Freefem++


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

F. Hecht

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


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

Frédéric Hecht
mailto:frederic.hecht@upmc.fr
https://www.ljll.math.upmc.fr/~hecht

In collaboration with:

- Syl an Auliac, mailto:auliac@ann.jussieu.fr, http://www.ann.jussieu.fr/auliac is PHD student at LJLL, he do all the new optimization interface with nlopt, ipopt, cmaes, ...

- 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. Jacques Morice is a Post-Doct at LJLL. His doing is Thesis in University of Bordeaux I on fast multipole method (FMM). In this version, he do all three dimensions 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.

- Pierre Jolivet, mailto:pierre.jolivet@enseeiht.fr, http://jolivet.perso.enseeiht.fr/ is a CNRS researcher, he did the MPI interface with PETSc, HPDDM...

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-CIST-002-01.

1Laboratoire Jacques-Louis Lions, Université Pierre et Marie Curie, Paris
2Projet Alpines, Inria Rocquencourt.
# Contents

1 Introduction

1.1 Installation

1.1.1 For everyone:

1.1.2 For the pros: Installation from sources

1.2 How to use FreeFem++

1.3 Environment variables, and the init file

1.4 History

2 Getting Started

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

2.0.2 Some Features of FreeFem++

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

3 Learning by Examples

3.1 Membranes

3.2 Heat Exchanger

3.3 Acoustics

3.4 Thermal Conduction

3.4.1 Axisymmetry: 3D Rod with circular section

3.4.2 A Nonlinear Problem: Radiation

3.5 Irrotational Fan Blade Flow and Thermal effects

3.5.1 Heat Convection around the airfoil

3.6 Pure Convection: The Rotating Hill

3.7 The System of elasticity

3.8 The System of Stokes for Fluids

3.9 A Projection Algorithm for the Navier-Stokes equations

3.10 Newton Method for the Steady Navier-Stokes equations

3.11 A Large Fluid Problem

3.12 An Example with Complex Numbers

3.13 Optimal Control

3.14 A Flow with Shocks

3.15 Classification of the equations

3.16 Optimize Time depend schema for Heat equation

3.17 Tutorial to write a transient Stokes solver in matrix form
5.11 Meshing examples
  5.11.1 Build a 3d mesh of a cube with a balloon
5.12 The output solution formats .sol and .solb
5.13 medit
5.14 Mshmet
5.15 FreeYams
5.16 mmg3d
5.17 A first 3d isotope mesh adaptation process
5.18 Build a 2d mesh from a isoline

6 Finite Elements
  6.1 Use of “fespace” in 2d
  6.2 Use of fespace in 3d
  6.3 Lagrangian Finite Elements
    6.3.1 P0-element
    6.3.2 P1-element
    6.3.3 P2-element
  6.4 P1 Nonconforming Element
  6.5 Other FE-space
  6.6 Vector valued FE-function
    6.6.1 Raviart-Thomas element
  6.7 A Fast Finite Element Interpolator
  6.8 Keywords: Problem and Solve
    6.8.1 Weak form and Boundary Condition
  6.9 Parameters affecting solve and problem
  6.10 Problem definition
  6.11 Numerical Integration
  6.12 Variational Form, Sparse Matrix, PDE Data Vector
  6.13 Interpolation matrix
  6.14 Finite elements connectivity

7 Visualization
  7.1 Plot
  7.2 link with gnuplot
  7.3 link with medit

8 Algorithms and Optimization
  8.1 conjugate Gradient/GMRES
  8.2 Algorithms for Unconstrained Optimization
    8.2.1 Example of utilization for BFGS or CMAES
  8.3 IPOPT
    8.3.1 Short description of the algorithm
    8.3.2 IPOPT in FreeFem++
  8.4 Some short examples using IPOPT
  8.5 3D constrained minimal surface with IPOPT
    8.5.1 Area and volume expressions
    8.5.2 Derivatives
8.5.3 The problem and its script ................................................. 225
8.6 The nlOpt optimizers ....................................................... 230
8.7 Optimization with MPI .................................................... 234

9 Mathematical Models ....................................................... 235

9.1 Static Problems ........................................................... 235
  9.1.1 Soap Film .............................................................. 235
  9.1.2 Electrostatics .......................................................... 237
  9.1.3 Aerodynamics .......................................................... 239
  9.1.4 Error estimation ....................................................... 241
  9.1.5 Periodic Boundary Conditions ..................................... 242
  9.1.6 Poisson Problems with mixed boundary condition .......... 245
  9.1.7 Poisson with mixed finite element ................................ 247
  9.1.8 Metric Adaptation and residual error indicator ............. 248
  9.1.9 Adaptation using residual error indicator ...................... 250

9.2 Elasticity ................................................................. 252
  9.2.1 Fracture Mechanics .................................................. 256

9.3 Nonlinear Static Problems .............................................. 259
  9.3.1 Newton-Raphson algorithm ....................................... 260

9.4 Eigenvalue Problems ..................................................... 261

9.5 Evolution Problems ...................................................... 265
  9.5.1 Mathematical Theory on Time Difference Approximations. 267
  9.5.2 Convection ............................................................ 269
  9.5.3 2D Black-Scholes equation for an European Put option ...... 271

9.6 Navier-Stokes Equation ................................................ 272
  9.6.1 Stokes and Navier-Stokes ......................................... 272
  9.6.2 Uzawa Algorithm and Conjugate Gradients .................... 277
  9.6.3 NSUzawaCahouetChabart.edp .................................... 278

9.7 Variational inequality .................................................... 281

9.8 Domain decomposition ................................................... 283
  9.8.1 Schwarz Overlap Scheme ......................................... 283
  9.8.2 Schwarz non Overlap Scheme .................................... 285
  9.8.3 Schwarz-gc.edp ..................................................... 286

9.9 Fluid/Structures Coupled Problem .................................. 289

9.10 Transmission Problem .................................................. 292

9.11 Free Boundary Problem ................................................ 295

9.12 Non linear Elasticity (nolinear-elas.edp) .......................... 297

9.13 Compressible Neo-Hookean Materials: Computational Solutions 301
  9.13.1 Notation ............................................................. 301
  9.13.2 A Neo-Hookean Compressible Material ....................... 302
  9.13.3 An Approach to Implementation in FreeFem++ ............... 303

9.14 Whispering gallery modes .............................................. 303
  9.14.1 Wave equation for the WGMs .................................... 304

9.15 Weak formulation ....................................................... 304
  9.15.1 A dielectric sphere example with FreeFem++ ................. 304
Preface

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) in dimension 2 and 3. 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, MUMPS. 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 manual is meant for students at Master level, for researchers at any level, and for engineers (including financial engineering) with some understanding of variational methods for partial differential equations.
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 implies, it is a free software (see the copyrights 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++ 3.3 or later, and OpenGL), on Window XP, Vista and 7,8,10 and on MacOS 10 intel.

Moreover FreeFem++ is highly adaptive. Many phenomena involve several coupled systems, for example: fluid-structure interactions, Lorentz forces for aluminium casting and ocean-atmosphere problems are three such systems. These require different finite element approximations and polynomial 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 and three-dimensional 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 part 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; the inner point density is proportional to the density of points on the boundaries.
• Metric-based anisotropic mesh adaptation. The metric can be computed automatically from the Hessian of any FreeFem++ function \[13\].

• 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, quadratic Lagrangian elements and more, discontinuous P1 and Raviart-Thomas elements, elements of a non-scalar type, the mini-element, . . . (but no quadrangles).

• Tools to define discontinuous Galerkin finite element formulations $P_0$, $P_{1dc}$, $P_{2dc}$ and keywords: jump, mean, intalledges.

• A large variety of linear direct and iterative solvers (LU, Cholesky, Crout, CG, GMRES, UMFPACK, MUMPS, SuperLU, . . .) and eigenvalue and eigenvector solvers (ARPACK).

• 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

1.1.1 For everyone:

First open the following web page

\[\text{http://www.freefem.org/ff++/}\]

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

Remark 1 : Binaries are available for Microsoft Windows, Apple Mac OS X and some Linux systems.

Install by double click on the appropriate file, under linux and MacOS the install file are respectively in directory /usr/local/bin, /usr/local/share/freefem++, /usr/local/lib/
1.1. INSTALLATION

Windows binaries install  First download the windows installation executable, then double click it. to install FreeFem++. In most cases just answer yes (or type return) to all questions. Otherwise in the Additional Task windows, check the box ”Add application directory to your system path your system path.” This is required otherwise the program ffglut.exe will not be found.

By now you should have two new icons on your desktop:

- FreeFem++ (VERSION).exe the FreeFem++ application.
- FreeFem++ (VERSION) Examples a link to the FreeFem++ folder of examples.

where (VERSION) is the version of the files (for example 3.3-0-P4).

By default, the installed files are in

C:\Programs Files\FreeFem++

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

The syntax for the command-line tools are the same as those of FreeFem.exe.

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 an change Open with: (choose FreeFem++.app) and click on button change All....

Where to go from here  An integrated environment called FreeFem+++-cs, written by Antoine Le Hyaric, is provided with FreeFem++. Unless you wish to profile now your own development environment, you may proceed to the next paragraph ”How to use FreeFem++”.

1.1.2 For the pros: Installation from sources

This section is for those who for some reason do not wish to use the binaries and hence need to recompile FreeFem++ or install it from the source code:

The documentation archive : The documentation is also open source; to regenerate it you need a L\TeXX environment capable of compiling a CVS archive; under MS-Windows you will have to use mingw/msys

http://www.mingw.org

and under MacOS X we have used Apple’s Developer Tools ”Xcode” and L\TeXX from http://www.ctan.org/system/mac/texmac
**The C++ archive**: FreeFem++ must be compiled from the source archive, as indicated in

[http://www.freefem.org/ff++/index.htm](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:

1. **FreeFem++**, standard version, with a graphical interface based on GLUT/OpenGL (use `ffglut` visualization tool) or not just add `-nw` parameter.

2. **ffglut** the visualization tools through a pipe of freefem++ (remark: *if ffglut is not in the system path, you will have no plot*).

3. **FreeFem++-nw**, postscript plot output only and `ffmedit` (batch version, no graphics windows via `ffglut`)

4. **FreeFem++-mpi**, parallel version, postscript output only, but you can use the shell script like

   (a) `ff-mpirun -np nbproc mympiscript.edp` ## without graphic

   (b) `ff-mpirun -np nbproc mympiscript.edp -glut ffglut` ## with graphic

   (c) `ff-mpirun -np nbproc mympiscript.edp -glut ffglut parameters` ## with graphic and script parameters

   (d) `ff-mpirun -np nbproc mympiscript.edp -glut ffglut` ## with graphic

   where `nbproc` in the number of process launch.

5. **/Applications/FreeFem++.app**, the Drag and Drop CoCoo MacOSX Application

6. **bamg**, the bamg mesh generator

7. **cvmsh2**, a mesh file convertor

8. **drawbdmesh**, a mesh file viewer

9. **ffmedit** the freefem++ version of medit software (thanks to P. Frey)

The syntax of tools **FreeFem++**, **FreeFem++-nw**, **FreeFem++-mpi**, on the command-line are
### 1.1. INSTALLATION

- `FreeFem++ [-?] [-v nn] [-fglut file1] [-glut file2] [-f] edpfilepath` where:
  - `-f` if the 
  - `FreeFem++-nw [-?] [-v nn] [-fglut file1] [-glut file2] [-f] edpfilepath` where:
    - `-?` show the usage.
    - `-fglut 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` and `medit`.
    - `-v nn` set the level of verbosity to `nn` before execution of the script.
    - `-ne` no edp script output.
    - `-wait` wait an return in text window before closing `FreeFem++`.
    - `-nowait` no wait an return in text window before closing `FreeFem++`.
    - `-cd` Change directory to script dir (the script path must by global).

The notation `[ ]` means "optional".

**Remark 2** 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.

**Link with other text editors**

- **emacs** form [https://www.gnu.org/s/emacs](https://www.gnu.org/s/emacs), you can download `freefem++-mode` here: [https://github.com/rrgalvan/freefem-mode](https://github.com/rrgalvan/freefem-mode)and follow the instruction to install.

- **notepad++** form [http://notepad-plus.sourceforge.net/uk/site.htm](http://notepad-plus.sourceforge.net/uk/site.htm)
  - Open Notepad++ and Enter F5
  - In the new window enter the command `launchff++ "$\{FULL\_CURRENT\_PATH\}"`
  - Click on Save, and enter `FreeFem++` in the box "Name", now choose the short cut key to launch directly `FreeFem++` (for example `alt+shift+R`)
  - To add Color Syntax Compatible with FreeFem++ In Notepad++,
– In Menu "Parameters"->"Configuration of the Color Syntax" proceed as follows:
– In the list "Language" select C++
– Add "edp" in the field "add ext"
– Select "INSTRUCTION WORD" in the list "Description" and in the field "supplementary key word", cut and past the following list:
P0 P1 P2 P3 P4 P5 P1dc P2dc P3dc P4dc P5dc RT0 RT1 RT2 RT3 RT4 RT5 macro plot int1d int2d solve movemesh adaptmesh trunc checkmovemesh on func buildmesh square Eigenvalue min max imag exec LinearCG NLCG Newton BFGS LinearGMRES catch try intalledges jump average mean load savemesh convect abs sin cos tan atan acos cotan sinh cosh tanh cotanh atanh asinh acosh pow exp log log10 sqrt dx dy endl cout
– Select "TYPE WORD" in the list "Description" and ... "supplementary key word", cut and past the following list
mesh real fespace varf matrix problem string border complex ifstream ofstream
– Click on Save & Close. Now nodepad++ is configured.

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 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. INSTALLATION

**textmate 2** for Mac 10.7 or better, download from [https://macromates.com/download](https://macromates.com/download) and install it, when get the textmate freefem++ syntax from [http://www.freefem.org/ff++/Textmate2-ff++.zip](http://www.freefem.org/ff++/Textmate2-ff++.zip) unzip and follow the explanation given in file How_To.rtf.

**Smultron/fraise** (Obsolete) on the Mac, available at [http://smultron.sourceforge.net](http://smultron.sourceforge.net). It comes ready with color syntax for .edp file. To teach it to launch FreeFem++ files, do a ”command B” (i.e. the menu Tools/Handle Command/new command) and create a command which does

```
/usr/local/bin/FreeFem++-CoCoa %%p
```

Figure 1.1: Integrated environment for FreeFem++ development with Windows
1.2 How to use FreeFem++

Under MacOS X with Graphical Interface  To test an .edp file, just drag and drop the file icon on the MacOS application icon FreeFem++.app. You can also launch this application and use the menu: File → Open.

One of the best ways however on the Mac is to use a text editor like Smultron.app (see above).

Figure 1.2: The 3 panels of the integrated environment built with the fraise Editor with FreeFem++ in action. The Tools menu has an item to launch FreeFem++ by a Ctrl+1 command.

In Terminal mode  Choose the type of application from FreeFem++, FreeFem++-nw, FreeFem++-mpi, ... according to your needs. Add at least the path name; 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 3 The variable verbosity changes the level of internal printing (0, nothing (unless there are syntax errors), 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:

```plaintext
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"
load += "MUMPS_seq"
```

The possible paths for this file are

- under unix and MacOs
  ```plaintext
  /etc/freefem++.pref
  $(HOME)/.freefem++.pref
  freefem++.pref
  ```

- under windows
  ```plaintext
  freefem++.pref
  ```

We can also use shell environment variable to change verbosity and the search rule before the init files.

```plaintext
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

```plaintext
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 [23] 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 [31], templates and bison for its implementation; more detail can be found in [17]. 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 Poisson’s equation:

for a given function $f(x, y)$, find a function $u(x, y)$ satisfying

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

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

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).

Example 2.1

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

Figure 2.1: mesh Th by build(C(50))  Figure 2.2: isovalue by plot(u)
// 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 Vh 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 re-
quired by the finite element method (FEM). Let us go through them one by one.

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

```cpp
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 \( T_h \) of Ω is automatically generated by buildmesh(C(50)) using
50 points on C as in Fig. 2.1.
The domain is assumed to be on the left side of the boundary which is implicitly oriented
by the parametrization. So an elliptic hole can be added by
If by mistake one had written
\[
\text{border } C(t=0,2\pi) \{ x=0.1+0.3\cos(t); \ y=0.5\sin(t); \}
\]
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, \texttt{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 \texttt{adaptmesh(Th,f)}. Now the name \( T_h \) (\texttt{Th} in \texttt{FreeFem++} ) refers to the family \( \{ T_k \}_{k=1}^{n_t} \) of triangles shown in figure \ref{fig:2.1}. Traditionally \( h \) refers to the mesh size, \( n_t \) to the number of triangles in \( T_h \) 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 \( \Omega_h = \bigcup_{k=1}^{n_t} T_k \). 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 \texttt{fespace Vh(Th,P1)} defines \( \mathcal{V}_h \) to be the space of continuous functions which are affine in \( x,y \) on each triangle of \( T_h \). As it is a linear vector space of finite dimension, basis can be found. The canonical basis is made of functions, called the \textit{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\footnote{The easiest way to define \( \phi_k \) is by making use of the \textit{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 \( \lambda_k \).}

Then
\[
\mathcal{V}_h(T_h,P_1) = \left\{ w(x,y) \bigg| w(x,y) = \sum_{k=1}^{M} w_k \phi_k(x,y), \ w_k \text{ are real numbers} \right\}
\] (2.3)
where \( M \) is the dimension of \( \mathcal{V}_h \), i.e. the number of vertices. The \( w_k \) are called the \textit{degree of freedom} of \( w \) and \( M \) the number of the degree of freedom.

It is said also that the \textit{nodes} of this finite element method are the vertices.

Currently \texttt{FreeFem++} implements the following elements in 2d, (see section 6 for the full description)
\begin{itemize}
  \item \texttt{P0} piecewise constant,
  \item \texttt{P1} continuous piecewise linear,
  \item \texttt{P2} continuous piecewise quadratic,
  \item \texttt{P3} continuous piecewise cubic, (need \texttt{load "Element_P3"})
  \item \texttt{P4} continuous piecewise cubic, (need \texttt{load "Element_P4"})
  \item \texttt{RT0} Raviart-Thomas piecewise constant,
\end{itemize}
RT1 Raviart-Thomas degree 1 piecewise constant (need load "Element_Mixte")
BDM1 Brezzi-Douglas-Marini degree 1 piecewise constant (need load "Element_Mixte")
RT0Ortho Nedelec type 1 degree 0 piecewise constant
RT1Ortho Nedelec type 1 degree 1 piecewise constant (need load "Element_Mixte")
BDM1Ortho Brezzi-Douglas-Marini degree 1 piecewise constant (need load "Element_Mixte")
P1nc piecewise linear non-conforming,
P1dc piecewise linear discontinuous,
P2dc piecewise quadratic discontinuous,
P2h quadratic homogene continuous (without P1)
P3dc piecewise cubic discontinuous,(need load "Element_P3dc")
P4dc piecewise quartic discontinuous,(need load "Element_P4dc")
P1b piecewise linear continuous plus bubble,
P2b piecewise quadratic continuous plus bubble.
Morley Morley finite element (need load "Morley")
HCT Hsieh-Clough-Tocher $C^1$ finite element (need load "Element_HCT", version 3.40)
P2BR P2 Bernardi-Raugel finite element (need load "BernardiRaugel.cpp")
P0edge a finite element constant per edge
P1edge to P5edge a finite element polynomial on edge (need load "Element_PkEdge")
...
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, Edge13d, Edge23d The Nedelec Edge element 0,1,2
P1b3d piecewise linear continuous plus bubble,
...
To get the full list, in a unix terminal, in directory examples++-tutorial do

FreeFem++ dumptable.edp
grep TypeOfFE lestables

Note that other elements can be added fairly easily.
Step 3: Setting the problem

4th line: \( V h \ u,v \) declares that \( u \) and \( v \) are approximated as above, namely

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

(2.4)

5th line: the right hand side \( f \) is defined analytically using the keyword \texttt{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 \( \Omega \):

\[
  - \int_{\Omega} v \Delta u \, dx \, dy = \int_{\Omega} vf \, dx \, dy
\]

Then, by Green’s formula, the problem is converted into finding \( u \) such that

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

(2.5)

with

\[
  a(u,v) = \int_{\Omega} \nabla u \cdot \nabla v \, dx \, dy, \quad \ell(f,v) = \int_{\Omega} fv \, dx \, dy
\]  

(2.6)

In FreeFem++ the \textbf{Poisson} problem can be declared only as in

\[
  V h \ u,v; \texttt{problem} \ \texttt{Poisson}(u,v) =
\]

and solved later as in

\[
  \ldots
  \texttt{Poisson;} \quad // \quad \text{the problem is solved here}
  \ldots
\]

or declared and solved at the same time as in

\[
  V h \ u,v; \texttt{solve} \ \texttt{Poisson}(u,v) = \texttt{int} (\ldots
\]

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

\[
  \int_{\Omega} \nabla u \cdot \nabla v \, dx \, dy \rightarrow \texttt{int2d}(Th)( dx(u) * dx(v) + dy(u) * dy(v) )
\]

\[
  \int_{\Omega} fv \, dx \, dy \rightarrow \texttt{int2d}(Th)( f*v ) \quad (\text{Notice here, } u \text{ is unused})
\]

In FreeFem++ bilinear terms and linear terms should not be under the same integral; indeed to construct the linear systems FreeFem++ finds out which integral contributes to the bilinear form by checking if both terms, the unknown (here \( u \)) and test functions (here \( v \)) are present.

Step 4: Solution and visualization

6th line: The current time in seconds is stored into the real-valued variable \( \texttt{cpu} \).

7th line The problem is solved.
The visualization is done as illustrated in Fig. 2.2 (see Section 7.1 for zoom, postscript and other commands).

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

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

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 $F_i = 10^{30} \times 0$ if $i$ is a boundary degree of freedom. Note, that the number $10^{30}$ is called tgv (très grande valeur) and it is generally possible to change this value, see the index item solve!tgv=.

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

If the user wants to access $A$ directly he can do so by using (see section 6.12 page 195 for details)

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

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

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

The problem can then be solved by

```c
u[] = A^-1\times 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 $(u[i]_{i=0,...,M-1}$ and $(F[])_{i=0,...,M-1}$). 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 $\phi_i, i = 0,..., M - 1$ are the basis functions of $Vh$ like in equation (2.3), and $M = Vh.n dof$ is the number of degree of freedom (i.e. the dimension of the space $Vh$).

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

```c
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 LaTeX. 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++ 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 documentation in the next chapter. If you are a FEM beginner, you may also have 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 (here written in file mycode.edp). note that this can also be done in terminal mode by:

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

**Visualization:** Use the keyword `plot` to display functions while FreeFem++ is running. Use the plot-parameter `wait=1` to stop the program at each 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`.

