` writing in binary. A real scalar functions  $f1$ , a vector fields  $\Phi = [\Phi1, \Phi2, \Phi3]$  and a symmetric tensor  $ST^{3d}$  (5.5) at the vertices of the three dimensional mesh  $\text{Th3}$  is stored in the file `"f1PhiTh3.sol"` using

```
savesol("f1PhiST3dTh3.sol",Th3, f1, [\Phi1, \Phi2, \Phi3], VV3, order=1);
```

where  $VV3 = [ST_{xx}^{3d}, ST_{yx}^{3d}, ST_{yy}^{3d}, ST_{zx}^{3d}, ST_{zy}^{3d}, ST_{zz}^{3d}]$ . For a two dimension mesh  $\text{Th}$ , A real scalar functions  $f2$ , a vector fields  $\Psi = [\Psi1, \Psi2]$  and a symmetric tensor  $ST^{2d}$  (5.5) at triangles is stored in the file `"f2PsiST2dTh3.solb"` using

```
savesol("f2PsiST2dTh3.solb",Th, f2, [\Psi1, \Psi2], VV2, order=0);
```

where  $VV2 = [ST_{xx}^{2d}, ST_{yx}^{2d}, ST_{yy}^{2d}]$  The arguments of `savesol` functions are the name of a file, a mesh and solutions. These arguments must be given in this order.

The parmameters of this keyword are

**order** = 0 is the solution is given at the center of gravity of elements. 1 is the solution is given at the vertices of elements.

In the file, solutions are stored in this order : scalar solutions, vector solutions and finally symmetric tensor solutions.

## 5.13 Call medit with the keyword medit

The keyword `medit` allows to display a mesh or mesh and solution using `medit`. `Medit` is a freeware display package by Pascal Frey using OpenGL. To use this command we need to install `medit`.

A vizualisation with `medit` of scalar solutions  $f1$  and  $f2$  at vertices of the mesh  $\text{Th}$  is obtained using

```
medit("sol1 sol2",Th, f1, f2, order=1);
```

The first plot named "sol1" display  $f1$ . The second plot names "sol2" display  $f2$ .

The arguments of function `medit` are the name of the differents scenes (separated by a space) of `medit`, a mesh and solutions. Each solution is associated with one scene. The scalar, vector and symmetric tensor solutions are given as in the keyword `savesol`.

The parmameters of this command line are

**order** = 0 is the solution is given at the center of gravity of elements. 1 is the solution is given at the vertices of elements.

**meditff** = set the name of execute command of `medit`. By default, this string is `medit`.



**save** = set the name of a file .sol or .solb to save solutions.

This commandline allows also to represent two different meshes with solutions in the same windows. The nature of solutions must be the same. Hence, we can visualize in the same window the different domains in a domain decomposition method. A visualization with medit of scalar solutions  $h1$  and  $h2$  at vertices of the mesh  $Th1$  and  $Th2$  respectively are obtained using

```
medit("sol2domain",Th1, h1, Th2, h2, order=1);
```

#### Example 5.25 (meditddm.edp)

```
load "medit"
```

```
// meditddm.edp
```

```

// Initial Problem:
// Resolution of the following EDP:
//  $-\Delta u = f$  on  $\Omega = \{(x,y)|1 \leq \sqrt{x^2 + y^2} \leq 2\}$ 
//  $-\Delta u = f1$  on  $\Omega_1 = \{(x,y)|0.5 \leq \sqrt{x^2 + y^2} \leq 1\}$ 
//  $u = 1$  on  $\Gamma$  + Null Neumann condition on  $\Gamma_1$  and on  $\Gamma_2$ 
// We find the solution  $u$  in solving two EDP defined on domain  $\Omega$  and  $\Omega_1$ 
// This solution is visualize with medit

verbosity=3;

border Gamma(t=0,2*pi){x=cos(t); y=sin(t); label=1;};
border Gamma1(t=0,2*pi){x=2*cos(t); y=2*sin(t); label=2;};
border Gamma2(t=0,2*pi){x=0.5*cos(t); y=0.5*sin(t); label=3;};

// construction of mesh of domain  $\Omega$ 
mesh Th=buildmesh(Gamma1(40)+Gamma(-40));

fespace Vh(Th,P2);
func f=sqrt(x*x+y*y);
Vh us,v;
macro Grad2(us) [dx(us),dy(us)] // EOM

problem Lap2dOmega(us,v,init=false)=int2d(Th)(Grad2(v)' *Grad2(us)) - int2d(Th)(f*v)+on(Gamma,us=1)
;

// Resolution of EDP defined on the domain  $\Omega$ 
//  $-\Delta u = f$  on  $\Omega$ 
//  $u = 1$  on  $\Gamma$ 
// + Null Neumann condition on  $\Gamma_1$ 

Lap2dOmega;

// construction of mesh of domain  $\Omega_1$ 
mesh Th1=buildmesh(Gamma(40)+Gamma2(-40));

fespace Vh1(Th1,P2);
func f1=10*sqrt(x*x+y*y);
Vh1 u1,v1;
macro Grad21(u1) [dx(u1),dy(u1)] // EOM

problem Lap2dOmega1(u1,v1,init=false)=int2d(Th1)(Grad21(v1)' *Grad21(u1)) - int2d(Th1)(f1*v1)+on(Ga
;

// Resolution of EDP defined on the domain  $\Omega_1$ 
```

```

//      -Δu = f1 on Ω1
//      u = 1 on Γ
//      + Null Neumann condition on Γ2

Lap2dOmega1;

//      vizualisation of solution of the initial problem
medit("solution",Th,us,Th1,u1,order=1,save="testsavemedit.solb");

```

**Example 5.26 (StockesUzawa.edp)***// signe of pressure is correct*

```

assert(version>1.18);
real s0=clock();
mesh Th=square(10,10);
fespace Xh(Th,P2),Mh(Th,P1);
Xh u1,u2,v1,v2;
Mh p,q,ppp;

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

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

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

Xh bc1; bc1[] = a(0,Xh);
Xh b;

matrix A= a(Xh,Xh,solver=CG);
matrix Bx= bx(Xh,Mh);
matrix By= by(Xh,Mh);
Xh bcx=1,bcy=0;

func real[int] divup(real[int] & pp)
{
  int verb=verbosity;
  verbosity=0;
  b[] = Bx'*pp; b[] += bc1[] .*bcx[];
  u1[] = A^-1*b[];
  b[] = By'*pp; b[] += bc1[] .*bcy[];
  u2[] = A^-1*b[];
  ppp[] = Bx*u1[];
  ppp[] += By*u2[];
  verbosity=verb;
  return ppp[] ;
};

p=0;q=0;u1=0;v1=0;

LinearCG(divup,p[],q[],eps=1.e-6,nbiter=50);

divup(p[]);

plot([u1,u2],p,wait=1,value=true,coef=0.1);
medit('velocity pressure',Th,[u1,u2],p,order=1);

```

# 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  $w_k$  ( $k = 0, \dots, M-1$ ). 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

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

is easily created by

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

or with  $\ell$  pairs of periodic boundary condition in 2d

```
fespace IDspace(IDmesh,<IDFE>,  
                periodic=[[1a_1,sa_1],[1b_1,sb_1],  
                ...  
                [1a_k,sa_k],[1b_k,sb_k]]);
```

and in 3d

```
fespace IDspace(IDmesh,<IDFE>,  
                periodic=[[1a_1,sa_1,ta_1],[1b_1,sb_1,tb_1],  
                ...  
                [1a_k,sa_k,ta_k],[1b_k,sb_k,tb_k]]);
```

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 2d case, a pair of periodic boundary condition, if `[1a_i,sa_i],[1b_i,sb_i]` is a pair of `int`, the 2 labels `1a_i` and `1b_i` define the 2 piece of boundary to be in equivalence; If `[1a_i,sa_i],[1b_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.

In 2d case, a pair of periodic boundary condition,if `[1a_i,sa_i,ta_i],[1b_i,sb_i,tb_i]` is a pair of `int`, the 2 labels `1a_i` and `1b_i` define the 2 piece of boundary to be in equivalence;

If `[1a_i,sa_i,ta_i],[1b_i,sb_i,tb_i]` is a pair of `real`, then `sa_i,ta_i` and `sb_i,tb_i` give two common parameter on the two boundary surface, and two points are identified as one if the two parameter are equal.

Remark, the 2d mesh of the two identified border must be the same, so to be sure, used the parameter `fixeborder=true` in `buildmesh` command (see 5.1.2) like in example `periodic2bis.edp` (see 9.7).

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

**P0,P03d** piecewise constante discontinuous finite element (2d, 3d), the degrees of freedom are the barycenter element value.

$$P0_h = \left\{ v \in L^2(\Omega) \mid \text{for all } K \in \mathcal{T}_h \text{ there is } \alpha_K \in \mathbb{R} : v|_K = \alpha_K \right\} \quad (6.1)$$

**P1,P13d** piecewise linear continuous finite element (2d, 3d), the degrees of freedom are the vertices values.

$$P1_h = \left\{ v \in H^1(\Omega) \mid \forall K \in \mathcal{T}_h \quad v|_K \in P_1 \right\} \quad (6.2)$$

**P1dc** piecewise linear discontinuous finite element

$$P1dc_h = \left\{ v \in L^2(\Omega) \mid \forall K \in \mathcal{T}_h \quad v|_K \in P_1 \right\} \quad (6.3)$$

Warning, due to interpolation problem, the degree of freedom is not the vertices but three vertices move inside with  $T(X) = G + .99(X - G)$  where  $G$  is the barycenter, (version 2.24-4).

**P1b,P1b3d** piecewise linear continuous finite element plus bubble (2d, 3d)

The 2d case:

$$P1b_h = \left\{ v \in H^1(\Omega) \mid \forall K \in \mathcal{T}_h \quad v|_K \in P_1 \oplus \text{Span}\{\lambda_0^K \lambda_1^K \lambda_2^K\} \right\} \quad (6.4)$$

The 3d case:

$$P1b_h = \left\{ v \in H^1(\Omega) \mid \forall K \in \mathcal{T}_h \quad v|_K \in P_1 \oplus \text{Span}\{\lambda_0^K \lambda_1^K \lambda_2^K \lambda_3^K\} \right\} \quad (6.5)$$

where  $\lambda_i^K, i = 0, \dots, d$  are the  $d + 1$  barycentric coordinate functions of the element  $K$  (triangle or tetrahedron).

**P2,P23d** piecewise  $P_2$  continuous finite element (2d, 3d),

$$P2_h = \left\{ v \in H^1(\Omega) \mid \forall K \in \mathcal{T}_h \quad v|_K \in P_2 \right\} \quad (6.6)$$

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

**P2b** piecewise  $P_2$  continuous finite element plus bubble,

$$P2_h = \left\{ v \in H^1(\Omega) \mid \forall K \in \mathcal{T}_h \quad v|_K \in P_2 \oplus \text{Span}\{\lambda_0^K \lambda_1^K \lambda_2^K\} \right\} \quad (6.7)$$

**P2dc** piecewise  $P_2$  discontinuous finite element,

$$P2dc_h = \left\{ v \in L^2(\Omega) \mid \forall K \in \mathcal{T}_h \quad v|_K \in P_2 \right\} \quad (6.8)$$

Warning, due to interpolation problem, the degree of freedom is not the six  $P_2$  nodes but six nodes move inside with  $T(X) = G + .99(X - G)$  where  $G$  is the barycenter, (version 2.24-4).

**RT0,RT03d** Raviart-Thomas finite element of degree 0.

The 2d case:

$$RT0_h = \left\{ \mathbf{v} \in H(\text{div}) \mid \forall K \in \mathcal{T}_h \quad \mathbf{v}|_K(x, y) = \begin{bmatrix} \alpha_K^1 \\ \alpha_K^2 \end{bmatrix} + \beta_K \begin{bmatrix} x \\ y \end{bmatrix} \right\} \quad (6.9)$$

The 3d case:

$$RT0_h = \left\{ \mathbf{v} \in H(\text{div}) \mid \forall K \in \mathcal{T}_h \quad \mathbf{v}|_K(x, y, z) = \begin{bmatrix} \alpha_K^1 \\ \alpha_K^2 \\ \alpha_K^3 \end{bmatrix} + \beta_K \begin{bmatrix} x \\ y \\ z \end{bmatrix} \right\} \quad (6.10)$$

where by writing  $\text{div } \mathbf{w} = \sum_{i=1}^d \partial w_i / \partial x_i$  with  $\mathbf{w} = (w_i)_{i=1}^d$ ,

$$H(\text{div}) = \left\{ \mathbf{w} \in L^2(\Omega)^d \mid \text{div } \mathbf{w} \in L^2(\Omega) \right\}$$

and where  $\alpha_K^1, \alpha_K^2, \alpha_K^3, \beta_K$  are real numbers.

**Edge03d** Nedelec finite element or Edge Element of degree 0.

The 3d case:

$$RT0_h = \left\{ \mathbf{v} \in H(\text{Curl}) \mid \forall K \in \mathcal{T}_h \quad \mathbf{v}|_K(x, y, z) = \begin{bmatrix} \alpha_K^1 \\ \alpha_K^2 \\ \alpha_K^3 \end{bmatrix} + \begin{bmatrix} \beta_K^1 \\ \beta_K^2 \\ \beta_K^3 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} \right\} \quad (6.11)$$

where by writing  $\text{curl} \mathbf{w} = \begin{bmatrix} \partial w_2 / \partial x_3 - \partial w_3 / \partial x_2 \\ \partial w_3 / \partial x_1 - \partial w_1 / \partial x_3 \\ \partial w_1 / \partial x_2 - \partial w_2 / \partial x_1 \end{bmatrix}$  with  $\mathbf{w} = (w_i)_{i=1}^d$ ,

$$H(\text{curl}) = \left\{ \mathbf{w} \in L^2(\Omega)^d \mid \text{curl } \mathbf{w} \in L^2(\Omega)^d \right\}$$

and  $\alpha_K^1, \alpha_K^2, \alpha_K^3, \beta_K^1, \beta_K^2, \beta_K^3$  are real numbers.

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

## 6.1 Usage of two dimensional finite element spaces

If we get the two dimensional finite element spaces

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

$$X_{ph} = \{v \in X_h \mid v|_{\left[ \begin{smallmatrix} 0 \\ \cdot \end{smallmatrix} \right]} = v|_{\left[ \begin{smallmatrix} 1 \\ \cdot \end{smallmatrix} \right]}, v|_{\left[ \begin{smallmatrix} \cdot \\ 0 \end{smallmatrix} \right]} = v|_{\left[ \begin{smallmatrix} \cdot \\ 1 \end{smallmatrix} \right]}\}$$

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

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

when  $\mathcal{T}_h$  is a mesh  $10 \times 10$  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  $X_h, M_h, R_h$  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.
Wxh[5](0.5, 0.5) // the 6th function at point (0.5, 0.5)
Wxh[5][] // the array of the degree of freedom of the 6 function.
```

The functions  $U_h, V_h$  have two components so we have

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

## 6.2 Usage of three dimensional finite element spaces

If we get the three dimensional finite element spaces

$$X_h = \{v \in H^1([0, 1]^3) \mid \forall K \in \mathcal{T}_h \quad v|_K \in P_1\}$$

$$X_{ph} = \{v \in X_h \mid v(\cdot^0) = v(\cdot^1), v(\cdot^0) = v(\cdot^1)\}$$

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

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

when  $\mathcal{T}_h$  is a mesh  $10 \times 10 \times 10$  of the unit cubic  $[0, 1]^3$ , we only write in FreeFem++ as follows:

```
mesh3 Th=buildlayers(square(10,10),10, zbound=[0,1]);
// label: 0 up, 1 down; 2 front, 3 left, 4 back, 5: right
fespace Xh(Th,P1); // scalar FE
fespace Xph(Th,P1,
    periodic=[[0,x,y],[1,x,y],
              [2,x,z],[4,x,z],
              [3,y,z],[5,y,z]]); // three-periodic FE
fespace Mh(Th,P2); // scalar FE
fespace Rh(Th,RT03d); // 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,Uyzh],[Vxh,Vyh,Vyzh];
Xh[int] Uh(10); // array of 10 function in Xh
Rh[int] [Wxh,Wyh,Wzh](10); // array of 10 functions in Rh.
Wxh[5](0.5,0.5,0.5) // the 6th function at point (0.5,0.5,0.5)
Wxh[5][] // the array of the degree of freedom of the 6 function.
```

The functions  $U_h, V_h$  have three components so we have

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

## 6.3 Lagrange finite element

### 6.3.1 P0-element

For each triangle (d=2) or tetrahedron (d=3)  $T_k$ , the basis function  $\phi_k$  in  $Vh(Th,P0)$  is given by

$$\phi_k(x) = 1 \text{ if } (x) \in T_k, \quad \phi_k(x) = 0 \text{ if } (x) \notin T_k$$

If we write

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

then for vertices  $q^{k_i}$ ,  $i = 1, 2, \dots, d+1$  in Fig. 6.1(a),  $f_h$  is built as

$$\mathbf{f}h = f_h(x, y) = \sum_k f\left(\frac{\sum_i q^{k_i}}{d+1}\right) \phi_k$$

See Fig. 6.3 for the projection of  $f(x, y) = \sin(\pi x) \cos(\pi y)$  on  $\mathbf{V}h(\mathbf{Th}, \mathbf{P0})$  when the mesh  $\mathbf{Th}$  is a  $4 \times 4$ -grid of  $[-1, 1]^2$  as in Fig. 6.2.

### 6.3.2 P1-element

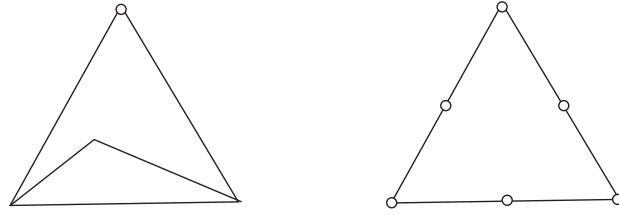


Figure 6.1:  $P_1$  and  $P_2$  degrees of freedom on triangle  $T_k$

For each vertex  $q^i$ , the basis function  $\phi_i$  in  $\mathbf{V}h(\mathbf{Th}, \mathbf{P1})$  is given by

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

The basis function  $\phi_{k_1}(x, y)$  with the vertex  $q^{k_1}$  in Fig. 6.1(a) at point  $p = (x, y)$  in triangle  $T_k$  simply coincide with the *barycentric coordinates*  $\lambda_1^k$  (*area coordinates*) :

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

If we write

$\mathbf{V}h(\mathbf{Th}, \mathbf{P1}); \mathbf{V}h \mathbf{f}h = g(x, y);$

then

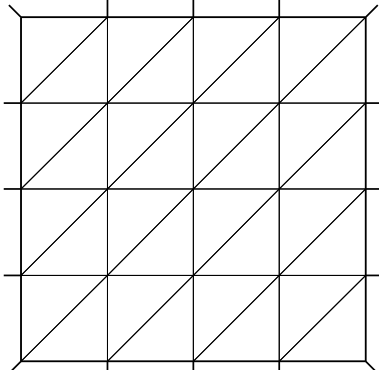
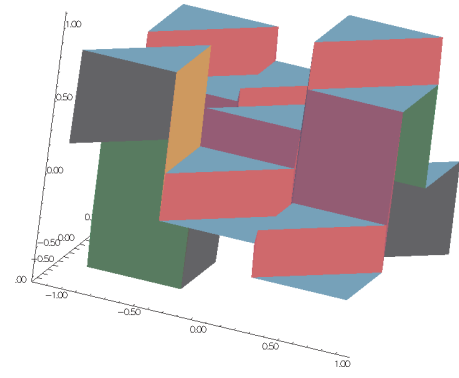
$$\mathbf{f}h = f_h(x, y) = \sum_{i=1}^{n_v} f(q^i) \phi_i(x, y)$$

See Fig. 6.4 for the projection of  $f(x, y) = \sin(\pi x) \cos(\pi y)$  into  $\mathbf{V}h(\mathbf{Th}, \mathbf{P1})$ .

### 6.3.3 P2-element

For each vertex or midpoint  $q^i$ , the basis function  $\phi_i$  in  $\mathbf{V}h(\mathbf{Th}, \mathbf{P2})$  is given by

$$\begin{aligned} \phi_i(x, y) &= a_i^k + b_i^k x + c_i^k y + d_i^k x^2 + e_i^k xy + f_i^k y^2 \text{ for } (x, y) \in T_k, \\ \phi_i(q^i) &= 1, \quad \phi_i(q^j) = 0 \text{ if } i \neq j \end{aligned}$$

Figure 6.2: Test mesh  $\mathcal{Th}$  for projectionFigure 6.3: projection to  $V_h(\mathcal{Th}, P_0)$ 

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  $q^{k_2}$

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

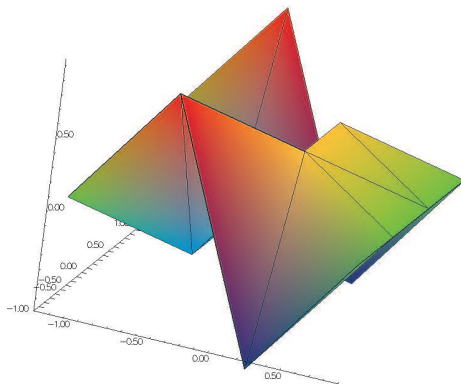
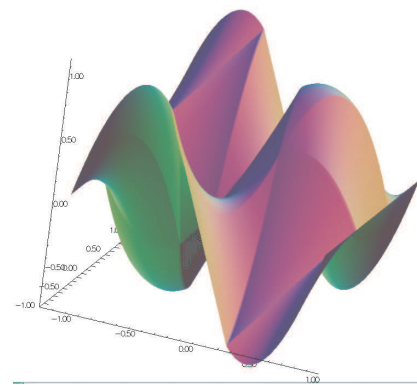
If we write

$V_h(\mathcal{Th}, \mathbf{P2})$ ;  $V_h \text{ fh} = f(x, y)$ ;

then

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

See Fig. 6.5 for the projection of  $f(x, y) = \sin(\pi x) \cos(\pi y)$  into  $V_h(\mathcal{Th}, \mathbf{P2})$ .

Figure 6.4: projection to  $V_h(\mathcal{Th}, P_1)$ Figure 6.5: projection to  $V_h(\mathcal{Th}, P_2)$



## 6.4 P1 Nonconforming Element

Refer to [23] 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

$$\text{Vh}(\text{Th}, \mathbf{P1nc}); \quad \text{Vh} \quad \text{fh} = f(x, y);$$

then

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

Here the basis function  $\phi_i$  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  $j = 3$ :

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

Strictly speaking  $\partial \phi_i / \partial x$ ,  $\partial \phi_i / \partial y$  contain Dirac distribution  $\rho \delta_{\partial T_k}$ . The numerical calculations will automatically ignore them. In [23], there is a proof of the estimation

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

The basis functions  $\phi_k$  have the following properties.

1. For the bilinear form  $a$  defined in (2.6) satisfy

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

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

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

$$\int_{\Omega} \phi_i \phi_j dx dy = 0 \quad \text{if } i \neq j$$

which is false for  $P_1$ -element.

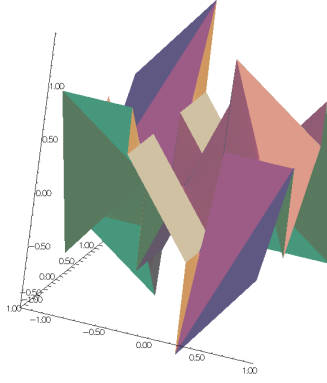
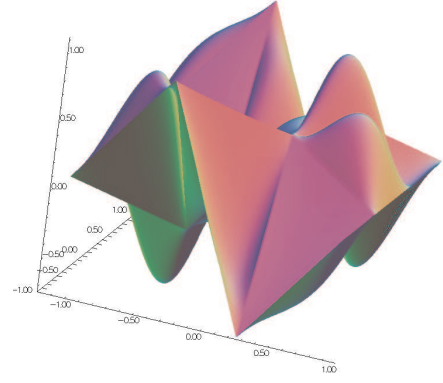
See Fig. 6.6 for the projection of  $f(x, y) = \sin(\pi x) \cos(\pi y)$  into  $\text{Vh}(\text{Th}, \mathbf{P1nc})$ . See Fig. 6.6 for the projection of  $f(x, y) = \sin(\pi x) \cos(\pi y)$  into  $\text{Vh}(\text{Th}, \mathbf{P1nc})$ .

## 6.5 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 coordinate of the triangle (see Fig. 6.1), and put

$$\beta_k(x, y) = 27 \lambda_{k_1}(x, y) \lambda_{k_2}(x, y) \lambda_{k_3}(x, y) \quad (6.12)$$

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

Figure 6.6: projection to  $V_h(\mathcal{T}_h, P1nc)$ Figure 6.7: projection to  $V_h(\mathcal{T}_h, P1b)$ 

1.  $\beta_k(x, y) = 0$  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

$V_h(\mathcal{T}_h, P1b)$ ;  $V_h \text{ fh} = f(x, y)$ ;

then

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

See Fig. 6.7 for the projection of  $f(x, y) = \sin(\pi x) \cos(\pi y)$  into  $V_h(\mathcal{T}_h, P1b)$ .

## 6.6 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`  $V_h(\mathcal{T}_h, [P0, P1])$  ;

make the space

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

### 6.6.1 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  $\mathbf{f} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  is  $\int_e \mathbf{f} \cdot \mathbf{n}_e$ ,  $\mathbf{n}_e$  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  $\text{sgn}(i_b - i_c)(\mathbf{b} - \mathbf{c})$ ,  $\text{sgn}(i_c - i_a)(\mathbf{c} - \mathbf{a})$ ,  $\text{sgn}(i_a - i_b)(\mathbf{a} - \mathbf{b})$ ,

We get three basis functions,

$$\phi_1^k = \frac{\text{sgn}(i_b - i_c)}{2|T_k|}(\mathbf{x} - \mathbf{a}), \quad \phi_2^k = \frac{\text{sgn}(i_c - i_a)}{2|T_k|}(\mathbf{x} - \mathbf{b}), \quad \phi_3^k = \frac{\text{sgn}(i_a - i_b)}{2|T_k|}(\mathbf{x} - \mathbf{c}), \quad (6.13)$$

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

$Vh(Th, \mathbf{RT0}); Vh [f1h, f2h] = [f1(x,y), f2(x,y)];$

then

$$f_h = f_h(x, y) = \sum_{k=1}^{n_t} \sum_{l=1}^6 n_{i_l j_l} |e^h| f_{j_l}(m^l) \phi_{i_l j_l}$$

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

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

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

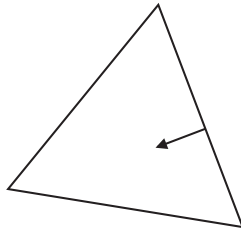


Figure 6.8: normal vectors of each edge

**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)];`

`vh= x^2+y^2;`

`Th = square(5,5);`

`uh = x^2+y^2;`

`Uxh = x;`

`vh = Uxh;`

`plot(uh,ps="onoldmesh.eps");`

`uh = uh;`

`plot(uh,ps="onnewmesh.eps");`

`vh([x-1/2,y])= x^2 + y^2;`

`// ok vectorial FE function`

`// vh`

`// change the mesh`

`// Xh is unchange`

`// compute on the new Xh`

`// error: impossible to set only 1 component`

`// of a vector FE function.`

`// ok`

`// and now uh use the 5x5 mesh`

`// but the fespace of vh is always the 2x2 mesh`

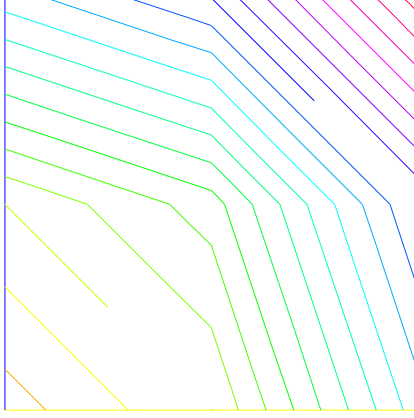
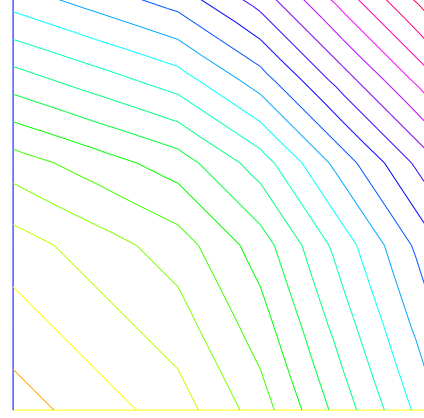
`// figure 6.9`

`// do a interpolation of vh (old) of 5x5 mesh`

`// to get the new vh on 10x10 mesh.`

`// figure 6.10`

`// interpolate vh = ((x - 1/2)^2 + y^2)`

Figure 6.9: vh Iso on mesh  $2 \times 2$ Figure 6.10: vh Iso on mesh  $5 \times 5$ 

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.

## 6.7 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  $\Omega_1$  and another one for  $\Omega_2$ . 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  $\Omega$ . 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 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  $\lambda_i$  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**

Two problems need to be solved:

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

**Note 6.2** Step 3 requires an array of pointers such that each vertex points to one triangle of the triangulation.

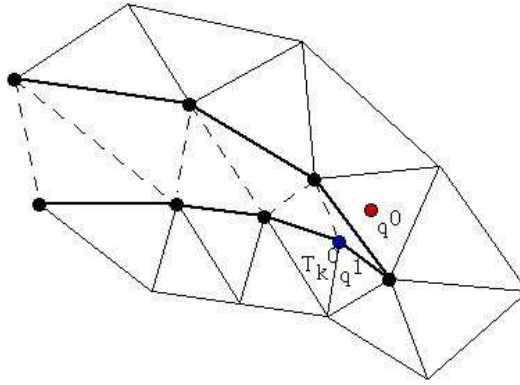


Figure 6.11: To interpolate a function at  $q^0$  the knowledge of the triangle which contains  $q^0$  is needed. The algorithm may start at  $q^1 \in T_k^0$  and stall on the boundary (thick line) because the line  $q^0q^1$  is not inside  $\Omega$ . But if the holes are triangulated too (doted line) then the problem does not arise.

**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");
plot(vs,vg,wait=1,ps="vs-vg.eps");

```

// see figure 6.12  
// see figure 6.13

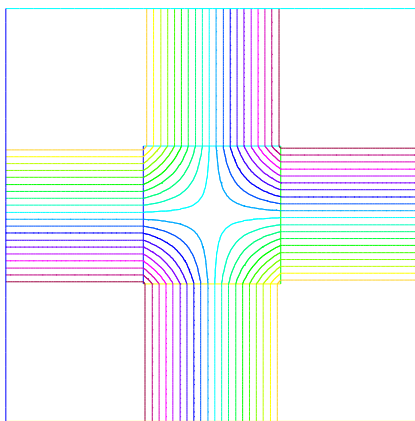


Figure 6.12: Extension of a continuous FE-function

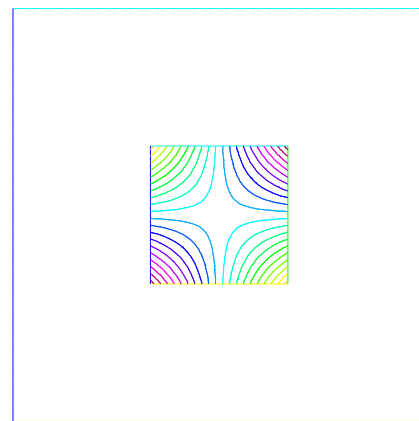


Figure 6.13: Extension of discontinuous FE-function, see warning 6

## 6.8 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 keyword `solve`.

### 6.8.1 Weak form and Boundary Condition

To present the principles of Variational Formulations or also called weak forms for 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  $\Omega$  of  $\mathbb{R}^d$  ( $d = 2, 3$ ) such that

$$-\nabla \cdot (\kappa \nabla u) = f, \quad \text{in } \Omega, \quad au + \kappa \frac{\partial u}{\partial n} = b \quad \text{on } \Gamma_r, \quad u = g \quad \text{on } \Gamma_d \quad (6.14)$$

where

- if  $d = 2$  then  $\nabla \cdot (\kappa \nabla u) = \partial_x(\kappa \partial_x u) + \partial_y(\kappa \partial_y u)$  with  $\partial_x u = \frac{\partial u}{\partial x}$  and  $\partial_y u = \frac{\partial u}{\partial y}$
- if  $d = 3$  then  $\nabla \cdot (\kappa \nabla u) = \partial_x(\kappa \partial_x u) + \partial_y(\kappa \partial_y u) + \partial_z(\kappa \partial_z u)$  with  $\partial_x u = \frac{\partial u}{\partial x}$ ,  $\partial_y u = \frac{\partial u}{\partial y}$  and  $\partial_z u = \frac{\partial u}{\partial z}$
- the border  $\Gamma = \partial\Omega$  is split in  $\Gamma_d$  and  $\Gamma_n$  such that  $\Gamma_d \cup \Gamma_n = \partial\Omega$  and  $\Gamma_d \cap \Gamma_n = \emptyset$
- $\kappa$  is a given positive function, such that  $\exists \kappa_0 \in \mathbb{R}, \quad 0 < \kappa_0 \leq \kappa$ .
- $a$  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  $\Gamma_d$  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  $\Gamma_d$ , by integration par part we get

$$-\int_{\Omega} \nabla \cdot (\kappa \nabla u) v \, d\omega = \int_{\Omega} \kappa \nabla v \cdot \nabla u \, d\omega = \int_{\Omega} f v \, d\omega - \int_{\Gamma} v \kappa \frac{\partial u}{\partial n} \, d\gamma, \quad (6.15)$$

where if  $d = 2$  the  $\nabla v \cdot \nabla u = (\frac{\partial u}{\partial x} \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \frac{\partial v}{\partial y})$ , where if  $d = 3$  the  $\nabla v \cdot \nabla u = (\frac{\partial u}{\partial x} \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \frac{\partial v}{\partial y} + \frac{\partial u}{\partial z} \frac{\partial v}{\partial z})$ , and where  $\mathbf{n}$  is the unitary outside normal of  $\partial\Omega$ .

Now we note that  $\kappa \frac{\partial u}{\partial n} = -au + g$  on  $\Gamma_r$  and  $v = 0$  on  $\Gamma_d$  and  $\partial\Omega = \Gamma_d \cup \Gamma_n$  thus

$$-\int_{\partial\Omega} v \kappa \frac{\partial u}{\partial n} = \int_{\Gamma_r} auv - \int_{\Gamma_r} bv$$

The problem become:

Find  $u \in V_g = \{v \in H^1(\Omega)/v = g \text{ on } \Gamma_d\}$  such that

$$\int_{\Omega} \kappa \nabla v \cdot \nabla u \, d\omega + \int_{\Gamma_r} a u v \, d\gamma = \int_{\Omega} f v \, d\omega + \int_{\Gamma_r} b v \, d\gamma, \quad \forall v \in V_0 \quad (6.16)$$

where  $V_0 = \{v \in H^1(\Omega)/v = 0 \text{ on } \Gamma_d\}$

The problem (6.16) 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:  $\int_{\Omega} f \, d\omega + \int_{\Gamma_r} b \, d\gamma$  and a way to fix the constant is to solve for  $u \in H^1(\Omega)$  such that:

$$\int_{\Omega} \varepsilon u v \, d\omega + \int_{\Omega} \kappa \nabla v \cdot \nabla u \, d\omega = \int_{\Omega} f v \, d\omega + \int_{\Gamma_r} b v \, d\gamma, \quad \forall v \in H^1(\Omega) \quad (6.17)$$

where  $\varepsilon$  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  $\int_{\Omega} u = 0$ , you can also add a Lagrange multiplier to solve the real mathematical problem like in the `examples++-tutorial/Laplace-lagrange-mult.edp` example.