```plaintext
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
```
CHAPTER 2. GETTING STARTED

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; watching the flow of graphs on the screen (while drinking coffee) 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, (we did our best)! Nevertheless they are displayed at the right place. For example, if you forget parenthesis as in

```cpp
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

```cpp
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, just as you would do in C++.

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

```cpp
real aaa =1;  //     real aaa;
```
Chapter 3

Learning by Examples

This chapter is for those, like us, who don’t like to read manuals. A number of simple examples cover a good deal of the capacity of FreeFem++ and are self-explanatory. For the modeling 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 $dx = dx_1dx_2$. 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.

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

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)

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

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

**WARNING** Unlike freefem+ which had both weak and strong forms, FreeFem++ implements only weak formulations. It is not possible to go further in using this software if you don’t know the weak form (i.e. variational formulation) of your problem: either you read a book, or ask help 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

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

**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)**

```plaintext
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 $T_h$ and $\varphi$ (remove $T_h$ 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.

```plaintext
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

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

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

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

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

**Example 3.2 (membranerror.edp)**

```plaintext
verbosity = 0; // to remove all default output
real theta = 4.*pi/3.; // the length of the semimajor axis and semiminor axis
real a = 1., b = 1.;
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);
```
3.1. MEMBRANES

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))<<endl;
    }
  }
}

We use the finite element numbering, where $Wh(i,j)$ is the global index of $j^{th}$ degrees of freedom of triangle number $i$.

Then open gnuplot and do

    set palette rgbformulae 30,31,32
    splot "graph.txt" w 1 pal

This works with $P_2$ and $P_1$, but not with $P_{1nc}$ because the 3 first degrees of freedom of $P_2$ or $P_2$ are on vertices and not with $P_{1nc}$.
### 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 \( \{C_i\}_{i=1,2} \) 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\(^\circ\)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

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

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:

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

The variational formulation for this problem is in the subspace \( H^1_0(\Omega) \subset H^1(\Omega) \) of functions which have zero traces on \( \Gamma \):

\[
u - g \in H^1_0(\Omega) : \quad \int_{\Omega} \nabla u \nabla v = 0 \quad \forall v \in H^1_0(\Omega)
\]

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 \( u_1 = 60^\circ C \).

#### Example 3.3 (heatex.edp)

```plaintext
// file heatex.edp
int C1=99, C2=98; // could be anything such that \( \neq 0 \) and \( C1 \neq C2 
border CO(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( CO(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");


plot(Th,wait=1);
```
3.2. HEAT EXCHANGER

```plaintext
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:

  ```plaintext
  // 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](image)

**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:

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

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

### 3.3 Acoustics

**Summary**  *Here we go to grip with ill posed problems and eigenvalue problems*

Pressure variations in air at rest are governed by the wave equation:

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

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

\[
k^2 v + c^2 \Delta v = 0 \quad \text{in } \Omega,
\]

\[
\frac{\partial v}{\partial n}|_{\Gamma} = g.
\]

(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:

```plaintext
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 ;}
```

```plaintext
// file sound.edp
```
3.4. THERMAL CONDUCTION

3.4.1 Mesh

```c
mesh Th = buildmesh( a0(20) + a1(20) + a2(20) + a3(20) + a4(20) + a5(20) + a6(20));
```

3.4.2 FESpace

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

3.4.3 Solve

```c
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 $kc^2$ 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.

```c
real sigma = 20; // value of the shift
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");
```

3.4.4 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 $u = u_0 + xu_1$. 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 $(0,T)$.

\[
\begin{align*}
\partial_t u - \nabla \cdot (\kappa \nabla u) &= 0 \text{ in } \Omega \times (0,T), \\
u(x,y,0) &= u_0 + xu_1 \\
\kappa \frac{\partial u}{\partial n} + \alpha (u - u_e) &= 0 \text{ on } \Gamma \times (0,T).
\end{align*}
\]
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 $w \in H^1(\Omega)$:

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

```plaintext
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 \equiv u^{n-1} = u^n \equiv u
  thermic; // here solve the thermic problem
  ff<<u(3,0.5)<<endl;
}```
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

```cpp
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):

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

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:

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

(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:

```cpp
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 $v = u - u_e$.

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

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

### 3.5 Irrotational Fan Blade Flow and Thermal effects

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

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

\[
u = \left( \begin{array}{c} \frac{\partial \psi}{\partial x} \\ -\frac{\partial \psi}{\partial x_1} \end{array} \right), \quad \text{where } \psi \text{ is solution of } \Delta \psi = 0\]

This equation expresses both incompressibility \((\nabla \cdot \nu = 0)\) and absence of vortex \((\nabla \times \nu = 0)\).

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

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

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)**

```plaintext
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 $v$ 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:

```plaintext
...  
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);
```
3.6. PURE CONVECTION : THE ROTATING HILL

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 Peclet 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

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

Pure convection by $u$ is

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

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

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

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.
CHAPTER 3. LEARNING BY EXAMPLES

Solution by a Characteristics-Galerkin Method  In FreeFem++ there is an operator called
\texttt{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 \(u = [u1,u2]\)
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 \(u\) 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; \texttt{convect}
returns \(c(x_{df}) = C \circ X\).

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

\begin{verbatim}
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=",min=c[].min +", max=", max=" + c[].max);
}
\end{verbatim}

Remark 4  3D plots can be done by adding the qualifyer ”dim=3” to the plot instruction.

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

Solution by Discontinuous-Galerkin FEM  Discontinuous Galerkin methods take advantage
of the discontinuities of \(c\) at the edges to build upwinding. There are may formulations
possible. We shall implement here the so-called dual-\(P^1_{DC}\) formulation (see Ern[16]):
\[
\int_{\Omega} \left( \frac{c_{n+1} - c^n}{\delta t} + u \cdot \nabla c \right) w + \int_{E^i} (\alpha |n \cdot u| - \frac{1}{2} n \cdot u)|c| w = \int_{E^\Gamma} |n \cdot u| cw \quad \forall w
\]
where \(E\) is the set of inner edges and \(E^\Gamma\) is the set of boundary edges where \(u \cdot n < 0\) (in our
case there is no such edges). Finally \([c]\) is the jump of \(c\) across an edge with the convention
that \(c^+\) refers to the value on the right of the oriented edge.

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

\begin{verbatim}
... 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;
\end{verbatim}
3.6. PURE CONVECTION : THE ROTATING HILL

```csharp
macro n() (N.x*v1+N.y*v2) // Macro without parameter
problem Adual(cc,w) =
  int2d(Th)((cc/dt+(v1*dx(cc)+v2*dy(cc))*w) + intalledges(Th)((1-nTonEdge)*w*(al*abs(n)-n/2)*jump(cc))
  // - int1d(Th,C)((n<0)*abs(n)*cc*w) // unused because cc=0 on \partial \Omega-
  - int2d(Th)(ccold*w/dt);
for ( t=0; t< 2*pi ; t+=dt) {
  ccold=cc; Adual;
  plot(cc,fill=1,cmm="t="+t + ", min="+ cc[].min + ", max=" + cc[].max);
}
real [int] viso=[-0.2,-0.1,0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.1];
plot(c,wait=1,fill=1,ps="convectCG.eps",viso=viso);
plot(c,wait=1,fill=1,ps="convectDG.eps",viso=viso);
```

Notice the new keywords, intalledges to integrate on all edges of all triangles

\[ \text{intalledges}(Th) \equiv \sum_{T \in Th} \int_{\partial T} \] (3.5)

(so all internal edges are see two times ), nTonEdge which is one if the triangle has a boundary edge and two 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
CHAPTER 3. LEARNING BY EXAMPLES

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

Notice the new keyword set to specify a solver in this framework; the modifier init is used
to tel the solver that the matrix has not changed (init=true), and the name parameter are
the same that in problem definition (see [6.9]).

Finite Volume Methods can also be handled with FreeFem++ but it requires program-
ing. 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:

\[
\int_{\sigma_i} \frac{1}{\Delta t} (c^{1n+1} - c^{1n}) + \int_{\partial\sigma_i} u \cdot n c^- = 0 \quad \forall i
\]

It can be programmed as

```plaintext
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;
```
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

```c++
// 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 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 $u(x)$ in a section $\Omega$ subject to forces $f$ perpendicular to the axis are governed by

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

where $\lambda, \mu$ are the Lamé coefficients introduced above. Remark, we do not used this equation because the associated variational form does not give the right boundary condition, we simply use

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

where the corresponding variational form is:

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

where $: \epsilon$ 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

$$
\sigma \cdot n = g = 0 \quad \text{on} \quad \Gamma_1, \Gamma_4, \Gamma_3,
$$

$$
u = 0 \quad \text{on} \quad \Gamma_2
$$

Here $u = (u,v)$ 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 3.8 (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
```
3.8. THE SYSTEM OF STOKES FOR FLUIDS

3.8 The System of Stokes for Fluids

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

\[-\Delta \mathbf{u} + \nabla p = 0\]
\[\nabla \cdot \mathbf{u} = 0\]

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
CHAPTER 3. LEARNING BY EXAMPLES

Figure 3.7: 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

\[
\text{int } n=3; \\
\text{mesh } Th=\text{square}(10*n,10*n); \\
\text{fespace } Uh(Th,P1b); Uh u,v,uu,vv; \\
\text{fespace } Ph(Th,P1); Ph p,pp; \\
\text{solve} \ \text{stokes}([u,v,p],[uu,vv,pp]) = \\
\text{int2d}(Th)(dx(u)*dx(uu)+dy(u)*dy(uu) + dx(v)*dx(vv) + dy(v)*dy(vv) \\
+ dx(p)*uu + dy(p)*vv + pp*(dx(u)+dy(v)) \\
- 1e-10*p*pp) \\
+ \text{on}(1,2,4,u=0,v=0) + \text{on}(3,u=1,v=0); \\
\text{plot}([u,v],p,wait=1);
\]

Remark, we add a stabilization term \(-10e-10*p*pp\) to fixe the constant part of the pressure.

Results are shown on figure 3.8

3.9 A Projection Algorithm for the Navier-Stokes equations

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

An incompressible viscous fluid satisfies:

\[
\partial_t u + u \cdot \nabla u + \nabla p - \nu \Delta u = 0, \quad \nabla \cdot u = 0 \quad \text{in } \Omega \times ]0, T[, \\
u|_{t=0} = u^0, \quad u|_{\Gamma} = u_\Gamma.
\]
A possible algorithm, proposed by Chorin, is

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

\[-\Delta p^{m+1} = -\nabla \cdot u^m \delta t u^m, \quad \partial_n p^{m+1} = 0 \text{ on } \Gamma, p^{m+1} = 0 \text{ on } \Gamma_{out}\]

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

We use the s Chorin’s algorithm with free boundary condition at outlet (i.e. \( p = 0, \nu \partial_n u = 0 \)), to compute a correction, \( q \), to the pressure.

\[-\Delta q = \nabla \cdot u, \quad q = 0 \text{ on } \Gamma_{out}\]

and define

\[u^{m+1} = \bar{u} + P q \delta t, \quad p^{m+1} = p^m - q\]

where \( \bar{u} \) is the \((u^{m+1}, v^{m+1})\) of Chorin’s algorithm, and where \( P \) is the \(L^2\) projection with mass lumping (a sparse matrix).

**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. Remark (FH), The are a technical difficulty is the example, is the output B.C., here we put \( p = 0 \) and \( \nu \partial_n u = 0 \).
Example 3.9 (NSprojection.edp) // file NSprojection.edp // Version july 2014, // FH. Change B.C on u on output, more simple .. // ............

verbosity=0;

border a0(t=1,0){ x=-2; y=t; label=1;}
border a1(t=-2,0){ x=t; y=0; label=2;}
border a2(t=0,-0.5){ x=0; y=t; label=2;}
border a3(t=0,1){ x=18*t^1.2; y=-0.5; label=2;}
border a4(t=-0.5,1){ x=18; y=t; label=3;}
border a5(t=1,0){ x=-2+20*t; y=1; label=4;}
int n=1;

mesh Th = buildmesh(a0(3*n)+a1(20*n)+a2(10*n)+a3(150*n)+a4(5*n)+a5(100*n));

plot(Th);

fespace Vh(Th,P1);

real nu = 0.0025, dt = 0.2;
// Reynolds=200

Vh w,u = 0, v =0, p = 0, q=0;

real epsv = 1e-6, epsu = 1e-6, epsp = 1e-6;
// Eps CG ..

matrix dtM1x,dtM1y;

macro BuildMat()
{
    /* for memory managenemnt */
    varf vM(unused,v) = int2d(Th){v} ;
    varf vdx(u,v) = int2d(Th){v*dx(u)*dt} ;
    varf vdy(u,v) = int2d(Th){v*dy(u)*dt} ;

    real[int] Mlump = vM(0,Vh);
    real[int] one(Vh.ndof); one = 1;
    real[int] M1 = one ./ Mlump;
    matrix dM1 = M1;
    matrix Mdx = vdx(Vh,Vh);
    matrix Mdy = vdy(Vh,Vh);
    dtM1x = dM1*Mdx;
    dtM1y = dM1*Mdy;
}
// EOF
BuildMat

real err=1, outflux=1;

for(int n=0;n<300;n++)
{
    Vh uold = u, vold = v, pold=p;

    solve pb4u(u,w,init=n,solver=CG,eps=epsu) =
        int2d(Th){u*w/dt +nu*(dx(u)*dx(w)+dy(u)*dy(w))}
        -int2d(Th){(convect([uold,vold],[-dt,uold]/dt-dx(p)))*w}
        + on(1,u = 4*y*(1-y)) + on(2,4,u = 0) ;
    // Neuman on Gamma3

    plot(u);

    solve pb4v(v,w,init=n,solver=CG,eps=epsv)
3.9. A PROJECTION ALGORITHM FOR THE NAVIER-STOKES EQUATIONS

\[
\begin{align*}
= & \text{int2d}(Th) \left( v \times w/dt + \nu \star (dx(v) \times dx(w) + dy(v) \times dy(w)) \right) \\
- & \text{int2d}(Th) \left( \text{convect} \left( [uold, vold], -dt, vold \right) / dt - dy(p) \right) \times w \\
+ & \text{on}(1, 2, 3, 4, v = 0);
\end{align*}
\]

\[\text{solve pb4p}(q, w, \text{solver=CG, init=n, eps=epsp}) = \text{int2d}(Th) \left( dx(q) \times dx(w) + dy(q) \times dy(w) \right) - \text{int2d}(Th) \left( (dx(u) + dy(v)) \times w/dt \right) + \text{on}(3, q=0);\]

\[
/ \text{ to have absolute epsilon in CG algorithm.}
\]

epsv = -abs(epsv);
epsu = -abs(epsu);
epsp = -abs(epsp);

p = pold-q;
u[] += dtM1x*q[];
v[] += dtM1y*q[];

if(n%50==49){
    Th = adaptmesh(Th, [u,v], q, err=0.04, nbvx=100000);
    plot(Th, wait=true); BuildMat // rebuild mat.
}

er = sqrt(int2d(Th) (square(u-uold)+square(v-vold))/Th.area) ;
outflux = int1d(Th) ( [u,v]'*[N.x,N.y] ) ;
cout << " iter " << n << " Err L2 = " << err
    << " flux sortant = " << outflux << endl;
    if(err < 1e-3) break;
}
assert(abs(outflux)< 2e-3); // verification ...
plot(p,wait=1,ps="NSprojP.eps");
plot(u,wait=1,ps="NSprojU.eps");

Figure 3.9: 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 3.9 the numerical results obtained for a Reynolds number of 400 where mesh adaptation is done after 50 iterations on the first mesh.
3.10 Newton Method for the Steady Navier-Stokes equations

The problem is find the velocity field $u = (u_i)_{i=1}^d$ and the pressure $p$ of a Flow satisfying in the domain $\Omega \subset \mathbb{R}^d (d = 2, 3)$:

$$(u \cdot \nabla) u - \nu \Delta u + \nabla p = 0,$$

$$\nabla \cdot u = 0$$

where $\nu$ is the viscosity of the fluid, $\nabla = (\partial_i)_{i=1}^d$, the dot product is $\cdot$, and $\Delta = \nabla \cdot \nabla$ with the some boundary conditions ($u$ is given on $\Gamma$)

The weak form is find $u, p$ such than for $\forall v$ (zero on $\Gamma$), and $\forall q$

$$\int_{\Omega} ((u \cdot \nabla) u \cdot v + \nu \nabla u : \nabla v - p \nabla \cdot v - q \nabla \cdot u = 0$$

The Newton Algorithm to solve nonlinear Problem is

Find $u \in V$ such that $F(u) = 0$ where $F : V \mapsto V$.

1. choose $u_0 \in \mathbb{R}^n$ ;
2. for ($i = 0; i < \text{iter}; i = i + 1$)
   (a) solve $DF(u_i)w_i = F(u_i)$;
   (b) $u_{i+1} = u_i - w_i$;
   break \(||w_i|| < \varepsilon\).

Where $DF(u)$ is the differential of $F$ at point $u$, this is a linear application such that:

$F(u + \delta) = F(u) + DF(u)\delta + o(\delta)$

For Navier Stokes, $F$ and $DF$ are :

$$F(u,p) = \int_{\Omega} ((u \cdot \nabla) u \cdot v + \nu \nabla u : \nabla v - p \nabla \cdot v - q \nabla \cdot u$$

$$DF(u,p)(\delta u, \delta p) = \int_{\Omega} ((\delta u \cdot \nabla) u \cdot v + (u \cdot \nabla) \delta u \cdot v$$

$$+ \nu \nabla \delta u : \nabla v - \delta p \nabla \cdot v - q \nabla \cdot \delta u$$

So the Newton algorithm become

Example 3.10 (NSNewton.edp) ...

```plaintext
for ( n=0; n< 15; n++)
{ solve Oseen([du1,du2,dp],[v1,v2,q]) =
  int2d(Th) ( nu*(Grad(du1,du2)'*Grad(v1,v2) )
                 + UgradV(du1,du2, u1, u2)*[v1,v2]
                 + UgradV( u1, u2,du1,du2)*[v1,v2]
                 - div(du1,du2)*q - div(v1,v2)*dp
                 - 1e-8*dp*q                      // stabilization term
       )
}
```

```
3.10. NEWTON METHOD FOR THE STEADY NAVIER-STOKES EQUATIONS

\[
- \text{int2d}(\theta) \{ \text{nu} \times (\text{Grad}(u_1, u_2)') \times \text{Grad}(v_1, v_2) \\
+ \text{UgradV}(u_1, u_2, u_1, u_2)' \times \{v_1, v_2\} \\
- \text{div}(u_1, u_2) * q - \text{div}(v_1, v_2) * p \}
\]

+ \text{on}(1, du_1=0, du_2=0) ;  \\
\text{u_1[]} -= du_1[]; \text{u_2[]} -= du_2[]; \text{p[]} -= dp[];  \\
\text{err} = \text{du_1[].linfty} + \text{du_2[].linfty} + \text{dp[].linfty};  \\
\text{if}(\text{err} < \text{eps}) \text{break};  \\
\text{if}(n>3 \&\& \text{err} > 10.) \text{break}; // blowup ????
\}

With the operator:

\[
\text{macro Grad}(u_1, u_2) \{ \text{dx}(u_1), \text{dy}(u_1), \text{dx}(u_2), \text{dy}(u_2) \}  \\
\text{macro UgradV}(u_1, u_2, v_1, v_2) \{ [u_1, u_2]' \times [\text{dx}(v_1), \text{dy}(v_1)], \\
[u_1, u_2]' \times [\text{dx}(v_2), \text{dy}(v_2)] \}  \\
\text{macro div}(u_1, u_2) \{ \text{dx}(u_1) + \text{dy}(u_2) \}
\]

We build a computation mesh the exterior of a 2d cylinder.

\[
\text{real } R = 5, L = 15;  \\
\text{border } cc(t=0, 2 \times \pi) \{ x = \cos(t) / 2; y = \sin(t) / 2; \text{label} = 1; \}  \\
\text{border } ce(t=\pi/2, 3 \times \pi/2) \{ x = \cos(t) \times R; y = \sin(t) \times R; \text{label} = 1; \}  \\
\text{border } beb(tt=0, 1) \{ \text{real } t = tt^{1.2}; x = t \times L; y = -R; \text{label} = 1; \}  \\
\text{border } beu(tt=0, 1) \{ \text{real } t = tt^{1.2}; x = t \times L; y = R; \text{label} = 1; \}  \\
\text{border } beo(t=-R, R) \{ x = L; y = t; \text{label} = 0; \}  \\
\text{border } bei(t=-R/4, R/4) \{ x = L/2; y = t; \text{label} = 0; \}  \\
\text{mesh } Th = \text{buildmesh}(cc(-50) + ce(30) + beb(20) + beu(20) + beo(10) + bei(10));  \\
\text{plot}(Th);  \\
\]

\[
\text{func } \text{bb} = \{ [-1, -2], [4, 2] \};  \\
\]

// bounding box for the plot

/ FE Space Taylor Hood
\text{fespace } Xh(Th, P2);  \\
\text{fespace } Mh(Th, P1);  \\
Xh u_1, u_2, v_1, v_2, du_1, du_2, ulp, u2p;  \\
Mh p, q, dp, pp;  \\
\]

// initial guess with B.C.
\[
u_1 = \{ x^2 + y^2 > 2;  \\
u_2 = 0;  \\
\]

\[
\text{Finally we use trick to make continuation on the viscosity } \nu, \text{ because the Newton method blowup owe start with the final viscosity } \nu
\]

\[
\text{real } \text{nu} = 1. / 50, \text{nufinal} = 1 / 200; , \text{cnu} = 0.5;  \\
\]

\[
\text{real } \text{eps} = 1 \times 10^{-6};  \\
\text{verbosity} = 0;  \\
\text{while}(1)  \\
\{ \text{int } n;  \\
\text{real } \text{err} = 0;  \\
\}
\]

// Physical parameter

// stop test for Newton

// Loop on viscosity

// err on Newton algo ...
if(err < eps)
{
    if( nu == nufinal) break; // converge decrease $\nu$ (more difficult)
    plot([u1,u2],p,wait=1,cmm=" rey = " + 1./nu , coef=0.3,bb=bb);
    if( n < 4) cnu=cnu^1.5; // fast converge => change faster
    nu = max(nufinal, nu* cnu); // new viscosity
    u1p=u1; u2p=u2; pp=p; // save correct solution ...
}
else
{
    assert(cnu< 0.95); // blowup increase $\nu$ (more simple)
    nu = nu/cnu; // the method finally blowup
    cnu= cnu^1.5; // get previous value of viscosity
    nu = nu* cnu; // no conv. => change lower
    cout << " restart nu = " << nu << " Rey= "<< 1./nu << " (cnu = " << cnu << " 
); // new viscosity
    u1=u1p;
    u2=u2p;
    p=pp;
}

3.11 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

\[
\begin{align*}
\partial_t \theta + u \nabla \theta - \nabla \cdot (\kappa^m \nabla \theta) &= 0 \\
\partial_t u + u \nabla u - \nabla \cdot (\mu^m \nabla u) + \nabla p + e(\theta - \theta_0) \mathbf{e}_2, \quad \nabla \cdot u &= 0 \\
\mu_T &= c_\mu \frac{\epsilon}{\epsilon^2}, \quad \kappa_T = \kappa \mu_T \\
\partial_t k + u \nabla k + \epsilon - \nabla \cdot (\mu_T \nabla k) &= \frac{\mu_T}{2} |\nabla u + \nabla u_T|^2 \\
\partial_t \epsilon + u \nabla \epsilon + c_2 \frac{\epsilon^2}{k} - \frac{c_\epsilon}{c_\mu} \nabla \cdot (\mu_T \nabla \epsilon) &= \frac{c_1}{2} k |\nabla u + \nabla u_T|^2 = 0
\end{align*}
\] (3.7)

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

\[
\begin{align*}
\frac{1}{\delta t}(\theta^{n+1} - \theta^m \circ X^m) - \nabla \cdot (\kappa^m \nabla \theta^{n+1}) &= 0 \\
\frac{1}{\delta t}(u^{n+1} - u^m \circ X^m) - \nabla \cdot (\mu^m \nabla u^{n+1}) + \nabla p^{n+1} + e(\theta^{n+1} - \theta_0) \mathbf{e}_2, \quad \nabla \cdot u^{n+1} &= 0 \\
\frac{1}{\delta t}(k^{n+1} - k^m \circ X^m) + \frac{\mu^m \kappa^m \epsilon^m}{k^m} - \nabla \cdot (\mu_T \nabla k^{n+1}) &= \frac{\mu_T^m}{2} |\nabla u^m + \nabla u_T^m|^2 \\
\frac{1}{\delta t}(\epsilon^{n+1} - \epsilon^m \circ X^m) + c_2 \frac{\epsilon^{n+1}}{k^m} - \frac{c_\epsilon}{c_\mu} \nabla \cdot (\mu_T \nabla \epsilon^{n+1}) &= \frac{c_1}{2} \frac{k^m}{k^m} |\nabla u^m + \nabla u_T^m|^2 \\
\mu_T^{n+1} &= c_\mu \frac{k^{m+1}}{\epsilon^{n+1}}, \quad \kappa_T^{n+1} = \kappa \mu_T^{n+1}
\end{align*}
\] (3.8)

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

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

fespace Vh2(Th,P1b);    // velocity space
fespace Vh(Th,P1);     // pressure space
fespace V0h(Th,P0);    // for gradients
Vh2 u2,v2,up1=0,up2=0;
Vh2 u1,v1;
Vh u1x=0, u1y,u2x,u2y, vv;
real reylnods=500;    // cout << " Enter the reynolds number :"; cin >> reylnods;
assert(reylnods>1 && reylnods < 100000);
```
up1=0;  
up2=0;  
func  
g=(x)*(1-x)*4; // inflow  
Vh p=0,q, temp1,temp=35, k=0.001,k1,ep=0.0001,ep1;  
V0h muT=1,prodk,prode, kappa=0.25e-4, stress;  
real alpha=0, eee=9.81/303, c1m = 1.3/0.09;  
real nu=1, numu=nu/sqrt( 0.09), nuep=pow(nu,1.5)/4.1;  
int i=0,iter=0;  
real dt=0;  
problem TEMPER(temp,q) = // temperature equation  
  int2d(Th){  
    alpha*temp*q + kappa * ( dx(temp)*dx(q) + dy(temp)*dy(q) )  
    + 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) )  
    + 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) )  
    + 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*0.0000001  
    - 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,uu= -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);  
for (int i=0;i<21;i++)  
{  
  yy[i]=i/20.;
3.11. A LARGE FLUID PROBLEM

\begin{verbatim}
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./reynods;

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(u1)],splitbedge=1,
    abserror=0,cutoff=cut,err=tol, inquire=0,ratio=1.5,hmin=1./1000);
plot(Th,ps="ThNS.eps");
dt = dt*coefdt;
tol = tol *coeftol;
cut = cut *coefcut;
}
cout << "CPU " << clock()-s0 << "s " << endl;
\end{verbatim}
3.12 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 \( v = \sin(\pi \frac{y-c}{d-c}) \).