In FreeFem++, the bidimensional problem (6.16) become

```
problem Pw(u,v) =
  int2d(Th)( kappa*( dx(u)*dx(u) + dy(u)*dy(u)) ) //  $\int_{\Omega} \kappa \nabla v \cdot \nabla u \, d\omega$ 
+ int1d(Th,gn)( a * u*v ) //  $\int_{\Gamma_r} a u v \, d\gamma$ 
- int2d(Th)( f*v ) //  $\int_{\Omega} f v \, d\omega$ 
- int1d(Th,gn)( b * v ) //  $\int_{\Gamma_r} b v \, d\gamma$ 
+ on(gd)(u= g) ; //  $u = g \text{ on } \Gamma_d$ 
```

where Th is a mesh of the the bidimensional domain  $\Omega$ , and gd and gn are respectively the boundary label of boundary  $\Gamma_d$  and  $\Gamma_n$ .

And the the three dimensional problem (6.16) become

```
macro Grad(u) [dx(u),dy(u),dz(u)] // EOM : definition of the 3d Grad macro
problem Pw(u,v) =
  int3d(Th)( kappa*( Grad(u)'*Grad(v)) ) //  $\int_{\Omega} \kappa \nabla v \cdot \nabla u \, d\omega$ 
+ int2d(Th,gn)( a * u*v ) //  $\int_{\Gamma_r} a u v \, d\gamma$ 
- int3d(Th)( f*v ) //  $\int_{\Omega} f v \, d\omega$ 
- int2d(Th,gn)( b * v ) //  $\int_{\Gamma_r} b v \, d\gamma$ 
+ on(gd)(u= g) ; //  $u = g \text{ on } \Gamma_d$ 
```

where Th is a mesh of the three dimensional domain  $\Omega$ , and gd and gn are respectively the boundary label of boundary  $\Gamma_d$  and  $\Gamma_n$ .

## 6.9 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 unpredictable, Sorry.*

The parameters are:

**solver=** LU, CG, Crout,Cholesky,GMRES,sparsesolver, UMFPACK ...

The default solver is `sparsesolver` ( it is equal to UMFPACK if not other sparse solver is defined) or is set to LU if no direct sparse solver is available. 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, `sparsesolver` or UMFPACK the matrix is just sparse.

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

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

if it is positive then the stopping test is

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

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

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

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

**tgval=** Huge value ( $10^{30}$ ) used to implement Dirichlet boundary conditions.

**tolpivot=** set the tolerance of the pivot in UMFPACK ( $10^{-1}$ ) and, LU, Crout, Cholesky factorisation ( $10^{-20}$ ).

**tolpivotsym=** set the tolerance of the pivot sym in UMFPACK

**strategy=** set the integer UMFPACK strategy (0 by default).

## 6.10 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 unknown 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,

$$\text{-) } \text{int3d}(\text{Th})(K^*v^*w) = \sum_{T \in \text{Th}} \int_T K v w$$

$$\text{-) } \text{int3d}(\text{Th},1)(K^*v^*w) = \sum_{T \in \text{Th}, T \subset \Omega_1} \int_T K v w$$

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

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

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

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

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

- ) they contribute to the sparse matrix of type `matrix` which, whether declared explicitly or not is constructed by `FreeFem++`.

- the right handside of the PDE, volumic terms of the linear form: for given functions  $K, f$ :

$$\text{-) } \text{int3d}(\text{Th})(K^*w) = \sum_{T \in \text{Th}} \int_T K w$$

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

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

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

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

- The boundary condition terms :

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

The meaning is for all degree of freedom  $i$  of this associated boundary, the diagonal term of the matrix  $a_{ii} = \text{tgv}$  with the *terrible geant value* `tgv` ( $=10^{30}$  by default) and the right hand side  $b[i] = "(\Pi_h g)[i]" \times \text{tgv}$ , where the  $"(\Pi_h g)[i]"$  is the boundary node value given by the interpolation of  $g$ .

- An "on" vectorial form (for Dirichlet) : `on(1,u1=g1,u2=g2)` If you have vectorial finite element like `RT0`, the 2 components are coupled, and so you have :  $b[i] = "(\Pi_h(g1, g2))[i]" \times \text{tgv}$ , where  $\Pi_h$  is the vectorial finite element interpolant.
- a linear form on  $\Gamma$  (for Neumann in 2d) `-int1d(Th))(f^*w)` or `-int1d(Th,3))(f^*w)`
- a bilinear form on  $\Gamma$  or  $\Gamma_2$  (for Robin in 2d) `int1d(Th))(K^*v^*w)` or `int1d(Th,2))(K^*v^*w)`.

- a linear form on  $\Gamma$  (for Neumann in 3d) `-int2d(Th))( f*w)` or `-int2d(Th,3))( f*w)`
- a bilinear form on  $\Gamma$  or  $\Gamma_2$  (for Robin in 3d) `int2d(Th))( K*v*w)` or `int2d(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.*
- *$N.x$ ,  $N.y$  and  $N.z$  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)` .

## 6.11 Numerical Integration

Let  $D$  be a  $N$ -dimensional bounded domain. For an arbitrary polynomials  $f$  of degree  $r$ , if we can find particular points  $\xi_j$ ,  $j = 1, \dots, J$  in  $D$  and constants  $\omega_j$  such that

$$\int_D f(\mathbf{x}) = \sum_{\ell=1}^L c_\ell f(\xi_\ell) \quad (6.18)$$

then we have the error estimation (see Crouzeix-Mignot (1984)), and then there exists a constant  $C > 0$  such that,

$$\left| \int_D f(\mathbf{x}) - \sum_{\ell=1}^L \omega_\ell f(\xi_\ell) \right| \leq C|D|h^{r+1} \quad (6.19)$$

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

$$\{(x, y) | x = (1 - t)q_x^i + tq_x^j, y = (1 - t)q_y^i + tq_y^j, 0 \leq t \leq 1\}.$$

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

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

where  $*$  stands for the name of the quadrature formula or the precision (order) of the Gauss formula.

Quadrature formula on an edge					
$L$	(qfe=)	qforder=	point in $[q^i q^j](= t)$	$\omega_\ell$	exact on $P_k, k =$
1	qf1pE	2	1/2	$ q^i q^j $	1
2	qf2pE	3	$(1 \pm \sqrt{1/3})/2$	$ q^i q^j /2$	3
3	<b>qf3pE</b>	6	$(1 \pm \sqrt{3/5})/2$ 1/2	$(5/18) q^i q^j $ $(8/18) q^i q^j $	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} q^i q^j $ $\frac{18+\sqrt{30}}{72} q^i q^j $	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} q^i q^j $ $\frac{64}{225} q^i q^j $ $\frac{322+13\sqrt{70}}{1800} q^i q^j $	9
2	qf1pElump	2	0 +1	$ q^i q^j /2$ $ q^i q^j /2$	1

where  $|q^i q^j|$  is the length of segment  $\overline{q^i q^j}$ . For a part  $\Gamma_1$  of  $\Gamma_h$  with the label “1”, we can calculate the integral over  $\Gamma_1$  by

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

The integral over  $\Gamma_1, \Gamma_3$  are given by

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

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

$$\begin{aligned} |T_k| &= \frac{1}{2} \begin{vmatrix} 1 & q_x^{k_1} & q_y^{k_1} \\ 1 & q_x^{k_2} & q_y^{k_2} \\ 1 & q_x^{k_3} & q_y^{k_3} \end{vmatrix} \quad D_1 = \begin{vmatrix} 1 & x & y \\ 1 & q_x^{k_2} & q_y^{k_2} \\ 1 & q_x^{k_3} & q_y^{k_3} \end{vmatrix} \quad D_2 = \begin{vmatrix} 1 & q_x^{k_1} & q_y^{k_1} \\ 1 & x & y \\ 1 & q_x^{k_3} & q_y^{k_3} \end{vmatrix} \quad D_3 = \begin{vmatrix} 1 & q_x^{k_1} & q_y^{k_1} \\ 1 & q_x^{k_2} & q_y^{k_2} \\ 1 & x & y \end{vmatrix} \\ \xi &= \frac{1}{2} D_1 / |T_k| \quad \eta = \frac{1}{2} D_2 / |T_k| \quad \text{then } 1 - \xi - \eta = \frac{1}{2} D_3 / |T_k| \end{aligned}$$

For a two dimensional domain or border of three dimensional domain  $\Omega_h = \sum_{k=1}^{n_t} T_k$ ,  $\mathcal{T}_h = \{T_k\}$ , we can calculate the integral over  $\Omega_h$  by

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

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

Quadrature formula on a triangle					
$L$	qft=	qforder=	point in $T_k$	$\omega_\ell$	exact on $P_k, k =$
1	qf1pT	2	$\left(\frac{1}{3}, \frac{1}{3}\right)$	$ T_k $	1
3	qf2pT	3	$\left(\frac{1}{2}, \frac{1}{2}\right)$ $\left(\frac{1}{2}, 0\right)$ $\left(0, \frac{1}{2}\right)$	$ T_k /3$ $ T_k /3$ $ T_k /3$	2
7	<b>qf5pT</b>	6	$\left(\frac{1}{3}, \frac{1}{3}\right)$ $\left(\frac{6-\sqrt{15}}{21}, \frac{6-\sqrt{15}}{21}\right)$ $\left(\frac{6-\sqrt{15}}{21}, \frac{9+2\sqrt{15}}{21}\right)$ $\left(\frac{9+2\sqrt{15}}{21}, \frac{6-\sqrt{15}}{21}\right)$ $\left(\frac{6+\sqrt{15}}{21}, \frac{6+\sqrt{15}}{21}\right)$ $\left(\frac{6+\sqrt{15}}{21}, \frac{9-2\sqrt{15}}{21}\right)$ $\left(\frac{9-2\sqrt{15}}{21}, \frac{6+\sqrt{15}}{21}\right)$	$0.225 T_k $ $\frac{(155-\sqrt{15}) T_k }{1200}$ $\frac{(155-\sqrt{15}) T_k }{1200}$ $\frac{(155-\sqrt{15}) T_k }{1200}$ $\frac{(155+\sqrt{15}) T_k }{1200}$ $\frac{(155+\sqrt{15}) T_k }{1200}$ $\frac{(155+\sqrt{15}) T_k }{1200}$	5
3	qf1pTlump		$(0, 0)$ $(1, 0)$ $(0, 1)$	$ T_k /3$ $ T_k /3$ $ T_k /3$	1
9	qf2pT4P1		$\left(\frac{1}{4}, \frac{3}{4}\right)$ $\left(\frac{3}{4}, \frac{1}{4}\right)$ $\left(0, \frac{1}{4}\right)$ $\left(0, \frac{3}{4}\right)$ $\left(\frac{1}{4}, 0\right)$ $\left(\frac{3}{4}, 0\right)$ $\left(\frac{1}{4}, \frac{1}{4}\right)$ $\left(\frac{1}{4}, \frac{1}{2}\right)$ $\left(\frac{1}{2}, \frac{1}{4}\right)$	$ T_k /12$ $ T_k /12$ $ T_k /12$ $ T_k /12$ $ T_k /12$ $ T_k /12$ $ T_k /6$ $ T_k /6$ $ T_k /6$	1
15	qf7pT	8	see [38] for detail		7
21	qf9pT	10	see [38] for detail		9

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

$$\begin{aligned}
 \int_{\Omega_h} f(x, y) &= \text{int3d}(\mathcal{T}_h)(f) \\
 &= \text{int3d}(\mathcal{T}_h, \text{qfV}=\*)(f) \\
 &= \text{int3d}(\mathcal{T}_h, \text{qforder}=\*)(f)
 \end{aligned}$$

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

Quadrature formula on a tetraedron					
$L$	qfV=	qforder=	point in $T_k \in \mathbb{R}^3$	$\omega_\ell$	exact on $P_k, k =$
1	qfV1	2	$\left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right)$	$ T_k $	1
4	qfV2	3	$G4(0.58\dots, 0.13\dots, 0.13\dots)$	$ T_k /4$	2
14	<b>qfV5</b>	6	$G4(0.72\dots, 0.092\dots, 0.092\dots)$ $G4(0.067\dots, 0.31\dots, 0.31\dots)$ $G6(0.45\dots, 0.045\dots, 0.45\dots)$	$0.073\dots T_k $ $0.11\dots T_k $ $0.042\dots T_k $	5
4	qfV1lump		$G4(1, 0, 0)$	$ T_k /4$	1

Where  $G4(a, b, b)$  such that  $a + 3b = 1$  is the set of the four point in barycentric coordinate

$$\{(a, b, b, b), (b, a, b, b), (b, b, a, b), (b, b, b, a)\}$$

and where  $G6(a, b, b)$  such that  $2a + 2b = 1$  is the set of the six points in barycentric coordinate

$$\{(a, a, b, b), (a, b, a, b), (a, b, b, a), (b, b, a, a), (b, a, b, a), (b, a, a, b)\}.$$

Remark, all this tetraedral quadrature formula come from <http://www.cs.kuleuven.be/~nines/research/ecf/mtables.html>

**Note 6.10** *By default, we use the formula which is exact for polynomes of degrees 5 on triangles or edges (in bold in three tables).*

## 6.12 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

$$\forall w \in V_0; \quad \int_{\Omega} \frac{u^n - u^{n-1}}{\delta t} w + \kappa \nabla u^n \nabla w + \int_{\Gamma} \alpha (u^n - u_{ue}) w = 0$$

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).

$$u^n = A^{-1} b^n, \quad b' = b_0 + M u^{n-1}, \quad b'' = \frac{1}{\varepsilon} b_{cl}, \quad b_i^n = \begin{cases} b''_i & \text{if } i \in \Gamma_{24} \\ b'_i & \text{else if } i \notin \Gamma_{24} \end{cases} \quad (6.20)$$

Where with  $\frac{1}{\varepsilon} = \text{tgv} = 10^{30}$  :

$$A_{ij} = \begin{cases} \frac{1}{\varepsilon} & \text{if } i \in \Gamma_{24}, \text{ and } j = i \\ \int_{\Omega} w_j w_i / dt + k(\nabla w_j \cdot \nabla w_i) + \int_{\Gamma_{13}} \alpha w_j w_i & \text{else if } i \notin \Gamma_{24}, \text{ or } j \neq i \end{cases} \quad (6.21)$$

$$M_{ij} = \begin{cases} \frac{1}{\varepsilon} & \text{if } i \in \Gamma_{24}, \text{ and } j = i \\ \int_{\Omega} w_j w_i / dt & \text{else if } i \notin \Gamma_{24}, \text{ or } j \neq i \end{cases} \quad (6.22)$$

$$b_{0,i} = \int_{\Gamma_{13}} \alpha u_{ue} w_i \quad (6.23)$$

$$b_{cl} = u^0 \quad \text{the initial data} \quad (6.24)$$

*// 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 side 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 :  $i \in \Gamma_{24} \Leftrightarrow bcn[i] \neq 0$ 
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  $\forall i: b[i] = bcn[i] ? bcl[i] : b[i]$  ;
    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 and local to the **varf** definition, 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 "fespace" for the unknow fields with a correct number of component
// Wh is "fespace" for the test fields with a correct number of component
```

The possible named parameter " [, ... ] " of the construction are

**solver=** LU, CG, Crout, Cholesky, GMRES, sparsesolver, 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, sparsesolver 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.  $\varepsilon$  sets the stopping test for the iterative methods like CG. Note that if  $\varepsilon$  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 preconditioner. The prototype for the function P must be

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

**tgval=** Huge value ( $10^{30}$ ) used to implement Dirichlet boundary conditions.

**tolpivot=** set the tolerance of the pivot in UMFPACK ( $10^{-1}$ ) and, LU, Crout, Cholesky factorisation ( $10^{-20}$ ).

**tolpivotsym=** set the tolerance 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 constant / triangle
Nh areaK;
varf varea(used,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:

$$\eta_K[i] = \int 1_{|K_i} = \int_{K_i} 1;$$

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)=intalldges(Th,qfnpE=1)(v);
varf vedgecount(u,v)=intalldges(Th,qfnpE=1)(v/lenEdge);
// computation of the mesh size
// -----
```

```

count=vedgecount(0,Vh);
h[]=vmeshsizen(0,Vh);
h[]=h[]./count;

```

// number of edge / vertex  
// sum length edge / vertex  
// mean lenght edge / vertex

To compute error indicator for Poisson equation :

$$\eta_K = \int_K h_K^2 |(f + \Delta u_h)|^2 + \int_{\partial K} h_e \left| \frac{\partial u_h}{\partial n} \right|^2$$

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);
Nh etak;
varf vetaK(UNUSED,chiK) =
    intalldges(Th)(chiK*lenEdge*square(jump(N.x*dx(u)+N.y*dy(u))))
    +int2d(Th)(chiK*square(hTriangle*(f+dxx(u)+dyy(u))) );

etak[] = vetaK(0,Ph);

```

// the space function contant / triangle

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);
Vh->VH matrix BB= interpolate(Wh,Vh);
matrix Vh->Wh

```

// build interpolation matrix  
// build interpolation

and after some operations on sparse matrices are available for example

```

int N=10;
real [int,int] A(N,N);
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;

```

// a full matrix

```

sparseA = 2*sparseA+sparseA';
sparseA = 4*sparseA+sparseA*5;
matrix sparseB=sparseA+sparseA+sparseA; ;
cout << "sparseB = " << sparseB(0,0) << endl;

```

## 6.13 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  $\partial_x$

2 interpolate the  $\partial_y$

..

### 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
//      extended 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
//      extended 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;

```

## 6.14 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  $i$ th 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;
int kdf= Wh.ndofK ;
cout << " df of element " << k << ":" ;
for (int i=0;i<kdf;i++)
    cout << Wh(k,i) << " ";
cout << endl;

```

// element 2

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,
- ?** to show all actives keyboard char,

**enter** wait for the next plot,  
**ESC** close the graphics process.  
**otherwise** do nothing.

**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]` of increasing values)

**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  $3 \times n$  value in HSV color model declare with for example

```
real[int] colors = [h1,s1,v1,... , hn,vn,vn];
```

where  $h_i, s_i, v_i$  is the  $i$ th color to defined the color table.

**boundary=** (bool) to plot or not the boundary of the domain (true by default).

**dim=** (int) sets dim of the plot 2d or 3d (2 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,greyscale=1,fill=1,value=1,ps="threeg.eps");
plot([xx,yy],ps="likegnu.eps",wait=true);

```

// figure 7.2  
// figure 7.3

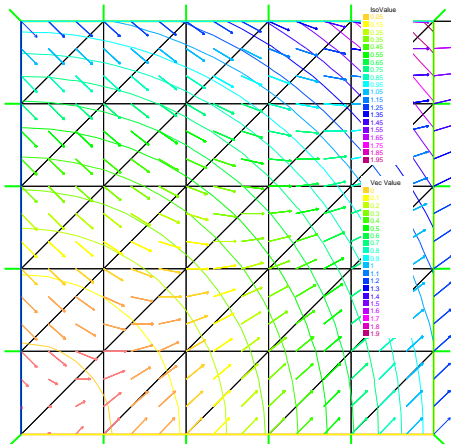


Figure 7.1: mesh, isovalue, and vector

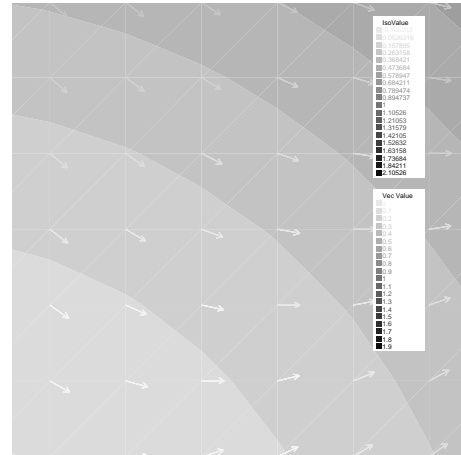


Figure 7.2: enlargement in grey of isovalue, and vector

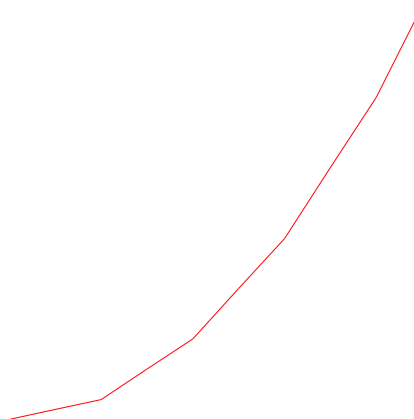


Figure 7.3: Plots a cut of uh. Note that a refinement of the same can be obtained in combination with gnuplot

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=[
    4./6., 1 , 0.5,
    4./6., 1 , 1,
    5./6., 1 , 1,
    1, 1. , 1,
    1, 0.5 , 1
];
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);

```

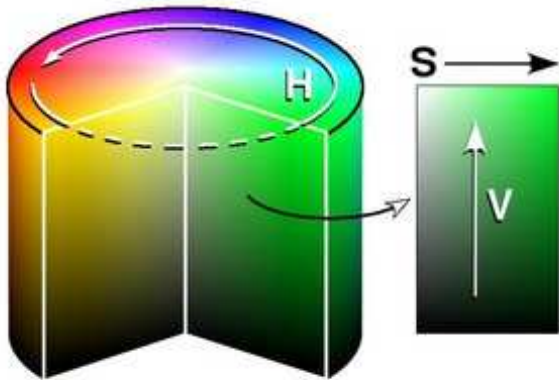


Figure 7.4: hsv color cylinder

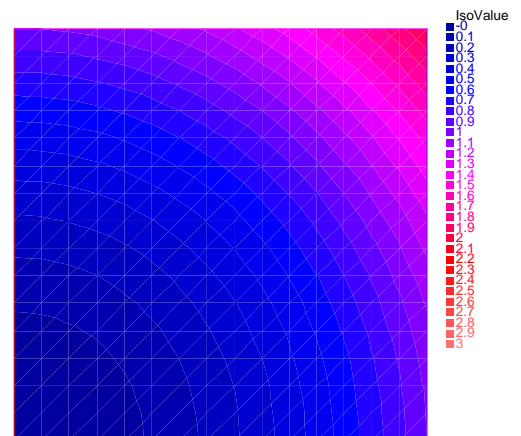


Figure 7.5: isovalue with an other color table



## 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<sup>1</sup> 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");
```

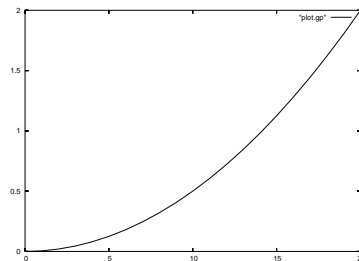


Figure 7.6: Plots a cut of uh with gnuplot

## 7.3 link with medit

First now medit<sup>2</sup>, a freeware display package by Pascal Frey using OpenGL. Then you may run the following example.

remark, now medit software is include in FreeFem++ under fmedit name.

Now with version 3.2 or better

```

load "medit"
mesh Th=square(10,10,[2*x-1,2*y-1]);
fespace Vh(Th,P1);
Vh u=2-x*x-y*y;
medit("mm",u);
```

<sup>1</sup><http://www.gnuplot.info/>

<sup>2</sup><http://www-rocq.inria.fr/gamma/medit/medit.html>

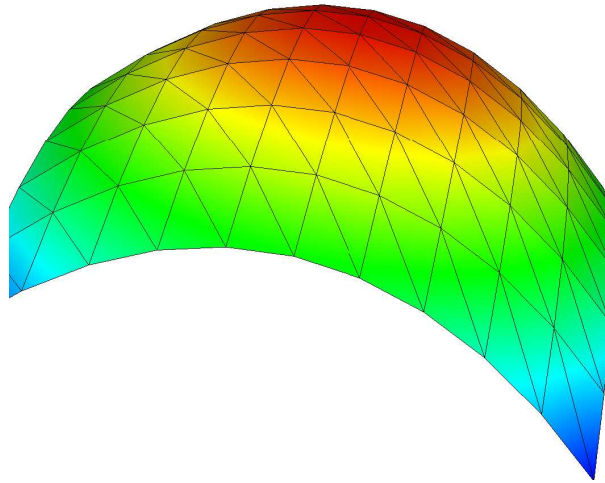


Figure 7.7: medit plot

Before:

```

mesh Th=square(10,10,[2*x-1,2*y-1]);
fespace Vh(Th,P1);
Vh u=2-x*x-y*y;
savemesh(Th,"mm",[x,y,u*.5]);           // save mm.points and mm.faces file
                                           // for medit
                                           // build a mm.bb file

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

                                           // call medit command

exec("ffmedit 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

$$\nabla J(x) = \left( \frac{\partial J}{\partial x_i}(x) \right) = 0 \quad (8.1)$$

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  $\nabla J$ , so the two parameters are the name of the function with prototype `func real[int] dJ(real[int] & xx)` which compute  $\nabla J$ , a vector `x` of type `real[int]` to initialize the process and get the result.

Given an initial value  $x^{(0)}$ , a maximum number  $i_{\max}$  of iterations, and an error tolerance  $0 < \epsilon < 1$ : Put  $x = x^{(0)}$  and write

```
NLCG( $\nabla J$ ,  $x$ , precon=  $M$ , nbiter=  $i_{\max}$ , eps=  $\epsilon$ );
```

will give the solution of  $\nabla J(x) = 0$ . We can omit parameters `precon`, `nbiter`, `eps`. Here  $M$  is the preconditioner whose default is the identity matrix. The stopping test is

$$\|\nabla J(x)\|_P \leq \epsilon \|\nabla J(x^{(0)})\|_P$$

Writing the minus value in `eps=`, i.e.,

```
NLCG( $\nabla J$ ,  $x$ , precon=  $M$ , nbiter=  $i_{\max}$ , eps=  $-\epsilon$ );
```

we can use the stopping test

$$\|\nabla J(x)\|_P^2 \leq \epsilon$$

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  $\varepsilon$  ( $= 10^{-6}$  by default) if positive then relative test  $\|\nabla J(x)\|_P \leq \varepsilon \|\nabla J(x_0)\|_P$ , otherwise the absolute test is  $\|\nabla J(x)\|_P^2 \leq |\varepsilon|$ .

**veps=** set and return the value of the stop test, if positive then relative test  $\|\nabla J(x)\|_P \leq \varepsilon \|\nabla J(x_0)\|_P$ , otherwise the absolute test is  $\|\nabla J(x)\|_P^2 \leq |\varepsilon|$ . 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  $\partial\Omega$ .

```
func real J(real[int] & u)
{
  Vh w;w[]=u; // copy array u in the finite element function w
  real r=int2d(Th)(0.5*f( dx(w)*dx(w) + dy(w)*dy(w) ) - b*w) ;
  cout << "J(u) =" << r << " " << u.min << " " << u.max << endl;
  return r;
}

// -----

Vh u=0; // the current value of the solution
Ph alpha; // of store df(|\nabla u|^2)
int iter=0;
alpha=df( dx(u)*dx(u) + dy(u)*dy(u) ); // optimization

func real[int] dJ(real[int] & u)
{
  int verb=verbosity; verbosity=0;
  Vh w;w[]=u; // copy array u in the finite element function w
  alpha=df( dx(w)*dx(w) + dy(w)*dy(w) ); // optimization
  varf au(uh,vh) = int2d(Th)( alpha*( dx(w)*dx(vh) + dy(w)*dy(vh) ) - b*vh)
    + on(1,2,3,4,u=0);
  u= au(0,Vh);
  verbosity=verb;
  return u; // 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

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

where  $\alpha = f'(|\nabla u|^2)$ . \*/

```
varf alap(uh,vh)= int2d(Th)( alpha *( dx(uh)*dx(vh) + dy(uh)*dy(vh) ))
  + on(1,2,3,4,u=0);

varf amass(uh)= int2d(Th)( uh*vh) + on(1,2,3,4,u=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] & u)
{
    real[int] w = A*mass*u;
    u = A\lap^-1*w;
    return u;
}

/* 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;
    alpha=df( dx(u)*dx(u) + dy(u)*dy(u) );
    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

$$J(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x}$$

then  $J(\mathbf{x})$  is minimized by the solution  $\mathbf{x}$  of  $A\mathbf{x} = \mathbf{b}$ . In this case, we can use the function `LinearCG`

```
LinearCG(A, x, precon= M, nbiter= i_max, eps= ±ε);
```

If  $A$  is not symmetric, we can use `GMRES` (Generalized Minimum Residual) algorithm by

```
LinearGMRES(A, x, precon= M, nbiter= i_max, eps= ±ε);
```

Also, we can use the non-linear version of `GMRES` algorithm (the functional  $J$  is just convex)

```
LinearGMRES(∇J, x, precon= M, nbiter= i_max, eps= ±ε);
```

For detail of these algorithms, refer to [14][Chapter IV, 1.3].

## 8.2 Optimization

Two algorithms of COOOL a package [27] 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

```

real[int] b(10),u(10);
func real J(real[int] & u)
{
    real s=0;

```

```

    for (int i=0;i<u.n;i++)
        s +=(i+1)*u[i]*u[i]*0.5 - b[i]*u[i];
    cout << "J ="<< s << " u =" << u[0] << " " << u[1] << "...\\n" ;
    return s;
}

// the grad of J (this is a affine version (the RHS is in )
func real[int] DJ(real[int] &u)
{
    for (int i=0;i<u.n;i++)
        u[i]=(i+1)*u[i]-b[i];
    return u; // return of global variable ok
};

b=1; u=2; // set right hand side and initial gest
BFGS(J,dJ,u,eps=1.e-6,nbiter=20,nbiterline=20);
cout << "BFGS: J(u) = " << J(u) << 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

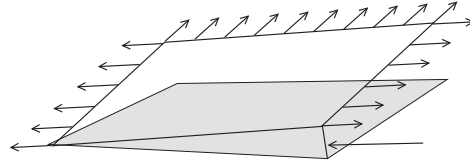
Our starting point here will be the mathematical model to find the shape of **soap film** which is glued to the ring on the  $xy$ -plane

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

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  $\mu$  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  $\mathbf{n}(x, y) = (n_x(x, y), n_y(x, y), n_z(x, y))$  the normal vector of the surface  $z = u(x, y)$ . We see that the vertical force due to the tension  $\mu$  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



$$-\mu n_y(x, y)\delta x, \quad \mu \left( n_y(x, y) + \frac{\partial n_y}{\partial y} \delta y \right) (x, y) \delta x.$$

The force in the vertical direction on the surface ABCD due to the tension  $\mu$  is given by

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

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

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

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

$$-\Delta u = f \quad \text{in } \Omega \quad (9.1)$$

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  $\rho$  is the charge density,  $\epsilon$  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

$$u = 0 \quad \text{on } \partial\Omega \quad (9.2)$$

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

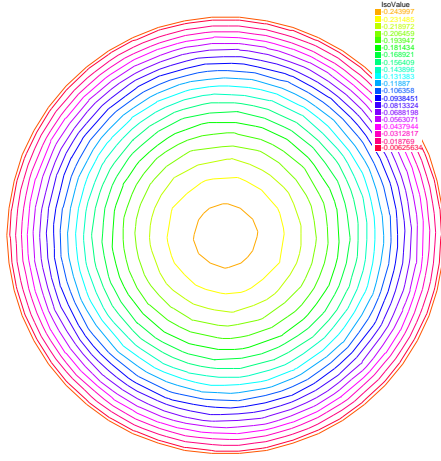
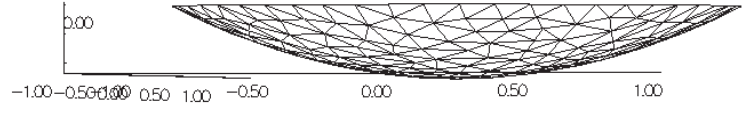
### Example 9.1 (a\_tutorial.edp)

```

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  $u_e$ ,

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

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

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

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 have  $\|u_h - u_e\|_{0,\Omega} = 0.000109043$ ,  $|u_h - u_e|_{1,\Omega} = 0.0188411$ . 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  $\mathbf{E}$  satisfy

$$\operatorname{div} \mathbf{E} = \rho/\epsilon, \quad \operatorname{curl} \mathbf{E} = 0 \quad (9.3)$$

where  $\rho$  is the charge density and  $\epsilon$  is called the permittivity of free space. From the second equation in (9.3), we can introduce the electrostatic potential such that  $\mathbf{E} = -\nabla \phi$ . Then we have Poisson equation  $-\Delta \phi = f$ ,  $f = -\rho/\epsilon$ . We now obtain the equipotential line which is the level curve of  $\phi$ , when there are no charges except conductors  $\{C_i\}_{1,\dots,K}$ . Let us assume  $K$  conductors  $C_1, \dots, C_K$  within an enclosure  $C_0$ . Each one is held at an electrostatic potential  $\varphi_i$ . We assume that the enclosure  $C_0$  is held at potential 0. In order to know  $\varphi(x)$  at any point  $x$  of the domain  $\Omega$ , we must solve

$$-\Delta \varphi = 0 \quad \text{in } \Omega, \quad (9.4)$$

where  $\Omega$  is the interior of  $C_0$  minus the conductors  $C_i$ , and  $\Gamma$  is the boundary of  $\Omega$ , that is  $\sum_{i=1}^N C_i$ . Here  $g$  is any function of  $x$  equal to  $\varphi_i$  on  $C_i$  and to 0 on  $C_0$ . The boundary equation is a reduced form for:

$$\varphi = \varphi_i \text{ on } C_i, \quad i = 1 \dots N, \quad \varphi = 0 \text{ on } C_0. \quad (9.5)$$

**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  $\Omega$  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");
fespace Vh(Th,P1);
Vh uh,vh;
problem Electro(uh,vh) =
    int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) )
    + on(C0,uh=0)
    + on(C1,uh=1)
    + on(C2,uh=-1) ;

Electro;
plot(uh,ps="electro.eps",wait=true);

```

// figure 9.3  
// P1 FE-space  
// unknown and test function.  
// definition of the problem  
// bilinear  
// boundary condition on  $C_0$   
// +1 volt on  $C_1$   
// -1 volt on  $C_2$

// solve the problem, see figure 9.4 for the solution  
// figure 9.4

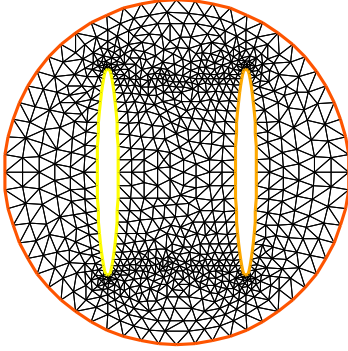
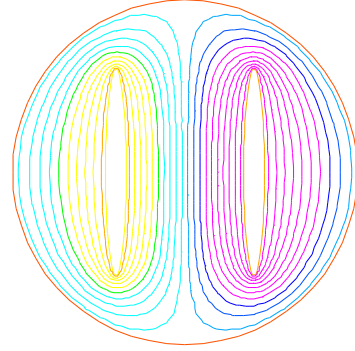


Figure 9.3: Disk with two elliptical holes

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  $\Gamma_\infty$ . As previously, we must solve

$$\Delta\varphi = 0 \quad \text{in } \Omega, \quad \varphi|_S = c, \quad \varphi|_{\Gamma_\infty} = u_{\infty 1x} - u_{\infty 2x} \quad (9.6)$$

where  $\Omega$  is the area occupied by the fluid,  $u_\infty$  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  $\varphi_c$  is a linear function of  $c$

$$\varphi_c = \varphi_0 + c\varphi_1, \quad (9.7)$$

where  $\varphi_0$  is a solution of (9.6) with  $c = 0$  and  $\varphi_1$  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:

$$y = 0.17735 \sqrt{x} - 0.075597x - 0.212836x^2 + 0.17363x^3 - 0.06254x^4. \quad (9.8)$$

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

$$-\Delta\varphi = 0 \quad \text{in } \Omega, \quad \varphi|_{\Gamma_1} = y - 0.1x, \quad \varphi|_{\Gamma_2} = c, \quad (9.9)$$

where  $\Gamma_2$  is the wing profile and  $\Gamma_1$  is an approximation of infinity. One finds  $c$  by solving:

$$-\Delta\varphi_0 = 0 \quad \text{in } \Omega, \quad \varphi_0|_{\Gamma_1} = y - 0.1x, \quad \varphi_0|_{\Gamma_2} = 0, \quad (9.10)$$

$$-\Delta\varphi_1 = 0 \quad \text{in } \Omega, \quad \varphi_1|_{\Gamma_1} = 0, \quad \varphi_1|_{\Gamma_2} = 1. \quad (9.11)$$

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  $\delta$ . 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  $\delta$  because the normal changes sign between the lower and upper surfaces. Thus

$$c = -\frac{\varphi_0|_{P^+} + \varphi_0|_{P^-}}{(\varphi_1|_{P^+} + \varphi_1|_{P^-} - 2)}, \quad (9.12)$$

which can be programmed as:

$$c = -\frac{\varphi_0(0.99, 0.01) + \varphi_0(0.99, -0.01)}{(\varphi_1(0.99, 0.01) + \varphi_1(0.99, -0.01) - 2)}. \quad (9.13)$$

### 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);

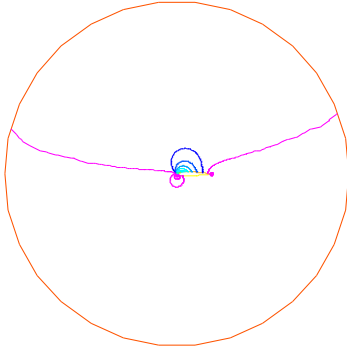
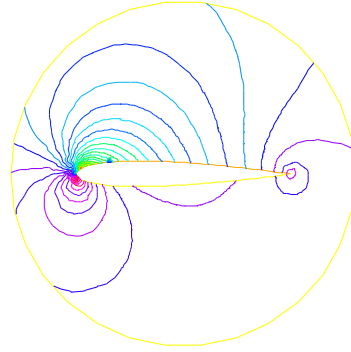
solve Joukowski1(psi1,vh) =                                     // definition of the problem
    int2d(Th)( dx(psi1)*dx(vh) + dy(psi1)*dy(vh) )              // bilinear form
    + on(a,psi1=0)                                               // boundary condition form
    + on(upper,lower,psi1=1);

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);

```

Figure 9.5: isovalue of  $cp = -(\partial_x \psi)^2 - (\partial_y \psi)^2$ Figure 9.6: Zooming of  $cp$ 

### 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

$$|\nabla u - \nabla u_h|_{0,\Omega} \leq C_1 h \quad (9.14)$$

$$\|u - u_h\|_{0,\Omega} \leq C_2 h^2 \quad (9.15)$$

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  $\Omega$  is convex.

In this section we check (9.14) and (9.15). We will pick up numerical error if we use the numerical derivative, so we will use the following for (9.14).

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

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);

```

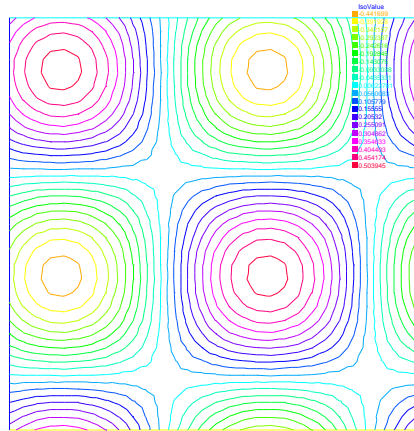


Figure 9.7: The isovalue of solution  $u$  with periodic boundary condition

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

//    1 first way
//    fespace Vh(Th,P2,periodic=[[2,1+x],[4,x],[1,x],[3,1+x]]);
//    2 second way
fespace Vh(Th,P2,periodic=[[2,x+y],[4,x+y],[1,x-y],[3,x-y]]);

Vh uh,vh;

func f=(y+x+1)*(y+x-1)*(y-x+1)*(y-x-1);
real intf = int2d(Th)(f);
real mTh = int2d(Th)(1);
real k = intf/mTh;
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");

```

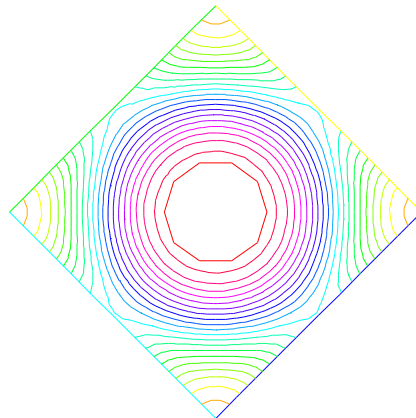


Figure 9.8: The isovalues of solution  $u$  for  $\Delta u = ((y+x)^2 + 1)((y-x)^2 + 1) - k$ , in  $\Omega$  and  $\partial_n u = 0$  on hole, and with two periodic boundary condition on external border

A other example with no equal border, just to see if the code works.

**Example 9.7 (periodic4bis.edp)**

```

//      irregular boundary condition.
//      to build border AB

macro LINEBORDER(A,B,lab) border A#B(t=0,1){real t1=1.-t;
  x=A#x*t1+B#x*t;y=A#y*t1+B#y*t;label=lab;} //      EOM
//      compute ||AB|| a=(ax,ay) et B =(bx,by)
macro dist(ax,ay,bx,by) sqrt(square((ax)-(bx))+ square((ay)-(by))) //      EOM
macro Grad(u) [dx(u),dy(u)] //      EOM

real Ax=0.9,Ay=1;          real Bx=2,By=1;
real Cx=2.5,Cy=2.5;        real Dx=1,Dy=2;
real gx = (Ax+Bx+Cx+Dx)/4.; real gy = (Ay+By+Cy+Dy)/4.;

LINEBORDER(A,B,1)
LINEBORDER(B,C,2)
LINEBORDER(C,D,3)
LINEBORDER(D,A,4)

int n=10;

real l1=dist(Ax,Ay,Bx,By);
real l2=dist(Bx,By,Cx,Cy);
real l3=dist(Cx,Cy,Dx,Dy);
real l4=dist(Dx,Dy,Ax,Ay);
func s1=dist(Ax,Ay,x,y)/l1; //      abscisse on AB = ||AX||/||AB||
func s2=dist(Bx,By,x,y)/l2; //      abscisse on BC = ||BX||/||BC||
func s3=dist(Cx,Cy,x,y)/l3; //      abscisse on CD = ||CX||/||CD||
func s4=dist(Dx,Dy,x,y)/l4; //      abscisse on DA = ||DX||/||DA||

mesh Th=buildmesh(AB(n)+BC(n)+CD(n)+DA(n),fixeborder=1); //

verbosity=6; //      to see the abscisse value pour the periodic condition.
fespace Vh(Th,P1,periodic=[[1,s1],[3,s3],[2,s2],[4,s4]]);
verbosity=1;
Vh u,v;
real cc=0;
cc= int2d(Th)((x-gx)*(y-gy)-cc)/Th.area;
cout << " compatibility =" << int2d(Th)((x-gx)*(y-gy)-cc) <<endl;

solve Poission(u,v)=int2d(Th)(Grad(u)'*Grad(v)+ 1e-10*u*v)
  -int2d(Th)(10*v*((x-gx)*(y-gy)-cc));
plot(u,wait=1,value=1);

```

**Example 9.8 (Period-Poisson-cube-ballon.edp)**

```

verbosity=1;
load "msh3"
load "tetgen"
load "medit"

bool buildTh=0;

```



```

mesh3 Th;
try {                                     // a way to build one time the mesh and read if the file exist.
  Th=readmesh3("Th-hex-sph.mesh");
}
catch(...) { buildTh=1;}
if( buildTh ){
  ...
  put the code example page                                     // 5.11.1121
  without the first line
}

fespace Ph(Th,P0);
verbosity=50;
fespace Vh(Th,P1,periodic=[[3,x,z],[4,x,z],[1,y,z],[2,y,z],[5,x,y],[6,x,y]]); // back
and front
verbosity=1;
Ph reg=region;

cout << " centre = " << reg(0,0,0) << endl;
cout << " exterieur = " << reg(0,0,0.7) << endl;

macro Grad(u) [dx(u),dy(u),dz(u)]                                     // EOM

Vh uh,vh;
real x0=0.3,y0=0.4,z0=0.6;
func f= sin(x*2*pi+x0)*sin(y*2*pi+y0)*sin(z*2*pi+z0);
real gn = 1.;
real cf= 1;
problem P(uh,vh)=
  int3d(Th,1)( Grad(uh)'*Grad(vh)*100)
+ int3d(Th,2)( Grad(uh)'*Grad(vh)*2)
+ int3d(Th) (vh*f)
;

P;

plot(uh,wait=1, nbiso=6);
medit(" uh ",Th, uh);

```

### 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

$$\begin{aligned}
 -\Delta u &= f && \text{in } \Omega \\
 u &= g && \text{on } \Gamma_D, \quad \partial u / \partial n = 0 \quad \text{on } \Gamma_N
 \end{aligned} \tag{9.16}$$

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

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

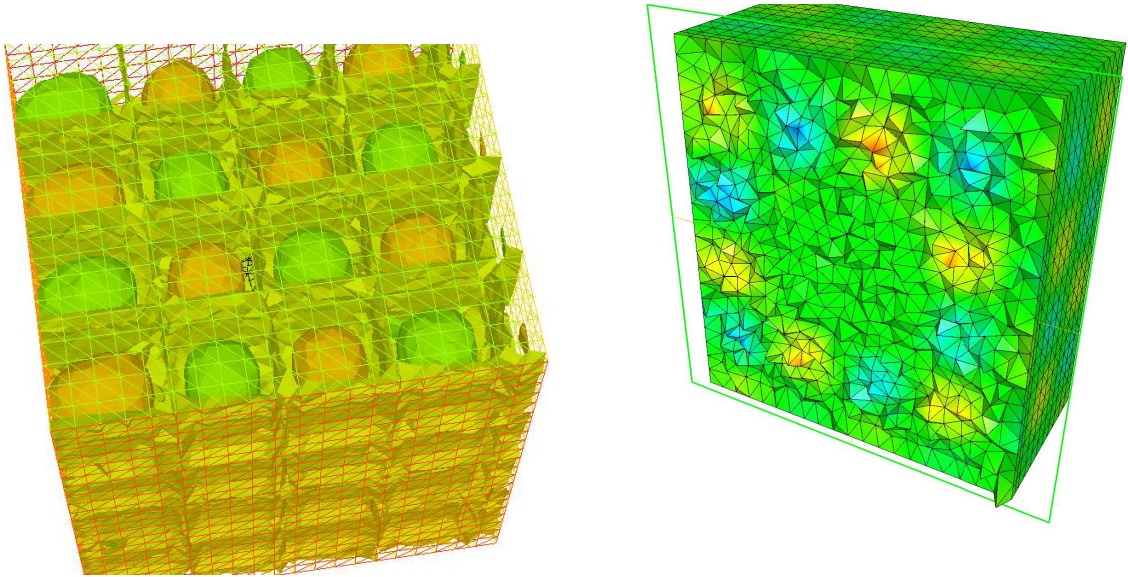


Figure 9.10:

Figure 9.9: view of the surface isovalue of periodic solution  $uh$  view a the cut of the solution  $uh$  with `ffmedit`

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  $\gamma_j$  such that  $(r_1, \theta_1) = (r, \theta)$ . Instead of poler coordinate system  $(r, \theta)$ , we use that  $r = \text{sqrt}(x^2 + y^2)$  and  $\theta = \text{atan2}(y, x)$  in FreeFem++ .

**Example 9.9** Assume that  $f = -2 \times 30(x^2 + y^2)$  and  $g = u_e = 10(x^2 + y^2)^{1/4} \sin([\tan^{-1}(y/x)]/2) + 30(x^2 y^2)$ , where  $u_e S$  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

$$\begin{aligned}-\Delta p &= 1 && \text{in } \Omega \\ p &= g_d && \text{on } \Gamma_D, \quad \partial p / \partial n = g_n && \text{on } \Gamma_N\end{aligned}\tag{9.17}$$

where  $\Gamma_D$  is a part of the boundary  $\Gamma$  and  $\Gamma_N = \Gamma \setminus \overline{\Gamma_D}$ .

The mixte formulation is: find  $p$  and  $\mathbf{u}$  such that

$$\begin{aligned}\nabla p + \mathbf{u} &= \mathbf{0} && \text{in } \Omega \\ \nabla \cdot \mathbf{u} &= f && \text{in } \Omega \\ p &= g_d && \text{on } \Gamma_D, \quad \partial u \cdot \mathbf{n} = \mathbf{g}_n \cdot \mathbf{n} && \text{on } \Gamma_N\end{aligned}\tag{9.18}$$

where  $\mathbf{g}_n$  is a vector such that  $\mathbf{g}_n \cdot \mathbf{n} = g_n$ .

The variationnal formulation is,

$$\begin{aligned} \forall \mathbf{v} \in \mathbb{V}_0, \quad \int_{\Omega} p \nabla \cdot \mathbf{v} + \mathbf{v} \mathbf{v} &= \int_{\Gamma_d} g_d \mathbf{v} \cdot \mathbf{n} \\ \forall q \in \mathbb{P} \quad \int_{\Omega} q \nabla \cdot \mathbf{u} &= \int_{\Omega} q f \\ \partial u \cdot \mathbf{n} &= \mathbf{g}_n \cdot \mathbf{n} \quad \text{on } \Gamma_N \end{aligned} \quad (9.19)$$

where the functionnal space are:

$$\mathbb{P} = L^2(\Omega), \quad \mathbb{V} = H(\text{div}) = \{\mathbf{v} \in L^2(\Omega)^2, \nabla \cdot \mathbf{v} \in L^2(\Omega)\}$$

and

$$\mathbb{V}_0 = \{\mathbf{v} \in \mathbb{V}; \quad \mathbf{v} \cdot \mathbf{n} = 0 \quad \text{on } \Gamma_N\}.$$

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  $\text{RT0}$  and  $\mathbb{P}$  is discretize by constant finite element  $\text{P0}$ .

#### Example 9.10 (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  $\Gamma_D$ 
+ on(4,u1=g1n,u2=g2n); // on  $\Gamma_N$ 

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  $\eta_T$  on the element  $T$  for the Poisson problem.

**Example 9.11 (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 Problem1(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  $\eta_T$  is:

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

where  $h_T$  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  $\eta_T^2$ , with test function constant function in each triangle.

```

29 : *****/
30 :
31 : varf indicator2(uu,chiK) =
32 :   intalldges(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 :   Problem1;
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  $\eta$  on your computer as in Fig. 9.11.

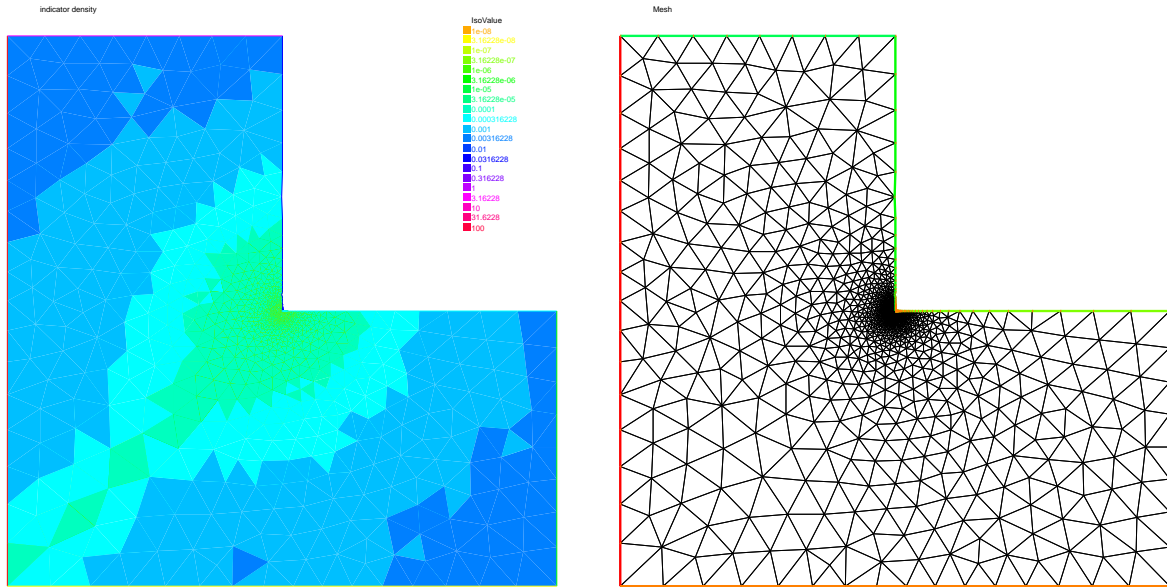


Figure 9.11: Density of the error indicator with isotropic  $P_2$  metric

### 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  $\eta_n^* = \text{mean}(\eta_n)c$ , and  $c$  is an user coefficient generally close to one.

#### Example 9.12 (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)=intalldges(Th,qfnpE=1)(v);
/* number of edge / par vertex */
varf vedgecount(u,v)=intalldges(Th,qfnpE=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,

```

fespace Vh(Th,P1); // FE space definition ---
fespace Ph(Th,P0); // for the mesh size and solution
// 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) =
    intalldges(Th)(chiK*lenEdge*square(jump(N.x*dx(u)+N.y*dy(u))))
    +int2d(Th)(chiK*square(hTriangle*(f+dxx(u)+dyy(u))) );

for (int i=0;i< 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);
}

```

## 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  $\mathbf{u} = (u_1, u_2, u_3) = (\xi_1 - x_1, \xi_2 - x_2, \xi_3 - x_3)$  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  $\tau$ , 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

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

then we have (see e.g. [16, p.32])

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



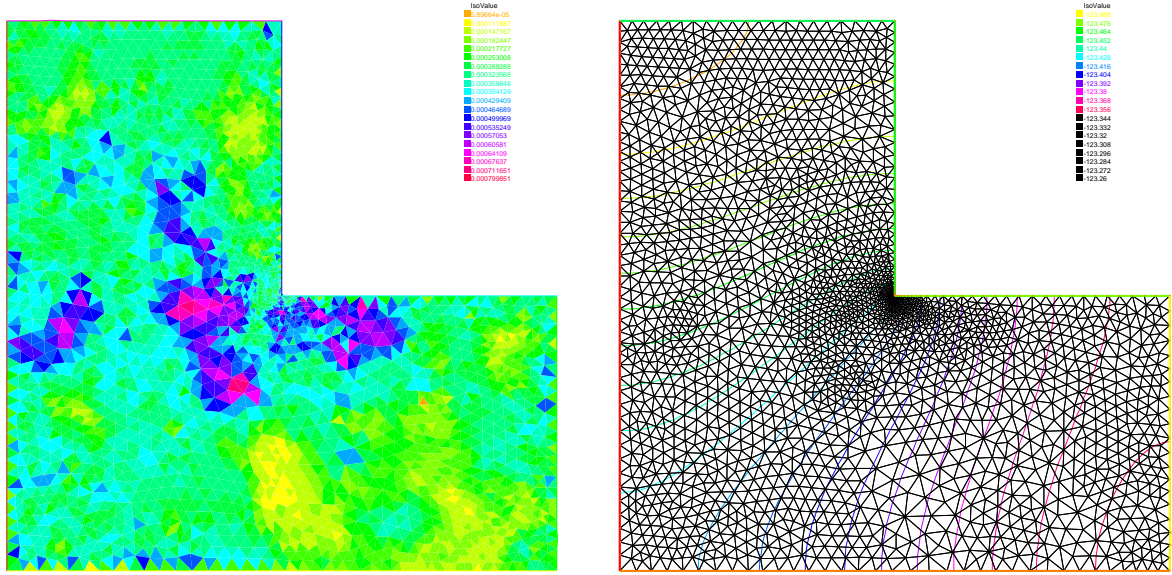


Figure 9.12: the error indicator with isotropic  $P_1$ , the mesh and isovalues of the solution

where  $v_i = \Delta x_i |\Delta \mathbf{x}|^{-1}$ . 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  $\varepsilon_{ij}$  is called *finite strain tensor*.

Consider the small plane  $\Delta \Pi(\mathbf{x})$  centered at  $\mathbf{x}$  with the unit normal direction  $\mathbf{n} = (n_1, n_2, n_3)$ , 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  $\sigma_{ij}$  and  $\varepsilon_{ij}$  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*  $\lambda, \mu$  (see e.g. [16, p.43]) satisfying

$$\sigma_{ij} = \lambda \delta_{ij} \text{div} u + 2\mu \varepsilon_{ij} \quad (9.20)$$

where  $\delta_{ij}$  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:

$$-\partial_j \sigma_{ij} = f_i \text{ in } \Omega \times ]-h, h[, \quad i = 1, 2, 3 \quad (9.21)$$

$$\sigma_{ij} n_j = g_i \text{ on } \Gamma_N \times ]-h, h[, \quad u_i = 0 \text{ on } \Gamma_D \times ]-h, h[, \quad i = 1, 2, 3 \quad (9.22)$$

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  $\mathbf{u}(x_1, x_2, x_3) = \bar{\mathbf{u}}(x_1, x_2)$  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  $\mathbf{u}(x_1, x_2, x_3) = \bar{\mathbf{u}}(x_1, x_2)$  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,

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

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

In what follows we omit the overlines of  $\bar{u}, \bar{f}, \bar{g}, \bar{\varepsilon}_{ij}$  and  $\bar{\sigma}_{ij}$ . Then we obtain similar equation of equilibrium given in (9.21) replacing  $\Omega \times ]-h, h[$  with  $\Omega$  and changing  $i = 1, 2$ . In the case of plane stress,  $\sigma_{ij} = \lambda^* \delta_{ij} \text{div} \mathbf{u} + 2\mu \varepsilon_{ij}$ ,  $\lambda^* = (2\lambda\mu)/(\lambda + \mu)$ .

The equations of elasticity are naturally written in variational form for the displacement vector  $\mathbf{u}(x) \in V$  as

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

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

**Example 9.13 (Beam.edp)** Consider elastic plate with the undeformed rectangle shape  $]0, 10[ \times ]0, 2[$ . The body force is the gravity force  $\mathbf{f}$  and the boundary force  $\mathbf{g}$  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;}; // righth beam
border d(t=0,10) { x=10-t; y=2; label=3;}; // top beam
real E = 21.5;
real sigma = 0.29;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -0.05;
mesh th = buildmesh( b(20)+c(5)+d(20)+a(5));
fespace Vh(th,[P1,P1]);
Vh [uu,vv], [w,s];
cout << "lambda,mu,gravity =" << lambda << " " << mu << " " << gravity << endl;
// deformation of a beam under its own weight
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);

```

**Example 9.14 (beam-3d.edp)** Consider elastic box with the undeformed parallelepiped shape  $]0, 5[ \times ]0, 1[ \times ]0, 1[$ . The body force is the gravity force  $f$  and the boundary force  $g$  is zero on all face except one the one vertical left face where the beam is fixed.

```

include "cube.idp"
int[int]   Nxyz=[20,5,5];
real [int,int] Bxyz=[[0.,5.],[0.,1.],[0.,1.]];
int [int,int] Lxyz=[[1,2],[2,2],[2,2]];
mesh3 Th=Cube(Nxyz,Bxyz,Lxyz);

real E = 21.5e4, sigma = 0.29;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -0.05;

fespace Vh(Th,[P1,P1,P1]);
Vh [u1,u2,u3], [v1,v2,v3];
cout << "lambda,mu,gravity ="<<lambda<< " " << mu << " " << gravity << endl;

real sqrt2=sqrt(2.);
macro epsilon(u1,u2,u3) [dx(u1),dy(u2),dz(u3),(dz(u2)+dy(u3))/sqrt2,
                        (dz(u1)+dx(u3))/sqrt2,(dy(u1)+dx(u2))/sqrt2]
macro div(u1,u2,u3) ( dx(u1)+dy(u2)+dz(u3) )

solve Lamé([u1,u2,u3],[v1,v2,v3])=
    int3d(Th)(
        lambda*div(u1,u2,u3)*div(v1,v2,v3)
        +2.*mu*( epsilon(u1,u2,u3)'*epsilon(v1,v2,v3) )
    )
    - int3d(Th) (gravity*v3)
    + on(1,u1=0,u2=0,u3=0)
;
real dmax= u1[].max;
cout << " max displacement = " << dmax << endl;
real coef= 0.1/dmax;
int[int] ref2=[1,0,2,0];
mesh3 Thm=movemesh3(Th,transfo=[x+u1*coef,y+u2*coef,z+u3*coef],label=ref2);
Thm=change(Thm,label=ref2);
plot(Th,Thm, wait=1,cmm="coef amplification = "+coef );

```

// EOM  
// EOM  
// ')

// see fig 9.13

coef amplification = 3997.95

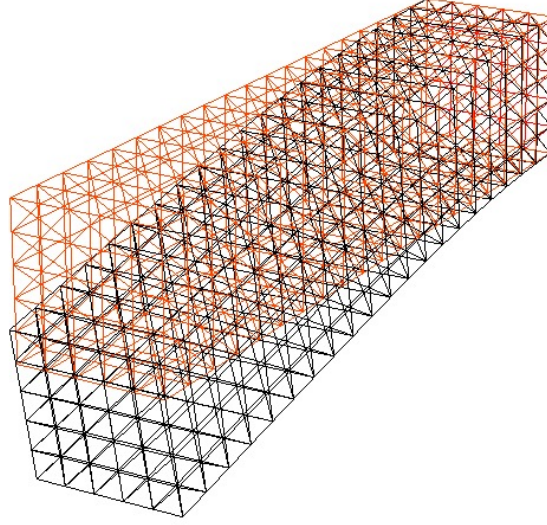


Figure 9.13: 3d Beam deformed and undeformed box

### 9.2.1 Fracture Mechanics

Consider the plate with the crack whose undeformed shape is a curve  $\Sigma$  with the two edges  $\gamma_1, \gamma_2$ . We assume the stress tensor  $\sigma_{ij}$  is the state of plate stress regarding  $(x, y) \in \Omega_\Sigma = \Omega \setminus \Sigma$ . Here  $\Omega$  stands for the undeformed shape of elastic plate without crack. If the part  $\Gamma_N$  of the boundary  $\partial\Omega$  is fixed and a load  $\mathcal{L} = (\mathbf{f}, \mathbf{g}) \in L^2(\Omega)^2 \times L^2(\Gamma_N)^2$  is given, then the displacement  $\mathbf{u}$  is the minimizer of the potential energy functional

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

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

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

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

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

If the elasticity is homogeneous isotropic, then the displacement  $\mathbf{u}(x)$  is decomposed in an open neighborhood  $U_k$  of  $\gamma_k$  as in (see e.g. [17])

$$\mathbf{u}(x) = \sum_{l=1}^2 K_l(\gamma_k) r_k^{1/2} S_{kl}^C(\theta_k) + \mathbf{u}_{k,R}(x) \quad \text{for } x \in \Omega_\Sigma \cap U_k, k = 1, 2 \quad (9.23)$$

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

$$\begin{aligned} S_{k1}^C(\theta_k) &= \frac{1}{4\mu} \frac{1}{(2\pi)^{1/2}} \begin{bmatrix} [2\kappa - 1] \cos(\theta_k/2) - \cos(3\theta_k/2) \\ -[2\kappa + 1] \sin(\theta_k/2) + \sin(3\theta_k/2) \end{bmatrix}, \\ S_{k2}^C(\theta_k) &= \frac{1}{4\mu} \frac{1}{(2\pi)^{1/2}} \begin{bmatrix} -[2\kappa - 1] \sin(\theta_k/2) + 3 \sin(3\theta_k/2) \\ -[2\kappa + 1] \cos(\theta_k/2) + \cos(3\theta_k/2) \end{bmatrix}. \end{aligned} \quad (9.24)$$

where  $\mu$  is the shear modulus of elasticity,  $\kappa = 3 - 4\nu$  ( $\nu$  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

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

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  $\Sigma$  is approximated by

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

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  $\gamma$ .
- After obtaining the displacement vector  $\mathbf{u} = (u, v)$ , we shall watch the deformation of the crack near  $\gamma$  as follows,

```
mesh Plate = movemesh(Zoom, [x+u, y+v]);
plot(Plate);
```

- Important technique is adaptive mesh, because the large singularity occur at  $\gamma$  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  $\Sigma$ , 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

$$\sigma_1 - \sigma_2 = \sqrt{(\sigma_{11} - \sigma_{22})^2 + 4\sigma_{12}^2} \quad (9.25)$$

where  $\sigma_1$  and  $\sigma_2$  are the principal stresses. In opening mode, the photoelasticity make symmetric pattern concentrated at  $\gamma$ .