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 3.11: A microwave oven: real (left) and imaginary (middle) parts of wave and temperature (right).](image)

Results are shown on figure 3.11

In the program below \( \beta = \frac{1}{(1 - I/2)} \) in the air and \( 2/(1 - I/2) \) in the object \((i = \sqrt{-1})\):

```plaintext
Example 3.11 (muwave.edp)

// file muwave.edp
real a=20, b=20, c=15, d=8, e=2, l=12, f=2, g=2;
border a0(t=0,1) {x=a*t; y=0; label=1;}
border a1(t=1,2) {x=a; y= b*(t-1);label=1;}
border a2 (t=2,3) {x=a*(3-t); y=b;label=1;}
border a3(t=3,4) {x=0; y=b-(b-c)*(t-3); label=1;}
border a4(t=4,5) {x=0; y=c-(c-d)*(t-4); label=2;}
border a5(t=5,6) {x=0; y= d*(6-t); label=1;}
border b0(t=0,1) {x=a-f+e*(t-1); y=g; label=3;}
border b1(t=1,4) {x=a-f; y=g+l*(t-1)/3; label=3;}
border b2(t=4,5) {x=a-f-e*(t-4); y=l+g; label=3;}
border b3(t=5,8) {x=a-e-f; y= 1+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))
```

3.13 Optimai Control

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

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

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 \( J'_{\kappa} \) 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

```plaintext
// 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=((xˆ2+yˆ2)<1.0001),
Ic=(((x+3)ˆ2+ yˆ2)<1.0001),
Id=((xˆ2+(y+3)ˆ2)<1.0001),
Ie=(((x-1)ˆ2+ yˆ2)<=4),
ud,u,uh,du;
real[int] z(3);
problem A(u,uh) =int2d(th)((1+z[0]*Ib+z[1]*Ic+z[2]*Id)*(dx(u)*dx(uh) +dy(u)*dy(uh))) + on(aa,u=xˆ3-yˆ3);
z[0]=2; z[1]=3; z[2]=4;
A; ud=u;
ofstream f("J.txt");
func real[int] J(real[int] & Z)
{
    for(int i=0;i<z.n;i++) z[i]=Z[i];
    A; real s=int2d(th)(Ie*(u-ud)ˆ2);
    f<<s<<" "; return s;
}
real[int] dz(3), dJdz(3);
problem B(du,uh) =int2d(th)((1+z[0]*Ib+z[1]*Ic+z[2]*Id)*(dx(du)*dx(uh)+dy(du)*dy(uh)))
    +int2d(th)((dz[0]*Ib+dz[1]*Ic+dz[2]*Id)*(dx(u)*dx(uh)+dy(u)*dy(uh)))
    +on(aa,du=0);
func real[int] DJ(real[int] &Z)
{
    for(int i=0;i<z.n;i++)
    {
        for(int j=0;j<dz.n;j++) dz[j]=0;
        dz[i]=1; B;
        dJdz[i]= 2*int2d(th)(Ie*(u-ud)*du);
    }
    return dJdz;
}
real[int] Z(3);
for(int j=0;j<z.n;j++) Z[j]=1;
BFGS(J,DJ,Z,eps=1.e-6,nbiter=15,nbiterline=20);
cout << "BFGS: J(z) = " << J(Z) << endl;
for(int j=0;j<z.n;j++) cout<<Z[j]<<endl;
plot(ud,value=1,ps="u.eps");

In this example the sets $B,C,D,E$ are circles of boundaries $bb,cc,dd,ee$ are the domain $\Omega$ 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 3.12 shows $u$ at convergence and the successive function evaluations of $J$. Note that an adjoint state could have been used. Define $p$ by

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

Consequently

$$\delta J = - \int_{\Omega} (\nabla \cdot (\kappa \nabla p)) \delta u$$
3.14. A FLOW WITH SHOCKS

Figure 3.12: 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)

$$
\int_\Omega (\kappa \nabla p \cdot \nabla \delta u) = - \int_\Omega (\delta \kappa \nabla p \cdot \nabla u)
$$

(3.9)

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.

3.14 A Flow with Shocks

Compressible Euler equations should be discretized with Finite Volumes or FEM with flux up-winding 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, and finally mesh adaptation.

$$
\frac{\partial \rho}{\partial t} + \bar{u} \nabla \rho + \bar{\rho} \nabla \cdot u = 0 \\
\bar{\rho} (\frac{\partial u}{\partial t} + \frac{\rho}{\bar{\rho}} \nabla \bar{u} + \nabla \bar{p}) = 0 \\
\frac{\partial \bar{p}}{\partial t} + \bar{u} \nabla \bar{p} + (\gamma - 1) \bar{\rho} \nabla \cdot u = 0
$$

(3.10)

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

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

(3.11)
A numerical result is given on Figure 3.13 and the FreeFem++ script is

```cpp
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 aaal1(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) + aaal1(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,uh,vh;
Vh r,rr,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>>ul[];
  ifstream gg("v.txt"); gg>>vl[];
}
```

Figure 3.13: Pressure for a Euler flow around a disk at Mach 2 computed by (3.11)
3.15 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.

\[ F(\varphi(x), \frac{\partial \varphi}{\partial x_1}(x), \ldots, \frac{\partial \varphi}{\partial x_d}(x), \frac{\partial^2 \varphi}{\partial x_1^2}(x), \ldots, \frac{\partial^m \varphi}{\partial x_d^m}(x)) = 0 \quad \forall x \in \Omega \subset \mathbb{R}^d. \]

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

\[ a\varphi + a \cdot \nabla \varphi + B : \nabla(\nabla \varphi) = f \quad \text{in} \quad \Omega \subset \mathbb{R}^d, \]

where \( f(x), a(x) \in \mathbb{R}, a(x) \in \mathbb{R}^d, B(x) \in \mathbb{R}^{d \times d} \) 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, \( \frac{\partial \varphi}{\partial x_i} \) by \( z_i \) and \( \frac{\partial^2 \varphi}{\partial x_i \partial x_j} \) by \( z_i z_j \), where \( z \) is a vector in \( \mathbb{R}^d \):

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

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

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

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

\[ \frac{\partial^2 u}{\partial t^2} - \frac{\partial u}{\partial x} \frac{\partial^2 u}{\partial x^2} = 0, \]

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:

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

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

\[ \frac{\partial \varphi}{\partial t} - \mu \Delta \varphi = f \quad \forall x \in \Omega \subset \mathbb{R}^d, \quad \forall t \in ]0, T[. \]
If $\mu > 0$, the wave equation is hyperbolic:
\[ \frac{\partial^2 \varphi}{\partial t^2} - \mu \Delta \varphi = f \quad \text{in} \quad Q. \]

The convection diffusion equation is parabolic if $\mu \neq 0$ and hyperbolic otherwise:
\[ \frac{\partial \varphi}{\partial t} + a \nabla \varphi - \mu \Delta \varphi = f. \]

The biharmonic equation is elliptic:
\[ \Delta (\Delta \varphi) = f \quad \text{in} \quad \Omega. \]

**Boundary conditions**  A relation between a function and its derivatives is not sufficient to define the function. Additional information on the boundary $\Gamma = \partial \Omega$ of $\Omega$, 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
\[ \varphi \text{ and } \frac{\partial \varphi}{\partial t} \text{ given at } t = 0 \text{ and Dirichlet or Neumann or mixed conditions on } \partial \Omega. \]
3.16 Optimize Time depend schema for Heat equation

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) (see the example Heat.edp in chapter directory).

For example solve the Thermal Conduction problem of section 3.4. We must solve the temperature equation in \( \Omega \) in a time interval \((0,T)\).

\[
\begin{aligned}
\frac{\partial u}{\partial t} - \nabla \cdot (\kappa \nabla u) &= 0 \text{ in } \Omega \times (0,T), \\
u(x,y,0) &= u_0 + xu_1 \\
u &= 30 \text{ on } \Gamma_24 \times (0,T), \\
\kappa \frac{\partial u}{\partial n} + \alpha (u - u_e) &= 0 \text{ on } \Gamma \times (0,T).
\end{aligned}
\] (3.12)

The variational formulation is in \( L^2(0,T; H^1(\Omega)) \); we shall seek \( u^n \) satisfying

\[
\forall w \in V_0 \colon \int_{\Omega} u^n - u^{n-1} w + \kappa \nabla u^n \nabla w + \int_{\Gamma} \alpha (u^n - u_{ue}) w = 0
\]

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 \( u^n, b^n, b', b'', b_d \) (notation if \( w \) is a vector then \( w_i \) is a component of the vector).

\[
u^n = A^{-1}b^n, \quad b' = b_0 + Mu^{-1}, \quad b'' = \frac{1}{\varepsilon}b_d, \quad b^n_i = \begin{cases} b''_i & \text{if } i \in \Gamma_24 \\ b'_i & \text{else} \end{cases}
\]

Where with \( \frac{1}{\varepsilon} = tgv = 10^{30} \):

\[
A_{ij} = \begin{cases} \frac{1}{\varepsilon} & \text{if } i \in \Gamma_24, \text{ and } j = i \\
\int_{\Omega} w_j w_i/\delta t + k(\nabla w_j, \nabla w_i) + \int_{\Gamma_13} \alpha w_j w_i & \text{else} \end{cases}
\]

\[
M_{ij} = \begin{cases} \frac{1}{\varepsilon} & \text{if } i \in \Gamma_24, \text{ and } j = i \\
n \int_{\Omega} w_j w_i/\delta t & \text{else} \end{cases}
\]

\[
b_{0,i} = n \int_{\Gamma_13} \alpha u_{ue} w_i
\]

\[
b_{cl} = u^0 \text{ the initial data}
\]

The Fast version script:

```plaintext
... \\
Vh u0=fu0,u=u0; \\
Create three variational formulation, and build the matrices A,M.

varf vthermic (u,v)= int2d(Th)(u*v/\delta t \\
+ 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/\delta t) + on(2,4,u=1); \\
```

Create three variational formulation, and build the matrices \( A,M \).
3.17. TUTORIAL TO WRITE A TRANSIENT STOKES SOLVER IN MATRIX FORM

\begin{verbatim}
real tgv = 1e30;
matrix A = vthermic(Vh, Vh, tgv=tgv, solver=CG);
matrix M = vMass(Vh, Vh);

Now, to build the right hand size we need 4 vectors.

real[int] b0 = vthermic0(0, Vh); // constant part of RHS
real[int] bcn = vthermic(0, Vh); // tgv on Dirichlet part
    // we have for the node i : i∈Γ_24 ⇔ bcn[i] ≠ 0
real[int] bcl= tgv*u0[]; // the Dirichlet B.C. part

// The fast loop ...
for(real t=0; t<T; t+=dt){
    real[int] b = b0; // for the RHS
    b += M*u[]; // add the the time dependent part
    u[] = A^-1*b;
    plot(u);
}
\end{verbatim}

3.17 Tutorial to write a transient Stokes solver in matrix form

Consider the following script to solve a time dependent Stokes problem in a cavity

\begin{verbatim}
\textbf{mesh} Th=square(10,10);
\textbf{fespace} Vh(Th,P2), Qh(Th,P1);
Vh u,v,uu,vv, uold=0, vold=0;
Qh p,pp;
real nu=0.1, T=1., dt = 0.1;
int m, M= T/dt;
\textbf{problem} stokes(u, v, p, uu, vv, pp)=
    \begin{align*}
    \int2d(Th) \left( (u \cdot uu + v \cdot vv)/dt + \frac{dy(u) \cdot dy(u) + dy(u) \cdot dy(u)}{dt} + \frac{dx(u) \cdot dx(u) + dx(v) \cdot dx(v)}{dt} \\
    - p \cdot pp \cdot 1.e-6 - p \cdot (dx(uu) + dy(vv)) - pp \cdot (dx(u) + dy(v)) \right)
    + \int2d(Th) \left( (uold \cdot uu + vold \cdot vv)/dt \right)
    + \text{on}(1,2,4,u=0,v=0) + \text{on}(3,u=1,v=0);
    \end{align*}

\textbf{for}(m=0; m<M; m++){
    stokes; uold=u; vold=v;
}
plot(p, [u,v], value=true, wait=true, cmm="t="+m*dt);
\end{verbatim}

Every iteration is in fact of the form \( A[u, v, p] = B[uold, vold, pold] + b \) where \( A, B \) are matrices and \( b \) is a vector containing the boundary conditions. The \( A, B, b \) are constructed by

\begin{verbatim}
\textbf{fespace} Xh(Th, [P2,P2,P1]);
\textbf{varf} aa ([u,v,p],[uu,vv,pp])
    = \int2d(Th) \left( (u \cdot uu + v \cdot vv)/dt + \frac{dy(u) \cdot dy(u) + dy(u) \cdot dy(u)}{dt} + \frac{dx(v) \cdot dx(vv) + dy(v) \cdot dy(vv)}{dt} \\
    - p \cdot pp \cdot 1.e-6 - p \cdot (dx(uu) + dy(vv)) - pp \cdot (dx(u) + dy(v)) \right)
    + \text{on}(1,2,4,u=0,v=0) + \text{on}(3,u=1,v=0);

\textbf{varf} bb ([uold,vold,pold],[uu,vv,pp])
\end{verbatim}
\[ \text{varf } bcl ([u_{\text{old}}, v_{\text{old}}, p_{\text{old}}], [u, v, p]) = \text{on}(1,2,4,u_{\text{old}}=0,v_{\text{old}}=0) + \text{on}(3,u_{\text{old}}=0,v_{\text{old}}=0); \]

\text{matrix } A = aa(Xh,Xh, solver=UMFPACK);
\text{matrix } B = bb(Xh,Xh);
\text{real } [\text{int}] b = bcl(0,Xh);

Note that the boundary conditions are not specified in \( b \). Removing the comment “//” would cause the compiler to multiply the diagonal terms corresponding to a Dirichlet degree of freedom by a very large term (tgv); if so \( b \) would not be needed, on the condition that \( u_{\text{old}} = 1 \) on boundary 3 initially. Note also that \( b \) has a tgv on the Dirichlet nodes, by construction, and so does \( A \).

The loop will then be

\text{real } [\text{int}] sol(Xh.n dof), aux(Xh.n dof);
\text{for } (m=0; m<M; m++) {
    aux = B * sol; aux += b;
    sol = A^{-1} * aux;
}

There is yet a difficulty with the initialization of \( sol \) and with the solution from \( sol \). For this we need a temporary vector in \( X_h \) and here is a solution

\text{Xh } [w_1, w_2, w_p] = [u_{\text{old}}, v_{\text{old}}, p];
\text{sol} = w_1[];
\text{for } (m=0; m<M; m++) {
    aux = B * sol; aux += b;
    sol = A^{-1} * aux;
}

\text{w}_1[] = \text{sol}; u = w_1; v = w_2; p = w_p;
\text{plot } (p, [u, v], \text{value=true, wait=true, cmm="t=\"}+m*\text{dt}"});

The freefem team agrees that the line \( \text{sol} = w_1[] \); is mysterious as it copies also \( w_2 \) and \( w_p \) into \( sol \). Structured data such as vectors of \( X_h \) here cannot be written component by component. Hence \( w_1 = u \) is not allowed.
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

```plaintext
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

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

The type declarations are compulsory in FreeFem++; in the end this feature is an asset because it is easy to make bugs in a language with many implicit 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

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

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

**int** declares an integer (i.e. `long` in C++)..

**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”. (i.e. `double` in C++).

**complex** Complex numbers, such as $1 + 2i$, FreeFem++ understand that $i = \sqrt{-1}$ (i.e. `complex<double>` in C++).

```cpp
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;

```plaintext
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.

```cpp
real[int] a(5);
cout << "a = " << a << endl;
```

This produces the output;

```plaintext
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.
4.3. GLOBAL VARIABLES

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

```plaintext
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)
- `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) and if the point is outside then `region == notaregion` where `notaregion` is a freefem++ integer constant.
- `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 and can be used in 3D.
- `N` gives the outward unit normal vector at the current point if it is on a curve defined 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 = |q^i - q^j|` gives the length of the current edge if the current edge is \([q^i, q^j]\)
- `edgeOrientation = \pm 1 : \text{sign of } (i - j)` if the current edge is \([q^i, q^j]\)
- `BoundaryEdge = 1` if the current edge the current edge on true boundary or 0 otherwise
\textbf{InternalEdge} = 0 if the current edge on true boundary or 1 otherwise

\textbf{hTriangle} gives the size of the current triangle

\textbf{nuTriangle} gives the index of the current triangle (int value).

\textbf{nuEdge} gives the index of the current edge in the triangle (int value).

\textbf{nTonEdge} gives the number of adjacent triangle of the current edge (integer).

\textbf{area} give the area of the current triangle (real value).

\textbf{volume} give the volume of the current tetrahedra (real value).

\textbf{cout} is the standard output device (default is console). On MS-Windows, the standard output is only the console, at this time.

\textbf{cin} is the standard input device (default is keyboard).

\textbf{endl} adds an “end of line” to the input/output flow.

\textbf{true} means “true” in \textbf{bool} value.

\textbf{false} means “false” in \textbf{bool} value.

\textbf{pi} is the real value approximation value of $\pi$.

\section*{4.4 System Commands}

Here is how to show all the types, and all the operator and functions of a \texttt{FreeFem++} program:

\begin{verbatim}
 dumptable(cout);
\end{verbatim}

To execute a system command in the string (not implemented on Carbon MacOS), which return the value of the system call.

\begin{verbatim}
 system("shell command");
 exec("shell command");
\end{verbatim}

This is useful to launch another executable from within \texttt{FreeFem++}. On MS-Windows, the full path of the executable. For example, if there is the command “ls.exe” in the subdirectory “c:\cygwin\bin\”, then we must write

\begin{verbatim}
 exec("c:\cygwin\bin\ls.exe");
\end{verbatim}

Another useful system command is \texttt{assert()} to make sure something is true.

\begin{verbatim}
 assert(version>=1.40);
\end{verbatim}
On integers, +, −, ∗ are 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 Computations with the integers

```cpp
int a = 12, b = 5;
int b=0;
real a=sqrt(2.), b = pi;
```

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 extended to real numbers or variables. However, % calculates the remainder of the integer parts of two real numbers.
The following are examples similar to Example 4.2

```cpp
real a=sqrt(2.), b = pi;
```
CHAPTER 4. SYNTAX

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

\[ \text{bool} \subset \text{int} \subset \text{real} \subset \text{complex}, \]

the operators “+”, “−”, “∗”, “/” and “^” are also applicable on complex-typed variables, but “%”, “max”, “min” cannot be used. Complex numbers such as 5+9i, i=√−1, are valid expressions. With real variables a=2.45, b=5.33, complex numbers like a + ib and a + i√2 must be declared by

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

The imaginary and real parts of a complex number z can be obtained with \texttt{imag} and \texttt{real}.

The conjugate of a + bi (a, b are reals) is defined by a − bi, which can also be computed with the operator "conj", by \texttt{conj(a+b*1i)} in FreeFem++. Internally the complex number z = a + ib is considered as the pair (a, b) of real numbers a, b. We can attach to it the point (a, b) in the Cartesian plane where the x-axis is for the real part and the y-axis for the imaginary part. The same point (a, b) has a representation with polar coordinate (r, φ), So z his also z = r(cos φ + i sin φ), r = √a^2 + b^2 and φ = tan^−1(b/a); r is called the \textit{modulus} and φ the \textit{argument} of z. In the following example, we shall show them using FreeFem++ programming, and \textit{de Moivre’s formula} \( z^n = r^n \left( \cos n\phi + i \sin n\phi \right) \).

**Example 4.3**

```cpp
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)<<") = "
```
4.6. STRING EXPRESSION

```cpp
<< 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)

Some usefull function exist also (version 3.51-1):

- projection(a,b,x) = min(max(a,x),b)
- dist(a,b) = sqrt(a^2+b^2)
- dist(a,b,c) = sqrt(a^2+b^2+c^2)
- min(a,b,c)
- max(a,b,c)
- diffpos(a,b) = max(b-a,0)
- invdiffpos(a,b) = a < b?1/(b-a) : 0
- diffnp(a,b) = (a < 0)&(0 < b)?(b-a) : 0
- invdiffnp(a,b) = (a < 0)&(0 < b)?1/(b-a) : 0

4.6 string expression

In the following example you can see some string example expression:

```cpp
string tt="toto1"+1" -- 77"; // string concatenation
string t1="0123456789";
string t2; // new operator
t2 ="12340005678";
t2(4:3) = "abcdefghijk-";
string t55=t2(4:3); // t2 = "12340abcdefghijk-005678"
cout << t2 << endl;
cout << " find abc " << t2.find("abc") << endl;
```
cout << "r find abc " << t2.rfind("abc") << endl;
cout << " find abc from 10 " << t2.find("abc",10) << endl;
cout << " ffind abc from 10 " << t2.rfind("abc",10) << endl;
cout << " " << string("abcc").length << endl;
cout << " t55 " << t55 << endl;
{
    // add getline version 3.0-6 jan 2009 FH
    string s;
    ifstream toto("xyf");
    for (int i=0;i<10;++i)
    {
        getline(toto,s);
        cout << i << " : " << s << endl;
    }
}

4.7 Functions of one Variable

Fundamental functions are built into FreeFem++ as well as The power function \( x^y \) = \texttt{pow}(x,y), the exponent function \( \texttt{exp}(x) = e^x \), the logarithmic function \( \texttt{log}(x) (= \ln x) \) or \( \texttt{log10}(x) (= \log_{10} x) \); the trigonometric functions \( \texttt{sin}(x) \), \( \texttt{cos}(x) \), \( \texttt{tan}(x) \) assume angles measured in radians; the inverse of sin, cos, tan \( x \) (called circular function or inverse trigonometric function ) \( \texttt{asin}(x) (=\arcsin x) \), \( \texttt{acos}(x) (=\arccos x) \), \( \texttt{atan}(x) (=\arctan x) \) are also implemented; the \( \texttt{atan2}(y,x) \) 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 functions,

\[
\sinh x = \left( e^x - e^{-x} \right)/2, \quad \cosh x = \left( e^x + e^{-x} \right)/2.
\]

and \( \tanh x = \sinh x / \cosh x \) called by \( \texttt{sinh}(x) \), \( \texttt{cosh}(x) \), \( \texttt{tanh}(x) \), \( \texttt{sinh}(x) \), \( \texttt{acos}(x) \) and \( \texttt{atanh}(x) \).

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

The real function which rounds a real to an integer \( \texttt{round}(x) \) (resp. \( \texttt{lround}(x) \)) round to integral value, regardless of rounding direction and return real (resp. integer). \( \texttt{floor}(x) \) rounds to largest integral value not greater than \( x \) vye as real, \( \texttt{ceil}(x) \) round to smallest integral value not less than \( x \); similarly \( \texttt{rint}(x) \) returns the integral value nearest to \( x \) (according to the prevailing rounding mode) in floating-point format) and \( \texttt{lrint}(x) \) return an integer. With a have also of the function from \texttt{cmath} : \( \texttt{fmod}(x,y) \), \( \texttt{fmin}(x,y) \), \( \texttt{fmax}(x,y) \), \( \texttt{fdim}(x,y) \) such that \( \texttt{fmod} \) functions compute the floating-point remainder of \( x/y \), \( \texttt{fmin} \) return the min, \( \texttt{fmax} \) return the max, \( \texttt{fdim}() \) functions return the ”positive difference” between their arguments: \( x - y \) if \( x > y \), \( +0 \) if \( x \leq y \).
and by composition \( f(g(x)) \), each applied a finite number of times. In FreeFem++, all elementary functions can thus be created. The derivative of an elementary function is also an elementary function; 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

```plaintext
real b = 1.;
real a = b;
func real phi_x(real t)
{
    return (a+b)*cos(t)-b*cos(t*(a+b)/b);
}
func real phi_y(real t)
{
    return (a+b)*sin(t)-b*sin(t*(a+b)/b);
}
border C(t=0,2*pi) { x=phi_x(t); y=phi_y(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 \( \exp(1+4i) \), \( \sin(\pi+i) \), \( \cos(\pi/2-1i) \) and \( \log(1+2i) \), we then have

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

**Random Functions** can be define as FreeFem++ has a Mersenne Twister function (see page [http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html](http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html) for full detail). It is a very fast and accurate random number generator of period \( 2^{219937} - 1 \), and the functions which calls it 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).
- `randreal53()` 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.
Remark, the plugin \texttt{ffrandom} ((add load "ffrandom") is a interface to \texttt{random}, \texttt{srandom}, \texttt{srandomdev} fonction random generator in the range from 0 to \((2^{31}) - 1\) of the unix libc library (if the function \texttt{srandomdev} do not exist the we use a seed based on the current time).