**Example 9.15 (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  $\gamma$ 
    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. 9.14
        plot(N,ps="1stPhoto.eps",bw=1); // Fig. 9.14
    } else if (i==5) {
        plot(Plate,ps="LastCOD.eps",bw=1); // Fig. 9.15
        plot(N,ps="LastPhoto.eps",bw=1); // Fig. 9.15
        break;
    }
    Th=adaptmesh(Th,[u,v]);
    Problem;
}

```

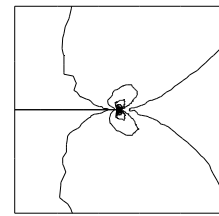
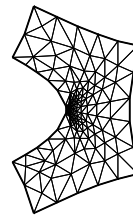
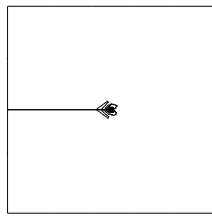
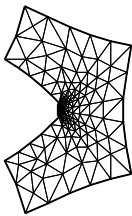


Figure 9.14: Crack open displacement (COD) and Principal stress difference in the first mesh      Figure 9.15: COD and Principal stress difference in the last adaptive mesh

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  $\Sigma$ , that is, the  $x$ -component  $f_1$  of the body force  $\mathbf{f}$  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 9.16 (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  $\gamma$

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. 9.17

**plot**(N,ps="1stPhoto2.eps",bw=1);

// Fig. 9.16

} **else if** (i==3) {

**plot**(Plate,ps="LastCOD2.eps",bw=1);

// Fig. 9.17

**plot**(N,ps="LastPhoto2.eps",bw=1);

// Fig. 9.17

**break**;

}

Th=**adaptmesh**(Th,[u,v]);

Problem;

}

## 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

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



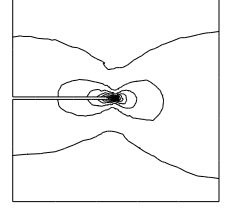
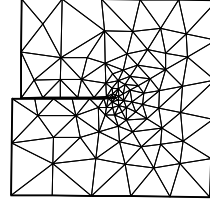
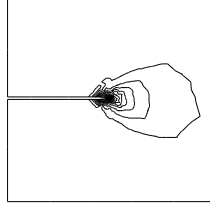
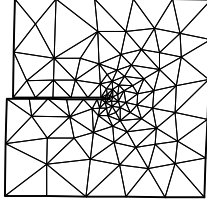


Figure 9.16: (COD) and Principal stress difference in the first mesh

Figure 9.17: COD and Principal stress difference in the last adaptive mesh

### 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 introduce the two variational form  $vdJ$  and  $vhJ$  to compute respectively  $\nabla J$  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''(|\nabla u|^2) optimisation

// the variational form of evaluate dJ = \nabla J
// -----
// dJ = f'()*( dx(u)*dx(vh) + dy(u)*dy(vh) )
varf vdJ(uh,vh) = int2d(Th)( alpha*( dx(u)*dx(vh) + dy(u)*dy(vh) ) - b*vh)
+ on(1,2,3,4, uh=0);

// the variational form of evaluate ddJ = \nabla^2 J
// hJ(uh,vh) = f'()*( dx(uh)*dx(vh) + dy(uh)*dy(vh) )
// + 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 = \nabla 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++`<sup>1</sup> the object-oriented version of ARPACK eigenvalue package [1]. The function `EigenValue` compute the generalized eigenvalue of  $Au = \lambda Bu$  where  $\sigma$  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 (6.20)). 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  $k$ th vector will contain the real part and the  $k + 1$ th vector the imaginary part of the corresponding complex conjugate eigenvectors.

**tol**= the relative accuracy to which eigenvalues are to be determined;

**sigma**= the shift value;

**maxit**= the maximum number of iterations allowed;

**ncv**= the number of Arnoldi vectors generated at each iteration of ARPACK.

---

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

**Example 9.17 (lapEigenValue.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:  $\lambda$ , and  $\nabla u_\lambda$  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 `arpac++` 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_0¹(Ω) such that:
//
//      ∫_Ω ∇u_λ ∇v = λ ∫_Ω u_λ v, ∀ v ∈ H_0¹(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=lgv 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

```



## 9.5 Evolution Problems

FreeFem++ also solve evolution problems such as the heat problem

$$\begin{aligned} \frac{\partial u}{\partial t} - \mu \Delta u &= f \quad \text{in } \Omega \times ]0, T[, \\ u(\mathbf{x}, 0) &= u_0(\mathbf{x}) \quad \text{in } \Omega; \quad (\partial u / \partial n)(\mathbf{x}, t) = 0 \quad \text{on } \partial\Omega \times ]0, T[. \end{aligned} \quad (9.26)$$

with a positive viscosity coefficient  $\mu$  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,

$$\frac{\partial u}{\partial t}(x, y, t) = \lim_{\tau \rightarrow 0} \frac{u(x, y, t) - u(x, y, t - \tau)}{\tau}$$

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:

$$\begin{aligned} \frac{u^{m+1} - u^m}{\tau} - \mu \Delta u^{m+1} &= f^{m+1} \quad \text{in } \Omega \\ u^0(\mathbf{x}) &= u_0(\mathbf{x}) \quad \text{in } \Omega; \quad \partial u^{m+1} / \partial n(\mathbf{x}) = 0 \quad \text{on } \partial\Omega, \quad \text{for all } m = 0, \dots, [T/\tau], \end{aligned} \quad (9.27)$$

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

$$\int_{\Omega} \{u^{m+1}v + \tau \nabla u^{m+1} \cdot \nabla v\} - \int_{\partial\Omega} \tau (\partial u^{m+1} / \partial n) v = \int_{\Omega} \{u^m v + \tau f^{m+1}v\}.$$

By the boundary condition  $\partial u^{m+1} / \partial n = 0$ , it follows that

$$\int_{\Omega} \{u^{m+1}v + \tau \nabla u^{m+1} \cdot \nabla v\} - \int_{\Omega} \{u^m v + \tau f^{m+1}v\} = 0. \quad (9.28)$$

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 9.18** We now solve the following example with the exact solution  $u(x, y, t) = tx^4$ .

$$\begin{aligned} \frac{\partial u}{\partial t} - \mu \Delta u &= x^4 - \mu 12tx^2 \quad \text{in } \Omega \times ]0, 3[, \quad \Omega = ]0, 1[^2 \\ u(x, y, 0) &= 0 \quad \text{on } \Omega, \quad u|_{\partial\Omega} = t * x^4 \end{aligned}$$

// heat equation  $\partial_t u = -\mu \Delta u = x^4 - \mu 12tx^2$

```
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;
uu = 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( $f, \Omega$ ):** For a given  $f \in L^2(0, T; V')$ ,  $u^0 \in H$

$$\begin{aligned} \frac{d}{dt}(u(t), v) + a(u(t), v) &= (f(t), v) \quad \forall v \in V, \quad a.e. t \in [0, T] \\ u(0) &= u^0 \end{aligned} \quad (9.29)$$

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  $\theta \in [0, 1]$

$$\begin{aligned} \frac{1}{\tau} (u_h^{n+1} - u_h^n, \phi_i) + a(u_h^{n+\theta}, \phi_i) &= (f^{n+\theta}, \phi_i) \\ i &= 1, \dots, m, \quad n = 0, \dots, N_T \\ u_h^{n+\theta} &= \theta u_h^{n+1} + (1 - \theta) u_h^n, \quad f^{n+\theta} = \theta f^{n+1} + (1 - \theta) f^n \end{aligned} \quad (9.30)$$

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  $u^n = (u_h^1, \dots, u_h^M)^T$  in

$$u_h^n(x) = u_1^n \phi_1(x) + \dots + u_m^n \phi_m(x), \quad u_1^n, \dots, u_m^n \in \mathbb{R}$$

are obtained from solving the matrix

$$(M + \theta\tau A)u^{n+1} = \{M - (1 - \theta)\tau A\}u^n + \tau \{\theta f^{n+1} + (1 - \theta)f^n\} \quad (9.31)$$

$$M = (m_{ij}), \quad m_{ij} = (\phi_j, \phi_i), \quad A = (a_{ij}), \quad a_{ij} = a(\phi_j, \phi_i)$$

Refer [22, pp.70–75] for solvability of (9.31). The stability of (9.31) is in [22, 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,

$$|u_h^n|^2 \leq \begin{cases} \frac{1}{\delta} \left\{ |u_h^0|^2 + \tau \sum_{k=0}^{n-1} \|f^{k+\theta}\|_{V_h'}^2 \right\} & \theta \in [0, 1/2) \\ |u_h^0|^2 + \tau \sum_{k=0}^{n-1} \|f^{k+\theta}\|_{V_h'}^2 & \theta \in [1/2, 1] \end{cases} \quad (9.32)$$

if the following are satisfied:

1. When  $\theta \in [0, 1/2)$ , then we can take a time step  $\tau$  in such a way that

$$\tau < \frac{2(1 - \delta)}{(1 - 2\theta)c_0^2} h^2 \quad (9.33)$$

for arbitrary  $\delta \in (0, 1)$ .

2. When  $1/2 \leq \theta \leq 1$ , we can take  $\tau$  arbitrary.

### Example 9.19

```

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;
Vh err = u - uex;
out<< abs(err[]).max)/abs(uex[]).max) <<" , ";
}
out << endl;
theta = theta + 0.1;
}

```

// exact sol.=  $tx^2(1-x)^2$   
// err=FE-sol - exact  
//  $\|err\|_{L^\infty(\Omega)}/\|u_{ex}\|_{L^\infty(\Omega)}$

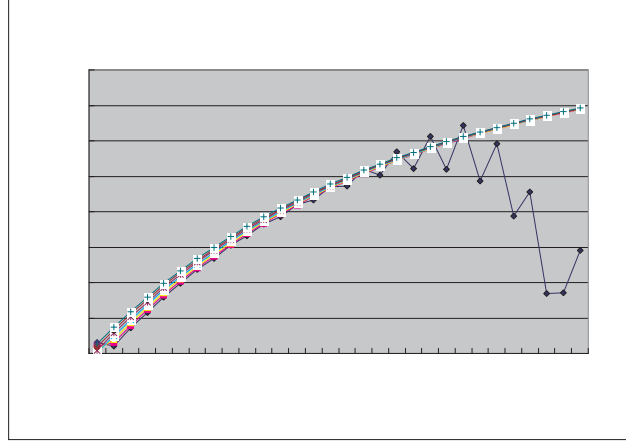


Figure 9.20:  $\max_{x \in \Omega} |u_h^n(\theta) - u_{ex}(n\tau)| / \max_{x \in \Omega} |u_{ex}(n\tau)|$  at  $n = 0, 1, \dots, 29$

We can see in Fig. 9.20 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

$$\partial_t u + \alpha \cdot \nabla u = f; \text{ for a vector-valued function } \alpha, \partial_t = \frac{\partial}{\partial t}, \quad (9.34)$$

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  $\alpha$  is constant,

$$\partial_t u + \alpha \partial_x u = 0, \quad u(x, 0) = u^0(x), \quad (9.35)$$

because  $\partial_t u + \alpha \partial_x u = -\alpha \dot{u}^0 + \alpha \dot{u}^0 = 0$ , where  $\dot{u}^0 = du^0(x)/dx$ . Even if  $\alpha$  is not constant construction, the principle is similar. One begins the ordinary differential equation (with convention which  $\alpha$  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  $\tau$  is the variable and  $x, t$  is parameters, and we denote the solution by  $X_{x,t}(\tau)$ . Then it is noticed that  $(x, t) \rightarrow v(X(\tau), \tau)$  in  $\tau = t$  satisfy the equation

$$\partial_t v + \alpha \partial_x v = \partial_t X \dot{v} + \alpha \partial_x X \dot{v} = 0$$

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 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

$$\partial_t u + \alpha \cdot \nabla u = 0 \text{ in } \Omega \times (0, T)$$

where  $\alpha(x, t) \in R^d$ . FreeFem++ implements the Characteristic-Galerkin method for convection operators. Recall that the equation (9.34) can be discretized as

$$\frac{Du}{Dt} = f \text{ i.e. } \frac{du}{dt}(X(t), t) = f(X(t), t) \text{ where } \frac{dX}{dt}(t) = \alpha(X(t), t)$$

where  $D$  is the total derivative operator. So a good scheme is one step of backward convection by the method of Characteristics-Galerkin

$$\frac{1}{\tau} (u^{m+1}(x) - u^m(X^m(x))) = f^m(x) \quad (9.36)$$

where  $X^m(x)$  is an approximation of the solution at  $t = m\tau$  of the ordinary differential equation

$$\frac{dX}{dt}(t) = \alpha^m(X(t)), \quad X((m+1)\tau) = x.$$

where  $\alpha^m(x) = (\alpha_1(x, m\tau), \alpha_2(x, m\tau))$ . Because, by Taylor's expansion, we have

$$\begin{aligned} u^m(X(m\tau)) &= u^m(X((m+1)\tau)) - \tau \sum_{i=1}^d \frac{\partial u^m}{\partial x_i}(X((m+1)\tau)) \frac{\partial X_i}{\partial t}((m+1)\tau) + o(\tau) \\ &= u^m(x) - \tau \alpha^m(x) \cdot \nabla u^m(x) + o(\tau) \end{aligned} \quad (9.37)$$

where  $X_i(t)$  are the  $i$ -th component of  $X(t)$ ,  $u^m(x) = u(x, m\tau)$  and we used the chain rule and  $x = X((m+1)\tau)$ . From (9.37), it follows that

$$u^m(X^m(x)) = u^m(x) - \tau \alpha^m(x) \cdot \nabla u^m(x) + o(\tau). \quad (9.38)$$

Also we apply Taylor's expansion for  $t \mapsto u^m(x - \alpha^m(x)t)$ ,  $0 \leq t \leq \tau$ , then

$$u^m(x - \alpha\tau) = u^m(x) - \tau \alpha^m(x) \cdot \nabla u^m(x) + o(\tau).$$

Putting

$$\text{convect}(\alpha, -\tau, u^m) \approx u^m(x - \alpha^m\tau),$$

we can get the approximation

$$u^m(X^m(x)) \approx \text{convect}([a_1^m, a_2^m], -\tau, u^m) \text{ by } X^m \approx x \mapsto x - \tau[a_1^m(x), a_2^m(x)].$$

A classical convection problem is that of the “rotating bell” (quoted from [14][p.16]). Let  $\Omega$  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)

$$u^{m+1}(x) = u^m(X^m(x)) \approx \text{convect}(\alpha, -\tau, u^m).$$

The exact solution is  $u(x, t) = u(X(t))$  where  $X$  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 9.20 (convect.edp) border C(t=0, 2*pi) { x=cos(t); y=sin(t); }; // the unit
circle
mesh Th = buildmesh(C(70)); // triangulates the disk
fespace Vh(Th,P1);
Vh u0 = exp(-10*((x-0.3)^2 +(y-0.3)^2)); // give u0

real dt = 0.17,t=0; // time step
Vh a1 = -y, a2 = x; // rotation velocity
Vh u; // um+1
for (int m=0; m<2*pi/dt ; m++) {
    t += dt;
    u=convect([a1,a2],-dt,u0); // um+1 = um(Xm(x))
    u0=u; // m++
    plot(u,cmm=" 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. 9.22).

convection: t=0, min=1.55289e-09, max=0.983612

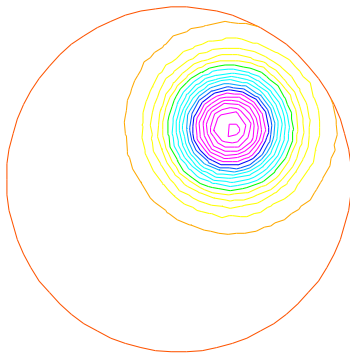


Figure 9.21:  $u^0 = e^{-10((x-0.3)^2 + (y-0.3)^2)}$

convection: t=6.29, min=1.55289e-09, max=0.40659m=37

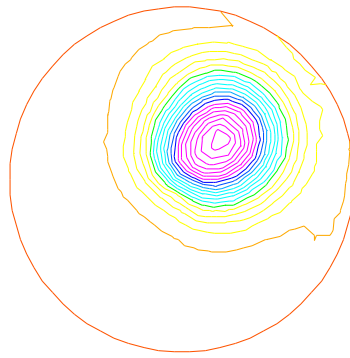


Figure 9.22: The bell at  $t = 6.29$

### 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[39] 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

$$\sigma_1 = 0.3, \quad \sigma_2 = 0.3, \quad \rho = 0.3, \quad r = 0.05, \quad K = 40, \quad T = 0.5 \quad (9.41)$$

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

$$\frac{\partial u}{\partial t} + a_1 \frac{\partial u}{\partial x} + a_2 \frac{\partial u}{\partial y} \approx \frac{1}{\tau} (u^{n+1}(x) - u^n(x - \alpha\tau)) \quad (9.42)$$

**Example 9.21** [*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. 9.21).

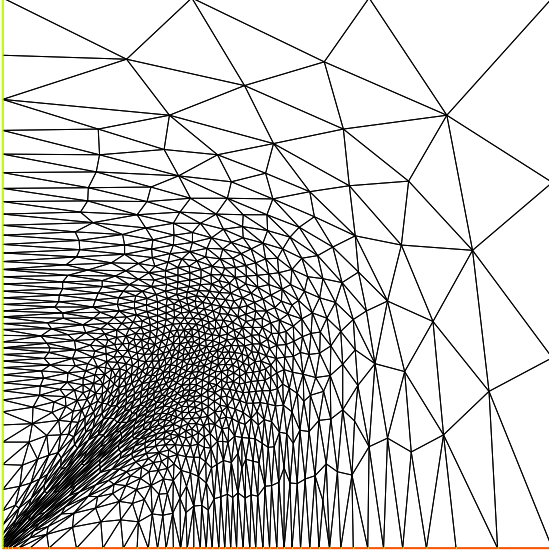


Figure 9.23: The adapted triangulation

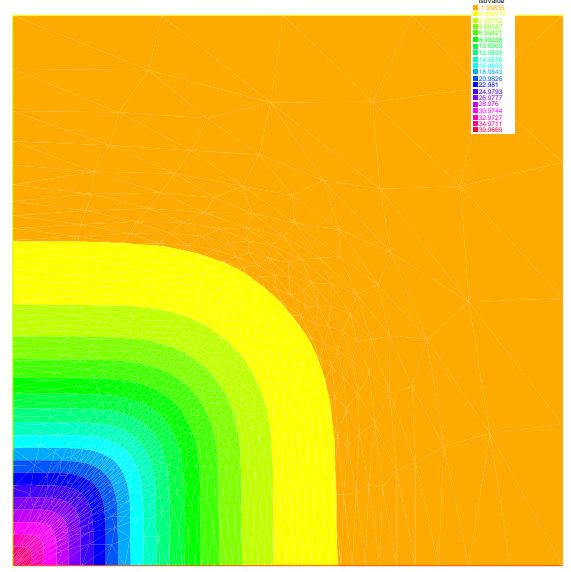


Figure 9.24: The level line of the European basquet put option

## 9.6 Navier-Stokes Equation

### 9.6.1 Stokes and Navier-Stokes

The Stokes equations are: for a given  $f \in L^2(\Omega)^2$ ,

$$\left. \begin{aligned} -\Delta \mathbf{u} + \nabla p &= \mathbf{f} \\ \nabla \cdot \mathbf{u} &= 0 \end{aligned} \right\} \quad \text{in } \Omega \quad (9.43)$$

where  $\mathbf{u} = (u_1, u_2)$  is the velocity vector and  $p$  the pressure. For simplicity, let us choose Dirichlet boundary conditions on the velocity,  $\mathbf{u} = \mathbf{u}_\Gamma$  on  $\Gamma$ .

In Temam [Theorem 2.2], there is a weak form of (9.43): Find  $\mathbf{v} = (v_1, v_2) \in \mathbf{V}(\Omega)$

$$\mathbf{V}(\Omega) = \{\mathbf{w} \in H_0^1(\Omega)^2 \mid \operatorname{div} \mathbf{w} = 0\}$$

which satisfy

$$\sum_{i=1}^2 \int_{\Omega} \nabla u_i \cdot \nabla v_i = \int_{\Omega} \mathbf{f} \cdot \mathbf{w} \quad \text{for all } \mathbf{v} \in \mathbf{V}$$

Here it is used the existence  $p \in H^1(\Omega)$  such that  $\mathbf{u} = \nabla p$ , if

$$\int_{\Omega} \mathbf{u} \cdot \mathbf{v} = 0 \quad \text{for all } \mathbf{v} \in \mathbf{V}$$

Another weak form is derived as follows: We put

$$\mathbf{V} = H_0^1(\Omega)^2; \quad \mathbf{W} = \left\{ q \in L^2(\Omega) \mid \int_{\Omega} q = 0 \right\}$$

By multiplying the first equation in (9.43) with  $v \in V$  and the second with  $q \in W$ , subsequent integration over  $\Omega$ , and an application of Green's formula, we have

$$\begin{aligned} \int_{\Omega} \nabla \mathbf{u} \cdot \nabla v - \int_{\Omega} \operatorname{div} v p &= \int_{\Omega} \mathbf{f} \cdot v \\ \int_{\Omega} \operatorname{div} \mathbf{u} q &= 0 \end{aligned}$$

This yields the weak form of (9.43): Find  $(\mathbf{u}, p) \in V \times W$  such that

$$a(\mathbf{u}, v) + b(v, p) = (\mathbf{f}, v) \quad (9.44)$$

$$b(\mathbf{u}, q) = 0 \quad (9.45)$$

for all  $(v, q) \in V \times W$ , where

$$a(\mathbf{u}, v) = \int_{\Omega} \nabla \mathbf{u} \cdot \nabla v = \sum_{i=1}^2 \int_{\Omega} \nabla u_i \cdot \nabla v_i \quad (9.46)$$

$$b(\mathbf{u}, q) = - \int_{\Omega} \operatorname{div} \mathbf{u} q \quad (9.47)$$

Now, we consider finite element spaces  $V_h \subset V$  and  $W_h \subset W$ , and we assume the following basis functions

$$\begin{aligned} V_h &= V_h \times V_h, \quad V_h = \{v_h \mid v_h = v_1 \phi_1 + \cdots + v_{M_V} \phi_{M_V}\}, \\ W_h &= \{q_h \mid q_h = q_1 \varphi_1 + \cdots + q_{M_W} \varphi_{M_W}\} \end{aligned}$$

The discrete weak form is: Find  $(\mathbf{u}_h, p_h) \in V_h \times W_h$  such that

$$\begin{aligned} a(\mathbf{u}_h, v_h) + b(v_h, p) &= (\mathbf{f}, v_h), \quad \forall v_h \in V_h \\ b(\mathbf{u}_h, q_h) &= 0, \quad \forall q_h \in W_h \end{aligned} \quad (9.48)$$

**Note 9.4** Assume that:

1. There is a constant  $\alpha_h > 0$  such that

$$a(v_h, v_h) \geq \alpha \|v_h\|_{1,\Omega}^2 \quad \text{for all } v_h \in Z_h$$

where

$$Z_h = \{v_h \in V_h \mid b(v_h, q_h) = 0 \quad \text{for all } q_h \in W_h\}$$

2. There is a constant  $\beta_h > 0$  such that

$$\sup_{v_h \in V_h} \frac{b(v_h, q_h)}{\|v_h\|_{1,\Omega}} \geq \beta_h \|q_h\|_{0,\Omega} \quad \text{for all } q_h \in W_h$$

Then we have an unique solution  $(\mathbf{u}_h, p_h)$  of (9.48) satisfying

$$\|\mathbf{u} - \mathbf{u}_h\|_{1,\Omega} + \|p - p_h\|_{0,\Omega} \leq C \left( \inf_{v_h \in V_h} \|\mathbf{u} - v_h\|_{1,\Omega} + \inf_{q_h \in W_h} \|p - q_h\|_{0,\Omega} \right)$$

with a constant  $C > 0$  (see e.g. [20, Theorem 10.4]).

Let us denote that

$$\begin{aligned} \mathbf{A} &= (A_{ij}), A_{ij} = \int_{\Omega} \nabla \phi_j \cdot \nabla \phi_i \quad i, j = 1, \dots, M_V \\ \mathbf{B} &= (Bx_{ij}, By_{ij}), Bx_{ij} = - \int_{\Omega} \partial \phi_j / \partial x \phi_i \quad By_{ij} = - \int_{\Omega} \partial \phi_j / \partial y \phi_i \\ &\quad i = 1, \dots, M_W; j = 1, \dots, M_V \end{aligned} \quad (9.49)$$

then (9.48) is written by

$$\begin{pmatrix} \mathbf{A} & \mathbf{B}^* \\ \mathbf{B} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{U}_h \\ \{p_h\} \end{pmatrix} = \begin{pmatrix} \mathbf{F}_h \\ 0 \end{pmatrix} \quad (9.50)$$

where

$$\mathbf{A} = \begin{pmatrix} A & 0 \\ 0 & A \end{pmatrix} \quad \mathbf{B}^* = \begin{pmatrix} Bx^T \\ By^T \end{pmatrix} \quad \mathbf{U}_h = \begin{pmatrix} \{u_{1,h}\} \\ \{u_{2,h}\} \end{pmatrix} \quad \mathbf{F}_h = \begin{pmatrix} \{\int_{\Omega} f_1 \phi_i\} \\ \{\int_{\Omega} f_2 \phi_i\} \end{pmatrix}$$

**Penalty method:** This method consists of replacing (9.48) by a more regular problem: Find  $(\mathbf{v}_h^\epsilon, p_h^\epsilon) \in \mathbf{V}_h \times \tilde{W}_h$  satisfying

$$\begin{aligned} a(\mathbf{u}_h^\epsilon, \mathbf{v}_h) + b(\mathbf{v}_h, p_h^\epsilon) &= (\mathbf{f}, \mathbf{v}_h), \quad \forall \mathbf{v}_h \in \mathbf{V}_h \\ b(\mathbf{u}_h^\epsilon, q_h) - \epsilon(p_h^\epsilon, q_h) &= 0, \quad \forall q_h \in \tilde{W}_h \end{aligned} \quad (9.51)$$

where  $\tilde{W}_h \subset L^2(\Omega)$ . Formally, we have

$$\operatorname{div} \mathbf{u}_h^\epsilon = \epsilon p_h^\epsilon$$

and the corresponding algebraic problem

$$\begin{pmatrix} \mathbf{A} & \mathbf{B}^* \\ \mathbf{B} & -\epsilon I \end{pmatrix} \begin{pmatrix} \mathbf{U}_h^\epsilon \\ \{p_h^\epsilon\} \end{pmatrix} = \begin{pmatrix} \mathbf{F}_h \\ 0 \end{pmatrix}$$

**Note 9.5** We can eliminate  $p_h^\epsilon = (1/\epsilon) \mathbf{B} \mathbf{U}_h^\epsilon$  to obtain

$$(\mathbf{A} + (1/\epsilon) \mathbf{B}^* \mathbf{B}) \mathbf{U}_h^\epsilon = \mathbf{F}_h^\epsilon \quad (9.52)$$

Since the matrix  $\mathbf{A} + (1/\epsilon) \mathbf{B}^* \mathbf{B}$  is symmetric, positive-definite, and sparse, (9.52) can be solved by known technique. There is a constant  $C > 0$  independent of  $\epsilon$  such that

$$\|\mathbf{u}_h - \mathbf{u}_h^\epsilon\|_{1,\Omega} + \|p_h - p_h^\epsilon\|_{0,\Omega} \leq C\epsilon$$

(see e.g. [20, 17.2])

**Example 9.22 (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  $\mathbf{u}_\Gamma \cdot \mathbf{n} = 0$  and  $\mathbf{u}_\Gamma \cdot \mathbf{s} = 1$  on the top boundary and zero elsewhere ( $\mathbf{n}$  is the unit normal to  $\Gamma$ , and  $\mathbf{s}$  the unit tangent to  $\Gamma$ ).

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

$$X_h = \{v \in H^1(\Omega, \mathbb{R}^2) \mid \forall K \in \mathcal{T}_h \quad v|_K \in P_2\}$$

and

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

The FE spaces and functions are constructed by

```
fespace Xh(Th,P2);           // definition of the velocity component space
fespace Mh(Th,P1);           // definition of the pressure space
Xh u2,v2;
Xh u1,v1;
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=Croux) =
  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  $\psi$  such that  $\text{rot}\psi = u$  or better,

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

```
Xh psi,phi;
```

```
solve streamlines(psi,phi) =
  int2d(Th)( dx(psi)*dx(phi) + dy(psi)*dy(phi))
+ int2d(Th)( -phi*(dy(u1)-dx(u2)))
+ on(1,2,3,4,psi=0);
```

Now the Navier-Stokes equations are solved

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

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



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

$$\begin{aligned} \frac{1}{\tau}(u^{n+1} - u^n \circ X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} &= 0, \\ \nabla \cdot u^{n+1} &= 0 \end{aligned} \quad (9.53)$$

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. [20, 17.3], [29, 13] or [30, 13]):

**Initialize:** Let  $p_h^0$  be an arbitrary chosen element of  $L^2(\Omega)$ .

**Calculate  $u_h^n$ :** Once  $p_h^n$  is known,  $v_h^n$  is the solution of

$$u_h^n = A^{-1}(f_h - B^* p_h^n)$$

**Advance  $p_h$ :** Let  $p_h^{n+1}$  be defined by

$$p_h^{n+1} = p_h^n + \rho_n B u_h^n$$

There is a constant  $\alpha > 0$  such that  $\alpha \leq \rho_n \leq 2$  for each  $n$ , then  $\mathbf{u}_h^n$  converges to the solution  $\mathbf{u}_h$ , and then  $B\mathbf{v}_h^n \rightarrow 0$  as  $n \rightarrow \infty$  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 9.23 (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;
```

$p_h^n \rightarrow \mathbf{B}\mathbf{A}^{-1}(-\mathbf{B}^*p_h^n) = -\text{div}\mathbf{u}_h$  is realized as the function `divup`.

```

func real[int] divup(real[int] & pp)
{
    b[] = Bx'*pp; b[] *=-1; b[] += bc1[] ;    u1[] = A^-1*b[];           // compute u1(pp)
    b[] = By'*pp; b[] *=-1; b[] += bc2[] ;    u2[] = A^-1*b[];           // compute u2(pp)
    // u^n = A^-1(Bx^T p^n By^T p^n)^T
    ppp[] = Bx*u1[];                         // ppp = Bxu1
    ppp[] += By*u2[];                         // +Byu2
    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 + B u_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 [31] 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  $\nabla \cdot ((\alpha Id + \nu \Delta)^{-1} \nabla$ , where  $Id$  is the identity operator. So the preconditioner suggested is  $\alpha \Delta^{-1} + \nu Id$ .

To implement this, we reuse the previous example, by including a file. Then we define the time step  $\Delta t$ , viscosity, and new variational form and matrix.

#### Example 9.24 (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

varf at(u1,u2)= int2d(Th)( xnu*dx(u1)*dx(u2) // the new variational form with mass term
+ 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

varf vfconv1(uu,vv) = int2d(Th,qforder=5) (convect([u1,u2],-dt,u1)*vv*alpha); // set the 2 convect variational form
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)
{
    pprec[] = LapMh-1* xx; // xx = ∫(divu)wi
    prhs[] = MassMh-1*xx; // αLapMh-1 + νMassMh-1
    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 => absolute stop test
    LinearCG(divup,p[],eps=-1.e-6,nbiter=50,precon=CahouetChabart);
    divup(p[]);                                //    computed the velocity

    plot([u1,u2],p,wait=!(idt%10),value= 1,coef=0.1);
}

```

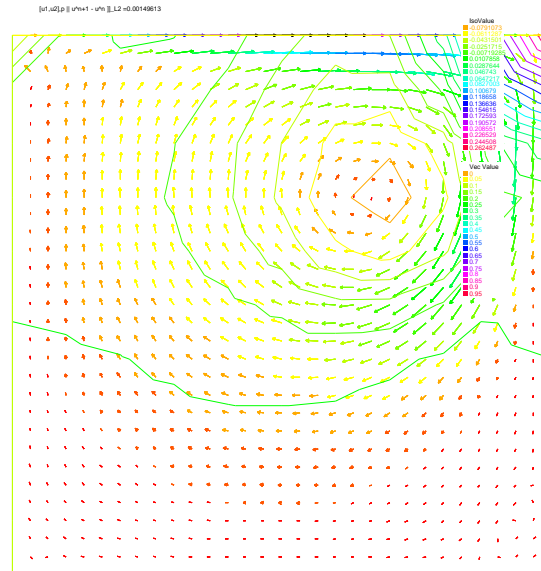


Figure 9.25: Solution of the cavity driven problem at Reynolds number 400 with the Cahouet-Chabart algorithm.

## 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 :

$$u = \arg \min_{u \in C} J(u) = \frac{1}{2} \int_{\Omega} \nabla u \cdot \nabla u - \int_{\Omega} f u$$

where  $f$  and  $g$  are given function.

The solution is a projection on the convex  $C$  of  $f^*$  for the scalar product  $((v, w)) = \int_{\Omega} \nabla v \cdot \nabla w$  of  $H_0^1(\Omega)$  where  $f^*$  is solution of  $((f^*, v)) = \int_{\Omega} f v, \forall v \in H_0^1(\Omega)$ . The projection on a convex satisfy clearly  $\forall v \in C, ((u - v, u - \tilde{f})) \leq 0$ , and after expanding, we get the classical inequality

$$\forall v \in C, \quad \int_{\Omega} \nabla(u - v) \cdot \nabla u \leq \int_{\Omega} (u - v) f.$$

We can also rewrite the problem as a saddle point problem

Find  $\lambda, u$  such that:

$$\max_{\lambda \in L^2(\Omega), \lambda \geq 0} \min_{u \in H_0^1(\Omega)} \mathcal{L}(u, \lambda) = \frac{1}{2} \int_{\Omega} \nabla u \cdot \nabla u - \int_{\Omega} f u + \int_{\Omega} \lambda (u - g)^+$$

where  $((u - g)^+ = \max(0, u - g))$

This saddle point problem is equivalent to find  $u, \lambda$  such that:

$$\begin{cases} \int_{\Omega} \nabla u \cdot \nabla v + \lambda v^+ d\omega = \int_{\Omega} f u, & \forall v \in H_0^1(\Omega) \\ \int_{\Omega} \mu (u - g)^+ = 0, & \forall \mu \in L^2(\Omega), \mu \geq 0, \lambda \geq 0, \end{cases} \quad (9.54)$$

A algorithm to solve the previous problem is:

1.  $k=0$ , and choose,  $\lambda_0$  belong  $H^{-1}(\Omega)$
2. loop on  $k = 0, \dots$ 
  - (a) set  $\mathcal{I}_k = \{x \in \Omega / \lambda_k + c * (u_{k+1} - g) \leq 0\}$
  - (b)  $V_{g,k+1} = \{v \in H_0^1(\Omega) / v = g \text{ on } \mathcal{I}_k\}$ ,
  - (c)  $V_{0,k+1} = \{v \in H_0^1(\Omega) / v = 0 \text{ on } \mathcal{I}_k\}$ ,
  - (d) Find  $u_{k+1} \in V_{g,k+1}$  and  $\lambda_{k+1} \in H^{-1}(\Omega)$  such that

$$\begin{cases} \int_{\Omega} \nabla u_{k+1} \cdot \nabla v_{k+1} d\omega = \int_{\Omega} f v_{k+1}, & \forall v_{k+1} \in V_{0,k+1} \\ \langle \lambda_{k+1}, v \rangle = \int_{\Omega} \nabla u_{k+1} \cdot \nabla v - f v d\omega \end{cases}$$

where  $\langle, \rangle$  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 [33].

Now how to do that in FreeFem++

The full example is:

#### Example 9.25 (VI.edp)

```

mesh Th=square(20,20);
real eps=1e-5;
fespace Vh(Th,P1);
int n = Vh.ndof;
Vh uh,uhp;
Vh Ik;
real[int] rhs(n);
real c=1000;

```

// P1 FE space  
 // number of Degree of freedom  
 // solution and previous one  
 // to def the set where the containt is reached.  
 // to store the right and side of the equation  
 // the penalty parameter of the algorith

```

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 variational 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,u=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

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

the Schwarz algorithm runs like this

$$\begin{aligned} -\Delta u_1^{n+1} &= f \text{ in } \Omega_1 & u_1^{n+1}|_{\Gamma_1} &= u_2^n \\ -\Delta u_2^{n+1} &= f \text{ in } \Omega_2 & u_2^{n+1}|_{\Gamma_2} &= u_1^n \end{aligned}$$

where  $\Gamma_i$  is the boundary of  $\Omega_i$  and on the condition that  $\Omega_1 \cap \Omega_2 \neq \emptyset$  and that  $u_i$  are zero at iteration 1.

Here we take  $\Omega_1$  to be a quadrangle,  $\Omega_2$  a disk and we apply the algorithm starting from zero.

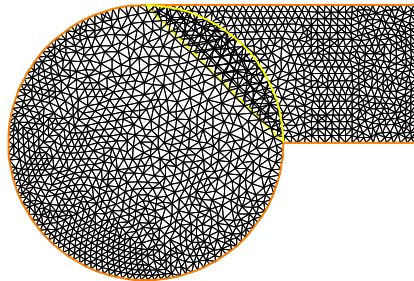


Figure 9.26: The 2 overlapping mesh TH and th

#### Example 9.26 (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);
};

```

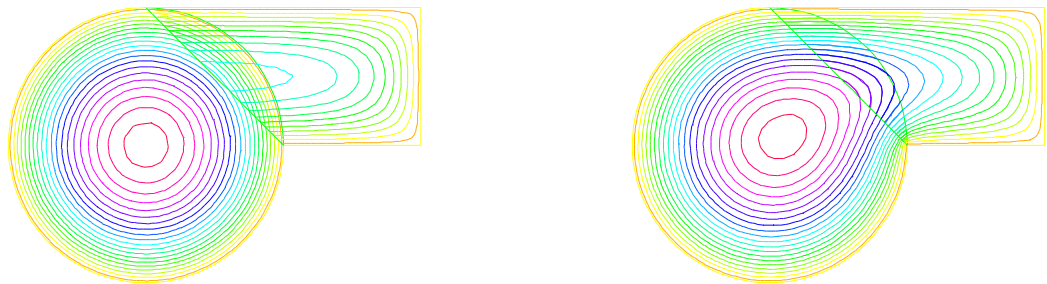


Figure 9.27: Isovalues of the solution at iteration 0 and iteration 9



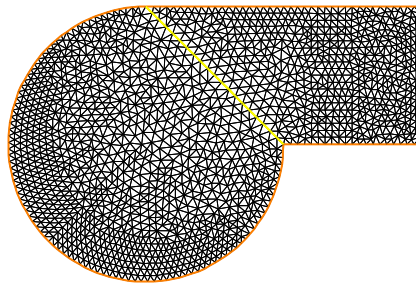


Figure 9.28: The two none overlapping mesh TH and th

### 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

Let introduce  $\Gamma_i$  is common the boundary of  $\Omega_1$  and  $\Omega_2$  and  $\Gamma_e^i = \partial\Omega_i \setminus \Gamma_i$ .

The problem find  $\lambda$  such that  $(u_1|_{\Gamma_i} = u_2|_{\Gamma_i})$  where  $u_i$  is solution of the following Laplace problem:

$$-\Delta u_i = f \text{ in } \Omega_i \quad u_i|_{\Gamma_i} = \lambda \quad u_i|_{\Gamma_e^i} = 0$$

To solve this problem we just make a loop with upgrading  $\lambda$  with

$$\lambda = \lambda \pm \frac{(u_1 - u_2)}{2}$$

where the sign + or - of  $\pm$  is choose to have convergence.

#### Example 9.27 (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");

```

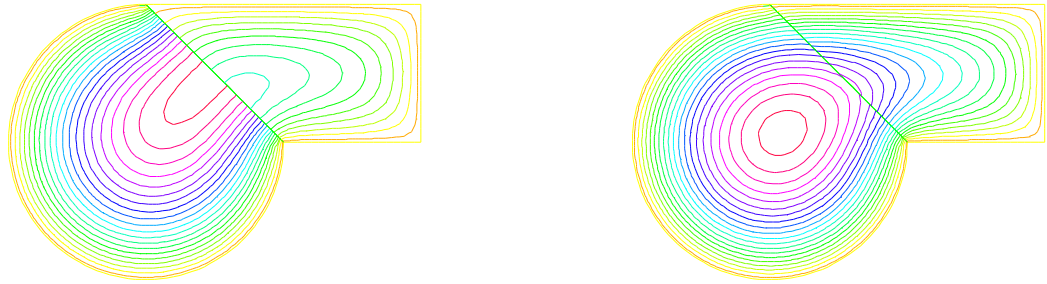


Figure 9.29: Isovalues of the solution at iteration 0 and iteration 9 without overlapping

### 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  $\Gamma_i$  is common the boundary of  $\Omega_1$  and  $\Omega_2$  and  $\Gamma_e^i = \partial\Omega_i \setminus \Gamma_i$ .

The problem find  $\lambda$  such that  $(u_1|_{\Gamma_i} = u_2|_{\Gamma_i})$  where  $u_i$  is solution of the following Laplace problem:

$$-\Delta u_i = f \text{ in } \Omega_i \quad u_i|_{\Gamma_i} = \lambda \quad u_i|_{\Gamma_e^i} = 0$$

The version of this example for Shur component. The border problem is solve with conjugate gradient. First, we construct the two domain

**Example 9.28 (Schwarz-gc.edp)**

```

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

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

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

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

//      build the mesh of  $\Omega_1$  and  $\Omega_2$ 
mesh Th1 = buildmesh( Gamma1(5*n) + Gamma2(5*n) + GammaInside(5*n) + Gamma3(5*n));
mesh Th2 = buildmesh ( GammaInside(-5*n) + GammaArc(25*n) );
plot(Th1,Th2);

//      defined the 2 FE space
fespace Vh1(Th1,P1),      Vh2(Th2,P1);

```

**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  $\Omega_1$  such as

```

Vh1 lambda=0; //      take  $\lambda \in V_{h1}$ 

```

The two Poisson problem:

```

Vh1 u1,v1;      Vh2 u2,v2;
int i=0; //      for factorization optimization
problem Pb2(u2,v2,init=i,solver=Cholesky) =
  int2d(Th2)( dx(u2)*dx(v2)+dy(u2)*dy(v2) )
+ int2d(Th2)( -v2)
+ int1d(Th2,inside)(-lambda*v2) + on(outside,u2= 0 ) ;
problem Pb1(u1,v1,init=i,solver=Cholesky) =
  int2d(Th1)( dx(u1)*dx(v1)+dy(u1)*dy(v1) )
+ int2d(Th1)( -v1)
+ int1d(Th1,inside)(+lambda*v1) + on(outside,u1 = 0 ) ;

```

or, we define a border matrix , because the  $\lambda$  function is none zero inside the domain  $\Omega_1$ :

```

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++;
v1=- (u1-u2);
lambda[]=B*v1[];
return lambda[] ;
};

```

*// no refactorization i !=0*

**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;
LinearCG(BoundaryProblem,p[],eps=1.e-6,nbiter=100);
BoundaryProblem(p[]);
cout << " -- CPU time  schwarz-gc:" <<  clock()-cpu << endl;
plot(u1,u2);

```

*// solve the problem with Conjugate Gradient*  
*// compute the final solution, because CG works with increment*  
*// solve again to have right 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  $\mathbf{u}$  and pressure  $p$ :

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

where  $\mathbf{u}_\Gamma$  is the velocity of the boundaries. The force that the fluid applies to the boundaries is the normal stress

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

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

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

where  $\delta$  is the Kronecker symbol and where  $\lambda, \mu$  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  $\mathbf{v}(x) \in V$  as

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

The data are the gravity force  $\mathbf{g}$  and the boundary stress  $\mathbf{h}$ .

**Example 9.29 (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  $10 \times 10$  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;}; //      righth beam
border d(t=0,10) { x=10-t; y=2; label=3;}; //      top beam
real E = 21.5;
real sigma = 0.29;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -0.05;
mesh th = buildmesh( b(20)+c(5)+d(20)+a(5));
fespace Vh(th,P1);
Vh uu,w,vv,s,fluidforce=0;
cout << "lambda,mu,gravity ="<<lambda<< " " << mu << " " << gravity << endl;
//      deformation of a beam under its own weight
solve bb([uu,vv],[w,s]) =
    int2d(th)(
        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 at low speed are solved in the box below the beam, but the beam has deformed the box (see border h):

```

//      Stokes on square b,e,f,g driven cavity 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:*

```

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

```

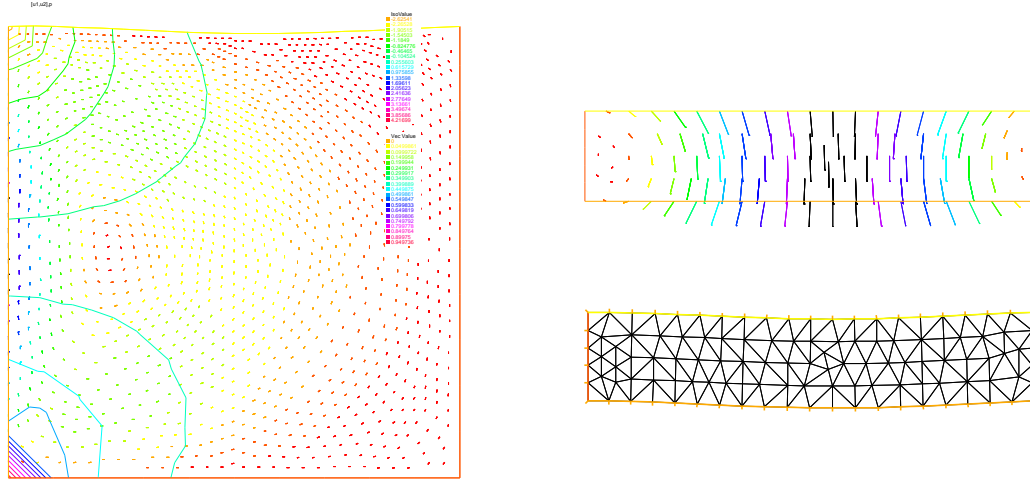


Figure 9.30: Fluid velocity and pressure (left) and displacement vector (center) of the structure and displaced geometry (right) in the fluid-structure interaction of a soft side and a driven cavity

```

+ 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

```

## 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  $\Omega_i$ ,  $i = 1, 2, 3$  be the domain occupied by  $i$ -th material with tension  $\mu_i$  (see Section 9.1.1). The computational domain  $\Omega$  is the interior of  $\overline{\Omega_1} \cup \overline{\Omega_2} \cup \overline{\Omega_3}$ . The vertical displacement  $u(x, y)$  is obtained from

$$-\mu_i \Delta u = f \text{ in } \Omega_i \quad (9.55)$$

$$\mu_i \partial_n u|_{\Gamma_i} = -\mu_j \partial_n u|_{\Gamma_j} \text{ on } \overline{\Omega_i} \cap \overline{\Omega_j} \quad \text{if } 1 \leq i < j \leq 3 \quad (9.56)$$

where  $\partial_n u|_{\Gamma_i}$  denotes the value of the normal derivative  $\partial_n u$  on the boundary  $\Gamma_i$  of the domain  $\Omega_i$ . By introducing the characteristic function  $\chi_i$  of  $\Omega_i$ , that is,

$$\chi_i(x) = 1 \quad \text{if } x \in \Omega_i; \quad \chi_i(x) = 0 \quad \text{if } x \notin \Omega_i \quad (9.57)$$

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

$$a(u, v) = \ell(f, v) \quad \text{for all } v \in H_0^1(\Omega) \quad (9.58)$$

$$a(u, v) = \int_{\Omega} \mu \nabla u \cdot \nabla v, \quad \ell(f, v) = \int_{\Omega} f v$$

where  $\mu = \mu_1 \chi_1 + \mu_2 \chi_2 + \mu_3 \chi_3$ . Here we notice that  $\mu$  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);

```

`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 ( $\nu$  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);

```



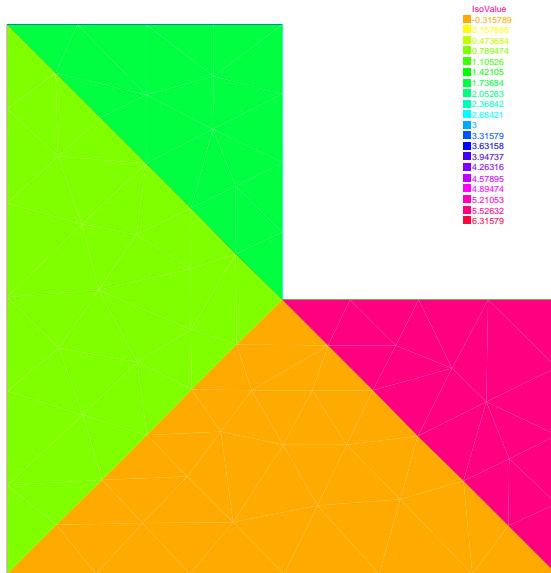


Figure 9.31: the function reg

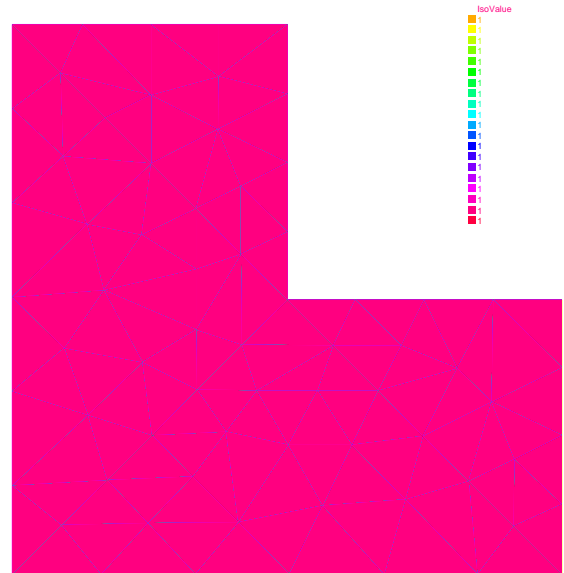


Figure 9.32: the function nu

## 9.11 Free Boundary Problem

The domain  $\Omega$  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

border a(t=0,L){x=t;y=0;};                // maillage d'un tapeze
border b(t=0,h1){x=L;y=t;};               // bottom:  $\Gamma_a$ 
border f(t=L,0){x=t;y=t*(h1-h)/L+h;};    // right:  $\Gamma_b$ 
border d(t=h,0){x=0;y=t;};               // free surface:  $\Gamma_f$ 
                                           // left:  $\Gamma_d$ 

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

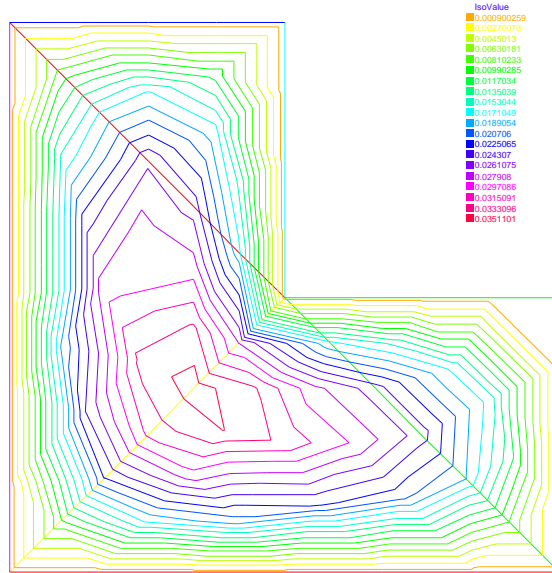
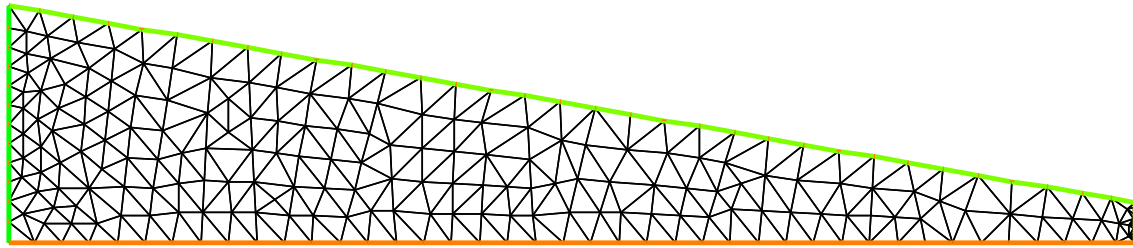
```

The free boundary problem is:

Find  $u$  and  $\Omega$  such that:

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

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

Figure 9.33: the isovalue of the solution  $u$ Figure 9.34: The mesh of the domain  $\Omega$ 

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

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

The variational formulation is:

find  $u$  on  $V = H^1(\Omega^n)$ , such that  $u = y$  on  $\Gamma_b^n$  and  $\Gamma_f^n$

$$\int_{\Omega^n} \nabla u \nabla u' = 0, \quad \forall u' \in V \text{ with } u' = 0 \text{ on } \Gamma_b^n \cup \Gamma_f^n$$

and secondly to construct a domain deformation  $\mathcal{F}(x, y) = [x, y - v(x, y)]$

where  $v$  is solution of the following problem:

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

The variational formulation is:

find  $v$  on  $V$ , such that  $v = 0$  on  $\Gamma_a^n$

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

Finally the new domain  $\Omega^{n+1} = \mathcal{F}(\Omega^n)$

**Example 9.30 (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) {
    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;
  }
}

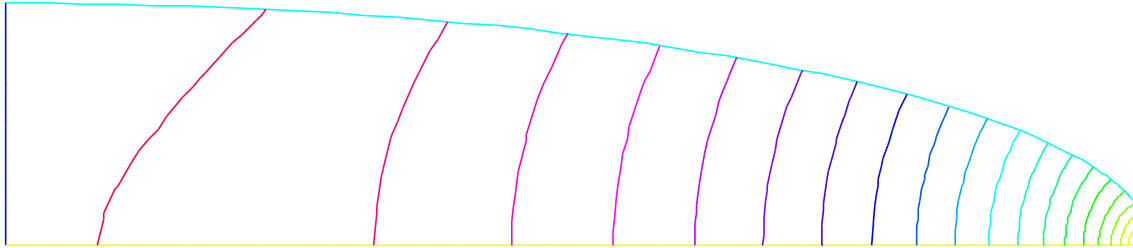
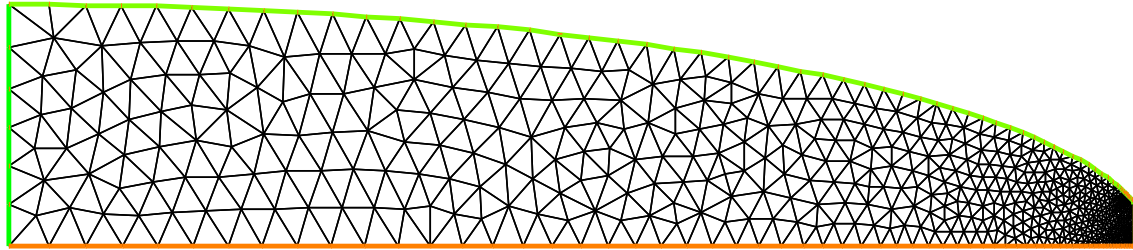
```

```

Th=movemesh(Th,[x,y-coef*v]);
cout << "\n\n"<<j <<"----- errv = " << errv << "\n\n";

}
plot(Th,ps="d_Thf.eps");
plot(u,wait=1,ps="d_u.eps");

```

Figure 9.35: The final solution on the new domain  $\Omega^{72}$ Figure 9.36: The adapted mesh of the domain  $\Omega^{72}$ 

## 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

$$DJ(\mathbf{u})(\mathbf{v}) = \int DF2(\mathbf{u})(\mathbf{v}) f'(F2(\mathbf{u})) - \int_{\Gamma_p} P_a v_2$$

where  $DF2(\mathbf{u})(\mathbf{v}) = 2 A( DE[\mathbf{u}](\mathbf{v}), E[\mathbf{u}] )$  and  $DE$  is the first differential of  $E$ .

The second order differential is

$$D^2 J(\mathbf{u})(\mathbf{v}, \mathbf{w}) = \int DF2(\mathbf{u})(\mathbf{v}) DF2(\mathbf{u})(\mathbf{w}) f''(F2(\mathbf{u})) \\ + \int D^2 F2(\mathbf{u})(\mathbf{v}, \mathbf{w}) f'(F2(\mathbf{u}))$$

where

$$D^2 F2(\mathbf{u})(\mathbf{v}, \mathbf{w}) = 2 A( D^2 E[\mathbf{u}](\mathbf{v}, \mathbf{w}), E[\mathbf{u}] ) + 2 A( DE[\mathbf{u}](\mathbf{v}), DE[\mathbf{u}](\mathbf{w}) ).$$

and  $D^2 E$  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  $u_0, v_0$  the initial displacement

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

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

- $u_n = u_n - du, \quad v_n = v_n - dv$
- 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 in cpp preprocessor of C++ , but this begin by `macro` and the end of the macro definition is before the comment `//`. In this case the macro is very useful because the type of parameter can be change. And it is easy to make automatic differentiation.

// non linear elasticity model

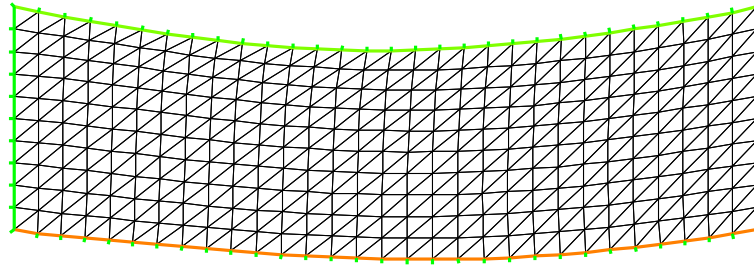


Figure 9.37: The deformed domain

```

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

macro f(u) (u)
macro df(u) (1)
macro ddf(u) (0)

//      end of macro
//      end of macro
//      end of macro

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

real mu = 0.012e5;
real lambda = 0.4e5;
//      kg/cm2
//      kg/cm2

//
//       $\sigma = 2\mu E + \lambda \text{tr}(E)Id$ 
//       $A(u, v) = \sigma(u) : E(v)$ 
//
//      ( a b )
//      ( b c )
//
//      tr*Id : (a,b,c) -> (a+c,0,a+c)
//      so the associated matrix is:
//      ( 1 0 1 )
//      ( 0 0 0 )
//      ( 1 0 1 )

//      -----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)( // (D2J(un)) part
    dF2(un,vn,uu,vv)*dF2(un,vn,w,s)*ddfe2
    + dF2(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] = (D2J(un))-1(DJ(un))

  w[] = M*uu[];
  real res = sqrt(w[]' * uu[]); // norme L2 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=" displacement " );
    if (res<1e-5) break;
}

plot([un,vn],wait=1);
mesh th1 = movemesh(Th, [x+un, y+vn]);
plot(th1,wait=1); // see figure 9.37

```

## 9.13 Compressible Neo-Hookean Materials: Computational Solutions

Author : Alex Sadovsky mailsashas@gmail.com

### 9.13.1 Notation

In what follows, the symbols  $\mathbf{u}, \mathbf{F}, \mathbf{B}, \mathbf{C}, \underline{\sigma}$  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  $I_1 := \text{tr } \mathbf{C}$  and  $J := \det \mathbf{F}$ . Use will be made of the identity

$$\frac{\partial J}{\partial \mathbf{C}} = J\mathbf{C}^{-1} \quad (9.59)$$

The symbol  $\mathbf{I}$  denotes the identity tensor. The symbol  $\Omega_0$  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  $\Omega$  involving the Cauchy stress  $\mathbf{T}$  to be rewritten as an integral of the Kirchhoff stress  $\kappa = J\mathbf{T}$  over  $\Omega_0$ .

### Recommended References

For an exposition of nonlinear elasticity and of the underlying linear- and tensor algebra, see [34]. For an advanced mathematical analysis of the Finite Element Method, see [35]. An explanation of the Finite Element formulation of a nonlinear elastostatic boundary value problem, see <http://www.engin.brown.edu/courses/en222/Notes/FEMfinitetrain/FEMfinitetrain.htm>.

### 9.13.2 A Neo-Hookean Compressible Material

**Constitutive Theory and Tangent Stress Measures** The strain energy density function is given by

$$W = \frac{\mu}{2}(I_1 - \text{tr } \mathbf{I} - 2 \ln J) \quad (9.60)$$

(see [32], formula (12)).

The corresponding 2nd Piola-Kirchhoff stress tensor is given by

$$\mathbf{S}_n := \frac{\partial W}{\partial \mathbf{E}}(\mathbf{F}_n) = \mu(\mathbf{I} - \mathbf{C}^{-1}) \quad (9.61)$$

The Kirchhoff stress, then, is

$$\kappa = \mathbf{F}\mathbf{S}\mathbf{F}^T = \mu(\mathbf{B} - \mathbf{I}) \quad (9.62)$$



The tangent Kirchhoff stress tensor at  $\mathbf{F}_n$  acting on  $\delta\mathbf{F}_{n+1}$  is, consequently,

$$\frac{\partial\kappa}{\partial\mathbf{F}}(\mathbf{F}_n)\delta\mathbf{F}_{n+1} = \mu \left[ \mathbf{F}_n(\delta\mathbf{F}_{n+1})^T + \delta\mathbf{F}_{n+1}(\mathbf{F}_n)^T \right] \quad (9.63)$$

**The Weak Form of the BVP in the Absence of Body (External) Forces** The  $\Omega_0$  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

$$0 = \int_{\Omega_0} \kappa[\mathbf{F}] : \left\{ (\nabla \otimes \mathbf{w})(\mathbf{F})^{-1} \right\} - \int_{\partial\Omega_0^t} P \cdot \hat{\mathbf{N}}_0$$

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  $\mathbf{u}_n$  of the displacement field  $\mathbf{u}$  satisfying the BVP, we seek the correction  $\delta\mathbf{u}_{n+1}$  to obtain a better approximation

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \delta\mathbf{u}_{n+1}$$

by solving the weak formulation

$$\left. \begin{aligned} 0 &= \int_{\Omega_0} \kappa[\mathbf{F}_n + \delta\mathbf{F}_{n+1}] : \left\{ (\nabla \otimes \mathbf{w})(\mathbf{F}_n + \delta\mathbf{F}_{n+1})^{-1} \right\} - \int_{\partial\Omega_0} P \cdot \hat{\mathbf{N}}_0 \\ &= \int_{\Omega_0} \left\{ \kappa[\mathbf{F}_n] + \frac{\partial\kappa}{\partial\mathbf{F}}[\mathbf{F}_n]\delta\mathbf{F}_{n+1} \right\} : \left\{ (\nabla \otimes \mathbf{w})(\mathbf{F}_n + \delta\mathbf{F}_{n+1})^{-1} \right\} \\ &= \int_{\Omega_0} \left\{ \kappa[\mathbf{F}_n] + \frac{\partial\kappa}{\partial\mathbf{F}}[\mathbf{F}_n]\delta\mathbf{F}_{n+1} \right\} : \left\{ (\nabla \otimes \mathbf{w})(\mathbf{F}_n^{-1} + \mathbf{F}_n^{-2}\delta\mathbf{F}_{n+1}) \right\} \\ &= \int_{\Omega_0} \kappa[\mathbf{F}_n] : \left\{ (\nabla \otimes \mathbf{w})\mathbf{F}_n^{-1} \right\} \\ &\quad - \int_{\Omega_0} \kappa[\mathbf{F}_n] : \left\{ (\nabla \otimes \mathbf{w})(\mathbf{F}_n^{-2}\delta\mathbf{F}_{n+1}) \right\} \\ &\quad + \int_{\Omega_0} \left\{ \frac{\partial\kappa}{\partial\mathbf{F}}[\mathbf{F}_n]\delta\mathbf{F}_{n+1} \right\} : \left\{ (\nabla \otimes \mathbf{w})\mathbf{F}_n^{-1} \right\} \end{aligned} \right\} \quad \text{for all test functions } \mathbf{w}, \quad (9.64)$$

where we have taken

$$\delta\mathbf{F}_{n+1} = \nabla \otimes \delta\mathbf{u}_{n+1}$$

**Note:** Contrary to standard notational use, the symbol  $\delta$  here bears no variational context. By  $\delta$  we mean simply an increment in the sense of Newton's Method. The role of a variational virtual displacement here is played by  $\mathbf{w}$ .

### 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

$$\langle TanK \rangle := \frac{\partial\kappa}{\partial\mathbf{F}}[\mathbf{F}_n]\delta\mathbf{F}_{n+1} \quad (9.65)$$

will be implemented as the macros `< TanK11 >`, `< TanK12 >`, ...

The individual components of the tensor quantities

$$\mathbf{D}_1 := \mathbf{F}_n(\delta\mathbf{F}_{n+1})^T + \delta\mathbf{F}_{n+1}(\mathbf{F}_n)^T,$$

$$\mathbf{D}_2 := \mathbf{F}_n^{-T}\delta\mathbf{F}_{n+1},$$

$$\mathbf{D}_3 := (\nabla \otimes \mathbf{w}) \mathbf{F}_n^{-2} \delta \mathbf{F}_{n+1},$$

and

$$\mathbf{D}_4 := (\nabla \otimes \mathbf{w}) \mathbf{F}_n^{-1},$$

will be implemented as the macros

$$\left. \begin{aligned} &< d1Aux11 >, < d1Aux12 >, \quad \dots \quad, < d1Aux22 >, \\ &< d2Aux11 >, < d2Aux12 >, \quad \dots \quad, < d2Aux22 > \\ &< d3Aux11 >, < d3Aux12 >, \quad \dots \quad, < d3Aux22 > \\ &< d4Aux11 >, < d4Aux12 >, \quad \dots \quad, < d4Aux22 > \end{aligned} \right\}, \quad (9.66)$$

respectively.

In the above notation, the tangent Kirchhoff stress term becomes

$$\frac{\partial \kappa}{\partial \mathbf{F}}(\mathbf{F}_n) \delta \mathbf{F}_{n+1} = \mu \mathbf{D}_1 \quad (9.67)$$

while the weak BVP formulation acquires the form

$$\left. \begin{aligned} 0 &= \int_{\Omega_0} \kappa[\mathbf{F}_n] : \mathbf{D}_4 \\ &- \int_{\Omega_0} \kappa[\mathbf{F}_n] : \mathbf{D}_3 \\ &+ \int_{\Omega_0} \left\{ \frac{\partial \kappa}{\partial \mathbf{F}}[\mathbf{F}_n] \delta \mathbf{F}_{n+1} \right\} : \mathbf{D}_4 \end{aligned} \right\} \quad \text{for all test functions } \mathbf{w} \quad (9.68)$$

# Chapter 10

## MPI Parallel version

A first attempt of parallelization of FreeFem++ is made here with **mpi**. We add big interface with MPI in version 3.5, (see MPI doc for the definition functionality at <http://www.mpi-forum.org/docs/mpi21-report.pdf>) in the language:

### 10.1 MPI keywords

”

**mpiGroup** to defined a group of processor in communication world

**mpiCom** to defined a communication world

**mpiRequest** to defined a request of wait the end of the communication

### 10.2 MPI constants

**mpisize** The total number of processes,

**mpirank** the number of my current process in  $\{0, \dots, mpisize - 1\}$ ,

**mpirank** the number of my current process in  $\{0, \dots, mpisize - 1\}$ ,

**mpiUndefined** The MPI\_Undefined constant,

**mpiAnySource** The MPI\_ANY\_SOURCE constant,

**mpiCommWorld** The MPI\_COMM\_WORLD constant ,

... All the basic MPI.Op for reduce operator: **mpiMAX**, **mpiMIN**, **mpiSUM**, **mpiPROD**, **mpiLAND**, **mpiLOR**, **mpiLXOR**, **mpiBA**



where a processor is just a integer rank, pointer to a MPI\_comm and pointer to a MPI\_Request, and processorblock with a special MPI\_Request.

## 10.5 MPI communicator operator

```

int status;                                // to get the MPI status of send / recv
processor(10) << a << b;                   // send a,b asynchronously to the process 1,
processor(10) >> a >> b;                   // receive a,b synchronously from the process 10,
broadcast(processor(10,comm),a);           // broadcast from processor
                                           // of com to other comm processor

status=Send( processor(10,comm) , a);       // send synchronously
                                           // to the process 10 the data a
status=Recv( processor(10,comm) , a);       // receive synchronously
                                           // from the process 10 the data a;
status=Isend( processor(10,comm) , a);      // send asynchronously to
                                           // the process 10 , the data a without request
status=Isend( processor(10,rq,comm) , a );  // send asynchronously to to
                                           // the process 10, the data a with request
status=Irecv( processor(10,rq) , a );       // receive synchronously from
                                           // the process 10, the data a;
status=Irecv( processor(10) , a );          // Error
                                           // Error asynchronously without request .

broadcast(processor(comm,a));              // Broadcast to all process of comm

```

where the data type of a can be of type of int,real, complex, int[int], double[int], complex[int], int[int,int], double[int,int],complex[int,int] and for communication with no request the type can be also mesh,mesh3,mesh[int],mesh3[int],matrix,matrix<complex because in this case the communication are multiple (header + data).

```

processor(10,rq) << a ;                    // send asynchronously to the process 10
                                           // the data a with request
processor(10,rq) >> a ;                    // receive asynchronously from the process 10
                                           // the data a with request

```

If a,b are arrays or full matrices of int, real, or complex, we can use the following MPI functions:

```

mpiAlltoall(a,b[,comm]) ;
mpiAllgather(a,b[,comm]) ;
mpiGather(a,b,processor(..) ) ;
mpiScatter(a,b,processor(..)) ;
mpiReduce(a,b,processor(..),mpiMax) ;
mpiAllReduce(a,b,comm,mpiMax) ;
mpiReduceScatter(a,b,comm,mpiMax) ;

```

See the examples++-mpi/essai.edp to test of all this functionality and Thank, to Guy Antoine Atenekeng Kahou, for the help of coding this interface.

## 10.6 Schwarz example in parallel

This example is just the rewriting 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

## Parallel sparse solvers

Parallel sparse solvers is solver which use several computer processors to solve linear systems of equation. Like sequential, parallel linear solver can be direct or iteratif. In freefem++ we have two types of parallels solvers.

### 11.1 Using parallel sparse solvers in Freefem++

We recall that the solver parameters are defined in the following command line: `solve`, `problem`, `set` (setting parameter of a matrix) and in the construction of the matrix corresponding to a bilinear form. In these command, the parameter `solver` must be set to `sparsesolver` for parallel sparse solver. We have added specify parameters for these command lines for parallel sparse solvers. These parameters are

- `lparams`: vector of integer parameters (l is for the c++ type long)
- `dparams`: vector of real parameters
- `sparams`: string parameters
- `datafilename`: name of the file which contains solver parameters

The following four parameters are only for direct solver and are vector. These parameters allow to give user's preprocessing matrix (see section sparse direct solver for more information).

- `permr`: row permutation (integer vector)
- `permc`: column permutation or inverse row permutation (integer vector)
- `scaler`: row scaling (real vector)
- `scalec`: column scaling (real vector)

There is two possibilities to control solver parameters. The first method is defined parameters with `lparams`, `dparams` and `sparams` in `.edp` file. In the second one, the solver parameters are read in a data file which contains these parameters. The name of this file will be specified in `datafilename`. If `lparams`, `dparams`, `sparams` and `datafilename` is not provided by the user, we use the solver's default options.

To use parallel solver in FreeFem++, we need to load the dynamic library corresponding to this solver. For example to use MUMPS solver as parallel solver in FreeFem, write in the `.edp` file `load "MUMPS.FreeFem"`. If the librairies are not loaded, the default sparse solver will be loaded (default sparse solver is umfpack). The table 11.1 gives this new value for the different libraries.

Libraries	default sparse solver	
	real	complex
MUMPS_FreeFem	mumps	mumps
real_SuperLU_DIST_FreeFem	SuperLU_DIST	previous solver
complex_SuperLU_DIST_FreeFem	previous solver	SuperLU_DIST
real_pastix_FreeFem	pastix	previous solver
complex_pastix_FreeFem	previous solver	pastix
hips_FreeFem	hips	previous solver
hypre_FreeFem	hypre	previous solver
parms_FreeFem	parms	previous solver

Table 11.1: Default sparse solver for real and complex arithmetics when we load a parallel sparse solver library

We also add functions (see Table 11.2) with no parameter to change the default sparse solver in the .edp file. To use these functions, we need to load the library corresponding to the solver. An example of using different parallel sparse solvers for the same problem is given in testdirectsolvers.edp (directory example+ + -mpi).

function	default sparse solver	
	real	complex
defaulttoMUMPS()	mumps	mumps
realdefaulttoSuperLUdist()	SuperLU_DIST	previous solver
complexdefaulttoSuperLUdist()	previous solver	SuperLU_DIST
realdefaulttopastix()	pastix	previous solver
complexdefaulttopastix()	previous solver	pastix
defaulttohips()	hips	previous solver
defaulttohypre()	hypre	previous solver
defaulttoparms()	parms	previous solver

Table 11.2: Functions that allow to change default sparse solver for real and complex arithmetics and the result of these functions

### Example 11.1 (testdirectsolvers.edp)

```

load "../src/solver/MUMPS_FreeFem"
                                // default solver : real-> MUMPS, complex -> MUMPS
load "../src/solver/real_SuperLU_DIST_FreeFem"
                                // default solver : real-> SuperLU_DIST, complex -> MUMPS
load "../src/solver/real_pastix_FreeFem"
                                // default solver : real-> pastix, complex -> MUMPS
                                // solving with pastix
{

  matrix A =
    [[ 1,  2,      2,  1, 1],
     [ 2,   12,    0, 10, 10],
     [ 2,    0,    1,  0, 2],
     [ 1,   10,    0, 22, 0],
     [ 1,   10,    2,  0, 22]];

```

```

real[int] xx = [ 1,32,45,7,2], x(5), b(5), di(5);
b=A*xx;
cout << "b=" << b << endl;
cout << "xx=" << xx << endl;

set(A,solver=sparsesolver,datafilename="ffpastix_iparm_dparm.txt");
cout << "solving solution" << endl;
x = A^-1*b;
cout << "b=" << b << endl;
cout << "x=" << endl; cout << x << endl;
di = xx-x;
if(mpirank==0){
cout << "x-xx="<< endl; cout << "Linf "<< di.linf << " L2 " << di.l2 << endl;
}
}

// solving with SuperLU_DIST
realdefaulttoSuperLUDist();
// default solver : real-> SuperLU_DIST, complex -> MUMPS

{

matrix A =
[[ 1, 2, 2, 1, 1],
[ 2, 12, 0, 10, 10],
[ 2, 0, 1, 0, 2],
[ 1, 10, 0, 22, 0.],
[ 1, 10, 2, 0., 22]];

real[int] xx = [ 1,32,45,7,2], x(5), b(5), di(5);
b=A*xx;
cout << "b=" << b << endl;
cout << "xx=" << xx << endl;

set(A,solver=sparsesolver,datafilename="ffsuperlu_dist_fileparam.txt");
cout << "solving solution" << endl;
x = A^-1*b;
cout << "b=" << b << endl;
cout << "x=" << endl; cout << x << endl;
di = xx-x;
if(mpirank==0){
cout << "x-xx="<< endl; cout << "Linf "<< di.linf << " L2 " << di.l2 << endl;
}
}

// solving with MUMPS
defaulttoMUMPS();
// default solver : real-> MUMPS, complex -> MUMPS

{

```

```

matrix A =
  [[ 1,  2,      2,  1, 1],
   [ 2,      12,      0, 10, 10],
   [ 2,      0,      1,  0, 2],
   [ 1,      10,      0, 22, 0.],
   [ 1,      10,      2,  0., 22]];

real[int] xx = [ 1,32,45,7,2], x(5), b(5), di(5);
b=A*xx;
cout << "b=" << b << endl;
cout << "xx=" << xx << endl;

set(A,solver=sparse solver,datafilename="ffmumps_fileparam.txt");
cout << "solving solution" << endl;
x = A^-1*b;
cout << "b=" << b << endl;
cout << "x=" << endl; cout << x << endl;
di = xx-x;
if(mpirank==0){
  cout << "x-xx=" << endl; cout << "Linf " << di.linf << " L2 " << di.l2 << endl;
}
}

```

## 11.2 Sparse direct solver

In this section, we present different sparse direct solvers interfacing with *FreeFem++*.