Remark, now all random function of the gssl are now interfaced in plugin \texttt{gsl} ((add load "gsl") See gsl section \ref{D.1} for more information.

**Library Functions** form the mathematical library (version 2.17).

- the functions \texttt{j0(x)}, \texttt{j1(x)}, \texttt{jn(n,x)}, \texttt{y0(x)}, \texttt{y1(x)}, \texttt{yn(n,x)} are the Bessel functions of first and second kind.
  
The functions \texttt{j0(x)} and \texttt{j1(x)} compute the Bessel function of the first kind of the order 0 and the order 1, respectively; the function \texttt{jn(n,x)} computes the Bessel function of the first kind of the integer order \texttt{n}.
  
The functions \texttt{y0(x)} and \texttt{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 \texttt{x} (expressed as a real); the function \texttt{yn(n,x)} computes the Bessel function of the second kind for the integer order \texttt{n} for the positive integer value \texttt{x} (expressed as a real).

- the function \texttt{tgamma(x)} calculates the \(\Gamma\) function of \texttt{x}. \texttt{lgamma(x)} calculates the natural logarithm of the absolute value of the \(\Gamma\) function of \texttt{x}.

- The \texttt{erf(x)} function calculates the error function, where \(\texttt{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) dt\).
  
The \texttt{erfc(x)} = function calculates the complementary error function of \texttt{x}, i.e. \(\texttt{erfc}(x) = 1 - \texttt{erf}(x)\).

Remark, all special function of the gssl are now interfaced in plugin \texttt{gsl} ((add load "gsl")

### 4.8 Functions of two Variables

#### 4.8.1 Formula

The general form of real functions of two independent variables \(a, b\) is usually written as \(c = f(a, b)\). In FreeFem++, \texttt{x}, \texttt{y} and \texttt{z} are reserved word as explained in in Section \ref{4.3}. So when the two variables of the function are \texttt{x} and \texttt{y}, we may define the function without its argument, for example

\begin{verbatim}
func f=cos(x)+sin(y) ;
\end{verbatim}

Remark that the function type is given by the expression type. The power operator can be used in functions such as \(x^1\), \(y^{0.23}\). In \texttt{func}, we can write an elementary function as follows

\begin{verbatim}
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));
\end{verbatim}
4.8. FUNCTIONS OF TWO VARIABLES

Complex valued functions create functions with 2 variables \( x, y \) as follows,

\[
\text{mesh } Th = \text{square}(20, 20, [-\pi+2*\pi*x, -\pi+2*\pi*y]); \quad // \quad [-\pi, \pi]^2
\]
\[
\text{fespace } Vh(Th, P2);
\]
\[
\text{func } z = x + y*1i; \quad // \quad z = x + iy
\]
\[
\text{func } f = \text{imag}(\text{sqrt}(z)); \quad // \quad f = 3\sqrt{2}
\]
\[
\text{func } g = \text{abs}(\sin(z/10)*\exp(z^2/10)); \quad // \quad g = |\sin z/10\exp z^2/10|
\]
\[
\text{Vh fh = f; plot(fh); } \quad // \quad \text{contour lines of } f
\]
\[
\text{Vh gh = g; plot(gh); } \quad // \quad \text{contour lines of } g
\]

We call also construct **elementary functions of two variables** from elementary functions \( f(x) \) or \( g(y) \) by the four arithmetic operations plus composition applied a finite number of times.

### 4.8.2 FE-functions

Finite element functions are also constructed like elementary functions by an arithmetic formula involving elementary functions. The difference is that they are evaluated at declaration time and FreeFem++ stores the array of its values at the places associated with the degree of freedom of the finite element type. By opposition elementary functions are evaluated only when needed. Hence FE-functions are not defined only by their formula but also by the mesh and the finite element which enter in their definitions. If the value of a FE-function is requested at a point which is not a degree of freedom, an interpolation is used, leading to an interpolation error, while by contrast, an elementary function can be evaluated at any point exactly.

\[
\text{func } f = x^2*(1+y)^3+y^2;
\]
\[
\text{mesh } Th = \text{square}(20, 20, [-2+4*x, -2+4*y]); \quad // \quad \text{square } [-2, 2]^2
\]
\[
\text{fespace } Vh(Th, P1);
\]
\[
\text{Vh fh = f; } \quad // \quad \text{fh is the projection of } f \text{ to } Vh \text{ (real value)}
\]
\[
\text{func } zf = (x^2*(1+y)^3+y^2)*\exp(x+1i*y);
\]
\[
\text{Vh<complex> zh = zf; } \quad // \quad \text{zh is the projection of } zf \text{ to complex value Vh space}
\]

The construction of \( fh (=f_{th}) \) is explained in Section 6.

**Note 4.1** The command \texttt{plot} applies only for real or complex FE-functions (2d or 3d) and not to elementary functions.

Complex valued functions create functions with 2 variables \( x, y \) as follows,

\[
\text{mesh } Th = \text{square}(20, 20, [-\pi+2*\pi*x, -\pi+2*\pi*y]); \quad // \quad [-\pi, \pi]^2
\]
\[
\text{fespace } Vh(Th, P2);
\]
\[
\text{func } z = x + y*1i; \quad // \quad z = x + iy
\]
\[
\text{func } f = \text{imag}(\text{sqrt}(z)); \quad // \quad f = 3\sqrt{2}
\]
\[
\text{func } g = \text{abs}(\sin(z/10)*\exp(z^2/10)); \quad // \quad g = |\sin z/10\exp z^2/10|
\]
\[
\text{Vh fh = f; plot(fh); } \quad // \quad \text{Fig. 4.1 isovalue of } f
\]
\[
\text{Vh gh = g; plot(gh); } \quad // \quad \text{Fig. 4.2 isovalue of } g
\]
4.9 Arrays

An array stores multiple objects, and there are 2 kinds of arrays: The first is similar to vector, i.e. arrays with integer indices and the second type is arrays with string indices. In the first case, the size of the array must be known at execution time, and implementation is done with the $\text{KN<>()}$ class and all the vector operator of $\text{KN<>()}$ are implemented. For instance

```cpp
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;
<< tab.n « " min: " « tab.min « " max: " « tab.max
<< " sum : " « tab.sum « « endl; //

// change the size of array tab
// to 12 with preserving first value
tab.resize(12);

// set unset value
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;

int [int] ii(0:d.n-1); // set array ii to 0,1,...,d.n-1
d=-1:-5; // set d to -1,-2,..,-5
```

4.9.1 Arrays with integer indices

Example of an implementation for $\text{KN<>()}$

```cpp
real [int] tab(10); // 2 array of 10 real
real [int] tab2; // bug array with no size

tab = 1.03; // set all the array to 1.03

// change the size of array tab
// to 12 with preserving first value
```

4.9.2 Arrays with string indices

Example of an implementation for $\text{KN<>()}$

```cpp
real [string] tt["*"]=1.5;
cout << tt["a"] << " " << tt["*"] << endl;

// set unset value
real [int] a(5), b(5), c(5), d(5);
a = 1;
b = 2;
c = 3;
```

4.9.3 Accessing array elements

Example of accessing array elements in $\text{KN<>()}$

```cpp
// for i = 0, n-1 : d[i] = a[i] ? b[i] : c[i],
cout « " d = (a ? b : c) is " << d << endl;
```

4.9.4 Slicing arrays

Example of slicing arrays in $\text{KN<>()}$

```cpp
int [int] ii(0:d.n-1); // set array ii to 0,1,...,d.n-1
```

4.9.5 Sorting arrays

Example of sorting arrays in $\text{KN<>()}$

```cpp
tab.sort ; // sort the array tab (version 2.18)
```

4.9.6 Vector-like arrays

Example of vector-like arrays in $\text{KN<>()}$

```cpp
int [int] ii(0:d.n-1); // set array ii to 0,1,...,d.n-1
```
sort(d, ii); // sort array d and ii in parallel
cout << " d " << d << "\n ii = " << ii << endl;
produces 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
  3.14 3.14
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

Arrays can be set like in matlab or scilab with the 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 ⌊(b−a)/c⌋ + 1 and the
value i is set by a + i(b − a)/c.
There are int, real, complex arrays with, in the third case, two operators (.in, .re) to
generate the real and imaginary real array from the 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 0
2

the all integer array operators are:

```
b . /= a; // same b = b . / a

cout << "b . /= a; b = " << b << endl;
c = a+b;
cout << " c = a+b; c = " << c << endl;
c = 2*a+4*b;
cout << " c = 2*a+4*b; c = " << c << endl;
c = a+4*b;
cout << " c = a+4*b; c = " << c << endl;
c = -a+4*b;
cout << " c = -a+4*b; c = " << c << endl;
c = -a-4*b;
cout << " c = -a-4*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 = a . / b;
cout << " c = a . / b; c = " << c << endl;
c = 2 * b;
cout << " c = 2 * b; c = " << c << endl;
c = b*2;
cout << " c = b*2; c = " << c << endl;

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

// ---- the methods ----
cout << " || a ||_1 = " << a.l1 << endl;
cout << " || a ||_2 = " << a.l2 << endl;
cout << " || a ||_\infty = " << a.linfty << endl;
cout << " sum a_i = " << a.sum << endl;
cout << " max a_i = " << a.max << " a[ " << a.imax << " ] = " << a[a.imax] << endl;
cout << " min a_i = " << a.min << " a[ " << a.imin << " ] = " << a[a.imin] << endl;

cout << " a'*a = " << (a' *a) << endl;  // a quantile 0.2 = " << a.quantile(0.2) << endl;

int[int] I=[2,3,4,-1,3];
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 << " c(I) = a " << c << endl;
c(I) += a;  // for( i=0;i<I.n;i++) if(I[i] >=0) C(I[i])+=a[i];
cout << " b = a(I) : " << b << " c(I) = a " << c << endl;
}

produce the output

5
3
3
2
3
3
\[ a = 5 \\
\]
\[ b = a + a : 5 \]
\[ b += a : 5 \]
\[ b += 2*a : 5 \]
\[ b /= 2 : 5 \]
\[ b.*=a; b =5 \]
\[ b./=a; b =5 \]
\[ c =a+b : c=5 \\
\]
\[ c =2*a+4b : c= 5 \]
\[ c =a+4b : c= 5 \]
\[ c =-a+4b : c= 5 \]
\[ c =-a-4b : c= 5 \]
\[ c =-a-b : c= 5 \]
\[ c =a.*b : c= 5 \]
\[ c =a./b : c= 5 \]
\[ c =2*b : c= 5 \]
\[ c =b*2 : c= 5 \]
\[ ||a||_1 = 13 \]
\[ ||a||_2 = 6.403124237 \]
\[ ||a||_\infty = 4 \]
4.9. ARRAYS

\[
\begin{align*}
\text{sum } a_i &= 13 \\
\text{max } a_i &= 4 \text{ a[3] = 4} \\
\text{min } a_i &= 1 \text{ a[1] = 1} \\
a' &= 41 \\
a \text{ quantile 0.2} &= 2 \\
b &= a(I) : 5 \\
2 & 4 & 4 & -3 & 4 \\
c(I) &= a 5 \\
-3 & -3 & 2 & 4 & 2 \\
b &= a(I) : 5 \\
2 & 4 & 4 & -3 & 4 \\
c(I) &= a 5 \\
-3 & -3 & 4 & 9 & 4 \\
\end{align*}
\]

**Note 4.2** Quantiles are points taken at regular intervals from the cumulative distribution function of a random variable. Here the array values are random. This statistical function \(a.\text{quantile}(q)\) computes \(v\) from an array \(a\) of size \(n\) for a given number \(q \in [0, 1]\) such that

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

it is equivalent to \(v = a[q \times n]\) 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 mapping is not imaged, so the value is not set.

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

The output is

\[
\begin{align*}
b &= a(I) : 5 \\
2 & 4 & 4 & -3 & 2 \\
c(I) &= a 5 \\
4 & -3 & 2 & 1 & 2
\end{align*}
\]
4.9.1 Arrays with two integer indices versus matrices

Some examples are given below to transform full matrices into sparse matrices.

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

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

b=[1,2,3];

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

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

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

// this transforms an array into a sparse matrix
matrix B;
B = A;
B=A(I,J); // B(i,j)= A(I(i),J(j))
B=A(I^-1,J^-1); // B(I(i),J(j))= A(i,j)

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

the output is

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

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

\[
A = \begin{bmatrix}
1 & 5 & 2 & 1 \\
3 & 3 & 3 & 1 \\
4 & 5 & 2 & 4
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
-50 & -40 \\
-100 & -80 \\
-150 & -120 \\
-200 & -160
\end{bmatrix}
\]

\[
(-50,-40) \quad (-100,-80) \quad (-150,-120) \quad (-200,-160)
\]

\[
(-100,-80) \quad (-200,-160) \quad (-300,-240) \quad (-400,-320)
\]

\[
(-150,-120) \quad (-300,-240) \quad (-450,-360) \quad (-600,-480)
\]

\[
A = \begin{bmatrix}
8 & 10 & 12 & 14 \\
16 & 20 & 24 & 28 \\
24 & 30 & 36 & 42
\end{bmatrix}
\]

4.9.2 Matrix construction and setting

- To change the linear system solver associated to a matrix do

  \[
  \text{set}(M, \text{solver} = \text{sparsesolver});
  \]

  The default solver is GMRES.

- from a variational form: (see section 6.12 page 195 for details)

  \[
  \text{varf} \quad vDD(u,v) = \int 2d(Thm)(u \times v * 1e-10);
  \]

  \[
  \text{matrix} \quad DD = vDD(Lh,Lh);
  \]

- To set from a constant matrix

  \[
  \text{matrix} \quad A =
  \begin{bmatrix}
  0 & 1 & 0 & 10 \\
  0 & 0 & 2 & 0 \\
  0 & 0 & 0 & 3 \\
  4 & 0 & 0 & 0
  \end{bmatrix}
  \]

- To set from a block matrix

  \[
  \text{matrix} \quad M =
  \begin{bmatrix}
  \text{Asd}[0],0 ,0 ,0 ,\text{Csd}[0] \\
  0 ,\text{Asd}[1],0 ,0 ,\text{Csd}[1] \\
  0 ,0 ,\text{Asd}[2],0 ,\text{Csd}[2] \\
  0 ,0 ,0 ,\text{Asd}[3],\text{Csd}[3] \\
  \text{Csd}[0]’,\text{Csd}[1]’,\text{Csd}[2]’,\text{Csd}[3]’,\text{DD}
  \end{bmatrix}
  \]

    // to now to pack the right hand side

  \[
  \text{real} \quad \text{bb} = [\text{rhssd}[0][] , \text{rhssd}[1][] , \text{rhssd}[2][] , \text{rhssd}[3][] , \text{rhsl}[] ];
  \]

  \[
  \text{set}(M, \text{solver} = \text{sparsesolver});
  \]

  \[
  xx = M^{-1} \times \text{bb};
  \]

  \[
  [\text{usd}[0][] , \text{usd}[1][] , \text{usd}[2][] , \text{usd}[3][] , \text{lh}[] ] = xx;
  \]

  // to dispatch

  // the solution on each part.
where Asd and Csd are arrays of matrices (from example mortar-DN-4.edp of examples++-tutorial).

- To set or get all the indices and coefficients of the sparse matrix $A$, let $I, J, C$ be respectively two int[int] arrays and a real[int] array. The three arrays define the matrix as follows

$$A = \sum_k C[k] M[I[k], J[k]]$$

where $M_{ab} = (\delta_{ia}\delta_{jb})_{ij}$

one has: $M_{ab}$ a basic matrix with the only non zero term $m_{ab} = 1$.

One can write $[I, J, C] = A$; to get all the term of the matrix $A$ (the arrays are automatically resized), and $A = [I, J, C]$; to change all the term matrices. Note that the size of the matrix is with $n = I\.\text{max}$ and $m = J\.\text{max}$. Remark that $I, J$ is forgotten to build a diagonal matrix, and similarly for the $n, m$ of the matrix.

- matrix renumbering

\[
\begin{align*}
\text{int[int]} & \ I(15), J(15); \quad // \ two \ array \ for \ renumbering \\
\text{matrix} & \ B;
\end{align*}
\]

\[
\begin{align*}
B & = A; \quad // \ \text{copie matrix} \ A \\
B & = A(I, J); \quad // \ B(i, j) = A(I(i), J(j)) \\
B & = A(I^{-1}, J^{-1}); \quad // \ B(I(i), J(j)) = A(i, j) \\
B & .\text{resize}(10, 20); \quad // \ \text{resize the sparse matrix and remove out of bound terms}
\end{align*}
\]

where $A$ is a given matrix.

- complex versus real sparse matrix:

\[
\begin{align*}
\text{matrix<complex>} & \ C = \text{vv}(Xh, Xh); \\
\text{matrix} & \ R = \text{vr}(Xh, Xh); \\
\text{matrix<complex>} & \ CR = R; C = R; \quad // \ create \ or \ copy \ real \ matrix \ to \ complex \ matrix
\end{align*}
\]

\[
\begin{align*}
R & = C\.\text{im}; R = C\.\text{re}; \quad // \ \text{get real or imagery part of complex sparse matrix} \\
\text{matrix} & \ CI = C\.\text{im}, CR = C\.\text{re}; \quad // \ \text{get real or imagery part of complex sparse matrix}
\end{align*}
\]

### 4.9.3 Matrix Operations

The multiplicative operators $\ast$, $\div$, and $\%$ group left to right.

- ‘ is the (unary) right transposition for arrays, the matrix in real cases and Hermitian transpose in complex cases.

- .\* is the term to term multiply operator.
4.9. ARRAYS

- \(./\) is the term to term divide operator.

there are some compound operators also:

- \(^{-1}\) 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 \(a' \ast b = \overline{a}^T b\).

- \(a \ast b'\) is the outer product (example matrix \(B = a' \ast b\))

Example 4.5

```plaintext
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*i*x);
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;

For the second case, it is just a map of the STL\(^1\) so no operations on vector are allowed, except the selection of an item.
The transpose or Hermitian conjugation operator is ‘ as in Matlab or Scilab, so the way to compute the dot product of two array \(a,b\) is real \(ab=a'*b\).
The resizing of a sparse matrix \(A\) is also allowed:

\[
A \text{.resize}(10,100);
\]

Note that the new size can be greater or smaller than the previous size; all new term are set to zero.

On the triangulation of Figure 2.4, this produces the following:

\[
A = \begin{bmatrix}
10^{30} & 0.5 & 0. & 30. & -2.5 & 0. \\
0. & 10^{30} & 0.5 & 0. & 5 & -2.5 \\
0. & 0. & 10^{30} & 0. & 0. & 0.5 \\
0.5 & 0. & 0. & 10^{30} & 0. & 0. \\
-2.5 & 0.5 & 0. & 5 & 10^{30} & 0. \\
0. & -2.5 & 0. & 0. & 0.5 & 10^{30}
\end{bmatrix}
\]

\[
\{v\} = f[] = (0 \ 0 \ 0 \ 0 \ 0.5 \ 1)^T
\]

\[
\{w\} = g[] = (0 \ 1 \ 1.2 \times 10^{-16} \ 0 \ 1 \ 1.2 \times 10^{-16})
\]

\[
A * f[] = (-1.25 \ -2.25 \ 0.5 \ 0 \ 5 \times 10^{29} \ 10^{30})^T \quad (= A\{v\})
\]

\[
A'*f[] = (-1.25 \ -2.25 \ 0.25 \ 5 \times 10^{29} \ 10^{30})^T \quad (= A^T\{v\})
\]

\[
f[] \ast g[] = (0 \ 0 \ 0 \ 0.5 \ 1.2 \times 10^{-16})^T = (v_1w_1 \ \cdots \ v_Mw_M)^T
\]

\[
f[] / g[] = (-NaN \ 0 \ 0 \ -NaN \ 0.5 \ 8.1 \times 10^{15})^T = (v_1/w_1 \ \cdots \ v_M/w_M)^T
\]

\[
f[]' \ast g[] = 0.5 \quad (= \{v\}^T\{w\} = \{v\} \cdot \{w\})
\]

The output of the \(I,J,C\) array:

\[
I = 18
\]

\[
0 \ 0 \ 0 \ 1 \ 1
1 \ 1 \ 2 \ 2 \ 3
3 \ 4 \ 4 \ 4 \ 4
5 \ 5 \ 5
\]

\[
J = 18
\]

\[
0 \ 1 \ 4 \ 1 \ 2
4 \ 5 \ 2 \ 5 \ 0
3 \ 0 \ 1 \ 3 \ 4
1 \ 4 \ 5
\]

\[
C = 18
\]

\[
1e+30 \ 0.5 \ -2.5 \ 1e+30 \ 0.5
0.5 \ -2.5 \ 1e+30 \ 0.5 \ 0.5
\]

\(^1\)Standard template Library, now part of standard C++
The output of a diagonal sparse matrix \( D \) (Warning due to fortran interface the indices start on the output at one, but in FreeFem++ in index as in C begin at zero):

\[
\begin{array}{c|c|c|c|c}
1e+30 & -2.5 & 0.5 & 0.5 & 1e+30 \\
-2.5 & 0.5 & 1e+30 & & \\
\end{array}
\]

\( D = \# \) 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\}

\[
\begin{array}{c|c|c|c|c}
1 & 1 & 1.0000000000000000199e+30 \\
2 & 2 & 1.0000000000000000199e+30 \\
3 & 3 & 1.0000000000000000199e+30 \\
4 & 4 & 1.0000000000000000199e+30 \\
5 & 5 & 1.0000000000000000199e+30 \\
6 & 6 & 1.0000000000000000199e+30 \\
\end{array}
\]

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

\[
\text{matrix} \quad \text{AAA} = \text{A}^{-1};
\]

In examples++-load/lapack.edp a full matrix is inverted using the lapack library and this small dynamic link interface (see for more detail section C page 371).

load "lapack"
load "fflapack"
int n=5;
real[int,int] A(n,n),Al(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; // def in load "lapack"
Al=A^{-1};
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)*Al(k,j);
cout << B << endl; // A1+A^{-1}; attention ne marche pas

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

and the output is:

\[
\begin{array}{c|c|c|c|c}
5 & 5 & 6 & 1 & 1 1 1 1 \\
1 & 6 & 1 & 1 & 1 1 1 1 \\
1 & 1 & 6 & 1 & 1 1 6 1 \\
1 & 1 & 1 & 6 & 1 1 1 6 \\
1 & 1 & 1 & 1 & 6 \\
\end{array}
\]

error: dgesv_ 0
5 5
to compile lapack.cpp or fflapack.cpp you must have the library lapack on you system and try in directory examples++-load

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

4.9.4 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. , the syntax for space or vector finite function is

```cpp
int n = 100; // size of the array.
Vh[int] wh(n); // real scalar case
Wh[int] [uh,vh](n); // real vectorial case
Vh<complex>[int] cwh(n); // complex scalar case
Wh<complex>[int] [cuh,cvh](n); // complex vectorial case
[cuh[2],cvh[2]]= [x,y]; // set interpolation of index 2.
```

```cpp
// Array of Array
real [int][int] V(10); //
matrix[int] B(10); //
real [int,int][int] A(10); //
```

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.

```cpp
mesh Th=square(20,20,[2*x,2*y]);
fespace Vh(Th,P1);
```
4.10. Map arrays

For the second case, it is just a map of the STL\(^{31}\) so no operations on vector are allowed, except the selection of an item.

```cpp
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;
```

4.11 Loops

The for and while loops are implemented in FreeFem++ together 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.

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

An example below shows a sum from 1 to 10 with result is in \texttt{sum},

```cpp
int sum=0;
for (int i=1; i<=10; i++)
    sum += i;
```
sum += i;
The while-loop

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

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

```c
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**

```c
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;
}
```

From version 3.43, we have new kind of implicit loop to set array, map, sparse matrices. The syntaxe is

- **for** [i,ai : a] for array with one indice, so if `real[int] a(10)`, then i= 0 to 9 and ai is a reference to **a[i]**, same for `real[string] a(10), ...`

- **for** [i,j,aij : a] for array with two indice, so if `real[int,int] a(10,11)`, then i= 0 to 9, j= 0 to 10 and aij is a reference to **a(i,j)**, same for `matrix a(10), ...`

**Example 4.8**

```c
real [int,int] a(10,10);
real [int] b(10);
for [i,bi : b] {bi=i+1; cout << i << " " << bi << endl;}
cout << " b="<< b<< endl;
for [i,j,aij : a]
  { aij= 1./(2+i+j);}
```
4.12. **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|binary);` means opening a file in append and binary mode.

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

**Example 4.9**

```cpp
int i;

// Close the file f because the variable f is delete
```
Some functions are available to format the output.

- \texttt{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.

- \texttt{f.scientific} Formats floating-point numbers in scientific notation (\texttt{d.dddE\pm dd})

- \texttt{f.fixed} Used fixed point notation (\texttt{d.ddd}) for floating-point numbers. Opposite of scientific.

- \texttt{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.

- \texttt{f.noshowbase} unset showbase flags

- \texttt{f.showpos} inserts a plus sign (+) into a decimal conversion of a positive integral value.

- \texttt{f.noshowpos} unset showpos flags

- \texttt{f.default} reset all the previous flags (fmtflags) to the default expect precision.

- \texttt{setw(n)} Behaves as if member width were called with \texttt{n} as argument on the stream on which it is inserted as a manipulator (it can be inserted on output streams).

Where \texttt{f} is output stream descriptor, for example \texttt{cout}.

Remark, all these methods except the first return the stream \texttt{f}, so they can be chained as in

\begin{verbatim}
cout.scientific.showpos << 3 << endl;
\end{verbatim}

An usefull example to see usage of some function on file stream:

\begin{verbatim}
ofstream file("f.txt",binary); // for windows add binary version 3.30
file.precision(16);
file << f[] << endl;
where=file.tellp();
file << 0.1 ; // file << " " << 0.2 ;

cout << " where in file " << where << endl;
file << " # comment bla bla ... 0.3 \n";
file << 0.2 << endl;
\end{verbatim}
4.12. INPUT/OUTPUT

```cpp
file.flush; // to flush the buffer of file
}
// Idea to skip comment in a file ... start with # too EOL
func ifstream skipcomment(ifstream &ff)
{
    while(1)
    {
        int where = ff.tellg(); // store file position
        string comment;
        ff >> comment;
        if ( ! ff.good() ) break;
        if( comment(0:0)=="#") {
            getline(ff,comment);
            cout << " -- #" << comment << endl;
        }
        else {
            ff.seekg(where); // restore file position
            break;
        }
    }
    return ff;
}
{ 
    ifstream file("f.txt",binary); // for windows (pb CRNL EOL translation )
    cout << " where " << file.seekg << endl;
    file.seekg(where);
    cout << " xx = " << xx << " good ? " << file.good() << endl;
    assert(xx==0.1);
    skipcomment(file) >> xx;
    assert(xx==0.2);
    file.seekg(0); // rewind ...
    cout << " where " << file.tellg() << " " << file.good() << endl;
    file >> g[] ;
}

4.12.1 Script arguments

There is a very useful predefined array in FreeFem++ ARGV that contains all the arguments of the script used in the command line. The following code prints out the first three of these arguments:

```cpp
// version 3.8-1
for(int i=0;i<ARGV.n;++i)
{
    cout << ARGV[i] << endl;
}
```

The parameter -Dmacroname=macrovalue of ARGV is a simple way to add macro the the script like in "C++".

And to get argument unused in getARGV.idp include script file,
getARGV(n,defaultvalue)  // get the nth parameter unused
getARGV(after,defaultvalue) // get the arg after the string after if exist

The type of default value can be int, real, string,

4.13 **preprocessor**

The preprocessor handles directives for source file inclusion (include ”script-name.idp”), macro definitions.

There are two types of macros, object-like and function-like. Object-like macros do not take parameters; function-like macros do. The generic syntax for declaring an identifier as a macro of each type is, respectively,

```plaintext
macro <identifier>() <replacement token list>    // EOM
```  
```plaintext
macro <identifier>(<parameter list>) <replacement token list>    // EOM
```

An example of macro without parameter

```plaintext
macro xxx() {real i=0;int j=0;cout << i << " " << j << endl;}    // xxx /* replace xxx by the <replacement token list> */
```

The FreeFem++ code associated:

1 : // macro without parameter
2 : macro xxx {real i=0;int j=0;cout << i << " " << j << endl;} //
3 :
4 : {real i=0;int j=0;cout << i << " " << j << endl;}

An example of macro parameter

```plaintext
macro toto(i) i    // quoting parameter the {} are remove
toto({real i=0;int j=0;cout << i << " " << j << endl;})    // and only one level of {} are remove
toto({{real i=0;int j=0;cout << i << " " << j << endl;}})
```

The FreeFem++ code created:

6 : macro toto(i) i //
8 : // quoting parameter the \{\} are remove
9 : real i=0;int j=0;cout << i << " " << j << endl;
10 : // and only one level of \{\} are remove
11 : {real i=0;int j=0;cout << i << " " << j << endl;}

Use a macro as parameter of macro to transforme full matrix in formal array like in :

```plaintext
real[int,int] CC(7,7),EE(6,3),EEps(4,4);
```

```plaintext
macro VIL6(v,i) [ v(1,i), v(2,i),v(4,i), v(5,i),v(6,i) ]    // EOM
```

```plaintext
macro VIL3(v,i) [ v(1,i), v(2,i) ]    // EOM
```

Use a macro as parameter of macro to transforme full matrix in formal array like in :
4.13. PREPROCESSOR

\texttt{macro VV6(v,vv) [ v(vv,1), v(vv,2), v(vv,4), v(vv,5), v(vv,6) ] \quad // EOM}
\texttt{macro VV3(v,vv) [ v(vv,1), v(vv,2) ] \quad // EOM}

\texttt{func C5x5 = VV6(VIL6,CC);}
\texttt{func E5x2 = VV6(VIL3,EE);}
\texttt{func Eps = VV3(VIL3,EEps);}

The \texttt{FreeFem++} code created:

\begin{verbatim}
16: real[int,int] CC(7,7),EE(6,3),EEps(4,4);
17:
18: macro VIL6(v,i ) [ v(1,i), v(2,i),v(4,i), v(5,i),v(6,i) ] \quad // EOM
19: macro VIL3(v,i ) [ v(1,i), v(2,i) ] \quad // EOM
20: // apply v on array element:
21: macro VV6(v,vv ) [ v(vv,1), v(vv,2), v(vv,4), v(vv,5),v(vv,6) ] \quad // EOM
22: macro VV3(v,vv ) [ v(vv,1), v(vv,2) ] \quad // EOM
23: // so formal matrix to build problem..
24: func C5x5 =
25: 1 : [ [ CC(1,1), CC(2,1),CC(4,1), CC(5,1),CC(6,1) ]
26: 1 : [ CC(1,2), CC(2,2),CC(4,2), CC(5,2),CC(6,2) ] ,
27: 1 : [ CC(1,4), CC(2,4),CC(4,4), CC(5,4),CC(6,4) ] ,
28: 1 : [ CC(1,5), CC(2,5),CC(4,5), CC(5,5),CC(6,5) ] ,
29: 1 : [ CC(1,6), CC(2,6),CC(4,6), CC(5,6),CC(6,6) ] ] ;
30: func E5x2 =
31: 1 : [ [ EE(1,1), EE(2,1) ] , [ EE(1,2), EE(2,2) ] ,
32: 1 : [ EE(1,4), EE(2,4) ] , [ EE(1,5), EE(2,5) ] ,
33: 1 : [ EE(1,6), EE(2,6) ] ] ;
34: func Eps = [ [ EEps(1,1), EEps(2,1) ] ,
35: [ EEps(1,2), EEps(2,2) ] ] ;
36: finally the operator \# to do concatenation of parameter: to build vectorial operation, like in
\end{verbatim}

\texttt{macro div(u) (dx(u#1)+dy(u#2)) \quad // EOM}
\texttt{mesh Th=square(2,2); fespace Vh(Th,P1);
Vh v1=x,v2=y;
cout << int2d(Th)(div(v)) << endl;}

The \texttt{FreeFem++} code created:

\begin{verbatim}
31: macro div(u ) (dx(u#1)+dy(u#2)) \quad //EOM
32: mesh Th=square(2,2); fespace Vh(Th,P1);
33: Vh v1=x,v2=y;
34: cout << int2d(Th)( (dx(v1)+dy(v2)) ) << endl;
\end{verbatim}

And to finish a amazing test to verified the quoting:

\texttt{macro foo(i,j,k) i j k \quad // EOM}
foo(,,) 
// empty line
foo( {int [,}, {int} a(10),{ });}

the result:

36 : macro foo(i,j,k ) i j k//EOM
37 : // empty line
38 : int [ int] a(10 );

To defined macro in a macro you can use the two new word NewMacro, EndMacro key
word to set and and close de macro definition (version 3.11, and not well tested).
In version 3.51-2, we test in macro generation like #ifdef ... #endif in C++, the
syntaxe is :

```
macro AA tt //

IFMACRO(AA,tt)
// if AA == tt
cout << " AA " << Stringification(AA) << endl;
macro f1 cas1 //
ENDIFMACRO

IFMACRO(AA,bb)
// if AA == bb
macro CAS 2 //
ENDIFMACRO

IFMACRO(DIR)
// macro DIR exist
include Stringification(DIR/f1.edp)
ENDIFMACRO
```

the output is

Load: lg_fem lg_mesh lg_mesh3 eigenvalue

1 : macro AAtt //
2 :
3 :
1 : // if AA == tt
2 : cout << " AA " << Stringification(tt) << endl;
3 : macro f1cas1 //
4 :
5 :
6 :
1 : // macro DIR exist
2 : include/ Stringification(./cas1.edp)cout << "f1.edp" << endl;
3 :
7 : sizestack + 1024 =1072 ( 48 )


4.14 Exception handling

In the version 2.3 of FreeFem++, exception handling was added as in C++. But today only the C++ exceptions are caught. Note that in C++ all the errors attached to ExecError, assert, exit, ... call exceptions too so it may be hard to find the cause of the error. The exceptions handle all ExecError:

Example 4.10 A simple example: catch a division by zero:

```cpp
real a;
try {
  a=1./0.;
}
catch (...) { // in versions > 2.3 all exceptions can be caught
  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
```

Example 4.11 : a more realistic example with a none invertible matrix:

```cpp
int nn=5
```

And FILE, LINE, Stringification() to get line number and edp filename, add quote to a parameter after macro generation? Like in

```cpp
cout << "in " << FILE << " line " << LINE << " -- '" << Stringification("zzz" aa ({} + /* */ bb cc) << "'" << endl;
```

Remark, you can see the example [Stokes-macro2d-3d.edp in tutorial](tutorial) for a true usage of this functionality.
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) = // definition 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;
    plot(uh);
    cout << "-- lap Cholesky " << nn << "x" << nn << " : " << -cpu+clock()
        << " s, max =" << uh[].max << endl;
}

catch(...) {
    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

Let us begin with the two important keywords `border` and `buildmesh`

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

5.1.1 Square

The command “`square`” triangulates the unit square. The following

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

generates a $4 \times 5$ grid in the unit square $[0,1]^2$. The labels of the boundaries are shown in Fig. 5.1. To construct a $n \times m$ grid in the rectangle $[x_0,x_1] \times [y_0,y_1]$, proceeds as follows:

```plaintext
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]);
```

Figure 5.1: Boundary labels of the mesh by `square(10,10)`
Note 5.1 Adding the named parameter flags=icase with icase:

0 will produce a mesh where all quads are split with diagonal $x - y = cte$

1 will produce Union Jack flag type of mesh.

2 will produce a mesh where all quads are split with diagonal $x + y = cte$ (v 3.8)

3 same as case 0 except in two corners such that no triangle with 3 vertices on boundary (v 3.8)

4 same as case 2 except in two corners 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 will change the 4 default label numbers to labs[i-1], for example `int[int] labs={11,12,13,14}`, and adding the named parameter region=10 will change the region number to 10, for instance (v 3.8).

To see all these flags at work, try the file 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

Boundaries are defined piecewise by parametrized curves. The pieces can only intersect at their endpoints, but it is possible to join more than two endpoints. This can be used to structure the mesh if an area touches a border and create new regions by dividing larger ones:

```
int upper = 1;
int others = 2;
int inner = 3;

border C01(t=0,1){x = 0; y = -1+t; label = upper;}
border C02(t=0,1){x = 1.5-1.5*t; y = -1; label = upper;}
border C03(t=0,1){x = 1.5; y = -t; label = upper;}
border C04(t=0,1){x = 1+0.5*t; y = 0; label = others;}
border C05(t=0,1){x = 0.5+0.5*t; y = 0; label = others;}
border C06(t=0,1){x = 0.5*t; y = 0; label = others;}
border C11(t=0,1){x = 0.5; y = -0.5*t; label = inner;}
border C12(t=0,1){x = 0.5+0.5*t; y = -0.5; label = inner;}
```
5.1. COMMANDS FOR MESH GENERATION

```plaintext
border C13(t=0,1){x = 1; y = -0.5+0.5*t; label = inner;)

int n = 10;
plot(C01(-n)+C02(-n)+C03(-n)+C04(-n)+C05(-n)+C06(-n)+
    C11(n)+C12(n)+C13(n), wait=true);

mesh Th = buildmesh(C01(-n)+C02(-n)+C03(-n)+C04(-n)+C05(-n)+C06(-n)+
    C11(n)+C12(n)+C13(n));

plot(Th, wait=true); // figure 5.3

cout << "Part 1 has region number " << Th(0.75, -0.25).region << endl;
cout << "Part 2 has region number " << Th(0.25, -0.25).region << endl;
```

Figure 5.2: Multiple border ends intersect

Figure 5.3: Generated mesh

Triangulation keywords assume that the domain is defined as being on the left (resp right) of its oriented parameterized boundary

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

To check the orientation plot \( t \mapsto (\varphi_x(t), \varphi_y(t)), t_0 \leq t \leq t_1 \). If it is as in Fig. 5.4 then the domain lies on the shaded area, otherwise it lies on the opposite side.

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

```plaintext
mesh Mesh_Name = buildmesh(\Gamma_1(m_1) + \cdots + \Gamma_J(m_J) OptionalParameter);
```

where \( m_j \) are positive or negative numbers to indicate how many vertices should be 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.
- `fixedborder=<bool value>` , to say if the mesh generator can change the boundary mesh or not (by default the boundary mesh can change; beware that with periodic boundary conditions (see 6), it can be dangerous).
The orientation of boundaries can be changed by changing the sign of $m_j$. The following example shows how to change the orientation. The example generates the unit disk with a small circular hole, and assign “1” to the unit disk (“2” to the circle inside). The boundary label must be non-zero, but it can also be omitted.

```plaintext
1: border a(t=0,2*pi){ x=cos(t); y=sin(t); label=1;}
2: border b(t=0,2*pi){ x=0.3+0.3*cos(t); y=0.3*sin(t); label=2;}
3: plot(a(50)+b(+30)) ; // to see a plot of the border mesh
4: mesh Thwithouthole= buildmesh(a(50)+b(+30));
5: mesh Thwithhole = buildmesh(a(50)+b(-30));
6: plot(Thwithouthole,wait=1,ps="Thwithouthole.eps"); // figure 5.5
7: plot(Thwithhole,wait=1,ps="Thwithhole.eps"); // figure 5.6
```

**Note 5.2** Notice that the orientation is changed by “b(-30)” in 5th line. In 7th line, `ps="fileName"` is used to generate a postscript file with identification shown on the figure.
5.1. COMMANDS FOR MESH GENERATION

5.1.3 Multi-Border

(Version 3.30)
Some time it can be useful to make array of border, but unfortunately it is incompatible with the FreeFem++ syntax. So to bypass this problem, the idea is small, if the number of segment of the discretization $n$ is a array, we make a implicit loop on all the value of the array, and the index variable $i$ of the loop is defined after parameter definition, like in border $a(t=0,2\pi;i)$ ....

A first very small example:

1: \begin{verbatim}
  border a(t=0,2*pi;i){ x=(i+1)*cos(t); y=(i+1)*sin(t); label=1;}
\end{verbatim}

2: \begin{verbatim}
  int[int] nn=[10,20,30];
\end{verbatim}

3: \begin{verbatim}
  plot(a(nn)); // plot 3 circle with 10,20,30 points ..
\end{verbatim}

And more complex exemple (take from mesh.edp example ) to defined a square with small circles:

\begin{verbatim}
// multi border syntax version 3.30 avril 2014 ...
real[int] xx=[0,1,1,0], yy=[0,0,1,1]; // radius, centre of the 4 circles ..
real[int] RC=[ 0.1, 0.05, 0.05, 0.1],
        XC= [0.2,0.8,0.2,0.8],
        YC= [0.2,0.8,0.8,0.2];
int[int] NC=[-10,-11,-12,13]; // list number of ± segments
    // of the 4 circles borders

border bb(t=0,1;i)
{
  // i is the index variable of the multi border loop
  int ii = (i+1)%4; real t1 = 1-t;
  x = xx[ii]*t1 + xx[ii]*t;
  y = yy[ii]*t1 + yy[ii]*t;
  label = 0;
}

border cc(t=0,2*pi;i)
{
  x = RC[i]*cos(t)+XC[i];
  y = RC[i]*sin(t)+YC[i];
  label = i+1;
}

int[int] nn=[4,4,5,7]; // 4 border , with 4,4,5,7 segment respectively .
plot(bb(nn),cc(NC),wait=1);
mesh th= buildmesh(bb(nn)+cc(NC));
plot(th,wait=1);
\end{verbatim}
5.1.4 Data Structures and Read/Write Statements for a Mesh

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

```cpp
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. 5.7.

The informations about `Th` are saved in the file “mesh_sample.msh”. whose structure is shown on Table 5.1.
There `nv` denotes the number of vertices, `nt` number of triangles and `ns` the number of edges on boundary.
For each vertex `qi`, `i = 1, ⋯, nv`, denote by `(qx, qy)` the x-coordinate and y-coordinate.
Each triangle `Tk`, `k = 1, ⋯, 10` has three vertices `qk1, qk2, qk3` that are oriented counterclockwise. The boundary consists of 10 lines `Li`, `i = 1, ⋯, 10` whose end points are `qi1, q2i`.

In the left figure, we have the following.

\[ n_v = 14, \quad n_t = 16, \quad n_s = 10 \]

\[ q^1 = (-0.309016994375, 0.951056516295) \]
\[ q^{14} = (-0.309016994375, -0.951056516295) \]

The vertices of `T1` are `q^0, q^{12}, q^{10}`.

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

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

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

In FreeFem++ there are many mesh file formats available for communication with other tools such as emc2, modulef. (see Section 12). The extension of a file implies its format. More details can be found on the file format .msh in the article by F. Hecht “bamg : a bidimensional anisotropic mesh generator” (downloadable from the FreeFem web site.)

A mesh file can be read into FreeFem++ except that the names of the borders are lost and only their reference numbers are kept. So these borders have to be referenced by the number which corresponds to their order of appearance in the program, unless this number is overwritten by the keyword ”label”. Here are some examples:

```cpp
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;};
```
5.1. COMMANDS FOR MESH GENERATION

<table>
<thead>
<tr>
<th>Content of the file</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 16 10</td>
<td>( n_v )  ( n_t )  ( n_e )</td>
</tr>
<tr>
<td>-0.309016994375 0.951056516295 1</td>
<td>( q_x^1 )  ( q_y^1 )  boundary label=1</td>
</tr>
<tr>
<td>0.309016994375 0.951056516295 1</td>
<td>( q_x^2 )  ( q_y^2 )  boundary label=1</td>
</tr>
<tr>
<td>... ... ...</td>
<td>( q_x^{14} )  ( q_y^{14} )  boundary label=1</td>
</tr>
<tr>
<td>9 12 10 0</td>
<td>1 ( _1 )  2 ( _1 )  13 ( _1 )  region label=0</td>
</tr>
<tr>
<td>5 9 6 0</td>
<td>2 ( _1 )  2 ( _2 )  23 ( _1 )  region label=0</td>
</tr>
<tr>
<td>...</td>
<td>16 ( _1 )  16 ( _2 )  16 ( _3 )  region label=0</td>
</tr>
<tr>
<td>6 5 1</td>
<td>1 ( _1 )  2 ( _1 )  boundary label=1</td>
</tr>
<tr>
<td>5 2 1</td>
<td>2 ( _1 )  2 ( _2 )  boundary label=1</td>
</tr>
<tr>
<td>...</td>
<td>10 ( _1 )  10 ( _2 )  boundary label=1</td>
</tr>
</tbody>
</table>

Table 5.1: The structure of “meshsample.msh”

```plaintext
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 [14]
mesh th2 = readmesh("toto.msh"); // read the mesh
```

Example 5.1 (Readmesh.edp) ```plaintext
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 [14]
```
```
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);

5.1.5 Mesh Connectivity and data

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;
    real area = Th.measure, borderlen = Th.bordermeasure;  // Version 3.56
    cout << " nb of Triangles = " << nbtriangles << endl;
    for (int i=0;i<nbtriangles;i++)
        for (int j=0; j <3; j++)
            cout << i << " " << j << " Th[i][j] = "
            << Th[i][j] << " x = "<< Th[i][j].x << " , y= "<< Th[i][j].y
            << ", label= " << Th[i][j].label << endl;

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

    fespace femp1(Th,P1);
    femp1 Thx=x,Thy=y; // hack of get vertex coordinates
    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 nl100 = Th[it00].label; // same as region in this case.

    // Hack to get a triangle containing point x,y
5.1. COMMANDS FOR MESH GENERATION

```plaintext
// 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 containing (0.55,0,6);
int nr0=nuReg(0.55,0.6);  // number of region of Th’s containing (0.55,0,6);

// dump
// -------------------------------------------------------
cout << " point (0.55,0,6) :triangle number " << it0
     << " \n" << it0 << ", region = " << nr0 << " == " << nr0 << " , area K " << area00 << endl;

// new method to get boundary information 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 ∈{0,...,Th.nbe−1}
Th.be(k)[l];  // return the vertices l ∈{0,1} of boundary elmt k
Th.be(k).Element ;  // return the triangle containing the boundary elmt k
Th.be(k).whoinElement ;  // return the edge number of triangle containing
                           // 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;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
    // note : if k == int(Th[k].adj(\(ee=e\))) not adjacent element

int nbboundaryelement = Th.nbe;
for (int k=0;k<nboundaryelement;++k)
    cout << k << " : " << Th.be(k)[0] << " " << Th.be(k)[1] << " , label "
    << Th.be(k).label << " tria " << int(Th.be(k).Element)
```
CHAPTER 5. MESH GENERATION

```cpp
<< " " << Th.be(k).whoInElement << endl;

real[int] bb(4);
boundingbox(Th,bb);
cout << " boundingbox xmin: " << bb[0] << " xmax: " << bb[1]
}
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
```
The real characteristic function of a mesh $Th$ is $\chi(Th)$ in 2d and 3d where

$$\chi(Th)(P) = 1 \quad \text{if} \quad P \in Th; \quad \chi(Th)(P) = 0 \quad \text{if} \quad P \notin Th; \quad (5.1)$$

### 5.1.6 The keyword ”triangulate”

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

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

0.51387 0.175741 0.636237
0.308652 0.534534 0.746765
0.947628 0.171736 0.899823
0.702231 0.226431 0.800819
0.494773 0.124720 0.580623
0.0838988 0.389647 0.456045

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

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

```plaintext
mesh Thxy=triangulate("xyf"); // build the Delaunay mesh of the convex hull
plot (Thxy, ps="Thxyf.ps"); // (see figure 5.8)
```

```plaintext
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(fxy, ps="xyf.eps"); // plot the function (see figure 5.9)

One new way to build a mesh is to have two arrays 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 a boundary, we came up with the idea of a mesh with no internal points (call empty mesh). It can be useful to handle Lagrange multipliers in mixed and mortar methods.