### 11.2.1 MUMPS solver

Multifrontal Massively Parallel Solver (MUMPS) is a free library [?, ?, ?]. This package solves linear system of the form  $A x = b$  where  $A$  is a square sparse matrix with a direct method. The square matrix considered in MUMPS can be either unsymmetric, symmetric positive definite or general symmetric. The method implemented in MUMPS is a direct method based on a multifrontal approach [?]. It constructed a direct factorization  $A = L U$ ,  $A = L^t D L$  depending of the symmetry of the matrix  $A$ . MUMPS is written in Fortran 90. The parallel version is constructed using MPI [?] for message passing and use the following libraries : BLAS[?, ?], BLACS and ScaLAPACK[?].

**Remark:** MUMPS doesn't solve linear system with a rectangular matrix.

**Installation of MUMPS** To used MUMPS in *FreeFem++* , you have to install MUMPS package in first. MUMPS is written in Fortran 90. The parallel version is constructing using MPI [?] for message passing and used the following libraries : BLAS [?, ?], BLACS and ScaLAPACK[?]. Therefore, we need a fortran compiler, MPI, BLAS, BLACS and ScaLAPACK to do this compilation. An installation procedure to obtain this package is given in the file README\_COMPILE in the directory src/solver of *FreeFem++* .

**Creating Library of MUMPS interface for FreeFem++ :** The MUMPS interface for FreeFem++ is given in file MUMPS\_freefem.cpp (directory src/solver/ ). This interface work with the release 3.8.3 and 3.8.4 of MUMPS. To used MUMPS in FreeFem++ , we need the library corresponding to this interface. A description to obtain this library is given in the file README\_COMPILE in the directory src/solver of FreeFem++ . We recall here the procedure. Go to the directory src/solver in FreeFem++ package. Edit the file makefile-sparsesolver.inc to yours system: comment Section 1, comment line corresponding to libraries BLAS, BLACS, ScaLAPACK, Metis, scotch in Section 2 and comment in Section 3 the paragraph corresponding to MUMPS solver. And just type **make mumps** in the terminal to obtain the dynamic library of interface.

Now we give a short description of MUMPS parameters before describing the method to call MUMPS in FreeFem++ .

**MUMPS parameters:** There are four input parameters in MUMPS (see [?]). Two integers SYM and PAR, a vector of integer of size 40 INCTL and a vector of real of size 15 CNTL. The first parameter gives the type of the matrix: 0 for unsymmetric matrix, 1 for symmetric positive matrix and 2 for general symmetric. The second parameter defined if the host processor work during the factorization and solves steps : PAR=1 host processor working and PAR=0 host processor not working. The parameter INCTL and CNTL is the control parameter of MUMPS. The vectors ICNTL and CNTL in MUMPS becomes with index 1 like vector in fortran. A short description of all parameters of ICNTL and CNTL is given in ffmumps\_fileparam.txt. For more details see the users' guide [?]. We describe now some elements of the main parameters of ICNTL for MUMPS.

**input matrix parameter** The input matrix is controlled by parameters ICNTL(5) and ICNTL(18). The matrix format (resp. matrix pattern and matrix entries) are controlled by INCTL(5) (resp. INCTL(18)). The different values of ICNTL(5) are 0 for assembled format and 1 for element format. In the current release of FreeFem++, we consider that FE matrix or matrix is storage in assembled format. Therefore, INCTL(5) is treated as 0 value. The main option for ICNTL(18): INCTL(18)=0 centrally on the host processor, ICNTL(18)=3 distributed the input matrix pattern and the entries (recommended option for distributed matrix by developer of MUMPS). For other values of ICNTL(18) see the user's guide of MUMPS. These values can be used also in FreeFem++. The default option implemented in FreeFem++ are ICNTL(5)=0 and ICNTL(18)=0.

**preprocessing parameter** The preprocessed matrix  $A_p$  that will be effectively factored is defined by

$$A_p = P D_r A Q_c D_c P^t$$

where  $P$  is the permutation matrix,  $Q_c$  is the column permutation,  $D_r$  and  $D_c$  are diagonal matrix for respectively row and column scaling. The ordering strategy to obtain  $P$  is controlled by parameter ICNTL(7). The permutation of zero free diagonal  $Q_c$  is controlled by parameter ICNTL(6). The row and column scaling is controlled by parameter ICNTL(18). These option are connected and also strongly related with ICNTL(12) (see documentation of mumps for more details [?]). The parameters permr, scaler, and scalec in FreeFem++ allow to give permutation matrix( $P$ ), row scaling ( $D_r$ ) and column scaling ( $D_c$ ) of the user respectively.

**Calling MUMPS in FreeFem++** To calling MUMPS in FreeFem++ , we need to load the dynamic library MUMPS\_freefem.dylib (MacOSX), MUMPS\_freefem.so (Unix) or MUMPS\_freefem.dll (Windows). This done in typing load "MUMPS\_freefem" in the .edp file. We give now the two methods to give the option of MUMPS solver in FreeFem++ .

**Solver parameters is defined in .edp file:** In this method, we need to give the parameters lparams and dparams. These parameters are defined for MUMPS by

$$\begin{aligned} & \text{lparams}[0] = \text{SYM}, \\ & \text{lparams}[1] = \text{PAR}, \\ & \forall i = 1, \dots, 40, \quad \text{lparams}[i+1] = \text{ICNTL}(i). \\ & \forall i = 1, \dots, 15, \quad \text{dparams}[i-1] = \text{CNTL}(i). \end{aligned}$$

**Reading solver parameters on a file:** The structure of data file for MUMPS in FreeFem++ is : first line parameter SYM and second line parameter PAR and in the following line the different value of vectors ICNTL and CNTL. An example of this parameter file is given in `ffmumpsfileparam.txt`.

```

0          /* SYM :: 0 for unsymmetric matrix, 1 for symmetric definite positive matrix and
2 general symmetric matrix*/
1          /* PAR :: 0 host not working during factorization and solves steps, 1 host working
during factorization and solves steps*/
-1         /* ICNTL(1) :: output stream for error message */
-1         /* ICNTL(2) :: output for diagnostic printing, statics and warning message */
-1         /* ICNTL(3) :: for global information */
0          /* ICNTL(4) :: Level of printing for error, warning and diagnostic message */
0          /* ICNTL(5) :: matrix format : 0 assembled format, 1 elemental format. */
7          /* ICNTL(6) :: control option for permuting and/or scaling the matrix in analysis
phase */
3          /* ICNTL(7) :: pivot order strategy : AMD, AMF, metis, pord scotch*/
77         /* ICNTL(8) :: Row and Column scaling strategy */
1          /* ICNTL(9) :: 0 solve Ax = b, 1 solve the transposed system A^t x = b : parameter
is not considered in the current release of freefem++*/
0          /* ICNTL(10) :: number of steps of iterative refinement */
0          /* ICNTL(11) :: statics related to linear system depending on ICNTL(9) */
1          /* ICNTL(12) :: constrained ordering strategy for general symmetric matrix */
0          /* ICNTL(13) :: method to control splitting of the root frontal matrix */
20         /* ICNTL(14) :: percentage increase in the estimated working space (default 20%)*
0          /* ICNTL(15) :: not used in this release of MUMPS */
0          /* ICNTL(16) :: not used in this release of MUMPS */
0          /* ICNTL(17) :: not used in this release of MUMPS */
3          /* ICNTL(18) :: method for given : matrix pattern and matrix entries : */
0          /* ICNTL(19) :: method to return schur complement matrix */
0          /* ICNTL(20) :: right hand side form ( 0 dense form, 1 sparse form) : parameter
will be set to 0 for freefem++ */
0          /* ICNTL(21) :: 0, 1 kept distributed solution : parameter is not considered in
the current release of freefem++ */
0          /* ICNTL(22) :: controls the in-core/out-of-core (OOC) facility */
0          /* ICNTL(23) :: maximum size of the working memory in Megabyte than MUMPS can allocate
per working processor */
0          /* ICNTL(24) :: control the detection of null pivot */
0          /* ICNTL(25) :: control the computation of a null space basis */
0          /* ICNTL(26) :: This parameter is only significant with Schur option (ICNTL(19)
not zero). : parameter is not considered in the current release of freefem++ */
-8         /* ICNTL(27) (Experimental parameter subject to change in next release of MUMPS)
:: control the blocking factor for multiple righthand side during the solution phase : parameter
is not considered in the current release of freefem++ */
0          /* ICNTL(28) :: not used in this release of MUMPS*/
0          /* ICNTL(29) :: not used in this release of MUMPS*/
0          /* ICNTL(30) :: not used in this release of MUMPS*/
0          /* ICNTL(31) :: not used in this release of MUMPS*/

```

```

0      /* ICNTL(32) :: not used in this release of MUMPS*/
0      /* ICNTL(33) :: not used in this release of MUMPS*/
0      /* ICNTL(34) :: not used in this release of MUMPS*/
0      /* ICNTL(35) :: not used in this release of MUMPS*/
0      /* ICNTL(36) :: not used in this release of MUMPS*/
0      /* ICNTL(37) :: not used in this release of MUMPS*/
0      /* ICNTL(38) :: not used in this release of MUMPS*/
1      /* ICNTL(39) :: not used in this release of MUMPS*/
0      /* ICNTL(40) :: not used in this release of MUMPS*/
0.01   /* CNTL(1) :: relative threshold for numerical pivoting */
1e-8   /* CNTL(2) :: stopping criteria for iterative refinement */
-1     /* CNTL(3) :: threshold for null pivot detection */
-1     /* CNTL(4) :: determine the threshold for partial pivoting */
0.0    /* CNTL(5) :: fixation for null pivots */
0      /* CNTL(6) :: not used in this release of MUMPS */
0      /* CNTL(7) :: not used in this release of MUMPS */
0      /* CNTL(8) :: not used in this release of MUMPS */
0      /* CNTL(9) :: not used in this release of MUMPS */
0      /* CNTL(10) :: not used in this release of MUMPS */
0      /* CNTL(11) :: not used in this release of MUMPS */
0      /* CNTL(12) :: not used in this release of MUMPS */
0      /* CNTL(13) :: not used in this release of MUMPS */
0      /* CNTL(14) :: not used in this release of MUMPS */
0      /* CNTL(15) :: not used in this release of MUMPS */

```

If no solver parameters is given, we used default option of MUMPS solver.

**example** A simple example of calling MUMPS in FreeFem++ with this two methods is given in the file `testsolver_MUMPS.edp` in the directory `examples++-mpi`.

## 11.2.2 SuperLU distributed solver

The package SuperLU\_DIST [?, ?] solve linear system using LU factorization. It is a free scientific library under BSD license. The web site of this project is <http://crd.lbl.gov/~xiaoye/SuperLU>. This library provides routines to handle square or rectangular matrix in real and complex arithmetics. The method implemented in SuperLU\_DIST is a supernodal method [?]. New release of this package include a parallel symbolic factorization [?]. This scientific library is written in C and using MPI for communication.

**Installation of SuperLU\_DIST:** To use SuperLU\_DIST in FreeFem++ , you have to install SuperLU\_DIST package. We need MPI and ParMetis library to do this compilation. An installation procedure to obtain this package is given in the file `README.COMPILE` in the directory `src/solver/` of the `freefem++` package.

**Creating Library of SuperLU\_DIST interface for FreeFem++ :** The FreeFem++ interface to SuperLU\_DIST for real (resp. complex) arithmetics is given in file `real.SuperLU_DIST-FreeFem.cpp` (resp. `complex.SuperLU_DIST-FreeFem.cpp`). These files are in the directory `src/solver/`. These interfaces are compatible with the release 3.2.1 of SuperLU\_DIST. To use SuperLU\_DIST in FreeFem++ , we need libraries corresponding to these interfaces. A description to obtain these libraries is given in the file `README.COMPILE` in the directory `src/solver` of FreeFem++ . We recall here the procedure. Go to the directory `src/solver` in FreeFem++ package. Edit the file `makefile-sparsesolver.inc` in your system : comment Section 1, comment line corresponding to libraries BLAS, Metis, ParMetis in Section 2 and comment in Section 3 the paragraph corresponding to SuperLU\_DIST solver. And just type **make rsludist** (resp. **make csludist**) in the terminal to obtain the dynamic library of interface for real (resp. complex) arithmetics.

Now we give a short description of SuperLU\_DIST parameters before describing the method to call SuperLU\_DIST in FreeFem++ .

**SuperLU\_DIST parameters:** We describe now some parameters of SuperLU\_DIST. The SuperLU\_DIST library use a 2D-logical process group. This process grid is specifies by  $nprow$  (process row) and  $npcol$  (process column) such that  $N_p = nprow \cdot npcol$  where  $N_p$  is the number of all process allocated for SuperLU\_DIST.

The input matrix parameters is controlled by "matrix=" in sparams for internal parameter or in the third line of parameters file. The different value are

matrix = assembled                      global matrix are available on all process  
matrix = distributedglobal      the global matrix is distributed among all the process  
matrix = distributed                  the input matrix is distributed (not yet implemented)

The option arguments of SuperLU\_DIST are described in the section Users-callable routine of [?]. The parameter Fact and TRANS are specified in FreeFem++ interfaces to SuperLU\_DIST during the different steps. For this reason, the value given by the user for this option is not considered.

The factorization LU is calculated in SuperLU\_DIST on the matrix  $A_p$ .

$$A_p = P_c P_r D_r A D_c P_c^t$$

where  $P_c$  and  $P_r$  is the row and column permutation matrix respectively,  $D_r$  and  $D_c$  are diagonal matrix for respectively row and column scaling. The option argument RowPerm (resp. ColPerm) control the row (resp. column) permutation matrix.  $D_r$  and  $D_c$  is controlled by the parameter DiagScale. The parameter permr, permc, scaler, and scalec in FreeFem++ is provided to give row permutation, column permutation, row scaling and column scaling of the user respectively. The other parameters for LU factorization are ParSymFact and ReplaceTinyPivot. The parallel symbolic factorization works only on a power of two processes and need the ParMetis ordering [?]. The default option argument of SuperLU\_DIST are given in the file ffsuperlu\_dist\_fileparam.txt.

**Calling SuperLU\_DIST in FreeFem++** To calling SuperLU\_DIST in FreeFem++ , we need to load the library dynamic correspond to interface. This done by the following line **load "real\_superlu\_DIST\_FreeFem"** (resp. **load "complex\_superlu\_DIST\_FreeFem"**) for real (resp. complex) arithmetics in the file .edp.

**Solver parameters is defined in .edp file:** To calling SuperLU\_DIST with internal parameter, we used the parameters sparams. The value of parameters of SuperLU\_DIST in sparams is defined by

sparams = "nprow=1, npcol=1, matrix= distributedglobal, Fact= DOFACT, Equil=NO,  
ParSymbFact=NO, ColPerm= MMD\_AT\_PLUS\_A, RowPerm= LargeDiag,                      This value  
DiagPivotThresh=1.0, IterRefine=DOUBLE, Trans=NOTRANS,  
ReplaceTinyPivot=NO, SolveInitialized=NO, PrintStat=NO, DiagScale=NOEQUIL "

correspond to the parameter in the file ffsuperlu\_dist\_fileparam.txt. If one parameter is not specify by the user, we take the default value of SuperLU\_DIST.

**Reading solver parameters on a file:** The structure of data file for SuperLU\_DIST in FreeFem++ is given in the file ffsuperlu\_dist\_fileparam.txt (default value of the FreeFem++ interface).

```
1                                      /* nprow : integer value        */
1                                      /* npcol : integer value        */
distributedglobal                    /* matrix input : assembled, distributedglobal, distributed */
```



```

DOFACT          /* Fact : DOFACT, SamePattern, SamePattern_SameRowPerm, FACTORED */
NO              /* Equil : NO, YES */
NO              /* ParSymbFact : NO, YES */
MMD_AT_PLUS_A  /* ColPerm : NATURAL, MMD_AT_PLUS_A, MMD_ATA, METIS_AT_PLUS_A, PARMETIS, MY_PERMC
*/
LargeDiag       /* RowPerm : NOROWPERM, LargeDiag, MY_PERMR */
1.0             /* DiagPivotThresh : real value */
DOUBLE          /* IterRefine : NOREFINE, SINGLE, DOUBLE, EXTRA */
NOTRANS         /* Trans      : NOTRANS, TRANS, CONJ*/
NO              /* ReplaceTinyPivot : NO, YES*/
NO              /* SolveInitialized : NO, YES*/
NO              /* RefineInitialized : NO, YES*/
NO              /* PrintStat : NO, YES*/
NOEQUIL         /* DiagScale : NOEQUIL, ROW, COL, BOTH*/

```

If no solver parameter is given, we used default option of SuperLU\_DIST solver.

**Example 11.2** *A simple example of calling SuperLU\_DIST in FreeFem++ with this two methods is given in the file testsolver\_superLU\_DIST.edp in the directory examples++-mpi.*

### 11.2.3 Pastix solver

Pastix (Parallel Sparse matrix package) is a free scientific library under CECILL-C license. This package solves sparse linear system with a direct and block ILU(k) iterative methods. This solver can be applied to a real or complex matrix with a symmetric pattern [?].

**Installation of Pastix:** To used Pastix in FreeFem++ , you have to install pastix package in first. To compile this package, we need a fortran 90 compiler, scotch [?] or Metis [?] ordering library and MPI. An installation procedure to obtain this package is given in the file .src/solver/ README.COMPILE in the section pastix of the FreeFem++ package.

**Creating Library of pastix interface for FreeFem++ :** The FreeFem++ interface to pastix is given in file real\_pastix\_FreeFem.cpp (resp. complex\_pastix\_FreeFem.cpp) for real (resp.complex) arithmetics. This interface is compatible with the release 2200 of pastix and is designed for a global matrix. We have also implemented interface for distributed matrix. To use pastix in FreeFem++ , we need the library corresponding to this interface. A description to obtain this library is given in the file README.COMPILE in the directory src/solver of FreeFem++ . We recall here the procedure. Go to the directory src/solver in FreeFem++ package. Edit the file makefile-sparsesolver.inc to yours system : comment Section 1, comment line corresponding to libraries BLAS, METIS and SCOTCH in Section 2 and comment in Section 3 the paragraph corresponding to pastix solver. And just type **make rpastix** (resp. **make cpastix**) in the terminal to obtain the dynamic library of interface for real (resp. complex) arithmetics.

Now we give a short description of pastix parameters before describing the method to call pastix in FreeFem++ .

**Pastix parameters:** The input matrix parameter of FreeFem++ depend on pastix interface. matrix=assembled for non distributed matrix. It is the same parameter for SuperLU\_DIST. There are four parameters in Pastix : iparm, dparm, perm and invp. These parameters are respectively the integer parameters (vector of size 64), real parameters (vector of size 64), permutation matrix and inverse permutation matrix respectively. iparm and dparm vectors are described in [?]. The parameters permr and permc in FreeFem++ are provided to give permutation matrix and inverse permutation matrix of the user respectively.

direct solver	square matrix			rectangular matrix		
	sym	sym pattern	unsym	sym	sym pattern	unsym
SuperLU_DIST	yes	yes	yes	yes	yes	yes
MUMPS	yes	yes	yes	no	no	no
pastix	yes	yes	no	no	no	no

Table 11.3: Type of matrix used by the different direct sparse solver

**Solver parameters defined in .edp file:** To call Pastix in FreeFem++ in this case, we need to specify the parameters **lparams** and **dparams**. These parameters are defined by

$$\forall i = 0, \dots, 63, \quad \text{lparams}[i] = \text{iparm}[i].$$

$$\forall i = 0, \dots, 63, \quad \text{dparams}[i] = \text{dparm}[i].$$

**Reading solver parameters on a file:** The structure of data file for pastix parameters in FreeFem++ is : first line structure parameters of the matrix and in the following line the value of vectors iparm and dparm in this order.

```

assembled /* matrix input :: assembled, distributed global and distributed */
iparm[0]
iparm[1]
...
...
iparm[63]
dparm[0]
dparm[1]
...
...
dparm[63]

```

An example of this file parameter is given in `ffpastix_iparm_dparm.txt` with a description of these parameters. This file is obtained with the example file `iparm.txt` and `dparm.txt` including in the pastix package.

If no solver parameter is given, we use default option of pastix solver.

**example:** A simple example of calling pastix in FreeFem++ with this two methods is given in the file `testsolver_pastix.edp` in the directory `examples++-mpi`.

In Table 11.3, we recall the different matrix considering in the different direct solvers.

## 11.3 Parallel sparse iterative solver

Concerning **iterative solvers**, we have chosen *pARMS* [?], *HIPS* [?] and *Hypre* [?]. Each software implements a different type of parallel preconditioner. So, *pARMS* implements algebraic domain decomposition preconditioner type such as additive Schwartz [?] and interface method [?]; while *HIPS* implement hierarchical incomplete factorization [?] and finally *HYPRE* implements multilevel preconditioner are AMG (Algebraic MultiGrid) [?] and parallel approximated inverse [?].

To use one of these programs in FreeFem++, you have to install it independently of FreeFem++. It is also necessary to install the MPI communication library which is essential for communication between the processors and, in some cases, software partitioning graphs like METIS [?] or Scotch [?].

All this preconditioners are used with Krylov subspace methods accelerators. Krylov subspace methods are iterative methods which consist in finding a solution  $x$  of linear system  $Ax = b$  inside the affine space  $x_0 + K_m$  by imposing that  $b - Ax \perp \mathcal{L}_m$ , where  $K_m$  is Krylov subspace of dimension  $m$  defined by  $K_m = \{r_0, Ar_0, A^2r_0, \dots, A^{m-1}r_0\}$  and  $\mathcal{L}_m$  is another subspace of dimension  $m$  which depends on type of Krylov subspace. For example in GMRES,  $\mathcal{L}_m = AK_m$ .

We realized an interface which is easy to use, so that the call of these different softwares in FreeFem++ is done in the same way. You just have to load the solver and then specify the parameters to apply to the specific solvers. In the rest of this chapter, when we talk about Krylov subspace methods we mean one among GMRES, CG and BICGSTAB.

### 11.3.1 pARMS solver

**pARMS** (*parallel Algebraic Multilevel Solver*) is a software developed by Youssef Saad and al at University of Minnesota [?]. This software is specialized in the resolution of large sparse non symmetric linear systems of equation. Solvers developed in pARMS is the Krylov subspace type. It consists of variants of GMRES like FGMRES(Flexible GMRES) , DGMRES(Deflated GMRES) [?] and BICGSTAB. pARMS also implements parallel preconditioner like RAS (Restricted Additive Schwarz)[?] and Schur Complement type preconditioner [?].

All these parallel preconditioners are based on the principle of domain decomposition. Thus, the matrix  $A$  is partitioned into sub matrices  $A_i (i = 1, \dots, p)$  where  $p$  represents the number of partitions one needs. The union of  $A_i$  forms the original matrix. The solution of the overall system is obtained by solving the local systems on  $A_i$  (see [?]). Therefore, a distinction is made between iterations on  $A$  and the local iterations on  $A_i$ . To solve the local problem on  $A_i$  there are several preconditioners as **ilut** (Incomplete LU with threshold), **iluk**(Incomplete LU with level of fill in) and **ARMS**( Algebraic Recursive Multilevel Solver). But to use pAMRS in FreeFem++ you have first to install pAMRS.

**Installation of pARMS** To install pARMS, you must first download the pARMS package at [?]. Once the download is complete, you must unpack package pARMS and follow the installation procedure described in file README to create the library **libparms.a**.

**Using pARMS as interface to FreeFem++** Before calling pARMS solver inside FreeFem++, you must compile file *parms\_FreeFem.cpp* to create a dynamic library *parms\_FreeFem.so*. To do this, move to the directory *src/solver* of FreeFem++, edit the file *makefileparms.inc* to specify the following variables:

<b>PARMS_DIR :</b>	Directory of pARMS
<b>PARMS_INCLUDE :</b>	Directory for header of pARMS
<b>METIS :</b>	METIS directory
<b>METIS_LIB :</b>	METIS librairy
<b>MPI :</b>	MPI directory
<b>MPI_INCLUDE :</b>	MPI headers
<b>FREEFEM :</b>	FreeFem++ directory
<b>FREEFEM_INCLUDE :</b>	FreeFem++ header for sparse linear solver
<b>LIBBLAS :</b>	Blas library

After that, in the command line type **make parms** to create *parms\_FreeFem.so*.

As usual in FreeFem++, we will show by examples how to call pARMS in FreeFem++. There are three

ways of doing this:

**Example 1: Default parameters** This example comes from user guide of FreeFem++ [?] at page 12.

### Example 11.3

```

1: load parms_freefem           // Tell to FreeFem that you want to use pARMS as
sparse linear solver.
2: border C(t=0,2*pi){x=cos(t); y=sin(t);label=1;}
3: mesh Th = buildmesh (C(50));
4: fespace Vh(Th,P2);
5: Vh u,v;
6: func f= x*y;
7: problem Poisson(u,v,solver=sparsesolver) =
8:   int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v))           // bilinear part, specify
that the linear solver is sparse solver in this case pARMS
9:   - int2d(Th)( f*v)                                   // right hand side
10:   + on(1,u=0) ;                                     // Dirichlet boundary condition
11:
12: real cpu=clock();
13: Poisson;                                           // SOLVE THE PDE
14: plot(u);
15: cout << " CPU time = " << clock()-cpu << endl;

```

In line 1 of example 11.3 we load in memory the pARMS dynamic library with interface FreeFem++. After this, in line 7 we specify that the bilinear form will be solved by the last sparse linear solver load in memory which, in this case, is pARMS.

The parameter used in pARMS in this case is the default one since the user does not have to provide any parameter.

Here are some default parameters:

**solver**=FGMRES, **Krylov dimension**=30, **Maximum of Krylov**=1000, **Tolerance for convergence**= $1e-08$ .(see book of Saad [?] to understand all this parameters.)

**preconditionner**=Restricted Additif Schwarz [?], **Inner Krylov dimension**=5, **Maximum of inner Krylov dimension**=5, **Inner preconditionner**=ILUK.

To specify the parameters to apply to the solver, the user can either give an integer vector for **integer parameters** and real vectors for **real parameters** or provide a **file** which contains those parameters.

**Example 2: User specifies parameters inside two vectors** Lets us consider Navier Stokes example 11.4 . In this example we solve linear systems coming from discretization of Navier Stokes equation with pARMS. Parameters of solver is specified by user.

### Example 11.4 (Stokes.edp) include "manual.edp"

```

include "includes.edp";
include "mesh_with_cylinder.edp";
include "bc_poiseuille_in_square.edp";
include "fe_functions.edp";
0: load parms_FreeFem
1: int[int] iparm(16); real[int] dparm(6);
2: int ,ii;
3: for(ii=0;ii<16;ii++){iparm[ii]=-1;}  for(ii=0;ii<6;ii++) dparm[ii]=-1.0;
4: fespace Vh(Th,[P2,P2,P1]);

```

```

5: iparm[0]=0;
6: varf Stokes ([u,v,p],[ush,vsh,psh},{solver=sparsesolver}) =
    int2d(Th)( nu*( dx(u)*dx(ush) + dy(u)*dy(ush) + dx(v)*dx(vsh) + dy(v)*dy(vsh) )
        - p*psh*(1.e-6) // p epsilon
        - p*(dx(ush)+dy(vsh))
        // + dx(p)*ush + dy(p)*vsh
        // psh div(u)
        - (dx(u)+dy(v))*psh
    )
+ on(cylinder,infwall,supwall,u=0.,v=0.)+on(inlet,u=uc,v=0); // Boundary
conditions
7: matrix AA=Stokes(VVh,VVh);
8: set(AA,solver=sparsesolver,lparams=iparm,dparams=dparm); // Set pARMS as
Linear solver
9: real[int] bb= Stokes(0,VVh); real[int] sol(AA.n);
10: sol= AA^-1 * bb;

```

We need two vectors to specify the parameters of the linear solver. In line 1 of example 11.4 we have declared these vectors(**int[int] iparm(16); real[int] dparm(6);**). In line 3 we have initialized these vectors by negative values. We do this because all parameters values in pARMS are positive and if you do not change the negative values of one entry of this vector, the default value will be set. In tables (table 11.4 and 11.5), we have the meaning of different entries of these vectors.

We run example 11.4 on cluster paradiet of Grid5000 and report results in table 11.8.

In this example, we fix the matrix size (in term of finite element, we fix the mesh) and increase the number of processors used to solve the linear system. We saw that, when the number of processors increases, the time for solving the linear equation decreases, even if the number of iteration increases. This proves that, using pARMS as solver of linear systems coming from discretization of partial differential equation in FreeFem++ can decrease drastically the total time of simulation.

## 11.3.2 Interfacing with HIPS

**HIPS** (*Hierarchical Iterative Parallel Solver*) is a scientific library that provides an efficient parallel iterative solver for very large sparse linear systems. HIPS is available as free software under the CeCILL-C licence. The interface that we realized is compatible with release **1.2 beta.rc4** of HIPS.

HIPS implements two solver classes which are the iteratives class (GMRES, PCG) and the Direct class. Concerning preconditionners, HIPS implements a type of multilevel ILU. For further informations on those preconditionners see [?, ?].

**Installation of HIPS** To install HIPS, first download the HIPS package at [?], unpack it and go to the HIPS source directory. The installation of HIPS is machine dependence. For example, to install HIPS on a linux cluster copy the file *Makefile\_Inc\_Examples/makefile.inc.gnu* on the root directory of HIPS with the name **makefile.inc**. After this, edit **makefile.inc** to set values of different variables and type **make all**.

**Using HIPS as the interface to FreeFem++** Before calling the HIPS solver inside FreeFem++, you must compile file *hips\_FreeFem.cpp* to create dynamic library *hips\_FreeFem.so*. To do this, move to the directory *src/solver* of FreeFem++ and edit the file *makefile.inc* to specify the following variables:

Entries of iparm	Significations of each entries
iparm[0]	Krylov subspace methods. Differents values for this parameters are specify on table 11.6
iparm[1]	Preconditionner. Differents preconditionners for this parameters are specify on table 11.7
iparm[2]	Krylov subspace dimension in outer iteration: default value 30
iparm[3]	Maximum of iterations in outer iteration: default value 1000
iparm[4]	Number of level in arms when used.
iparm[5]	Krylov subspace dimension in inner iteration: default value 3
iparm[6]	Maximum of iterations in inner iteration: default value 3
iparm[7]	Symmetric(=1 for symmetric) or unsymmetric matrix: default value 0(unsymmetric matrix)
iparm[8]	Overlap size between different subdomain: default value 0(no overlap)
iparm[9]	Scale the input matrix or not: Default value 1 (Matrix should be scale)
iparm[10]	Block size in arms when used: default value 20
iparm[11]	lfl0 (ilut, iluk, and arms) : default value 20
iparm[12]	lfl for Schur complement const : default value 20
iparm[13]	lfl for Schur complement const : default value 20
iparm[14]	Multicoloring or not in ILU when used : default value 1
iparm[15]	Inner iteration : default value 0
iparm[16]	Print message when solving:default 0(no message print). 0: no message is print, 1: Convergence informations like number of iteration and residual , 2: Timing for a different step like preconditioner 3 : Print all informations.

Table 11.4: Significations of **lparams** corresponding variables for example 11.4

Entries of dparam	Significations of each entries
dparam[0]	precision for outer iteration : default value 1e-08
dparam[1]	precision for inner iteration: default value 1e-2
dparam[2]	tolerance used for diagonal domain: : default value 0.1
dparam[3]	drop tolerance droptol0 (ilut, iluk, and arms) : default value 1e-2
dparam[4]	droptol for Schur complement const: default value 1e-2
dparam[5]	droptol for Schur complement const: default value 1e-2

Table 11.5: Significations of **dparams** corresponding variables for example 11.4

Values of iparm[0]	Krylov subspace methods
0	FGMRES (Flexible GMRES)
1	DGMRES (Deflated GMRES)
2	BICGSTAB

Table 11.6: Krylov Solvers in pARMS

Values of iparm[1]	Preconditionners
0	Preconditioners type is <i>additive Schwartz preconditioner with ilu0 as local preconditioner,</i>
1	preconditioner type is <i>additive Schwartz preconditioner with iluk as local preconditioner,</i>
2	preconditioner type is <i>additive Schwartz preconditioner with ilut as local preconditioner,</i>
3	preconditioner type is <i>additive Schwartz preconditioner with arms as local preconditioner,</i>
4	preconditioner type is <i>Left Schur complement preconditioner with ilu0 as local preconditioner,</i>
5	preconditioner type is <i>Left Schur complement preconditioner with ilut as local preconditioner,</i>
6	preconditioner type is <i>Left Schur complement preconditioner with iluk as local preconditioner,</i>
7	preconditioner type is <i>Left Schur complement preconditioner with arms as local preconditioner,</i>
8	preconditioner type is <i>Right Schur complement preconditioner with ilu0 as local preconditioner,</i>
9	preconditioner type is <i>Right Schur complement preconditioner with ilut as local preconditioner,</i>
10	preconditioner type is <i>Right Schur complement preconditioner with iluk as local preconditioner,</i>
11	preconditioner type is <i>Right Schur complement preconditioner with arms as local preconditioner,</i>
12	preconditioner type is <i>sch_gilu0 , Schur complement preconditioner with global ilu0</i>
13	preconditioner type is <i>SchurSymmetric GS preconditioner</i>

Table 11.7: Preconditionners in pARMS

<b>n= 471281</b>		<b>nnz=<math>13 \times 10^6</math></b>		<b>Te=571,29</b>
np	add(iluk)		schur(iluk)	
	nit	time	nit	time
4	230	637.57	21	557.8
8	240	364.12	22	302.25
16	247	212.07	24	167.5
32	261	111.16	25	81.5

Table 11.8: Convergence and time for solving linear system from example 11.4

n	matrix size
nnz	number of non null entries inside matrix
nit	number of iteration for convergence
time	Time for convergence
Te	Time for constructing finite element matrix
np	number of processor

Table 11.9: Legend of table 11.8

<i>HIPS_DIR</i> :	Directory of HIPS
<i>HIPS_INCLUDE</i> :	-I\$( <i>HIPS_DIR</i> )/SRC/INCLUDE : Directory for HIPS headers
<i>LIB_DIR</i> :	-L\$( <i>HIPS_DIR</i> )/LIB : Librairies directory
<i>LIBHIPSSEQUENTIAL</i> :	\$( <i>HIPS_DIR</i> )/LIB/libhipssequential.a: HIPS utilities library
<i>LIBHIPS</i> :	\$( <i>HIPS_DIR</i> )/LIB/libhips.a: HIPS library
<i>FREEFEM</i> :	FreeFem++ directory
<i>FREEFEM_INCLUDE</i> :	FreeFem headers for sparse linear solver
<i>METIS</i> :	METIS directory
<i>METIS_LIB</i> :	METIS library
<i>MPI</i> :	MPI directory
<i>MPI_INCLUDE</i> :	MPI headers

After specifies all the variables, in the command line in the directory *src/solver* type **make hips** to create *hips-FreeFem.so*.