So the function emptymesh remove all the internal points of a mesh except points on internal boundaries.

{ // new stuff 2004 emptymesh (version 1.40)
   // -- useful to build Multiplicator space
   // build a mesh without internal point
   // with the same boundary
   // -----

   assert(version>=1.40);

   border a(t=0, 2*pi){ x=cos(t); y=sin(t); label=1;}
   mesh Th=buildmesh(a(20));
   Th=emptymesh(Th);
   plot(Th, wait=1, ps="emptymesh-1.eps"); // see figure 5.10
}

It is also possible to build an empty mesh of a pseudo subregion with emptymesh(Th, ssd) using the set of edges of the mesh Th; a edge e is in this set if with the two adjacent triangles \( e = t_1 \cap t_2 \) and \( ssd[T_1] \neq ssd[T_2] \) where ssd refers to the pseudo region numbering of triangles, when they are stored in an int[int] array of size the number of triangles.

{ // new stuff 2004 emptymesh (version 1.40)
   // -- useful to build Multiplicator space
   // build a mesh without internal point
   // of 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++)
      { 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;
      }
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 $\mathcal{T}_h(\Omega)$, and $\Phi$ is a displacement vector then $\Phi(\mathcal{T}_h)$ is obtained by

```c
mesh Th=movemesh(Th, [\Phi_1, \Phi_2]);
```

Sometimes the transformed mesh is invalid because some triangle have flip over (now has negative area). To spot such problems one may check the minimum triangle area in the transformed mesh with checkmovemesh before any real transformation.

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

```c
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;};
```
CHAPTER 5. MESH GENERATION

```plaintext
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 5.12
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 5.13
```

Figure 5.12: L-shape

Figure 5.13: moved L-shape

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

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

// simple movemesh example

```plaintext
mesh Th=square(10,10);
fespace Vh(Th,P1);
real t=0;
```

// ---
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 $\{T_h\}_{h \downarrow 0}$ of $\Omega$ is called regular if they satisfy the following:

1. $$\lim_{h \downarrow 0} \max\{\text{diam}(T_k); \ T_k \in T_h\} = 0$$

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

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

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

We put $h(T_h) = \max\{\text{diam}(T_k); \ T_k \in T_h\}$, which is obtained by

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

The function

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

sharply varies in value and the initial mesh given by one of the commands of Section 5.1 cannot reflect its sharp variations.

Example 5.4

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

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

Figure 5.14: 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 $ATh$ adapted to the Hessian

$$D^2 f = \left( \frac{\partial^2 f}{\partial x^2}, \frac{\partial^2 f}{\partial x \partial y}, \frac{\partial^2 f}{\partial y^2} \right)$$

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

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

![Figure 5.15: L-shape domain and its boundary name](image1)

![Figure 5.16: Final solution after 4-times adaptation](image2)

**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. 5.15).

```
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
fespace Vh(Th,P1); // 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 the default parameter err of adaptmesh is changed by hand; it specifies the required precision, so as to make the new mesh finer or coarser.
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, the slope of the final solution is correctly computed near the point of intersection of bc and bd as in Fig. 5.16.

This method is described in detail in [13]. It has a number of default parameters which can be modified:

Si f1, f2 sont des functions et thold, Thnew des maillages.

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

The additional parameters of adaptmesh not written here, hence the "..."

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

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

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

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

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

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

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

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

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

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

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

\[ M = \left( \frac{1}{	ext{err coef}^2} \frac{\|H\|}{	ext{max}(\text{CutOff},|\eta|)} \right)^p \] (5.2)

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

\[ M = \left( \frac{1}{	ext{err coef}^2} \frac{\|H\|}{\sup(\eta) - \inf(\eta)} \right)^p \] (5.3)

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

\[ \text{Th=adaptmesh(Th,fxx,fxy,fyy,IsMetric=1,nbvx=10000,hmin=hmin);} \]

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

thetamax= minimum corner angle of in degrees (default is 10°) where the corner is \( ABC \) and the angle is the angle of the two vectors \( AB, BC \), (0 imply no corner, 90 imply perp. corner , ...).

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 \(m_{11}, m_{12}, m_{22}\) are three P1 finite elements related to the mesh to adapt, you can write: `metric=[m_{11}[], m_{12}[], m_{22}[]]` (see file `convect-apt.edp` for a full example)

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

**periodic=** Writing `periodic=[[4,y], [2,y], [1,x], [3,x]]` 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)

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 \(\frac{1}{30}\) try:

**Example 5.6 uniformmesh.edp**

```plaintext
mesh Th=square(2,2);  // to have initial mesh
plot(Th,wait=1,ps="square-0.eps");
Th= adaptmesh(Th,1./3As writing 0.,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 5.17: Initial mesh](image1)
![Figure 5.18: first iteration](image2)
![Figure 5.19: last iteration](image3)

### 5.6 Trunc

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

**label=** sets the label number of new boundary item (one by default)
split\(=\) sets the level \(n\) of triangle splitting. Each triangle is splitted in \(n \times n\) (one by default).

To create the mesh \(Th3\) where all triangles of a mesh \(Th\) are splitted in \(3\times3\), just write:

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

The `truncmesh.edp` example construct all ”trunc” mesh to the support of the basic function of the space \(Vh\) (cf. \(\text{abs}(u)>0\)), split all the triangles in \(5\times5\), and put a label number to 2 on new boundary.

```cpp
mesh Th=square(3,3);
fespace Vh(Th,P1);
Vh u;
int i,n=u.n;
u=0;
for (i=0;i<n;i++)
{
  u[i]=1;
  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[]=0;
```

Figure 5.20: mesh of support the function \(P1\) number 0, splitted in \(5\times5\)

Figure 5.21: mesh of support the function \(P1\) number 6, splitted in \(5\times5\)

### 5.7 Splitmesh

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

```cpp
// new stuff 2004 splitmesh (version 1.37)
```
assert(version>=1.37);
border a(t=0,2*pi){ x=cos(t); y=sin(t); label=1; }
mesh Th=buildmesh(a(20));
plot(Th,wait=1,ps="nosplitmesh.eps"); // see figure 5.22
Th=splithmesh(Th,1+5*(x-0.5)^2+y^2); // see figure 5.23
plot(Th,wait=1,ps="splitmesh.eps");
}

5.8 Meshing Examples

Example 5.7 (Two rectangles touching by a side)

border a(t=0,1){x=t;y=0;};
border b(t=0,1){x=1;y=t;};
border c(t=1,0){x=t ;y=1;};
border d(t=1,0){x = 0; y=t;};
border c1(t=0,1){x=t ;y=1;};
border e(t=0,0.2){x=1;y=1+t;};
border f(t=1,0){x=t ;y=1.2;};
border g(t=0.2,0){x=0;y=1+t;};
int n=1;
mesh th = buildmesh(a(10*n)+b(10*n)+c(10*n)+d(10*n));
mesh TH = buildmesh( c1(10*n) + e(5*n) + f(10*n) + g(5*n) );
plot(th,TH,ps="TouchSide.esp"); // Fig. 5.24

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. 5.25

Figure 5.24: Two rectangles touching by a side

Figure 5.25: NACA0012 Airfoil

Example 5.9 (Cardioid)

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

mesh Th = buildmesh(C(50));
plot(Th,ps="Cardioid.eps",bw=1);

// Fig. 5.26

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. 5.27

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. 5.28

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); }
plot(Th,ps="Engine.eps",bw=1); // Fig. 5.29

Example 5.13 (Domain with U-shape channel)

real d = 0.1; // width of U-shape
border L1(t=0,1-d) { x=-1; y=-d-t; }
border L2(t=0,1-d) { x=-1; y=1-t; }
Example 5.14 (Domain with V-shape cut)

```
real dAg = 0.01; // angle of V-shape
border C(t=dAg,2*pi-dAg) { x=cos(t); y=sin(t); }
real[int] pa(2), pb(2), pc(2);
pa[0] = cos(dAg); pa[1] = sin(dAg);
pb[0] = cos(2*pi-dAg); pb[1] = sin(2*pi-dAg);
pc[0] = 0; pc[1] = 0;
border seg1(t=0,1) { x=(1-t)*pb[0]+t*pc[0]; y=(1-t)*pb[1]+t*pc[1]; }
border seg2(t=0,1) { x=(1-t)*pc[0]+t*pa[0]; y=(1-t)*pc[1]+t*pa[1]; }
mesh Th = buildmesh(seg1(20)+C(40)+seg2(20));
plot(Th,ps="V-shape.eps",bw=1); // Fig. 5.31
```

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;
}
```
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. 5.33
5.9 How to change the label of elements and border elements of a mesh

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.
- `flabel` = is an integer function with given the new value of the label (version 3.21).
- `fregion` = is an integer function with given the new value of the region.

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

\[
\text{label} = \left[ O_1, N_1, ..., O_{n_l}, N_{n_l} \right] \quad (5.4)
\]
\[
\text{region} = \left[ O_1, N_1, ..., O_{n_l}, N_{n_l} \right] \quad (5.5)
\]

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

**Example 5.17 (glumesh2D.edp)**

1: 2: \texttt{mesh Th1=square(10,10);}
3: \texttt{mesh Th2=square(20,10,\{x+1,y\});}
4: \texttt{verbosity=3;}
5: \texttt{int[int] r1=[2,0], r2=[4,0];}
6: \texttt{plot(Th1,wait=1);}
7: \texttt{Th1=change(Th1,label=r1); \quad // Change the label of Edges 2 in 0.}
8: \texttt{plot(Th1,wait=1);}
9: \texttt{Th2=change(Th2,label=r2); \quad // Change the label of Edges 4 in 0.}

In line 10 of previous file, the method to "gluing" different mesh of the same dimension in FreeFem++ is using. This function is the operator "+" between meshes. The method implemented need that the point in adjacent mesh are the same.

5.10 Mesh in three dimensions

5.10.1 cube

From version (3.38-2), a new function cube like the function square in 2d is the simple way to build cubic object, in plugin msh3 (need load "msh3").

The following code

```
mesh3 Th=cube(3,4,5);
```

generates a $3 \times 4 \times 5$ grid in the unit cube $[0,1]^3$.

By defaults the label (after version 3.56-2 and to correct otherwise add label=16 with
int[int] 16=[1,2,3,4,5,6];) are :

1. face $y = 0$,
2. face $x = 1$,
3. face $y = 1$,
4. face $x = 0$,
5. face $z = 0$,
6. face $z = 1$,

and the region number is 0.

A full examples of the this function to build a mesh of cube $[−1,1]^3$ with face label given by $(ix + 4*(iy + 1) + 16*(iz + 1))$ where $(ix,iy,iz)$ is coordinate of the barycenter of the current face.

```
load "msh3"
int[int] 16=[37,42,45,40,25,57];
int r11=11;
mesh3 Th=cube(4,5,6,[x*2-1,y*2-1,z*2-1],label=16,flags =3,region=r11);
cout << " volume " << Th.measure << ", border area "<< Th.bordermeasure <<endl;
```

// Check label dans region numbering
5.10. MESH IN THREE DIMENSIONS

```cpp
int err = 0;
for (int i = 0; i < 100; ++i)
{
    real s = int2d(Th, i)(1.);
    real sx = int2d(Th, i)(x);
    real sy = int2d(Th, i)(y);
    real sz = int2d(Th, i)(z);

    if (s)
    {
        int ix = (sx/s+1.5), iy = (sy/s+1.5), iz = (sz/s+1.5),
                   ii = (ix + 4*iy + 16*iz) ;
        // value of ix,iy,iz => face min 0 , face max 2 , no face 1
        cout << label = "" << i << " s " << s << " " << ix << iy << iz
             << " : " << ii << endl;
        if (i != ii) err++;
    }
}
real volr11 = int3d(Th, r11)(1.);
cout << " vol region " << 11 << ": " << volr11 << endl;
if ( (volr11 - Th.measure )>1e-8) err++;
plot (Th, fill=0);
cout << " nb err = " << err << endl;
assert (err==0);
```

The output of this script is:

```
Enter: BuildCube: 3
    kind = 3 n tet Cube = 6 / n slip 6 19
    Cube  nv=210 nt=720 nbe=296
Out: BuildCube
    volume 8, border area 24
    label=25 s 4 110 : 25
    label=37 s 4 101 : 37
    label=40 s 4 011 : 40
    label=42 s 4 211 : 42
    label=45 s 4 121 : 45
    label=57 s 4 112 : 57
    vol region 11: 8
    nb err = 0
    times: compile 0.005363s, execution 0.00218s, mpirank:0
    CodeAlloc : nb ptr 2856, size :352744 mpirank: 0
```

5.10.2 Read/Write Statements for a Mesh in 3D

In three dimensions, the file mesh format supported for input and output files by FreeFem++ are the extension .msh and .mesh. These formats are described in the chapter on Mesh Files in two dimensions.
extension file .msh The structure of the files with extension .msh in 3D is 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 \( q^i, i = 1, \ldots, n_v \), we denote by \((q^i_x, q^i_y, q^i_z)\) the \( x \)-coordinate, the \( y \)-coordinate and the \( z \)-coordinate. Each tetrahedra \( T_k, k = 1, \ldots, n_{tet} \) 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 \( b_{e_j}, j = 1, \ldots, n_{tri} \) has three vertices \( q^{p_1}, q^{p_2}, q^{p_3} \).

<table>
<thead>
<tr>
<th>( n_v )</th>
<th>( n_{tet} )</th>
<th>( n_{tri} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q^1_x )</td>
<td>( q^1_y )</td>
<td>( q^1_z )</td>
</tr>
<tr>
<td>( q^2_x )</td>
<td>( q^2_y )</td>
<td>( q^2_z )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( q^n_x )</td>
<td>( q^n_y )</td>
<td>( q^n_z )</td>
</tr>
<tr>
<td>( 1_1 )</td>
<td>( 1_2 )</td>
<td>( 1_3 )</td>
</tr>
<tr>
<td>( 2_1 )</td>
<td>( 2_2 )</td>
<td>( 2_3 )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( (n_{tet})_1 )</td>
<td>( (n_{tet})_2 )</td>
<td>( (n_{tet})_3 )</td>
</tr>
<tr>
<td>( 1_1 )</td>
<td>( 1_2 )</td>
<td>( 1_3 )</td>
</tr>
<tr>
<td>( 2_1 )</td>
<td>( 2_2 )</td>
<td>( 2_3 )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( (n_{tri})_1 )</td>
<td>( (n_{tri})_2 )</td>
<td>( (n_{tri})_3 )</td>
</tr>
</tbody>
</table>

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

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 tetrahedra. The tetrahedra of a three dimensional mesh are refereed using the following field:

- **Tetrahedra**
  
  (I) NbOfTetrahedrons
  
  \[
  \left( \left( @\text{Vertex}_j, j=1,4 \right), (I) \text{ RefId}_i, i=1, \text{NbOfTetrahedrons} \right)
  \]
5.10. MESH IN THREE DIMENSIONS

This field is express with the notation of Section [2.1].

5.10.3 TeGen: A tetrahedral mesh generator

**TetGen** TetGen is a software developed by Dr. Hang Si of Weierstrass Institute for Applied Analysis and Stochastics of Berlin in Germany [41]. TetGen is a free for research and non-commercial uses. For any commercial licence utilization, a commercial licence is available upon request to 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 tetrahedralization is a constrained Delaunay tetrahedralization.

The method used in TetGen to control the quality of the mesh is a Delaunay refinement due to Shewchuk [42]. The quality measure of this algorithm is the Radius-Edge Ratio (see Section 1.3.1 [41] 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 launch 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 [5.4].
- **switch** = A string expression. This string corresponds to the command line switch of Tetgen see Section 3.2 of [41].
- **nbofholes** = Number of holes (default value size of holelist/3 (version 3.11)).
- **holelist** = This array correspond to holelist of tetgenio data structure [41]. A real vector of size 3 × nbofholes. In TetGen, each hole is associated with a point inside this domain. This vector is $x_h^1, y_h^1, z_h^1, x_h^2, y_h^2, z_h^2, \cdots$ where $x_h^i, y_h^i, z_h^i$ is the associated point with the $i$th hole.
- **nbofregions** = Number of regions (size of regionlist/5 (version 3.11)).
- **regionlist** = This array corresponds to regionlist of tetgenio data structure [41]. The attribute and the volume constraint of region are given in this real vector of size 5 × nbofregions. The $i$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 tetrahedra ($mvol_i$) for this region. The regionlist vector is: $x_1, y_1, z_1, at_1, mvol_1, x_2, y_2, z_2, at_2, mvol_2, \cdots$.
- **noffacetcl** = Number of facets constraints size of facetcl/2 (version 3.11).
- **facetcl** = This array corresponds to facetconstraintlist of tetgenio data structure [41]. The $i$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: $Ref_1^{fc}, marea_1^{fc}, Ref_2^{fc}, marea_2^{fc}, \cdots$. This parameters has no effect if switch q is not selected.

Principal switch parameters in TetGen:

- **p** Tetrahedralization of boundary.
CHAPTER 5. MESH GENERATION

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 tetrahedra. 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 $Th_2$ by this three dimensional displacement is obtained using:

```plaintext
mesh3  Th3 = movemesh23(Th2, transf=[\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

\[ ptmerge = 1e-7 \ Vol(B) \]

where \( B \) is the smallest axis parallel boxes containing the discretized domain of \( \Omega \) and \( Vol(B) \) is the volume of this box.

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

Example 5.18 (tetgencube.edp)

```plaintext
// 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 = 0.5;
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",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, tranfo=[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 closed surface and define one region only. 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 tetrahedralization from a set of points. This tetrahedralization is a Delaunay mesh of the convex hull of the set of points.
The coordinates of the points can be initialized in two ways. The first is a file that contains the coordinate of points \( X_i = (x_i, y_i, z_i) \). This files is organized as follows:

\[
\begin{array}{ccc}
n_v & x_1 & y_1 \\
x_2 & y_2 & z_2 \\
\vdots & \vdots & \vdots \\
n_v & x_{n_v} & y_{n_v} \\
\end{array}
\]

The second way is to give three arrays that correspond respectively to the \( x \)–coordinates, \( y \)–coordinates and \( z \)–coordinates. The parameters of this command line are

\texttt{switch = A string expression. This string corresponds to the command line switch of TetGen see Section 3.2 of [7].}
5.10. MESH IN THREE DIMENSIONS

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

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

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

### 5.10.4 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

- **region**= an integer array that allow to change the region number of tetrahedra. 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 reel function. This function allows to constraint volume size of tetrahedra in the domain. (see example 5.31 to build 3d adapt mesh)

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 [41] for effect of ’r’ to other character.

The parameter `regionlist` allows 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)**

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

mesh Th=square(10,20,[x*pi-pi/2,2*y*pi]); // [-pi/2,pi/2]*[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^tDF
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, m1*vv, m21*vv, m22*vv, IsMetric=1, periodic=perio);
Th = adaptmesh(Th, m1*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, transf=[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=domain, refinesizeofvolume=0.0001);

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);

5.10.5 Moving mesh in three dimensions

Meshes in three dimensions 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 tetrahedrized as $T_h(\Omega)$, and $\Phi(x,y) = (\Phi_1(x,y,z), \Phi_2(x,y,z), \Phi_3(x,y,z))$ is a displacement vector then $\Phi(T_h)$ is obtained by

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

The parameters of movemesh in three dimensions are

region = set integer label of tetrahedra. 0 by default.

label = set the label of faces of border. This parameters is initialized as label for the keyword change [5.4].

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.
5.10. MESH IN THREE DIMENSIONS

Figure 5.35: Example of Layer mesh in three dimension.

\textbf{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

\[ ptmerge = 1 e^{-7} \, \text{Vol}(B), \]

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

\textbf{orientation} = An integer expression (1 by default), to reverse or not the orientation of tet if not positive.

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

5.10.6 Layer mesh

In this section, we present the command line to obtain a Layer mesh: \texttt{buildlayermesh}. This mesh is obtained by extending a two dimensional mesh in the z-axis. The domain \( \Omega_{3d} \) defined by the layer mesh is equal 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 \( \Omega_{3d} \).

For a vertex of a two dimensional mesh \( V_{2d}^{i} = (x_i, y_i) \), 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

\[ \forall j = 0, \ldots, M, \quad V_{i,j}^{3d} = (x_i, y_i, \theta_{i}(z_{i,j})), \]

where \( (z_{i,j})_{j=0,...,M} \) 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 \( (\theta_{i,j})_{j=0,...,M} \) are the \( M_{i} + 1 \) equidistant points on the interval \( [zmin(V_{i}^{2d}), zmax(V_{i}^{2d})] \).
Set a triangle $K = (V_{i_1}^{2d}, V_{i_2}^{2d}, V_{i_3}^{2d})$ of the two dimensional mesh. $K$ is associated with a triangle on the upper surface (resp. on the lower surface) of layer mesh: $(V_{i_1,M}^{3d}, V_{i_2,M}^{3d}, V_{i_3,M}^{3d})$ (resp. $(V_{i_1,0}, V_{i_2,0}, V_{i_3,0})$).

Also $K$ is associated with $M$ volume prismatic elements which are defined by $\forall j = 0, \ldots, M, H_j = (V_{i_1,j}^{3d}, V_{i_2,j}^{3d}, V_{i_3,j}^{3d}, V_{i_1,j+1}^{3d}, V_{i_2,j+1}^{3d}, V_{i_3,j+1})$.

Theses volume elements can have some merged point:

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

The elements with merged points are called degenerate elements. To obtain a mesh with tetrahedra, we decompose the pyramid into two tetrahedra and the prism into three tetrahedra. These tetrahedra are obtained by cutting the quadrilateral face of pyramid and prism with the diagonal which have the vertex with the maximum index (see [12] for the reasnp 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 tetrahedra 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:

- $z\text{bound} = [z\text{min}, z\text{max}]$ where $z\text{min}$ and $z\text{max}$ are functions expression. Theses functions define the lower surface mesh and upper mesh of surface mesh.

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

- $r\text{region} = This vector is used to initialized the region of tetrahedra. This vector contain successive pair of the 2d region number at index $2i$ and the corresponding 3d region number at index $2i + 1$, like (5.4), become the

- $l\text{abelmid} = This vector is used to initialized the 3d labels number of the vertical face or mid face form 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 (5.4).

- $l\text{abelup} = This vector is used to initialized the 3d labels numbers of the upper/top face form 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 (5.4).

- $l\text{abeldown} = Same as the previous case but 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 pairs of number $O_l, N_l$ 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)

```plaintext
load "medit"
load "msh3"
func mesh3 Cube(int[int] & NN, real[int,int] & BB ,int[int,int] & L)
{
    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 unit cube example:

```plaintext
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
int left,right,
    // front, back, down, right
mesh3 Th=Cube(NN,BB,L);
medit ("Th",Th); // see figure 5.36
```

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

Example 5.21 (cone.edp)

```plaintext
load "msh3"
load "medit"
```

```plaintext
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;
```

```plaintext
mesh Th2=buildmesh( Taxe(HH*nn)+ Hypo(sqrt(HH*HH+RR*RR)*nn) + Vert(RR*nn) ) ;
```

```plaintext
plot (Th2,wait=1); // the 2d mesh
```

```plaintext
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);
// see figure 5.37
medit("cone", Th3T);

Figure 5.36: the mesh of a cube made with cube.edp
Figure 5.37: the mesh of a cone made with cone.edp

Example 5.22 (buildlayermesh.edp)

// file buildlayermesh.edp

load "msh3"
load "tetgen"

// Test 1

int C1=99, C2=98;
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;}

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;}
5.11 Meshing examples

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

```plaintext
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[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=rlow);
savemesh(Th,'Th.meshb');
exec('medit Th; Th.meshb');

Example 5.24 (tetgenholeregion.edp) // file ‘tetgenholeregion.edp’
load "msh3"'
load "tetgen"

mesh Th=square(10,20,[x*pi-pi/2,2*y*pi]); // ]-pi/2,-pi/2[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 = DFtDF
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.;
5.11. MESHING EXAMPLES

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.,.145,0.001,0.5,0.,.18,0.001];
cout << "==================================" << endl;
cout " tetgen call without hole " << endl;
cout "==================================" << endl;

mesh3 Th3fin = tetg(Th3,switch="paAAQYY",nbofregions=2,regionlist=domain2);
cout "finish tetgen call without hole" << endl;
cout "==================================" << endl;
savemesh(Th3fin,"spherewithtworegion.mesh");

real[int] hole = [0.,0.,0.];
real[int] domain = [1.5,0.,.053,0.001];
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 "finish tetgen call with hole " << endl;
cout "==================================" << endl;
savemesh(Th3finhole,"spherewithahole.mesh");

5.11.1 Build a 3d mesh of a cube with a balloon

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 labels
    renumbering
    int[int] refz=[0,L(2,0)],refZ=[0,L(2,1)];  // Zmin, Zmax faces labels
    renumbering
```plaintext
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]); // [-pi/2, pi/2] [0, 2pi]
  // a parametrization of a sphere
  func f1 = cos(x) * cos(y);
  func f2 = cos(x) * sin(y);
  func f3 = sin(x);
  // partial 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^2
  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 unit 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

```plaintext
load "tetgen"
include "MeshSurface.idp"
real hs = 0.1; // mesh size on sphere
int [int] N=[20, 20, 20]; // mesh size on sphere
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 total boundary meshes
```
5.12. The output solution formats .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

- MeshVersionFormatt ed 0
- Dimension (I) dim

The following fields 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 the 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 values of the solutions on the elements.

The file must be ended with the keyword End.

The real element of symmetric tensor

\[
ST^{3d} = \begin{pmatrix}
ST_{xx}^{3d} & ST_{xy}^{3d} & ST_{xz}^{3d} \\
ST_{yx}^{3d} & ST_{yy}^{3d} & ST_{yz}^{3d} \\
ST_{zx}^{3d} & ST_{zy}^{3d} & ST_{zz}^{3d}
\end{pmatrix}
\]

\[
ST^{2d} = \begin{pmatrix}
ST_{xx}^{2d} & ST_{xy}^{2d} \\
ST_{yx}^{2d} & ST_{yy}^{2d}
\end{pmatrix}
\]

stored in the extension .sol are respectively \(ST_{xx}^{3d}, ST_{yy}^{3d}, ST_{zz}^{3d}, ST_{xy}^{3d}, ST_{yx}^{3d}, ST_{yz}^{3d}, ST_{zx}^{3d}, ST_{zy}^{3d}\) and \(ST_{xx}^{2d}, ST_{yy}^{2d}\)

An example of field with the keyword SolAtTetrahedra:

- \(\text{SolAtTetrahedra (I) NbOfTetrahedrons}\)
  - \(\text{nbsol typesol}^1 \ldots \text{typesol}^n\)
  - \(\left(\left(\left(U_{ijk}^k, \forall i \in \{1,...,\text{nbrealsol}^k\}\right), \forall k \in \{1,...,\text{nbsol}\}\right) \forall j \in \{1,...,\text{NbOfTetrahedrons}\}\right)\)

where

- \(\text{nbsol}\) is an integer equal to the number of solutions
- \(\text{typesol}^k\), type of the solution number \(k\), is
  - \(\text{typesol}^k = 1\) 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 discribe 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).
  - \(\text{nbrealsol}^k = \text{dim} \ast (\text{dim} + 1)/2\) the solution \(k\) is a symmetric tensor or symmetric matrix.
- \(U_{ijk}^k\) 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 written with the notation of Section 12.1. The format .solb is the same as format .sol but in binary (read/write is faster, storage is less).

A real scalar functions \(f_1\), a vector fields \(\Phi = [\Phi_1, \Phi_2, \Phi_3]\) and a symmetric tensor \(ST^{3d}\) at the vertices of the three dimensional mesh \(\text{Th3}\) is stored in the file "f1PhiTh3.sol" using

\[
savesol("f1PhiST3dTh3.sol",\text{Th3}, f1, [\Phi_1, \Phi_2, \Phi_3], \text{VV3}, \text{order}=1);\]

where \(\text{VV3} = [ST_{xx}^{3d}, ST_{yz}^{3d}, ST_{yy}^{3d}, ST_{zx}^{3d}, ST_{zy}^{3d}, ST_{zz}^{3d}]\). For a two dimensional mesh \(\text{Th}\), A real scalar functions \(f_2\), a vector fields \(\Psi = [\Psi_1, \Psi_2]\) and a symmetric tensor \(ST^{2d}\) at triangles is stored in the file "f2PsiST2dTh3.solb" using

\[
savesol("f2PsiST2dTh3.solb",\text{Th}, f2, [\Psi_1, \Psi_2], \text{VV2}, \text{order}=0);\]

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

The parameters 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 medit

The keyword medit allows to display a mesh alone or a mesh and one or several functions defined on the mesh using the Pascal Frey’s freeware medit. Medit opens its own window and uses OpenGL extensively. Naturally to use this command medit must be installed.

A visualization with medit of scalar solutions \( f_1 \) and \( f_2 \) continuous, piecewise linear and known at the vertices of the mesh \( Th \) is obtained using

\[
\text{medit(“sol1 sol2”,Th, f1, f2, order=1);}
\]

The first plot named “sol1” display \( f_1 \). The second plot names “sol2” display \( f_2 \).

The arguments of function medit are the name of the different 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 specified in the format described in the section dealing with the keyword savesol.

The parameters of this command line are

- \( \text{order} = 0 \) is the solution is given at the center of gravity of elements. \( 1 \) is the solution is given at the vertices of elements.
- \( \text{meditff} \) = set the name of execute command of medit. By default, this string is medit.
- \( \text{save} \) = set the name of a file .sol or .solb to save solutions.

This command line allows also to represent two different meshes and solutions on them in the same window. The nature of solutions must be the same. Hence, we can visualize in the same window the different domains in a domain decomposition method for instance. A visualization with medit of scalar solutions \( h_1 \) and \( h_2 \) at vertices of the mesh \( Th1 \) and \( Th2 \) respectively are obtained using

\[
\text{medit(“sol2domain”,Th1, h1, Th2, h2, order=1);}
\]

Example 5.25 (meditddm.edp)

\[
\text{// meditddm.edp}
\]

\[
\text{// Initial Problem:}
\]

\[
\text{// Resolution of the following EDP:}
\]

\[
\text{// } -\Delta u = f \text{ on } \Omega = \{(x,y)|1 \leq \sqrt{x^2 + y^2} \geq 2\}
\]

\[
\text{// } -\Delta u_1 = f_1 \text{ on } \Omega_1 = \{(x,y)|0.5 \leq \sqrt{x^2 + y^2} \geq 1.\}
\]

\[
\text{// } u = 1 \text{ on } \Gamma + \text{ Null Neumann condition on } \Gamma_1 \text{ and on } \Gamma_2
\]

\[
\text{// We find the solution } u \text{ in solving two EDP defined on domain } \Omega \text{ and } \Omega_1
\]

\[
\text{// This solution is visualize with medit}
\]

\[
\text{verbosity=3;}
\]

\[
\text{border Gamma(t=0,2*pi){x=cos(t); y=sin(t); label=1;}};
\]
CHAPTER 5. MESH GENERATION

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)]

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

// Definition of EDP defined on the domain \Omega
// \Delta u_s = f_1 on \Omega_1, u_s = 1 on \Gamma_1, \frac{\partial u_s}{\partial n} = 0 on \Gamma_2
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)]

problem Lap2dOmega1(u1,v1,init=false)=int2d(Th1)(Grad21(v1)'*Grad21(u1))
   - int2d(Th1)(f1*v1)+on(1,u1=1) ;

// Resolution of EDP defined on the domain \Omega_1
// \Delta u_1 = f_1 on \Omega_1, u_1 = 1 on \Gamma_1, \frac{\partial u_1}{\partial n} = 0 on \Gamma_2
Lap2dOmega1;

// visualization 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);
5.14. **Mshmet**

Mshmet is a software developed by P. Frey that allows to compute an anisotropic metric based on solutions (i.e. Hessian-based). This software can return also an isotropic metric. Moreover, mshmet can construct also a metric suitable for level sets interface capturing. The solution can be defined on 2D or 3D structured/unstructured meshes. For example, the solution can be an error estimate of a FE solution.

Solutions for mshmet are given as an argument. The solution can be a func, a vector func, a symmetric tensor, a FE func, a FE vector func and a FE symmetric tensor. The symmetric tensor argument is defined as this type of data for datasol argument. This software accepts more than one solution.

For example, the metric $M$ computed with mshmet for the solution $u$ defined on the mesh $Th$ is obtained by writing.

```c
fespace Vh(Th,P1);
Vh u;
real[int] M = mshmet(Th,u);
```

The parameters of the keyword mshmet are:

- **normalization** = <b> do a normalisation of all solution in $[0,1]$.  
- **aniso** = <b> build aniso metric if 1 (default 0: iso)  
- **levelset** = <b> build metric for level set method (default: false)  
- **verbosity** = <l>  
- **nbregul** = <l> number of regularization’s iteration of solutions given (default 0).
• **hmin** = \(<d>\)
• **hmax** = \(<d>\)
• **err** = \(<d>\) level of error.
• **width** = \(<d>\) the width
• **metric**= a vector of double. This vector contains an initial metric given to mshmet. The structure of the metric vector is described in the next paragraph.

• **loptions**=]a vector of integer of size 7. This vector contains the integer parameters of mshmet(for expert only).
  - loptions(0): normalization (default 1).
  - loptions(1): isotropic parameters (default 0). 1 for isotropic metric results otherwise 0.
  - loptions(2): level set parameters (default 0). 1 for building level set metric otherwise 0.
  - loptions(3): debug parameters (default 0). 1 for turning on debug mode otherwise 0.
  - loptions(4): level of verbosity (default 10).
  - loptions(5): number of regularization’s iteration of solutions given (default 0).
  - loptions(6): previously metric parameter (default 0). 1 for using previous metric otherwise 0.

• **doptions**= a vector of double of size 4. This vector contains the real parameters of mshmet (for expert only).
  - doptions(0): hmin : min size parameters (default 0.01).
  - doptions(1): hmax : max size parameters (default 1.0).
  - doptions(2): eps : tolerance parameters ( default 0.01).
  - doptions(2): width : relative width for Level Set (0 < w < 1) ( default 0.05).

The result of the keyword mshmet is a real[int] which contains the metric computed by mshmet at the different vertices \( V_i \) of the mesh.

With \( nv \) is the number of vertices, the structure of this vector is

\[
M_{iso} = (m(V_0), m(V_1), \ldots, m(V_{nv}))^t
\]

for a isotropic metric \( m \). For a symmetric tensor metric \( h = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \), the parameters metric is

\[
M_{aniso} = (H(V_0), \ldots, H(V_{nv}))^t
\]

where \( H(V_i) \) is the vector of size 6 defined by \( [m_{11}, m_{21}, m_{22}, m_{31}, m_{32}, m_{33}] \)

**Example 5.27 (mshmet.edp)**

```plaintext
load "mshmet"
load "medit"
load "msh3"

border a(t=0,1.0) {x=t; y=0; label=1;};
border b(t=0,0.5) {x=1; y=t; label=2;};
border c(t=0,0.5) {x=1-t; y=0.5; label=3;};
```
5.15. FREEYAMS

FreeYams is a surface mesh adaptation software which is developed by P. Frey. This software is a new version of yams. The adapted surface mesh is constructed with a geometric metric tensor field. This field is based on the intrinsic properties of the discrete surface. Also this software allows to construct a simplification of a mesh. This decimation is based on the Hausdorff distance between the initial and the current triangulation. Compared to the software yams, FreeYams can be used also to produce anisotropic triangulations adapted to level set simulations. A technical report on FreeYams is not available yet but a documentation on yams exists at http://www.ann.jussieu.fr/~frey/software.html [45].

To call FreeYams in FreeFem++, we used the keyword freeyams. The arguments of this function are the initial mesh and/or metric. The metric with freeyams are a function, a FE function, a symmetric tensor function, a symmetric tensor FE function or a vector of double. If the metric is vector of double, this data must be given in metric parameter. Otherwise, the metric is given in the argument.

For example, the adapted mesh of Thinit defined by the metric \( u \) defined as FE function is obtained in writing.
fespace Vh(Thinit,P1);
Vh u;
mesh3 Th=freeyams(Thinit,u);

The symmetric tensor argument for freeyams keyword is defined as this type of data for datasol argument.

- **aniso** = <b> aniso or iso metric (default 0, iso)
- **mem** = <l> memory of for freeyams in Mb (delaulf -1, freeyams choose)
- **hmin** = <d>
- **hmax** = <d>
- **gradation** = <d>
- **option** = <l>
  0 : mesh optimization (smoothing+swapping)
  1 : decimation+enrichment adapted to a metric map. (default)
-1 : decimation adapted to a metric map.
  2 : decimation+enrichment with a Hausdorff-like method
-2 : decimation with a Hausdorff-like method
  4 : split triangles recursively.
  9 : No-Shrinkage Vertex Smoothing
- **ridgeangle** = <d>
- **absolute** = <b>
- **verbosity** = <i>
- **metric**= vector expression. This parameters contains the metric at the different vertices on the initial mesh. With \( nv \) is the number of vertices, this vector is

\[
M_{iso} = (m(V_0), m(V_1), \ldots, m(V_{nv}))^t
\]

for a scalar metric \( m \). For a symmetric tensor metric \( h = \begin{pmatrix}
m_{11} & m_{12} & m_{13} \\
m_{21} & m_{22} & m_{23} \\
m_{31} & m_{32} & m_{33}
\end{pmatrix} \), the parameters metric is

\[
M_{aniso} = (H(V_0), \ldots, H(V_{nv}))^t
\]

where \( H(V_i) \) is the vector of size 6 defined by \( [m_{11}, m_{21}, m_{22}, m_{31}, m_{32}, m_{33}] \)

- **loptions**= a vector of integer of size 13. This vectors contains the integer options of FreeYams. (just for the expert)
  - loptions(0): anisotropic parameter (default 0). If you give an anisotropic metric 1 otherwise 0.
  - loptions(1): Finite Element correction parameter (default 0). 1 for no Finite Element correction otherwise 0.
  - loptions(2): Split multiple connected points parameter (default 1). 1 for splitting multiple connected points otherwise 0.
- loptions(3): maximum value of memory size in Mbytes (default -1: the size is given by freeyams).
- loptions(4): set the value of the connected component which we want to obtain. (Remark: freeyams give an automatic value at each connected component).
- loptions(5): level of verbosity
- loptions(6): Create point on straight edge (no mapping) parameter (default 0). 1 for creating point on straight edge otherwise 0.
- loptions(7): validity check during smoothing parameter. This parameter is only used with No-Shrinkage Vertex Smoothing optimization (optimization option parameter 9). 1 for No validity checking during smoothing otherwise 0.
- loptions(8): number of desired’s vertices (default -1).
- loptions(9): number of iteration of optimizations (default 30).
- loptions(10): no detection parameter (default 0). 1 for detecting the ridge on the mesh otherwise 0. The ridge definition is given in the parameter doptions(12).
- loptions(11): no vertex smoothing parameter (default 0). 1 for smoothing the vertices otherwise 0.
- loptions(12): Optimization level parameter (default 0).

\[
\begin{align*}
\ast & 0 : \text{mesh optimization (smoothing+swapping)} \\
\ast & 1 : \text{decimation+enrichment adapted to a metric map.} \\
\ast & -1 : \text{decimation adapted to a metric map.} \\
\ast & 2 : \text{decimation+enrichment with a Hausdorff-like method} \\
\ast & -2 : \text{decimation with a Hausdorff-like method} \\
\ast & 4 : \text{split triangles recursively.} \\
\ast & 9 : \text{No-Shrinkage Vertex Smoothing}
\end{align*}
\]

\[\text{doptions} = \text{a vector of double of size 11. This vector contains the real options of freeyams.}\]
- doptions(0): Set the geometric approximation (Tangent plane deviation) (default 0.01).
- doptions(1): Set the lambda parameter (default -1. ).
- doptions(2): Set the mu parameter (default -1. ).
- doptions(3): Set the gradation value (Mesh density control) (default 1.3).
- doptions(4): Set the minimal size(hmin) (default -2.0: the size is automatically computed).
- doptions(5): Set the maximal size(hmax) (default -2.0: the size is automatically computed).
- doptions(6): Set the tolerance of the control of Chordal deviation (default -2.0).
- doptions(7): Set the quality of degradation (default 0.599).
- doptions(8): Set the declic parameter (default 2.0).
- doptions(9): Set the angular walton limitation parameter (default 45 degree).
- doptions(10): Set the angular ridge detection (default 45 degree).
Example 5.28 (freeyams.edp)

```plaintext
load "msh3"
load "medit"
load "freeyams"
int nn=20;
mesh Th2=square(nn,nn);
fespace Vh2(Th2,P2);
Vh2 ux,uz,p2;
int[int] rup=[0,2], rdown=[0,1], rmid=[1,1,2,1,3,1,4,1];
real zmin=0,zmax=1;
mesh3 Th=buildlayers(Th2,nn, zbound=[zmin,zmax],
                       reffacemid=rmid, reffaceup = rup, reffacelow = rdown);

mesh3 Th3 = freeyams(Th);
medit("maillagesurfacique",Th3,wait=1);
```

### 5.16 mmg3d

Mmg3d is a 3D remeshing software developed by C. Dobrzynski and P. Frey (http://www.math.u-bordeaux1.fr/~dobj/logiciels/mmg3d.php). To obtain a version of this library send an e-mail at:

cecile.dobrzynski@math.ubordeaux1.fr or pascal.frey@upmc.fr.

This software allows to remesh an initial mesh made of tetrahedra. This initial mesh is adapted to a geometric metric tensor field or to a displacement vector (moving rigid body). The metric can be obtained with mshmet (see section 5.14).

**Remark 5 :**

(a) *If no metric is given, an isotropic metric is computed by analyzing the size of the edges in the initial mesh.*

(b) *If a displacement is given, the vertices of the surface triangles are moved without verifying the geometrical structure of the new surface mesh.*

The parameters of mmg3d are:

- **options=** vector expression. This vector contains the option parameters of mmg3d. It is a vector of 6 values, with the following meaning:

  (0) **optimization parameters:** (default 1)

    - 0: mesh optimization.
    - 1: adaptation with metric (deletion and insertion vertices) and optimization.
    - -1: adaptation with metric (deletion and insertion vertices) without optimization.
    - 4: split tetrahedra (be careful modify the surface).
    - 9: moving mesh with optimization.
    - -9: moving mesh without optimization.

  (1) **debug mode:** (default 0)

    - 1: turn on debug mode.
    - 0: otherwise.
(2) Specify the size of bucket per dimension (default 64)

(3) swapping mode: (default 0)
   1: no edge or face flipping.
   0: otherwise.

(4) insert points mode: (default 0)
   1: no edge splitting or collapsing and no insert points.
   0: otherwise.

(5) verbosity level (default 3)

- memory = integer expression. Set the maximum memory size of new mesh in Mbytes. By default the number of maximum vertices, tetrahedra and triangles are respectively 500 000, 3000 000, 100000 which represent approximately a memory of 100 Mo.

- metric = vector expression. This vector contains the metric given at mmg3d. It is a vector of size \( n_v \) or 6 \( n_v \) respectively for an isotropic and anisotropic metric where \( n_v \) is the number of vertices in the initial mesh. The structure of metric vector is described in the mshmet’s section (section 5.14).

- displacement = \([\Phi_1, \Phi_2, \Phi_3]\) set the displacement vector of the initial mesh \( \Phi(x, y) = [\Phi_1(x, y), \Phi_2(x, y), \Phi_3(x, y)] \).

- displVect = sets the vector displacement in a vector expression. This vector contains the displacement at each point of the initial mesh. It is a vector of size 3 \( n_v \).

An example using this function is given in "mmg3d.edp":

Example 5.29 (mmg3d.edp)

```plaintext
// test mmg3d
load "msh3"
load "medit"
load "mmg3d"
include ".../examples++-3d/cube.idp"

int n=6;
int[int] Nxyz=[12,12,12];
real [int,int] Bxyz=[[0.,1.],[0.,1.],[0.,1.]]; 
int [int,int] Lxyz=[[1,1],[2,2],[2,2]];
mesh3 Th=Cube(Nxyz,Bxyz,Lxyz);
real[int] isometric(Th.nv);
   for( int ii=0; ii<Th.nv; ii++)
      isometric[ii]=0.17;
}
mesh3 Th3=mmg3d( Th, memory=100, metric=isometric);

medit("init",Th);
medit("isometric",Th3);
```

An example of a moving mesh is given in fallingspheres.edp":

Example 5.30 (fallingspheres.edp)

```plaintext
load "msh3"  load "tetgen"  load "medit"  load "mmg3d"
include "MeshSurface.idp"

// build mesh of a box (311) wit 2 holes (300,310)
real hs = 0.8;
int[int] N=[4/hs,8/hs,11.5/hs];
real [int,int] B=[[-2,2],[-2,6],[-10,1.5]];
int [int,int] L=[[311,311],[311,311],[311,311]];
mesh3 ThH = SurfaceHex(N,B,L,1);
mesh3 ThSg =Sphere(1,hs,300,-1);
mesh3 ThSd =Sphere(1,hs,310,-1); ThSd=movemesh3(ThSd,transfo=[x,4+y,z]);
mesh3 ThHS=ThH+ThSg+ThSd;  // gluing surface meshes
medit("ThHS", ThHS);  // see surface mesh

real voltet=(hs^3)/6.;
real[int] domaine = [0,0,-4,1,voltet];
real [int] holes=[0,0,0,0,4,0];
mesh3 Th = tetg(ThHS,switch="pqaAAYYQ",nbofregions=1,regionlist=domaine, nbofholes=2,holelist=holes);
mesh3 Th = medit("Box-With-two-Ball",Th);
```

5.17 A first 3d isotope mesh adaptation process

Example 5.31 (Laplace-Adapt-3d.edp)

```plaintext
load "msh3"  load "tetgen"  load "mshmet"  load "medit"

// build initial mesh
int nn = 6;
int[int] l1111=[1,1,1,1],101=[0,1],111=[1,1];  // label numbering to have all label to 1
```
5.18. Build a 2D Mesh From an Isoline

The idea is to discretize an isoline to fluid meshes. This tool can be useful to construct meshes from images. First, we give an example of the isovalue meshes 0.2 of the analytical function \(\sqrt{(x-1/2)^2+(y-1/2)^2}\), on a unit square.

**Example 5.32 (isoline.edp)**

```cpp
load "isoline" // load the plugin "isoline"
real[int,int] xy(3,1); // to store the isoline points
int[int] be(1); // to store the begin, end couple of lines
{ // a block for memory management
    mesh Th=square(10,10); // ,{x*.5,y*0.5});
    fespace Vh(Th,P1);
    Vh u= sqrt(square(x-0.5)+square(y-0.5));
    real iso= 0.2 ;
    real[int] viso=[iso];
    plot(u,viso=viso,Th); // to see the iso line
}
int nbc= isoline(Th,u,xy,close=1,iso=iso,beginend=be,smoothing=0.1);
```

The isoline parameters are Th the mesh, the expression u, the bidimensional array xy to store the list coordinate of the points. The list of named parameters are:

- **Th**: the mesh to be considered.
- **u**: the expression to be evaluated on the mesh.
- **xy**: an array of coordinates of the isoline.
- **close**: a boolean indicating if the isoline should be closed.
- **iso**: the isoline value.
- **beginend**: an array indicating the begin and end of the isoline.
- **smoothing**: a smoothing factor.
iso= value of the isoline to compute (0 is the default value)

close= close the iso line with the border (def. true), we add the part of the mesh border such the value is less than the iso value

smoothing= nb of smoothing process is the l's where l is the length of the current line component, r the ratio, s is smoothing value. The smoothing default value is 0.

ratio= the ratio ( 1 by default).

eps= relative ε (see code ??) (def 1e-10)

beginend= array to get begin, end couple of each of sub line (resize automatically)

file= to save the data curve in data file for gnu plot

In the array xy you get the list of vertices of the isoline, each connex line go from \( i = i^c_0, \ldots, i^c_1 - 1 \) with \( i^c_0 = be(2 * c) \) \( i^c_1 = be(2 * c + 1) \), and where \( x_i = xy(0, i), y_i = xy(1, i), l_i = xy(2, i) \). Here \( l_i \) is the length of the line (the origin of the line is point \( i^c_0 \)).

The sense of the isoline is such that the upper part is at the left size of the isoline. So here : the minimum is a point 0.5, 05 so the curve 1 turn in the clockwise sense, the order of each component are sort such the the number of point by component is decreasing .