Like with pARMS, the calling of HIPS in FreeFem++ can be done in three different manners. We will present only one example where the user specifies the parameters through keywords *lparams* and *dparams*.

**Laplacian 3D solve with HIPS** Let us consider the 3D Laplacian example inside FreeFem++ package where after discretization we want to solve the linear equation with Hips. Example 11.5 is Laplacian3D using Hips as linear solver. We first load Hips solver at line 2. From line 4 to 15 we specify the parameters for the Hips solver and in line 46 of example 11.5 we set these parameters in the linear solver.

In Table 11.10 results of running example 11.5 on Cluster Paradent of Grid5000 are reported. We can see in this running example the efficiency of parallelism.

#### Example 11.5 (Laplacian3D.edp) 1: load "msh3"

```

2: load "hips-FreeFem"                                // load library
3: int nn=10,iii;
4: int[int] iparm(14);
5: real[int] dparm(6);
6: for(iii=0;iii<14;iii++)iparm[iii]=-1;
7: for(iii=0;iii<6;iii++) dparm[iii]=-1;
8: iparm[0]=0;                                         // use iterative solver
9: iparm[1]=1;                                         // PCG as Krylov method
10: iparm[4]=0;                                        // Matrix are symmetric
11: iparm[5]=1;                                        // Pattern are also symmetric
12: iparm[9]=1;                                        // Scale matrix
13: dparm[0]=1e-13;                                    // Tolerance to convergence
14: dparm[1]=5e-4;                                    // Threshold in ILUT
15: dparm[2]=5e-4;                                    // Threshold for Schur preconditionner
16: mesh Th2=square(nn,nn);
17: fespace Vh2(Th2,P2);
18: Vh2 ux,uz,p2;
```



$n = 4 \times 10^6$	$nnz = 118 \times 10^6$	<b>Te=221.34</b>
np	nit	time
8	190	120.34
16	189	61.08
32	186	31.70
64	183	23.44

Table 11.10: Iterations and Timing of solving linear system from example 11.5

```

19: int[int] rup=[0,2], rdown=[0,1], rmid=[1,1,2,1,3,1,4,1];
20: real zmin=0,zmax=1;
21: mesh3 Th=buildlayers(Th2,nn,
    zbound=[zmin,zmax],
    reffacemid=rmid,
    reffaceup = rup,
    reffacelow = rdown);
22: savemesh(Th,"copie.mesh");
23: mesh3 Th3("copie.mesh");
24: fespace Vh(Th,P2);
25: func ue = 2*x*x + 3*y*y + 4*z*z + 5*x*y+6*x*z+1;
26: func uex= 4*x+ 5*y+6*z;
27: func uey= 6*y + 5*x;
28: func uez= 8*z +6*x;
29: func f= -18. ;
30: Vh uhe = ue;
31: cout << " uhe min: " << uhe[].min << " max:" << uhe[].max << endl;
32: Vh u,v;
33: macro Grad3(u) [dx(u),dy(u),dz(u)]
34: varf va(u,v)= int3d(Th)(Grad3(v)' *Grad3(u))
    + int2d(Th,2)(u*v)
    - int3d(Th)(f*v)
    - int2d(Th,2) ( ue*v + (uex*N.x +uey*N.y +uez*N.z)*v )
    + on(1,u=ue);
35: real cpu=clock();
36: matrix Aa;
37: Aa=va(Vh,Vh);
38: varf l(unused,v)=int3d(Th)(f*v);
39: Vh F; F[]=va(0,Vh);
40: if(mpirank==0){
    cout << "Taille " << Aa.n << endl;
    cout << "Non zeros " << Aa.nbcoef << endl;
}
41: if(mpirank==0)
42:   cout << "CPU TIME FOR FORMING MATRIX = " << clock()-cpu << endl;
43:   set(Aa,solver=sparse solver,dparams=dparm, lparams=iparm);
44:   u[]=Aa^-1*F[];

```

//

// EOM

// ') for emacs

// Set hips as

Legend of table 11.10 are give in table 11.9.

Entries of <b>iparm</b>	Significations of each entries
iparm[0]	Strategy use for solving ( Iterative=0 or Hybrid=1 or Direct=2 ). Defaults values are : Iterative
iparm[1]	Krylov methods. If iparm[0]=0, give type of Krylov methods: 0 for GMRES, 1 for PCG
iparm[2]	Maximum of iterations in outer iteration: default value 1000
iparm[3]	Krylov subspace dimension in outer iteration: default value 40
iparm[4]	Symmetric(=0 for symmetric) and 1 for unsymmetric matrix: default value 1(unsymmetric matrix)
iparm[5]	Pattern of matrix are symmetric or not: default value 0
iparm[6]	Partition type of input matrix: default value 0
iparm[7]	Number of level that use the HIPS locally consistent fill-in: Default value 2
iparm[8]	Numbering in indices array will start at 0 or 1: Default value 0
iparm[9]	Scale matrix. Default value 1
iparm[10]	Reordering use inside subdomains for reducing fill-in: Only use for iterative. Default value 1
iparm[11]	Number of unknowns per node in the matrix non-zero pattern graph: Default value 1
iparm[12]	This value is used to set the number of time the normalization is applied to the matrix: Default 2.
iparm[13]	Level of informations printed during solving: Default 5.
iparm[14]	HIPS_DOMSIZE Subdomain size

Table 11.11: Significations of **lparams** corresponding to HIPS interface

dparm[0]	<i>HIPS_PREC</i> : Relative residual norm: Default=1e-9
dparm[1]	<i>HIPS_DROPTOL0</i> : Numerical threshold in ILUT for interior domain (important : set 0.0 in HYBRID: Default=0.005)
dparm[2]	<i>HIPS_DROPTOL1</i> : Numerical threshold in ILUT for Schur preconditioner: Default=0.005
dparm[3]	<i>HIPS_DROPTOLE</i> : Numerical threshold for coupling between the interior level and Schur: Default 0.005
dparm[4]	<i>HIPS_AMALG</i> : Numerical threshold for coupling between the interior level and Schur: Default=0.005
dparm[5]	<i>HIPS_DROPSCHUR</i> : Numerical threshold for coupling between the interior level and Schur: Default=0.005

Table 11.12: Significations of **dparams** corresponding to HIPS interface

### 11.3.3 Interfacing with HYPRE

**HYPRE** (*High Level Preconditioner*) is a suite of parallel preconditioner developed at Lawrence Livermore National Lab [?].

There are two main classes of preconditioners developed in HYPRE: AMG (Algebraic MultiGrid) and Parasails (Parallel Sparse Approximate Inverse).

Now, suppose we want to solve  $Ax = b$ . At the heart of AMG there is a series of progressively coarser(smaller) representations of the matrix  $A$ . Given an approximation  $\hat{x}$  to the solution  $x$ , consider solving the residual equation  $Ae = r$  to find the error  $e$ , where  $r = b - A\hat{x}$ . A fundamental principle of AMG is that it is an algebraically smooth error. To reduce the algebraically smooth errors further, they need to be represented by a smaller defect equation (coarse grid residual equation)  $A_c e_c = r_c$ , which is cheaper to solve. After solving this coarse equation, the solution is then interpolated in fine grid represented here by matrix  $A$ . The quality of AMG depends on the choice of coarsening and interpolating operators.

The *sparse approximate inverse* approximates the inverse of a matrix  $A$  by a sparse matrix  $M$ . A technical idea to construct matrix  $M$  is to minimize the Frobenius norm of the residual matrix  $I - MA$ . For more details on this preconditioner technics see [?].

HYPRE implement three Krylov subspace solvers: GMRES, PCG and BiCGStab.

**Installation of HYPRE** To install HYPRE, first download the HYPRE package at [?], unpack it and go to the HYPRE/src source directory and do `./configure` to configure Hypre. After this just type `make all` to create `libHYPRE.a`.

**Using HYPRE as interface to FreeFem++** Before calling HYPRE solver inside FreeFem++, you must compile the file `hypre_FreeFem.cpp` to create dynamic library `hypre_FreeFem.so`. To do this, move to the directory `src/solver` of FreeFem++, edit the file `makefile.inc` to specify the following variables:

```

HYPRE_DIR :           Directory of HYPRE
HYPRE_INCLUDE =       -I$(HYPRE_DIR)src/hypre/include/ :
                        Directory for header of HYPRE
HYPRE_LIB =           -L$(HIPS_DIR)/src/lib/ -lHYPRE : Hypre Library
FREEFEM :             FreeFem++ directory
FREEFEM_INCLUDE :     FreeFem header for sparse linear solver
METIS :               METIS directory
METIS_LIB :           METIS library
MPI :                 MPI directory
MPI_INCLUDE :         MPI headers

```

Like with pARMS, the calling of HIPS in FreeFem++ can be done in three manners. We will present only one example where the user specifies its parameters through keywords `lparams` and `dparams`.

**Laplacian 3D solve with HYPRE** Let us consider again the 3D Laplacian example inside FreeFem++ package where after discretization we want to solve the linear equation with Hypre. Example 11.6 is the Laplacian3D using Hypre as linear solver. Example 11.6 is the same as 11.5, so we just show here the lines where we set some Hypre parameters.

We first load the Hypre solver at line 2. From line 4 to 15 we specifies the parameters to set to Hypre solver and in line 43 we set parameters to Hypre solver.

It should be noted that the meaning of the entries of these vectors is different from those of Hips . In the case of HYPRE, the meaning of different entries of vectors **iparm** and **dparm** are given in tables 11.13 to 11.17.

In Table ?? the results of running example 11.6 on Cluster Paradent of Grid5000 are reported. We can see in this running example the efficiency of parallelism, in particular when AMG are use as preconditioner.

**Example 11.6 (Laplacian3D.edp)** 1: load "msh3"

```

2: load "hipre_FreeFem" // load librairie
3: int nn=10,iii;
4: int[int] iparm(20);
5: real[int] dparm(6);
6: for(iii=0;iii<20;iii++)iparm[iii]=-1;
7: for(iii=0;iii<6;iii++) dparm[iii]=-1;
8: iparm[0]=2; // PCG as krylov method
9: iparm[1]=0; // AMG as preconditionner 2: if ParaSails
10: iparm[7]=7; // Interpolation
11: iparm[9]=6; // AMG Coarsen type
12: iparm[10]=1; // Measure type
13: iparm[16]=2; // Additive schwarz as smoother
13: dparm[0]=1e-13; // Tolerance to convergence
14: dparm[1]=5e-4; // Threshold
15: dparm[2]=5e-4; // truncation factor
.
.
.
43: set(Aa,solver=sparse solver,dparams=dparm, lparams=iparm);

```

iparms[0]	Solver identification: 0: BiCGStab, 1: GMRES, 2: PCG. By <b>default=1</b>
iparms[1]	Preconditioner identification: 0: BOOMER AMG, 1: PILUT, 2: Parasails, 3: Schwartz <b>Default=0</b>
iparms[2]	Maximum of iteration: <b>Default=1000</b>
iparms[3]	Krylov subspace dim: <b>Default= 40</b>
iparms[4]	Solver print info level: <b>Default=2</b>
iparms[5]	Solver log : <b>Default=1</b>
iparms[6]	Solver stopping criteria only for BiCGStab : <b>Default=1</b>
dparms[0]	Tolerance for convergence : <i>Default = 1.0e – 11</i>

Table 11.13: Definitions of common entries of **iparms** and **dparms** vectors for every preconditioner in HYPRE

### 11.3.4 Conclusion

With the different runs presented here, we wanted to illustrate the gain in time when we increase the number of processors used during simulation. We saw that in every case time for construct finite element matrix is constant. This is normal because until now this phase is sequential in FreeFem++. Contrariwise, phases for solving the linear system are parallel. We saw on different examples presented here that when we increase the number of processor, in general we decrease the time used for solving linear system. But this not true in every case. In several case, when we increase the number of processors the time to convergence also increases. There are two main reasons for this. First, the increase of processors can lead to the increase of volume of exchange data across processors with consequence of increase the time for solving linear system. Second, in decomposition domain type preconditioner, the number of processors generally corresponds to the number of sub domains. In subdomain methods, generally when we increase the number of subdomains we decrease convergence quality of the preconditioner. This can have effect of increase time use for solving linear equations.

iparms[7]	AMG interpolation type: <b>Default=6</b>
iparms[8]	Specifies the use of GSMG - geometrically smooth coarsening and interpolation: <b>Default=1</b>
iparms[9]	AMG coarsen type: <b>Default=6</b>
iparms[10]	Defines whether local or global measures are used: <b>Default=1</b>
iparms[11]	AMG cycle type: <b>Default=1</b>
iparms[12]	AMG Smoother type: <b>Default=1</b>
iparms[13]	AMG number of levels for smoothers: <b>Default=3</b>
iparms[14]	AMG number of sweeps for smoothers: <b>Default=2</b>
iparms[15]	Maximum number of multigrid levels: <b>Default=25</b>
iparms[16]	Defines which variant of the Schwartz method is used: 0: hybrid multiplicative Schwartz method (no overlap across processor boundaries) 1: hybrid additive Schwartz method (no overlap across processor boundaries) 2: additive Schwartz method 3: hybrid multiplicative Schwartz method (with overlap across processor boundaries) <b>Default=1</b>
iparms[17]	Size of the system of PDEs: <b>Default=1</b>
iparms[18]	Overlap for the Schwarz method: <b>Default=1</b>
iparms[19]	Type of domain used for the Schwarz method 0: each point is a domain 1: each node is a domain (only of interest in “systems” AMG) 2: each domain is generated by agglomeration (default)
dparms[1]	AMG strength threshold: <b>Default=0.25</b>
dparms[2]	Truncation factor for the interpolation: <b>Default=1e-2</b>
dparms[3]	Sets a parameter to modify the definition of strength for diagonal dominant portions of the matrix: <b>Default=0.9</b>
dparms[3]	Defines a smoothing parameter for the additive Schwartz method <b>Default=1.</b>

Table 11.14: Definitions of other entries of **iparms** and **dparms** if preconditioner is **BOOMER AMG**

iparms[7]	Row size in Parallel ILUT: <b>Default=1000</b>
iparms[8]	Set maximum number of iterations: <b>Default=30</b>
dparms[1]	Drop tolerance in Parallel ILUT: <b>Default=1e-5</b>

Table 11.15: Definitions of other entries of **iparms** and **dparms** if preconditioner is **PILUT**

iparms[7]	Number of levels in Parallel Sparse Approximate inverse: <b>Default=1</b>
iparms[8]	Symmetric parameter for the ParaSails preconditioner: 0: nonsymmetric and/or indefinite problem, and nonsymmetric preconditioner 1: SPD problem, and SPD (factored) preconditioner 2: nonsymmetric, definite problem, and SPD (factored) preconditioner <b>Default=0</b>
dparms[1]	Filters parameters: The filter parameter is used to drop small nonzeros in the preconditioner, to reduce the cost of applying the preconditioner: <b>Default=0.1</b>
dparms[2]	Threshold parameter: <b>Default=0.1</b>

Table 11.16: Definitions of other entries of **iparms** and **dparms** if preconditioner is **ParaSails**

iparms[7]	Defines which variant of the Schwartz method is used: 0: hybrid multiplicative Schwartz method (no overlap across processor boundaries) 1: hybrid additive Schwartz method (no overlap across processor boundaries) 2: additive Schwartz method 3: hybrid multiplicative Schwartz method (with overlap across processor boundaries) <b>Default=1</b>
iparms[8]	Overlap for the Schwartz method: <b>Default=1</b>
iparms[9]	Type of domain used for the Schwartz method 0: each point is a domain 1: each node is a domain (only of interest in “systems” AMG) 2: each domain is generated by agglomeration (default)

Table 11.17: Definitions of other entries of **iparms** and **dparms** if preconditioner is **Schwartz**

<b><math>n=4 \times 10^6</math></b>	<b><math>nnz=13 \times 10^6</math></b>	<b><math>Te=571,29</math></b>
np	AMG	
	nit	time
8	6	1491.83
16	5	708.49
32	4	296.22
64	4	145.64

Table 11.18: Convergence and time for solving linear system from example 11.4

To end this, we should note that good use of preconditioner interfaced in FreeFem++ is empiric, because it is difficult to know what is a good preconditioner for some type of problems. Although, the efficiency of preconditioner sometimes depends on how preconditioner parameters are set. For this reason we advise user to check the meaning of different parameters in the user guide of different iterative solvers interface in FreeFem++.

## 11.4 Domain decomposition

In previous section, we saw that the phases of constructing matrix are sequential. One strategy to construct the matrix in parallel is to divide geometric domain into subdomains. In every subdomain we construct a local submatrix and after that we assemble every submatrix to form global matrix.

We can use this technic to solve pde directly in domain  $\Omega$ . In this case, in every subdomains you have to define artificial boundaries conditions to form consistent equations on every subdomains. After this, you solve equation on every subdomains and define strategy to obtain global solution.

In term of parallel programming on FreeFem++ with MPI, this means that the user must be able to divide processors group available for computation on subgroups of processors and also must be able to realise different type of communication in FreeFem++ script. Here is wrapper of some MPI functions.

### 11.4.1 Communicators and groups

#### Groups

**mpiGroup grpe(mpiGroup gp, KN\_ < long >):** Create *MPI\_Group* from existing group **gp** by given vector

#### Communicators

Communicators is an abstract MPI object which allows MPI user to communicate across group of processors. Communicators can be Intracommunicators (involves a single group) or Intercommunicators (involves two groups). When we not specify type of communicator it will be Intracommunicators

**mpiComm cc(mpiComm comm, mpiGroup gp):** Creates a new communicator. *comm* communicator(handle), *gp* group which is a subset of the group of *comm* (handle). Return new communicator

**mpiComm cc(mpiGroup gp):** Same as previous constructor but default *comm* here is MPI.COMM\_WORLD.

**mpiComm cc(mpiComm comm, int color, int key):** Creates new communicators based on *colors* and *key*. This constructor is based on MPI\_Comm\_split routine of MPI.

**mpiComm cc(MPIrank p, int key):** Same constructor than the last one. Here *colors* and *comm* is defined in *MPIrank*. This constructor is based on MPI\_Comm\_split routine of MPI.

**Example 11.7 (commsplit.edp)**

```
1: int color=mpiRank(comm)%2;
2: mpiComm ccc(processor(color,comm),0);
3: mpiComm qpp(comm,);
4: mpiComm cp(cc,color,0);
```

**mpiComm cc(mpiComm comm, int high):** Creates an intracommunicator from an intercommunicator. *comm* intercommunicator, *high* Used to order the groups within *comm* (logical) when creating the new communicator. This constructor is based on MPI\_Intercomm\_merge routine of MPI.

**mpiComm cc(MPIrank p1, MPIrank p2, int tag):** This constructor creates an intercommunicator from two intracommunicators. *p1* defined local (intra)communicator and rank in local\_comm of leader (often 0) while *p2* defined remote communicator and rank in peer\_comm of remote leader (often 0). *tag* Message tag to use in constructing intercommunicator. This constructor is based on MPI\_Intercomm\_create.

**Example 11.8 (merge.edp)**

```

1: mpiComm comm,cc;
2: int color=mpiRank(comm)%2;
3: int rk=mpiRank(comm);
4: int size=mpiSize(comm);
4: cout << "Color values " << color << endl;
5: mpiComm ccc(processor((rk<size/2),comm),rk);
6: mpiComm cp(cc,color,0);
7: int rleader;
8: if (rk == 0) { rleader = size/2; }
9: else if (rk == size/2) { rleader = 0; }
10: else { rleader = 3; }
11: mpiComm qqp(processor(0,ccc),processor(rleader,comm),12345);
12: int aaa=mpiSize(ccc);
13: cout << "number of processor" << aaa << endl;

```

### 11.4.2 Process

In FreeFem++ we wrap MPI process by function call **processor** which create internal FreeFem++ object call **MPIrank**. This mean that do not use **MPIrank** in FreeFem++ script.

**processor(int rk):** Keep process rank inside object **MPIrank**. Rank is inside **MPI\_COMM\_WORLD**.

**processor(int rk, mpiComm cc) and processor(mpiComm cc,int rk)** process rank inside communicator **cc**.

**processor(int rk, mpiComm cc) and processor(mpiComm cc,int rk)** process rank inside communicator **cc**.

**processorblock(int rk) :** This function is exactly the same than **processor(int rk)** but is use in case of blocking communication.

**processorblock(int rk, mpiComm cc) :** This function is exactly the same than **processor(int rk,mpiComm cc)** but use a synchronization point.

### 11.4.3 Points to Points communicators

In FreeFem++ you can call MPI points to points communications functions.

**Send(processor(int rk,mpiComm cc),Data D) :** Blocking send of *Data D* to processor of *rank rk* inside communicator *cc*. Note that *Data D* can be: *int, real,complex, int[int], real[int],complex[int], Mesh, Mesh3, Matrix*.

**Recv(processor(int rk,mpiComm cc),Data D):** Receive *Data D* from process of rank *rk* in communicator *cc*. Note that *Data D* can be: *int, real,complex, int[int], real[int],complex[int], Mesh, Mesh3, Matrix* and should be the same type than corresponding send.

**Isend(processor(int rk,mpiComm cc),Data D) :** Non blocking send of *Data D* to processor of *rank rk* inside communicator *cc*. Note that *Data D* can be: *int, real,complex, int[int], real[int],complex[int], Mesh, Mesh3, Matrix*.

**Recv(processor(int rk,mpiComm cc),Data D):** Receive corresponding to send.

### 11.4.4 Global operations

In FreeFem++ you can call MPI global communication functions.

**broadcast(processor(int rk,mpiComm cc),Data D):** Process *rk* Broadcast *Data D* to all process inside communicator *cc*. Note that *Data D* can be: *int, real,complex, int[int], real[int],complex[int], Mesh,*



*Mesh3, Matrix.*

**broadcast(processor(int rk),Data D):** Process *rk* Broadcast *Data D* to all process inside MPI\_COMM\_WORLD. Note that *Data D* can be: *int, real,complex, int[int], real[int],complex[int], Mesh, Mesh3, Matrix.*

**mpiAlltoall(Data a,Data b):** Sends *data a* from all to all processes. Receive buffer is *Data b*. This is done inside communicator MPI\_COMM\_WORLD.

**mpiAlltoall(Data a,Data b, mpiComm cc):** Sends *data a* from all to all processes. Receive buffer is *Data b*. This is done inside communicator *cc*.

**mpiGather(Data a,Data b,processor(mpiComm,int rk) :** Gathers together values *Data a* from a group of processes. Process of rank *rk* get data on communicator *rk*. This function is like MPI\_Gather

**mpiAllgather(Data a,Data b) :** Gathers *Data a* from all processes and distribute it to all in *Data b*. This is done inside communicator MPI\_COMM\_WORLD. This function is like MPI\_Allgather

**mpiAllgather(Data a,Data b, mpiComm cc) :** Gathers *Data a* from all processes and distribute it to all in *Data b*. This is done inside **communicator cc**. This function is like MPI\_Allgather

**mpiScatter(Data a,Data b,processor(int rk, mpiComm cc)) :** Sends **Data a** from one process with rank **rk** to all other processes in group represented by communicator *mpiComm cc*.

**mpiReduce(Data a,Data b,processor(int rk, mpiComm cc),MPI\_Op op),** Reduces values *Data a* on all processes to a single value *Data b* on process of rank *rk* and communicator *cc*. Operation use in reduce is: *MPI\_Op op* which can be: *mpiMAX, mpiMIN, mpiSUM, mpiPROD, mpiLAND, mpiLOR, mpiLXOR, mpiBAND, mpiBXOR, mpiMAXLOC, mpiMINLOC.*

Note that, for all global operations, only *int[int]* and *real[int]* are data type take in account in FreeFem++.

The following example present in details of Schwartz domain decomposition algorithm for solving Laplacian2d problem. In this example we use two level of parallelism to solve simple Laplacian2d in square domain. We have few number of subdomain and in every subdomain we use parallel sparse solver to solve local problem.

```

Example 11.9 (schwarz.edp) 1:load "hypr-FreeFem"; // Load Hypr solver
2:func bool AddLayers(mesh & Th,real[int] &ssd,int n,real[int] &unssd)
{
    // build a continuous function uussd (P1) :
    // ssd in the characteristics function on the input sub domain.
    // such that :
    // unssd = 1 when ssd =1;
    // add n layer of element (size of the overlap)
    // and unssd = 0 outside of this layer ...
    // -----
    fespace Vh(Th,P1);
    fespace Ph(Th,P0);
    Ph s;
    assert(ssd.n==Ph.ndof);
    assert(unssd.n==Vh.ndof);
    unssd=0;
    s[] = ssd;
}

```

```

//      plot(s,wait=1,fill=1);

Vh u;
varf vM(u,v)=int2d(Th,qforder=1)(u*v/area);
matrix M=vM(Ph,Vh);

for(int i=0;i<n;++i)
{
    u[] = M*s[];
//      plot(u,wait=1);

    u = u>.1;
//      plot(u,wait=1);

    unssd+= u[];
    s[] = M'*u[];
//      '
    s = s >0.1;
}
unssd /= (n);
u[]=unssd;
ssd=s[];
return true;
}

3: mpiComm    myComm;                //      Create communicator with value MPI_COMM_WORLD

4: int membershipKey,rank,size;        //      Variables for manage communicators
5: rank=mpiRank(myComm); size=mpiSize(myComm);    //      Rank of process and size of
communicator
6: bool withmetis=1, RAS=0;            //      Use or not metis for partitioning Mesh
7: int sizeoverlaps=5;                //      size off overlap
8: int withplot=1;
9: mesh Th=square(100,100);
10: int[int] chlab=[1,1 ,2,1 ,3,1 ,4,1 ];
11: Th=change(Th,refe=chlab);
12: int nn=2,mm=2, npart= nn*mm;
13: membershipKey = mpiRank(myComm)%npart; //      Coloring for partitioning process group
14: mpiComm cc(processor(membershipKey,myComm),rank); //      Create MPI communicator
according previous coloring
15: fespace Ph(Th,P0), fespace Vh(Th,P1);
16: Ph part;
17: Vh sun=0,unssd=0;
18: real[int] vsum=sun[], reducesum=sun[];    //      Data use for control partitioning.
19: Ph xx=x,yy=y,nupp;
20: part = int(xx*nn)*mm + int(yy*mm);
21: if(withmetis)
{
    load "metis";
    int[int] nupart(Th.nt);
    metisdual(nupart,Th,npart);
    for(int i=0;i<nupart.n;++i)
        part[][i]=nupart[i];
}
22: if(withplot>1)
21: plot(part,fill=1,cmm="dual",wait=1);
22: mesh[int] aTh(npart);
23: mesh Thi=Th;
24: fespace Vhi(Thi,P1);
25: Vhi[int] au(npart),pun(npart);
26: matrix[int] Rih(npart), Dih(npart), aA(npart);

```

```

27: Vhi[int] auntgv(npart), rhsi(npart);
28: i=membershipKey;
    Ph suppi= abs(part-i)<0.1;
    AddLayers(Th,suppi[],sizeoverlaps,unssd[]);
    Thi=aTh[i]=trunc(Th,suppi>0,label=10,split=1);
    Rih[i]=interpolate(Vhi,Vh,inside=1);           //    Vh -> Vhi
    if(RAS)
    {
        suppi= abs(part-i)<0.1;
        varf vSuppi(u,v)=int2d(Th,qforder=1)(suppi*v/area);
        unssd[]= vSuppi(0,Vh);
        unssd = unssd>0.;
        if(withplot>19)
            plot(unssd,wait=1);
    }
    pun[i][]=Rih[i]*unssd[];                       //    this is global operation
    sun[] += Rih[i]*pun[i][];                      //    also global operation like broadcast';
    vsum=sun[];
    if(withplot>9)
        plot(part,aTh[i],fill=1,wait=1);
        //    Add mpireduce for sum all sun and pun local contribution.
29: mpiReduce(vsum, reducesum,processor(0,myComm),mpiSUM); //    MPI global operation
MPI_Reduce on global communicator
30: broadcast(processor(0,myComm),reducesum); //    Broadcast sum on process 0 to all
process
31: sun[]=reducesum;
32: plot(sun,wait=1);
33: i=membershipKey
34: Thi=aTh[i];
35: pun[i]= pun[i]/sun;
36: if(withplot>8) plot(pun[i],wait=1);
37: macro Grad(u) [dx(u),dy(u)]                  //    EOM
38: sun=0;
39: i=membershipKey
    Thi=aTh[i];
    varf va(u,v) =
        int2d(Thi)(Grad(u)*Grad(v))              //    ')
        +on(1,u=1) + int2d(Th)(v)
        +on(10,u=0) ;
40: aA[i]=va(Vhi,Vhi);
41: set(aA[i],solver=sparsesolver,mpicomm=cc); //    Set parameters for Solver Hypre.
mpicomm=cc means you not solve on global process but in group on of process define by cc
42: rhsi[i][]= va(0,Vhi);
43: Dih[i]=pun[i][];
44: real[int] un(Vhi.ndof);
45: un=1.;
46: real[int] ui=Dih[i]*un;
47: sun[] += Rih[i]*ui;                          //    ');
48: varf vaun(u,v) = on(10,u=1);
49: auntgv[i][]=vaun(0,Vhi);                      //    store array of tgv on Gamma intern.
56: reducesum=0; vsum=sun;
57: mpiReduce(vsum, reducesum,processor(0,myComm),mpiSUM); //    MPI global operation
MPI_Reduce on global communicator
58: broadcast(processor(0,myComm),reducesum); //    Broadcast sum on process 0 to all
other process
59: sun[]=reducesum;

```

```

60: if(withplot>5)
61: plot(sun,fill=1,wait=1);
62: cout << sun[].max << " " << sun[].min<< endl;
63: assert( 1.-1e-9 <= sun[].min && 1.+1e-9 >= sun[].max);
64: int nitermax=1000;
{
  Vh un=0;
  for(int iter=0;iter<nitermax;++iter)
  {
    real err=0,rerr=0;
    Vh un1=0;
    i=membershipKey;
    Thi=aTh[i];
    real[int] ui=Rih[i]*un[];                                     //  '
    real[int] bi = ui .* auntgv[i][];
    bi = auntgv[i][] ? bi : rhsi[i][];
    ui=au[i][];
    ui= aA[i] ^-1 * bi;      // Solve local linear system on group of process
    represented by color membershipKey
    bi = ui-au[i][];
    err += bi'*bi;          //  '
    au[i][]= ui;
    bi = Dih[i]*ui;      // Prolongation of current solution to obtain right hand
    un1[] += Rih[i]'*bi;  //  '
  }
65: reducesum=0; vsum=un1[];
66: mpiReduce(vsum, reducesum,processor(0,myComm),mpiSUM);      // MPI global operation
MPI_Reduce on global communicator
67: broadcast(processor(0,myComm),reducesum);      // Broadcast sum on process 0 to all
other process
68: un1[]=reducesum;
69: real residrela=0;
70: mpiReduce(err,residrela ,processor(0,myComm),mpiSUM);
71: broadcast(processor(0,myComm),residrela);
72: err=residrela; err= sqrt(err);
73: if(rank==0) cout << iter << " Err = " << err << endl;
74:   if(err<1e-5) break;
75:   un[]=un1[];
76:   if(withplot>2)
77:     plot(au,dim=3,wait=0,cmm=" iter "+iter,fill=1 );
78:   }
79: plot(un,wait=1,dim=3);
80: }

```

# 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  
( (R)  $x_i^j$ ,  $j=1,\text{dim}$  ), (I)  $Ref\phi_i^v$ ,  $i=1,\text{NbOfVertices}$  )
- Edges (I) NbOfEdges  
( @@Vertex<sub>i</sub><sup>1</sup>, @@Vertex<sub>i</sub><sup>2</sup>, (I)  $Ref\phi_i^e$ ,  $i=1,\text{NbOfEdges}$  )
- Triangles (I) NbOfTriangles  
( ( @@Vertex<sub>i</sub><sup>j</sup>,  $j=1,3$  ), (I)  $Ref\phi_i^t$ ,  $i=1,\text{NbOfTriangles}$  )
- Quadrilaterals (I) NbOfQuadrilaterals  
( ( @@Vertex<sub>i</sub><sup>j</sup>,  $j=1,4$  ), (I)  $Ref\phi_i^t$ ,  $i=1,\text{NbOfQuadrilaterals}$  )
- Geometry  
(C\*) FileNameOfGeometricSupport
  - VertexOnGeometricVertex  
(I) NbOfVertexOnGeometricVertex  
( @@Vertex<sub>i</sub>, @@Vertex<sub>i</sub><sup>geo</sup>,  $i=1,\text{NbOfVertexOnGeometricVertex}$  )
  - EdgeOnGeometricEdge  
(I) NbOfEdgeOnGeometricEdge  
( @@Edge<sub>i</sub>, @@Edge<sub>i</sub><sup>geo</sup>,  $i=1,\text{NbOfEdgeOnGeometricEdge}$  )
- CrackedEdges (I) NbOfCrackedEdges  
( @@Edge<sub>i</sub><sup>1</sup>, @@Edge<sub>i</sub><sup>2</sup>,  $i=1,\text{NbOfCrackedEdges}$  )

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

- MeshSupportOfVertices  
(C\*) FileNameOfMeshSupport
  - VertexOnSupportVertex  
(I) NbOfVertexOnSupportVertex  
( @@Vertex<sub>i</sub>, @@Vertex<sub>i</sub><sup>supp</sup>, i=1,NbOfVertexOnSupportVertex )
  - VertexOnSupportEdge  
(I) NbOfVertexOnSupportEdge  
( @@Vertex<sub>i</sub>, @@Edge<sub>i</sub><sup>supp</sup>, (R)  $u_i^{supp}$ , i=1,NbOfVertexOnSupportEdge )
  - VertexOnSupportTriangle  
(I) NbOfVertexOnSupportTriangle  
( @@Vertex<sub>i</sub>, @@Tria<sub>i</sub><sup>supp</sup>, (R)  $u_i^{supp}$ , (R)  $v_i^{supp}$ ,  
i=1 ,NbOfVertexOnSupportTriangle )
  - VertexOnSupportQuadrilaterals  
(I) NbOfVertexOnSupportQuadrilaterals  
( @@Vertex<sub>i</sub>, @@Quad<sub>i</sub><sup>supp</sup>, (R)  $u_i^{supp}$ , (R)  $v_i^{supp}$ ,  
i=1 ,NbOfVertexOnSupportQuadrilaterals )

## 12.2 bb File type for Store Solutions

The file is formatted such that:

2 nbso1 nbv 2  
 $((U_{ij}, \forall i \in \{1, \dots, \text{nbso1}\}), \forall j \in \{1, \dots, \text{nbv}\})$   
 where

- nbso1 is a integer equal to the number of solutions.
- nbv is a integer equal to the number of vertex .
- $U_{ij}$  is a real equal the value of the  $i$  solution at vertex  $j$  on the associated mesh background if read file, generated if write file.

## 12.3 BB File Type for Store Solutions

The file is formatted such that:

2 n typesol<sup>1</sup> ... typesol<sup>n</sup> nbv 2  
 $((U_{ij}^k, \forall i \in \{1, \dots, \text{typesol}^k\}), \forall k \in \{1, \dots, n\}) \forall j \in \{1, \dots, \text{nbv}\})$   
 where

- n is a integer equal to the number of solutions
- typesol<sup>k</sup>, type of the solution number  $k$ , is
  - typesol<sup>k</sup> = 1 the solution  $k$  is scalar (1 value per vertex)

- $\text{typesol}^k = 2$  the solution  $k$  is vectorial (2 values per unknown)
- $\text{typesol}^k = 3$  the solution  $k$  is a  $2 \times 2$  symmetric matrix (3 values per vertex)
- $\text{typesol}^k = 4$  the solution  $k$  is a  $2 \times 2$  matrix (4 values per vertex)
- $\text{nbv}$  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

$\text{nbv}$  1

$h_i \quad \forall i \in \{1, \dots, \text{nbv}\}$

where

- $\text{nbv}$  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

$\text{nbv}$  3

$a11_i, a21_i, a22_i \quad \forall i \in \{1, \dots, \text{nbv}\}$

where

- $\text{nbv}$  is a integer equal to the number of vertices,
- $a11_i, a21_i, a22_i$  is metric  $\mathcal{M}_i = \begin{pmatrix} a11_i & a21_i \\ a21_i & a22_i \end{pmatrix}$  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:

$\text{nbt}$  is the number of triangles.

$\text{nbv}$  is the number of vertices.

$\text{nu}(1:3, 1:\text{nbt})$  is an integer array giving the three vertex numbers counterclockwise for each triangle.

$\text{c}(1:2, \text{nbv})$  is a real array giving the two coordinates of each vertex.

$\text{refs}(\text{nbv})$  is an integer array giving the reference numbers of the vertices.

$\text{reft}(\text{nbv})$  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  $f$  taking value in  $\mathbb{R}^N$ ,  $N = 1, 2, \dots$ , we define the finite element approximation  $\Pi_h f$  of  $f$ . Let us denote the number of the degrees of freedom of the finite element by  $NbDoF$ . Then the  $i$ -th base  $\omega_i^K$  ( $i = 0, \dots, NbDoF - 1$ ) of the finite element space has the  $j$ -th component  $\omega_{ij}^K$  for  $j = 0, \dots, N - 1$ .

The operator  $\Pi_h$  is called the interpolator of the finite element. We have the identity  $\omega_i^K = \Pi_h \omega_i^K$ . Formally, the interpolator  $\Pi_h$  is constructed by the following formula:

$$\Pi_h f = \sum_{k=0}^{kPi-1} \alpha_k f_{j_k}(P_{p_k}) \omega_{i_k}^K \quad (13.1)$$

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  $\alpha_k$  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:

$$RT0_h = \{ \mathbf{v} \in H(div) / \forall K \in \mathcal{T}_h \quad \mathbf{v}|_K(x, y) = \begin{bmatrix} \alpha_K \\ \beta_K \end{bmatrix} + \gamma_K \begin{bmatrix} x \\ y \end{bmatrix} \} \quad (13.2)$$

The degree of freedom are the flux throw an edge  $e$  of the mesh, where the flux of the function  $\mathbf{f} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  is  $\int_e \mathbf{f} \cdot \mathbf{n}_e$ ,  $\mathbf{n}_e$  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:

$$\omega_0^K = \frac{sgn(i_b - i_c)}{2|T|}(x - a), \quad \omega_1^K = \frac{sgn(i_c - i_a)}{2|T|}(x - b), \quad \omega_2^K = \frac{sgn(i_a - i_b)}{2|T|}(x - c), \quad (13.3)$$

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

So we have  $N = 2$ ,  $kPi = 6$ ;  $npPi = 3$ ; and:

- $P_p = \left\{ \frac{\mathbf{b}+\mathbf{c}}{2}, \frac{\mathbf{a}+\mathbf{c}}{2}, \frac{\mathbf{b}+\mathbf{a}}{2} \right\}$
- $\alpha_0 = -\mathbf{e}_2^0, \alpha_1 = \mathbf{e}_1^0, \alpha_2 = -\mathbf{e}_2^1, \alpha_3 = \mathbf{e}_1^1, \alpha_4 = -\mathbf{e}_2^2, \alpha_5 = \mathbf{e}_1^2$  (effectively, the vector  $(-\mathbf{e}_2^m, \mathbf{e}_1^m)$  is orthogonal to the edge  $\mathbf{e}^m = (e_1^m, e_2^m)$  with a length equal to the side of the edge or equal to  $\int_{e^m} 1$ ).
- $i_k = \{0, 0, 1, 1, 2, 2\}$ ,
- $p_k = \{0, 0, 1, 1, 2, 2\}$ ,  $j_k = \{0, 1, 0, 1, 0, 1\}$ .

## 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  $\alpha_k$  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  $\hat{K}$ 

        for (int p=0, kk=0; p<3; p++) {
            P_Pi_h[p]=Pt[p];
            for (int j=0; j<2; j++)
                pij_alpha[kk++] = IPJ(p, p, j); } // definition of  $i_k, p_k, j_k$  in (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 finally 6 for element support.

The function to defined the function  $\omega_i^K$ , this function return the value of all the basics function or this derivatives in array val, computed at point PHat on the reference triangle corresponding to point R2  $P=K(\text{Phat})$ ; on the current triangle K.

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

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

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

$k$  is the type of computed value  $f(P), dx(f)(P), dy(f)(P), \dots i \in [0, \text{last\_operatortype}[$ . Remark for optimization, this value is computed only if  $\text{whatd}[k]$  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));
    RN_ f1(val('.',1,0));
    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])
{
    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  $\alpha_k$ :

```

void TypeOfFE_RT::Pi_h_alpha(const baseFEElement & K,KN_<double> & v) const
{
    const Triangle & T(K.T);

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

```

Now , we just need to add a new key work in FreeFem++, 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

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

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

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

```
void init_static_FE() // correct Problem of static library link with new make file
{ // 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 1.7 2008/11/24 19:10:15 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 \
ConjuguedGradientNL.cpp DOperator.hpp Drawing.cpp Element_P2h.cpp \
Element_P3.cpp Element_RT.cpp fem3.hpp fem.cpp fem.hpp FESpace.cpp \
FESpace.hpp FESpace-v0.cpp FQuadTree.cpp FQuadTree.hpp gibbs.cpp \
glutdraw.cpp gmres.hpp MatriceCreuse.hpp MatriceCreuse_tpl.hpp \
MeshPoint.hpp mortar.cpp mshptg.cpp QuadratureFormular.cpp \
QuadratureFormular.hpp RefCounter.hpp RNM.hpp RNM_opc.hpp RNM_op.hpp \
RNM_tpl.hpp FE_ADD.cpp
```

and do in the `freefem++` root directory

```
autoreconf
./reconfigure
make
```

For codewarrior compilation add the file in the project and remove the flag in panel PPC linker FreeFEM++ Setting Dead-strip Static Initialization Code Flag.



# Appendix A

## Table of Notations

Here mathematical expressions and corresponding FreeFem++ commands are noted.

### A.1 Generalities

$\delta_{ij}$  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 ( $a \in \mathbb{N} \Leftrightarrow \text{int } a$ ); “int” means *long integer* inside FreeFem++

$\mathbb{R}$  the set of real numbers ( $a \in \mathbb{R} \Leftrightarrow \text{real } a$ ); *double* inside FreeFem++

$\mathbb{C}$  the set of complex numbers ( $a \in \mathbb{C} \Leftrightarrow \text{complex } a$ ); *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  $E$ .

$\{x \in E \mid P\}$  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  $\{x \in E \mid x \notin A\}$

$E + F$   $E \cup F$  with  $E \cap F = \emptyset$

$E \times F$  the cartesian product of  $E$  and  $F$

$E^n$  the  $n$ -th power of  $E$  ( $E^2 = E \times E$ ,  $E^n = E \times E^{n-1}$ )

$f : E \rightarrow F$  the mapping from  $E$  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  $f : F \rightarrow G$  and  $g : E \rightarrow F$ ,  $E \ni x \mapsto (f \circ g)(x) = f(g(x)) \in G$  (see Section 4.6)

$f|_A$  the restriction of  $f : E \rightarrow F$  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  $\{x \in \mathbb{R} \mid a \leq x \leq b\}$

$]a, b]$  is the interval  $\{x \in \mathbb{R} \mid a < x \leq b\}$

$[a, b[$  is the interval  $\{x \in \mathbb{R} \mid a \leq x < b\}$

$]a, b[$  is the interval  $\{x \in \mathbb{R} \mid a < x < b\}$

### A.4 Differential Calculus

$\partial f / \partial x$  the partial derivative of  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  with respect to  $x$  ( $\text{dx}(f)$ )

$\nabla f$  the gradient of  $f : \Omega \rightarrow \mathbb{R}$ , i.e.,  $\nabla f = (\partial f / \partial x, \partial f / \partial y)$

**div** $f$  or  $\nabla \cdot f$  the divergence of  $f : \Omega \rightarrow \mathbb{R}^d$ , i.e.,  $\text{div } f = \partial f_1 / \partial x + \partial f_2 / \partial y$

$\Delta f$  the Laplacian of  $f : \Omega \rightarrow \mathbb{R}$ , i.e.,  $\Delta f = \partial^2 f / \partial x^2 + \partial^2 f / \partial y^2$

## A.5 Meshes

$\Omega$  usually denotes a domain on which PDE is defined

$\Gamma$  denotes the boundary of  $\Omega$ , i.e.,  $\Gamma = \partial\Omega$  (keyword **border**, see Section 5.1.2)

$\mathcal{T}_h$  the triangulation of  $\Omega$ , 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”)

$\Omega_h$  denotes the approximated domain  $\Omega_h = \cup_{k=1}^{n_t} T_k$  of  $\Omega$ . If  $\Omega$  is polygonal domain, then it will be  $\Omega = \Omega_h$

$\Gamma_h$  the boundary of  $\Omega_h$

$n_v$  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  $(\text{Th}[\mathbf{k}-1][\mathbf{j}-1].\mathbf{x}, \text{Th}[\mathbf{k}-1][\mathbf{j}-1].\mathbf{y}))$

$I_\Omega$  the set  $\{i \in \mathbb{N} \mid q^i \notin \Gamma_h\}$

## A.6 Finite Element Spaces

$L^2(\Omega)$  the set  $\left\{ w(x, y) \mid \int_{\Omega} |w(x, y)|^2 dx dy < \infty \right\}$

$$\text{norm: } \|w\|_{0,\Omega} = \left( \int_{\Omega} |w(x, y)|^2 dx dy \right)^{1/2}$$

$$\text{scalar product: } (v, w) = \int_{\Omega} vw$$

$H^1(\Omega)$  the set  $\left\{ w \in L^2(\Omega) \mid \int_{\Omega} (|\partial w / \partial x|^2 + |\partial w / \partial y|^2) dx dy < \infty \right\}$

$$\text{norm: } \|w\|_{1,\Omega} = \left( \|w\|_{0,\Omega}^2 + \|\nabla w\|_{0,\Omega}^2 \right)^{1/2}$$

$H^m(\Omega)$  the set  $\left\{ w \in L^2(\Omega) \mid \int_{\Omega} \frac{\partial^{|\alpha|} w}{\partial x^{\alpha_1} \partial y^{\alpha_2}} \in L^2(\Omega) \quad \forall \alpha = (\alpha_1, \alpha_2) \in \mathbb{N}^2, |\alpha| = \alpha_1 + \alpha_2 \right\}$

$$\text{scalar product: } (v, w)_{1,\Omega} = \sum_{|\alpha| \leq m} \int_{\Omega} D^{\alpha} v D^{\alpha} w$$

$H_0^1(\Omega)$  the set  $\{w \in H^1(\Omega) \mid u = 0 \text{ on } \Gamma\}$

$L^2(\Omega)^2$  denotes  $L^2(\Omega) \times L^2(\Omega)$ , and also  $H^1(\Omega)^2 = H^1(\Omega) \times H^1(\Omega)$

$V_h$  denotes the finite element space created by “`fespace Vh(Th,*)`” in FreeFem++ (see Section 6 for “\*”)

$\Pi_h f$  the projection of the function  $f$  into  $V_h$  (“`func f=x^2*y^3; Vh v = f;`” means  $v = \Pi_h f$ )

$\{v\}$  for FE-function  $v$  in  $V_h$  means the column vector  $(v_1, \dots, v_M)^T$  if  $v = v_1\phi_1 + \dots + v_M\phi_M$ , 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 '`' ;

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

```

// transpose



## 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>... )

- intalldges, 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 fonctionnalites in FreeFem++ under the three environnements Linux, Windows and MacOS X 10.3 or newer. It is a good 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 environment 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 automatically added.*
- *Under Windows, the file `xxx.dll` to be loaded must be in the `loadLibrary` search directory which includes the directory of the application,*

**The compilation of your module:** the script `ff-c++` compiles and makes the link with FreeFem++, but be careful, the script has no way to know 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 [12].

```

// A class build the link with FreeFem++
// generally this class are already in AFunction.hpp
// but unfortunately, 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, use the ff-c++ script :

```
Brochet% ff-c++ 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, v1.1062010/02/0818 : 15 : 34hechtExp
 4 :
 5 : load "myfunction"
    load: 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 trymake.

The 1D dfft (fast discret fourier transform) for a simple array  $f$  of size  $n$  is defined by the following formula

$$\text{dfft}(f, \varepsilon)_k = \sum_{j=0}^{n-1} f_j e^{\varepsilon 2\pi i k j / n}$$

The 2D DFT for an array of size  $N = n \times m$  is

$$\text{dfft}(f, m, \varepsilon)_{k+nl} = \sum_{j'=0}^{m-1} \sum_{j=0}^{n-1} f_{i+nj} e^{\varepsilon 2\pi i (kj/n + lj'/m)}$$

Remark: the value  $n$  is given by  $\text{size}(f)/m$ , and the numbering is row-major order.

So the classical discrete DFT is  $\hat{f} = \text{dfft}(f, -1)/\sqrt{n}$  and the reverse dFT  $f = \text{dfft}(\hat{f}, 1)/\sqrt{n}$

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  $k \leq n/2$  else  $\tilde{k} = k - n$  and  $\tilde{l} = l$  if  $l \leq m/2$  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**

```
% ff-c++ 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 UMFPAK
1 :                                     //      Example of dynamic function load
2 :                                     //      -----
3 :                                     //      Id : freefem + +doc.tex, v1.1062010/02/0818 : 15 : 34hechtExp
4 :                                     //      Discret Fast Fourier Transform
5 :                                     //      -----
6 : load "dfft" load: 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 associated edp file is `examples++-load/convect_dervieux.edp`

```

// ----- Implementation of P1-P0 FVM-FEM -----
// ----- Id : freefem + +doc.tex,v1.1062010/02/0818 : 15 : 34hechtExp -----
// compile and link with ff-c++ 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];
        expTh= to<pmesh>(args[1]);
        expc = CastTo<double>(args[2]);
        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]);           //    first 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) {
        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 (fvmPlP0(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(Ty
// 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  $\lambda_i^K$  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  $P_1^2 + 3$  normals bubbles edges function ( $P_2$ )
// 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 : MSP2BRP0.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 baseFEelement & 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 baseFEelement & 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;
  plot([a1,a2], wait=1); // a bascis functions

  [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 beginnig 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

Warning the sparse solver interface as been completely rewrite in version 3.2 , so the section is obsolete, the example in are correct/

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 notation.

```
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 information, to solve 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(DCL_ARG_SPARSE_SOLVER(double,A))
{
    if(verbosity>9)
        cout << " BuildSolverSuperLU<double>" << endl;
    return new SolveSuperLU<double>(*A,ds.strategy,ds.tgv,ds.epsilon,ds.tol_pivot,ds.tol_pivot_sym,ds)
}
```

```
MatriceMorse<Complex>::VirtualSolver *
BuildSolverSuperLU(DCL_ARG_SPARSE_SOLVER(Complex,A))
{
    if(verbosity>9)
        cout << " BuildSolverSuperLU<Complex>" << endl;
    return new SolveSuperLU<Complex>(*A,ds.strategy,ds.tgv,ds.epsilon,ds.tol_pivot,ds.tol_pivot_sym,ds)
}
```

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,defaulttoSuperLUin 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 superlu_3.0.ta
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 :

```
ff-c++ SuperLU.cpp -L$HOME/work/LinearSolver/SuperLU_3.0/ -lsuperlu_3.0
```

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)defaultttoGMRES();
  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>
- Jacques Morice <morice@ann.jussieu.fr>

All its programs except files the coming from COOOL software (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 INACCURATE 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. SORESENSEN, 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. Outils et algorithmes pour la méthode des éléments finis, HdR, Université Pierre et Marie Curie, France, 1992
- [9] HECHT, F: The mesh adapting software: bamg. INRIA report 1998.
- [10] Library Modulef, INRIA, <http://www.inria-rocq/modulef>
- [11] 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.
- [12] F. HECHT. C++ Tools to construct our user-level language. Vol 36, N°, 2002 pp 809-836, *Modél. math et Anal Numér.*
- [13] J.L. LIONS, O. PIRONNEAU: Parallel Algorithms for boundary value problems, *Note CRAS*. Dec 1998. Also : Superpositions for composite domains (to appear)
- [14] B. LUCQUIN, O. PIRONNEAU: *Introduction to Scientific Computing* Wiley 1998.
- [15] I. DANAILA, F. HECHT, AND O. PIRONNEAU. *Simulation numérique en C++*. Dunod, Paris, 2003.

- [16] J. NEČAS AND L/ HLAVÁČEK, Mathematical theory of elastic and elasto-plastic bodies: An introduction, Elsevier, 1981.
- [17] K. OHTSUKA, O. PIRONNEAU AND F. HECHT: Theoretical and Numerical analysis of energy release rate in 2D fracture, *INFORMATION* **3** (2000), 303–315.
- [18] F. PREPARATA, M. SHAMOS *Computational Geometry* Springer series in Computer sciences, 1984.
- [19] 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
- [20] ROBERTS, J.E. AND THOMAS J.-M: Mixed and Hybrid Methods, Handbook of Numerical Analysis, Vol.II, North-Holland, 1993
- [21] J.L. STEGER: The Chimera method of flow simulation, Workshop on applied CFD, Univ of Tennessee Space Institute, August 1991.
- [22] TABATA, M: Numerical solutions of partial differential equations II (in Japanese), Iwanami Applied Math., 1994
- [23] THOMASSET, F: Implementation of finite element methods of Navier-Stokes Equations, Springer-Verlag, 1981
- [24] N. WIRTH: *Algorithms + Data Structures = Programs*, Prentice Hall, 1976
- [25] Bison documentation
- [26] Bjarne Stroustrup: The C++ , programming language, Third edition, Addison-Wesley 1997.
- [27] COOOL: a package of tools for writing optimization code and solving optimization problems, <http://cool.mines.edu>
- [28] 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).
- [29] 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.
- [30] R. GLOWINSKI, *Numerical Methods for Nonlinear Variational Problems*, Springer-Verlag, New York, NY, 1984.
- [31] 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.
- [32] 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).



- [33] KAZUFUMI ITO, AND KARL KUNISCH, Semi smooth newton methods for variational inequalities of the first kind , M2AN, vol 37, N°, 2003, pp 41-62.
- [34] Ogden, RW; Non-Linear Elastic Deformations, Dover, 1984.
- [35] Raviart, P; Thomas J; Introduction à l'analyse numérique des équations aux dérivées partielles, Masson, 1983.
- [36] Hang Si, *TetGen Users' Guide: A quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator* // <http://tetgen.berlios.de>
- [37] Shewchuk J. R., *Tetrahedral Mesh Generation by Delaunay Refinement* Proceeding of the Fourteenth Annual Symposium on Computational Geometry (Minneapolis, Minnesota), pp 86–95, 1998.
- [38] 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>
- [39] Wilmott, Paul and Howison, Sam and Dewynne, Jeff : A student introduction to mathematical finance, Cambridge University Press (1995).

# Index

<<, 78  
     matrix, 74  
 >>, 78  
 .\*, 72  
   \*, 148  
 ./, 72  
 =, 138  
 ?:, 58  
 [], 16, 136, 216  
 →, 55  
 ', 72  
 3d  
     beam-3d.edp, 183  
     periodic, 172  
  
 time dependent, 28  
  
 accuracy, 19  
 acos, 61  
 acosh, 61  
 adaptmesh, 95, 98  
     abserror=, 98  
     cutoff=, 99  
     err=, 98  
     errg=, 98  
     hmax=, 98  
     hmin=, 98  
     inquire=, 99  
     isMetric=, 99  
     iso=, 98  
     keepbackvertices=, 99  
     maxsubdiv=, 99  
     metric=, 99  
     nbjacoby=, 98  
     nbsmooth=, 98  
     nbvx=, 98  
     nomeshgeneration=, 99  
     omega=, 98  
     periodic=, 99  
     powerin=, 99  
     ratio=, 98  
     rescaling=, 99  
     splitin2, 99  
     splitpbedge=, 99  
     uniform, 99  
     verbosity=, 99  
 alphanumeric, 55  
 append, 78  
 area, 58  
 area coordinate, 144  
 argument, 60  
 array, 56, 64, 76, 147  
     .i1, 67  
     .i2, 67  
     .linfty, 67  
     .max, 67  
     .min, 67  
     .sum, 67  
     ::, 65  
     ?::, 65  
     = + - \* / .\* ./ += -= /= \*= , 67  
 column, 69  
 dot product, 68  
 FE function, 76  
 fespace, 129, 130  
 im, 65, 66  
 line, 69  
 max, 65  
 mesh, 234  
 min, 65, 76  
 quantile, 68  
 re, 65, 66  
 renumbering, 69  
 resize, 65  
 sort, 65  
 sum, 65  
 varf, 149  
 asin, 61  
 asinh, 61

- assert(), 58
- atan, 61
- atanh, 61
- axisymmetric, 28
- backward Euler method, 194
- bang, 86
- barycentric coordinates, 13
- bessel, 62
- BFGS, 161
- block matrix, 71
- bool, 56
- border, 84
- boundary condition, 148
- break, 77
- broadcast, 234
- bubble, 133
- buildmesh
  - fixeborder, 85
  - fixeborder=1, 172
  - nbvx=, 85
- catch, 79
- Cauchy, 53
- ceil, 61
- CG, 141
- change, 107
- Characteristics-Galerkin, 33
- checkmovemesh, 93
- Cholesky, 141, 160
- cin, 58, 78
- column, 69
- compatibility condition, 140
- compiler, 55
- Complex, 56
- complex, 46, 60, 72
- Complex geometry,, 30
- concatenation, 97
- cone, 117
- connectivity, 152
- continue, 77
- convect, 34, 205, 207
- cos, 61
- cosh, 61
- cout, 58, 78
- Crout, 141
- cube, 116
- de Moivre's formula, 60
- default, 78
- degree of freedom, 13
- DFFT, 294
- diag, 73, 210
- diagonal matrix, 73
- Dirichlet, 19, 25, 53, 139
- discontinuous functions, 220
- Discontinuous-Galerkin, 33
- displacement vector, 180
- divide
  - term to term, 72
- domain decomposition, 213, 214
- dot product, 72, 76
- dumptable, 58
- Edge03d, **129**
- EigenValue, 191
  - ivalue=, 190
  - maxit=, 190
  - ncv=, 190
  - nev=, 190
  - rawvector=, 190
  - sigma=, 190
  - sym=, 190
  - tol=, 190
  - value=, 190
  - vector=, 190
- eigenvalue problems, 26
- elements, 13, 14
- emptymesh, 92
- endl, 78
- erf, 62
- erfc, 62
- exception, 79
- exec, 58, 157, 158
- exp, 61
- expression optimization, 150
- external C++ function, 37
- factorize=, 160
- false, 56, 58
- FE function
  - [], 136**
  - complex, 46, 64, 72
  - n, 136**
  - value, 136

- FE space, 129, 130
- FE-function, 63, 129, 130
- FESpace
  - (int ,int ), 152
  - ndof, 152
  - nt, 152
- fespace, 127
  - Edge03d, 129
  - P0, 128
  - P1, 128
  - P13d, 128
  - P1b, 128
  - P1b3d, 128
  - P1dc, 128
  - P1nc, 129
  - P2, 128
  - P2dc, 128
  - periodic=, 141, 170
  - RT0, 128
- ffmedit, 157
- FFT, 294
- file
  - am, 267, 268
  - am\_fmt, 87, 267, 268
  - amdba, 267, 268
  - bamg, 87, 265
  - data base, 265
  - ftq, 269
  - mesh, 87
  - msh, 268
  - nopo, 87
- finite element space, 127
- Finite Volume Methods, 36
- fixed, 78
- floor, 61
- fluid, 203
- for, 77
- formulas, 63
- Fourier, 28
- func, 57
- function
  - tables, 91
- functions, 61
- gamma, 62
- geometry input , 24
- GMRES, 141
- gnuplot, 157
- graphics packages, 19
- hat function, 13
- Helmholtz, 46
- hTriangle, 57, 177
- ifstream, 78
- ill posed problems, 26
- im, 65, 66
- imag, 60
- include, 7, 207
- includepath, 7
- init, 36
- init=, 141
- initial condition, 53
- inside=, 151
- int1d, 141
- int2d, 141
- intalldges, 141, 177
- interpolate, 150
  - inside=, 151
  - op=, 151
  - t=, 151
- interpolation, 138
- Irecv, 233
- Isend, 233
- isotropic, 181
- jump, 177
- label, 57, 83, 84
- label on the boundaries, 25
- label=, 100
- lagrangian, 94
- Laplace operator, 19
- lenEdge, 57, 177
- level line, 35
- line, 69
- linearCG
  - eps=, 159
  - nbiter=, 159
  - precon=, 159
  - veps=, 159
- LinearGMRES
  - eps=, 159
  - nbiter=, 159
  - precon=, 159

- veps=, 159
- load, 7
- loadpath, 7
- log, 61
- log10, 61
- LU, 141
- macro, 35, 225
- mass lumping, 37
- matrix, 15, 57, 160
  - =, 207
  - array, 70
  - block, 71
  - complex, 73
  - constant, 71
  - diag, 73, 210
  - factorize=, 189
  - interpolate, 150
  - renumbering, 70, 72
  - resize, 72, 73
  - set, 71
  - solver=, 207
  - stiffness matrix, 15
  - varf, 71, 148
    - eps=, 149
    - precon=, 149
    - solver=, 148
    - solver=factorize, 148
    - tg=, 149
    - tolpivot =, 149
- max, 65
- maximum, 58
- medit, 157, 158
- membrane, 19
- mesh, 57
  - (), 88
  - +, 107
  - [], 88
  - 3point bending, 106
  - beam, 101
  - Bezier curve, 103
  - Cardioid, 102
  - Cassini Egg, 103
  - change, 107
  - connectivity, 88
  - NACA0012, 102
  - regular, 95
  - Section of Engine, 104
  - Smiling face, 105
  - U-shape channel, 104
  - uniform, 99, 100
  - V-shape cut, 105
- mesh adaptation, 37
- min, 65, 76
- minimum, 58
- mixed, 28
- mixed Dirichlet Neumann, 19
- modulus, 60
- movemesh, 93
- mpiAllgather, 233
- mpiAllReduce, 233
- mpiAlltoall, 233
- mpiAnySource, 231
- mpiBarrier, 232
- mpiBXOR, 231
- mpiCom, 231
- mpiCommWorld, 231
- mpiGather, 233
- mpiGroup, 231
- mpiLAND, 231
- mpiLOR, 231
- mpiLXOR, 231
- mpiMAX, 231
- mpiMIN, 231
- mpiPROD, 231
- mpiRank, 232
- mpirank, 231
- mpiReduce, 233
- mpiReduceScatter, 233
- mpiRequest, 231
- mpiScatter, 233
- mpiSize, 232
- mpisize, 231
- mpiSUM, 231
- mpiUndefined, 231
- mpiWait, 232
- mpiWtick, 232
- mpiWtime, 232
- multi-physics system, 30
- multiple meshes, 30
- N, 57, 143
- n, 136
- Navier-Stokes, 205, 207

- ndof, 152
- ndofK, 152
- Neumann, 53, 142, 143
- Newton, 161, 188
- NLCG, 161
  - eps=, 159
  - nbiter=, 159
  - veps=, 159
- nodes, 13
- non-homogeneous Dirichlet, 19
- nonlinear, 30
- nonlinear problem, 28
- normal, 35, 143
- noshowbase, 78
- noshowpos, 78
- nt, 152
- nTonEdge, 35, 58
- nuEdge, 57
- number of degree of freedom, 152
- number of element, 152
- nuTriangle, 57
- ofstream, 78
  - append, 78
- on, 142
  - intersection, 206
  - scalar, 142
- optimize=, 150
- outer product, 69, 72
- P, 57
- P0, **128**
- P1, **128**
- P1b, **128**
- P1dc, **128**
- P1nc, **129**
- P2, **128**
- P2dc, **128**
- parabolic, 28
- periodic, 127, 141, 170
  - 3d, 172
- pi, 58
- plot
  - aspectratio =, 154
  - nbiso =, 154
  - bb=, 154
  - border, 85
  - boundary =, 154
  - bw=, 154
  - cmm=, 154
  - coef=, 154
  - cut, 154
  - dim=, 154
  - grey=, 154
  - hsv=, 154
  - mesh, 85
  - nbarrow=, 154
  - ps=, 154
  - value=, 154
  - varrow=, 154
  - viso=, 154
- point
  - region, 88
  - triange, 88
- pow, 61
- precision, 78
- precon=, 141, 149, 208
- problem, 36, 57, 139
  - eps=, 141
  - init=, 141
  - precon=, 141
  - solver=, 141
  - strategy =, 141, 149
  - tgV=, 141
  - tolpivot =, 141
  - tolpivotsym =, 141, 149
- processor, 232, 234
- processorblock, 232
- product
  - Hermitian dot, 72
  - dot, 72, 76
  - outer, 72
  - term to term, 72
- qforder=, 207
- quadrature: qf5pT, 145
- quadrature: qfV5, 146
- quadrature: default, 146
- quadrature: qf1pE, 144
- quadrature: qf1pElump, 144
- quadrature: qf1pT, 145
- quadrature: qf1pTlump, 145
- quadrature: qf2pE, 144
- quadrature: qf2pT, 145

quadrature:qf2pT4P1, 145  
 quadrature:qf3pE, 144  
 quadrature:qf7pT, 145  
 quadrature:qfe=, 145, 146  
 quadrature:qforder=, 145, 146  
 quadrature:qft=, 145, 146  
 quadrature:qfV, 145  
 quadrature:qfV1, 146  
 quadrature:qfV1lump, 146  
 quadrature:qfV2, 146  
 quantile, 69

radiation, 30  
 rand, 62  
 randinit, 62  
 randint31, 62  
 randint32, 62  
 random, 62  
 re, 65, 66  
 read files, 87  
 readmesh, 86, 108  
 real, 56, 60  
 region, 57, 88, 220  
 region indicator, 30  
 renumbering, 69  
 resize, 65, 73  
 Reusable matrices, 205  
 rint, 61  
 Robin, 28, 139, 142, 143  
 RT0, **128**

savemesh, 86, 108  
 schwarz, 234  
 scientific, 78  
 sec:Plot, 153  
 Secv, 233  
 Send, 233  
 set, 36  
     matrix, 71  
 showbase, 78  
 showpos, 78  
 shurr, 213, 214  
 sin, 61  
 singularity, 97  
 sinh, 61  
 solve, 57, 139  
     eps=, 141

init=, 141  
 linear system, 72  
 precon=, 141  
 solver=, 141  
 strategy=, 141, 149  
 tgv=, 15, 141  
 tolpivot=, 141  
 tolpivotsym=, 141, 149  
 solver=, 160  
     CG, 97, 141  
     Cholesky, 141  
     Crout, 141  
     GMRES, 141  
     LU, 141  
     sparsesolver, 141  
     UMFPACK, 141

sort, 65  
 sparsesolver, 141  
 split=, 100  
 square, 177  
     flags=, 84  
     label=, 84  
     region=, 84  
 Stokes, 203  
 stokes, 201  
 stop test, 141  
     absolue, 207  
 strain tensor, 181  
 streamlines, 204  
 stress tensor, 181  
 string, 56  
 subdomains, 220  
 sum, 65

tan, 61  
 Taylor-Hood, 206  
 tetgconvexhull, 112  
 tetgtransfo, 112  
 transpose, 72, 76, 284  
 triangle  
     [], 88  
     area, 88  
     label, 88  
     region, 88  
 triangulate, 91  
 triangulation files, as well as read and write, 24  
 true, 56, 58

- trunc, 100
  - label=, 100
  - split=, 100
- try, 79
- tutorial
  - LaplaceRT.edp, 176
  - adapt.edp, 97
  - adaptindicatorP2.edp, 177
  - AdaptResidualErrorIndicator.edp, 178
  - aTutorial.edp, 164
  - beam.edp, 182
  - BlackSchol.edp, 200
  - convect.edp, 199
  - fluidStruct.edp, 216
  - freeboundary.edp, 223
  - movemesh.edp, 94
  - NSUzawaCahouetChabart.edp, 207
  - periodic.edp, 170
  - periodic4.edp, 170
  - periodic4bis.edp, 172
  - readmesh.edp, 87
  - Schwarz-gc.edp, 215
  - Schwarz-no-overlap.edp, 213
  - Schwarz-overlap.edp, 211
  - StokesUzawa.edp, 206
- tutorial
  - VI.edp, 209
- type of finite element, 127
- UMFPACK, 141
- upwinding, 33
- varf, 15, 57, 141, 147, 207
  - array, 148
  - matrix, 148
  - optimize=, 150
- variable, 55
- variational formulation, 20
- veps=, 161
- verbosity, 3, 7
- vertex
  - label, 88
  - x, 88
  - y, 88
- viso, 35
- volume, 58
- weak form, 20
- while, 77
- write files, 87
- x, 57
- y, 57
- z, 57





## 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, Российской академии наук .