```cpp
cout << " nb of the line component = " << nbc << endl;
cout << " n = " << xy.m << endl; // number of points
cout << "be = " << be << endl; // begin end of the each component

for( int c=0; c<nbc; ++c)
{
    int i0 = be[2*c], i1 = be[2*c+1]-1; // begin,end of the line component
    cout << " Curve " << c << endl;
    for(int i=i0; i<= i1; ++i)
        cout << " x= " << xy(0,i) <<" y= " << xy(1,i) << " s= " << xy(2,i) << endl;
    plot([xy(i0:i1),xy(1,i0:i1)],wait=1,viso=viso,cmm = " curve "+c);
}
```
// end of block for memory management

cout << " len of last curve = " << xy(2,xy.m-1) << endl;;

We also have a new function to parametrize easly a discret Curve defined by couple be,xy.

```cpp
border Curve0(t=0,1) // the extern boundary
{
    int c =0;
    // component 0
    int i0 = be[2*c], i1 = be[2*c+1]-1;
    P=Curve(xy,i0,i1,t); // Curve 0
    label=1;
}
```

```cpp
border Curvel(t=0,1) // component 1
{
    int c =1;
    int i0 = be[2*c], i1 = be[2*c+1]-1;
    P=Curve(xy,i0,i1,t); // Curve 1
    label=1;
}
5.18. BUILD A 2D MESH FROM A ISOLINE

plot(Curve1(100));                          // show curve.
mesh Th= buildmesh(Curve1(-100));           // because
plot(Th,wait=1);                            //

Secondly, we use this idea to build meshes from image, we use the plugins ppm2rnm to read pgm gray scale image, and we extract the gray contour at level 0.25.

Example 5.33 (Leman-mesh.edp)

load "ppm2rnm" load "isoline"
string leman="lg.pgm";                      // see figure 5.40
real AreaLac = 580.03;                       // in Km^2
real hsize= 5;                               // mesh sir in pixel..
real[int,int] Curves(3,1);
int[int] be(1);
int nc;                                      // nb of curve
{
    real[int,int] ff1(leman);                // read image (figure 5.40)
    // and set to an rect. array
    int nx = ff1.n, ny=ff1.m;                // grey value between 0 to 1 (dark)
    // build a Cartesian mesh such that the origin is at the right place.
    mesh Th=square(nx-1,ny-1,[(nx-1)*(x),(ny-1)*(1-y)]);
    // warning the numbering is of the vertices (x,y) is
    // given by i=x/nx + nx*y/ny
    fespace Vh(Th,P1);
    Vh f1; f1[]=ff1;                         // transforme array in finite element function.
    nc=isoline(Th,f1,iso=0.25,close=1,Curves,beginend=be,smoothing=.1,ratio=0.5);
}
    // the longest isoline : the lac..
int ic0=be(0), ic1=be(1)-1;
plot([Curves(0,ic0:ic1),Curves(1,ic0:ic1)], wait=1);
int NC= Curves(2,ic1)/hsize;
border G(t=0,1) {
    P=Curve(Curves,ic0,ic1,t); label= 1 + (x>xl)*2 + (y<yl);}
plot(G(-NC),wait=1);

mesh Th=buildmesh(G(-NC));
plot(Th,wait=1);
real scale = sqrt(AreaLac/Th.area);
Th=movemesh(Th,[x*scale,y*scale]);        // resize the mesh
cout << " Th.area = " << Th.area << " Km^2 " << " == " << AreaLac << " Km^2 " << endl;
plot(Th,wait=1,ps="leman.eps");           // see figure 5.41
Figure 5.40: The image of the leman lac meshes

Figure 5.41: the mesh of lac
Chapter 6

Finite Elements

As stated in Section 2, FEM approximates all functions \( w \) as

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

with finite element basis functions \( \phi_k(x, y) \) and numbers \( w_k \) (\( k = 0, \cdots, M - 1 \)). The functions \( \phi_k(x, y) \) are constructed from the triangle \( T_k \), and called shape functions. In FreeFem++ the finite element space

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

is easily created by

\[
\text{fespace IDspace(IDmesh,<IDFE>) ;}
\]

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

\[
\text{fespace IDspace(IDmesh,<IDFE>,}
\]

\[
\text{   periodic=[[la_1,sa_1],[lb_1,sb_1]},\]
\]

\[
\text{   ...}
\]

\[
\text{   [la_k,sa_k],[lb_k,sb_k]]) ;}
\]

and in 3d

\[
\text{fespace IDspace(IDmesh,<IDFE>,}
\]

\[
\text{   periodic=[[la_1,sa_1,ta_1],[lb_1,sb_1,tb_1]},\]
\]

\[
\text{   ...}
\]

\[
\text{   [la_k,sa_k,ta_k],[lb_k,sb_k,tb_k]]) ;}
\]

where

IDspace is the name of the space (e.g. \( V_h \)),

IDmesh is the name of the associated mesh and \( <\text{IDFE}> \) is a identifier of finite element type.

In 2D we have a pair of periodic boundary condition, if \( [la_i,sa_i],[lb_j,sb_j] \) is a pair of int, and the 2 labels \( la_i \) and \( lb_j \) refer to 2 pieces of boundary to be in equivalence.

If \( [la_i,sa_i],[lb_j,sb_j] \) is a pair of real, then \( sa_i \) and \( sb_j \) give two common abscissa on the two boundary curve, and two points are identified as one if the two abscissa are equal.

In 2D, we have a pair of periodic boundary condition, if \( [la_i,sa_i,ta_i],[lb_j,sb_j,tb_j] \) is a pair of int, the 2 labels \( la_i \) and \( lb_j \) define the 2 piece of boundary to be in equivalence.
CHAPTER 6. FINITE ELEMENTS

If \([l_{a,i}, s_{a,i}, t_{a,i}], [l_{b,i}, s_{b,i}, t_{b,i}]\) is a pair of real, then \(s_{a,i}, t_{a,i}\) and \(s_{b,i}, t_{b,i}\) give two common parameters on the two boundary surface, and two points are identified as one if the two parameters are equal.

**Remark 6** The 2D mesh of the two identified borders must be the same, so to be sure, use the parameter `fixedborder=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 constant discontinuous finite element (2d, 3d), the degrees of freedom are the barycenter element value.

\[
P_{0h} = \{ v \in L^2(\Omega) \mid \text{for all } K \in \mathcal{T}_h \text{ there is } \alpha_K \in \mathbb{R} : \ v|_K = \alpha_K \} \tag{6.1}
\]

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

\[
P_{1h} = \{ v \in H^1(\Omega) \mid \forall K \in \mathcal{T}_h \ v|_K \in P_1 \} \tag{6.2}
\]

**P1dc** piecewise linear discontinuous finite element

\[
P_{1dch} = \{ v \in L^2(\Omega) \mid \forall K \in \mathcal{T}_h \ v|_K \in P_1 \} \tag{6.3}
\]

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:

\[
P_{1bh} = \{ v \in H^1(\Omega) \mid \forall K \in \mathcal{T}_h \ \ v|_K \in P_1 \oplus \text{Span}\{\lambda_0^K \lambda_1^K \lambda_2^K\} \} \tag{6.4}
\]

The 3d case:

\[
P_{1bh} = \{ v \in H^1(\Omega) \mid \forall K \in \mathcal{T}_h \ v|_K \in P_1 \oplus \text{Span}\{\lambda_0^K \lambda_1^K \lambda_2^K \lambda_3^K\} \} \tag{6.5}
\]

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

**P1bl, P1bl3d** piecewise linear continuous finite element plus linear bubble (2d, 3d) the bulle is build by splitting the \(K\) a barycenter in \(d+1\) sub element. (need `load "Element_P1bl"`)  

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

\[
P_{2h} = \{ v \in H^1(\Omega) \mid \forall K \in \mathcal{T}_h \ v|_K \in P_2 \} \tag{6.6}
\]

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,

\[
P_{2bh} = \{ v \in H^1(\Omega) \mid \forall K \in \mathcal{T}_h \ v|_K \in P_2 \oplus \text{Span}\{\lambda_0^K \lambda_1^K \lambda_2^K\} \} \tag{6.7}
\]
\textbf{P2dc} piecewise \( P_2 \) discontinuous finite element,
\[
P_{2dc_h} = \left\{ v \in L^2(\Omega) \mid \forall K \in T_h \ v|_K \in P_2 \right\} \tag{6.8}
\]
Warning, due to interpolation problem, the degree of freedom is not the six \( P_2 \) nodes but six nodes move inside with \( T(X) = G + 0.99(X - G) \) where \( G \) is the barycenter, (version 2.24-4).

\textbf{P3} piecewise \( P_3 \) continuous finite element (2d) (need \texttt{load \ "Element\_P3\"} )
\[
P_{2h} = \left\{ v \in H^1(\Omega) \mid \forall K \in T_h \ v|_K \in P_3 \right\} \tag{6.9}
\]
where \( P_3 \) is the set of polynomials of \( \mathbb{R}^2 \) of degrees \( \leq 3 \).

\textbf{P3dc} piecewise \( P_3 \) discontinuous finite element (2d) (need \texttt{load \ "Element\_P3dc\"} )
\[
P_{2h} = \left\{ v \in L^2(\Omega) \mid \forall K \in T_h \ v|_K \in P_3 \right\} \tag{6.10}
\]
where \( P_3 \) is the set of polynomials of \( \mathbb{R}^2 \) of degrees \( \leq 3 \).

\textbf{P4} piecewise \( P_4 \) continuous finite element (2d) (need \texttt{load \ "Element\_P4\"} )
\[
P_{2h} = \left\{ v \in H^1(\Omega) \mid \forall K \in T_h \ v|_K \in P_4 \right\} \tag{6.11}
\]
where \( P_4 \) is the set of polynomials of \( \mathbb{R}^2 \) of degrees \( \leq 4 \).

\textbf{P4dc} piecewise \( P_4 \) discontinuous finite element (2d) (need \texttt{load \ "Element\_P4dc\"} )
\[
P_{2h} = \left\{ v \in L^2(\Omega) \mid \forall K \in T_h \ v|_K \in P_3 \right\} \tag{6.12}
\]
where \( P_4 \) is the set of polynomials of \( \mathbb{R}^2 \) of degrees \( \leq 3 \).

\textbf{P0Edge} piecewise \( P_0 \) discontinuous finite element (2d) constanl on each edge of the mesh.

\textbf{P1Edge} piecewise \( P_1 \) discontinuous finite element (2d) (need \texttt{load \ "Element\_PkEdge\"} ) \( P_1 \) on each edge of the mesh.

\textbf{P2Edge} piecewise \( P_2 \) discontinuous finite element (2d) (need \texttt{load \ "Element\_PkEdge\"} ) \( P_2 \) on each edge of the mesh.

\textbf{P3Edge} piecewise \( P_3 \) discontinuous finite element (2d) (need \texttt{load \ "Element\_PkEdge\"} ) \( P_3 \) on each edge of the mesh.

\textbf{P4Edge} piecewise \( P_4 \) discontinuous finite element (2d) (need \texttt{load \ "Element\_PkEdge\"} ) \( P_4 \) on each edge of the mesh.

\textbf{P5Edge} piecewise \( P_5 \) discontinuous finite element (2d) (need \texttt{load \ "Element\_PkEdge\"} ) \( P_5 \) on each edge of the mesh.

\textbf{P2Morley} piecewise \( P_2 \) non conform finite element (2d) (need \texttt{load \ "Morley\"} )
\[
P_{2h} = \left\{ v \in L^2(\Omega) \mid \forall K \in T_h \ v|_K \in P_3, \begin{cases} v \text{ continuous at vertices}, \\ \partial_n v \text{ continuous at middle of edge} \end{cases} \right\} \tag{6.13}
\]
where \( P_2 \) is the set of polynomials of \( \mathbb{R}^2 \) of degrees \( \leq 2 \).

Warning to build the interplant of a function \( u \) (scalar) for this finite element, we need the function and 2 partial derivatives \( (u, u_x, u_y) \), so this vectorial finite element with 3 components \( (u, u_x, u_y) \).
See example bilapMorley.edp of examples+++-load for solving BiLaplacien problem:

```plaintext
load "Morley"

fespace Vh(Th,P2Morley); // the Morley finite element space

macro bilaplacien(u,v) ( dxx(u)*dxx(v)+dyy(u)*dyy(v)+2.*dxy(u)*dxy(v)) // fin macro

real f=1;

Vh [u,ux,uy],[v,vx,vy];

solve bilap([u,ux,uy],[v,vx,vy]) =
    int2d(Th)( bilaplacien(u,v) )
    - int2d(Th)(f*v)
    + on(1,2,3,4,u=0,ux=0,uy=0)
```

**HCT** $P_3$ $C^1$ conform finite element (2d) (need `load "Element_HCT"`) one 3 sub triangles (version 3.40).

Let call $\mathcal{T}_h^\Delta$ the sub mesh of $\mathcal{T}_h$ where all triangle are split in 3 at the barycenter.

$$PHCT_h = \left\{ v \in C^1(\Omega) \mid \forall K \in \mathcal{T}_h^\Delta \quad v|_K \in P_3 \right\}$$

(6.14)

where $P_3$ is the set of polynomials of $\mathbb{R}^2$ of degrees $\leq 3$. The degree of freedom are the value and derivative at vertices and normal derivative a middle edge point of initial meshes, and thank to [13].

Warning to build the interplant of a function $u$ (scalar) for this finite element, we need the function and 2 partial derivatives $(u, u_x, u_y)$, so this vectorial finite element with 3 components $(u, u_x, u_y)$ like in previous Finite Element.

**P2BR** (need `load "BernardiRaugel"`) the Bernadi Raugel Finite Element is a Vectorial element (2d) with 2 components. 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, with the Polynomial space is $P^2 + 3$ normals bubbles edges function ($P_2$) and the degree of freedom is 6 values at of the 2 components at the 3 vertices and the 3 flux on the 3 edges so the number degrees of freedom is 9.

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

The 2d case:

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

(6.15)

The 3d case:

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

(6.16)

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

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

and where $\alpha_k^1, \alpha_k^2, \alpha_k^3, \beta_K$ are real numbers.
RT0Ortho Raviart-Thomas Orthogonal, or Nédélec finite element type I of degree 0 in dimension 2

$$RT0Orthoh = \left\{ v \in H(\text{curl}) \mid \forall K \in T_h \quad v_{|K}(x, y) = \left[ \frac{\alpha_K^1}{\alpha_K^2} + \beta_K \right] \frac{x}{y} \right\} \quad (6.17)$$

Edge03d 3d Nédélec finite element or Edge Element of degree 0.

The 3d case:

$$Edge0h = \left\{ v \in H(\text{Curl}) \mid \forall K \in T_h \quad v_{|K}(x, y, z) = \left[ \frac{\alpha_K^1}{\alpha_K^2} + \frac{\beta_K^{13}}{\beta_K^{12}} \times \left[ \frac{x}{y} \frac{y}{z} \right] \right] \right\} \quad (6.18)$$

where by writing $\text{curl} w = \left| \frac{\partial w_2}{\partial x_3} - \frac{\partial w_3}{\partial x_2} \right| \frac{\partial w_3}{\partial x_1} - \frac{\partial w_1}{\partial x_3} \frac{\partial w_2}{\partial x_1}$ with $w = (w_i)_{i=1}^d$,

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

and $\alpha_K^1, \alpha_K^2, \alpha_K^3, \beta_K^{12}, \beta_K^{13}, \beta_K^{23}$ are real numbers.

Edge13d (need load "Element_Mixte3d", version 3.34) 3d Nédélec finite element or Edge Element of degree 1.

Edge23d (need load "Element_Mixte3d", version 3.34) 3d Nédélec finite element or Edge Element of degree 2.

P1nc piecewise linear element continuous at the middle of edge only in 2D.

RT1 (need load "Element_Mixte", version 3.13)

$$RT1h = \left\{ v \in H(\text{div}) \mid \forall K \in T_h \quad (\alpha_K^2, \alpha_K^2, \beta_K) \in P^3_1, v_{|K}(x, y) = \left[ \frac{\alpha_K^1}{\alpha_K^2} + \beta_K \right] \frac{x}{y} \right\} \quad (6.19)$$

RT1Ortho (need load "Element_Mixte", version 3.13, dimension 2)

$$RT1Orthoh = \left\{ v \in H(\text{curl}) \mid \forall K \in T_h \quad (\alpha_K^2, \alpha_K^2, \beta_K) \in P^3_1, v_{|K}(x, y) = \left[ \frac{\alpha_K^1}{\alpha_K^2} + \beta_K \right] \frac{x}{y} \right\} \quad (6.20)$$

BDM1 (need load "Element_Mixte", version 3.13, dimension 2) the Brezzi-Douglas-Marini finite element

$$BDM1h = \left\{ v \in H(\text{div}) \mid \forall K \in T_h \quad v_{|K} \in P^2_1 \right\} \quad (6.21)$$

BDM1Ortho (need load "Element_Mixte", version 3.13, dimension 2) the Brezzi-Douglas-Marini Orthogonal also call Nédélec of type II, finite element

$$BDM1Orthoh = \left\{ v \in H(\text{curl}) \mid \forall K \in T_h \quad v_{|K} \in P^2_1 \right\} \quad (6.22)$$

FEQF (need load "Element_QF", version 3.45, dimension 2 or 3) the finite element to store function at default quadrature points (so the quadrature is qf5pT in 2d and is qfV5 in 3d). For over quadrature you have the following correspondence finite element, quadrature formula.

- FEQF1 $\mapsto$ qf1pT,
- FEQF2 $\mapsto$ qf2pT,
- FEQF5 $\mapsto$ qf5pT,
You can use this element to store and reuse function with long formula in non linear process in integral.

6.1 Use of “fespace” in 2d

With the 2d finite element spaces

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

\[
X_{ph} = \{ v \in X_h \mid v(\{0\}) = v(\{1\}) \} 
\]

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

\[
R_h = \{ v \in H^1([0,1]^2) \mid \forall K \in \mathcal{T}_h \quad v|_{K(x,y)} = |^{\alpha}_K + |^{\gamma}_K \frac{x}{y} \} 
\]

when \( \mathcal{T}_h \) is a mesh \( 10 \times 10 \) of the unit square \( [0,1]^2 \), we only write in FreeFem++:

```
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.

To use FE-functions \( u_h, v_h \in X_h, p_h, q_h \in M_h \) and \( U_h, V_h \in R_h \), we write:

```
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

\[
U_h = |^{Uxh}_{Uyh} \quad \text{and} \quad V_h = |^{Vxh}_{Vyh}
\]
6.2 Use of fespace in 3d

With the 3d finite element spaces

\[
X_h = \{ v \in H^1([0,1]^3) \mid \forall K \in T_h \; v|_K \in P_1 \} \\
X_{ph} = \{ v \in X_h \mid v(0) = v(1), v(\hat{0}) = v(\hat{1}) \} \\
M_h = \{ v \in H^1([0,1]^2) \mid \forall K \in T_h \; v|_K \in P_2 \} \\
R_h = \{ v \in H^1([0,1]^2)^2 \mid \forall K \in T_h \; v|_K(x,y) = \beta_K + \gamma_K |^x_y \}
\]

when \( T_h \) is a mesh \( 10 \times 10 \times 10 \) of the unit cubic \([0,1]^2\), we write in FreeFem++:

```cpp
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 (see Note 6.1)
fespace Mh(Th,P2); // scalar FE
fespace Rh(Th,RT03d); // vectorial FE
```

where \( X_h, M_h, R_h \) expresses finite element spaces (called FE spaces) \( X_h, M_h, R_h \), respectively.

To define and 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:

\[
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

\[
U_h = \begin{bmatrix} Uxh \\ Uyh \\ Uzh \end{bmatrix} \quad \text{and} \quad V_h = \begin{bmatrix} Vxh \\ Vyh \\ Vzh \end{bmatrix}
\]

**Note 6.1** One hard problem of the periodic boundary condition is the mesh must be the same as equivalence face, the BuildLayer mesh generator split each quadrilateral faces with the diagonal passing through vertex with maximal number, so to be sure to have the same mesh one both face periodic the 2d numbering in corresponding edges must be compatible (for example the same variation). By Default, the numbering of square vertex are correct.

To change the mesh numbering you can used the change function like:

```cpp
\{
    // for cleaning memory..
    int[int] old2new(0:Th.nv-1); // array set on 0, 1, .., nv-1
    fespace Vh2(Th,P1);
    Vh2 sorder=x+y; // choose an ordering increasing on 4 square borders with x or y
    sort(sorder[],old2new); // build the inverse permutation
    int[int] new2old=old2new^-1; // inverse the permutation
\}
```
**CHAPTER 6. FINITE ELEMENTS**

Th= change(Th, renumv=new2old); change;

the full example is in examples++-3d/periodic-3d.edp

6.3 Lagrangian Finite Elements

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

$$
\phi_k(x) = \begin{cases} 
1 & \text{if } (x) \in T_k, \\
0 & \text{if } (x) \notin T_k
\end{cases}
$$

If we write $Vh(Th, P0)$; $Vh fh=f(x,y);$

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

$$
f_h = f_h(x, y) = \sum_k f(\frac{\sum q^{k_i}}{d+1}) \phi_k
$$

See Fig. 6.3 for the projection of $f(x, y) = \sin(\pi x) \cos(\pi y)$ on $Vh(Th, P0)$ when the mesh $Th$ is a $4 \times 4$-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 $T_k$](image)

For each vertex $q^i$, the basis function $\phi_i$ in $Vh(Th, P1)$ is given by

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

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

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

If we write
6.3. LAGRANGIAN FINITE ELEMENTS

\[ V_h(\mathbf{Th}, P_1); \ V_h \ f_h = g(x,y); \]
then

\[ f_h = f_h(x,y) = \sum_{i=1}^{n_v} f(q^i) \phi_i(x,y) \]

See Fig. 6.4 for the projection of \( f(x,y) = \sin(\pi x) \cos(\pi y) \) into \( V_h(\mathbf{Th}, P_1) \).

Figure 6.2: Test mesh \( \mathbf{Th} \) for projection

Figure 6.3: projection to \( V_h(\mathbf{Th}, P_0) \)

6.3.3 P2-element

For each vertex or midpoint \( q^i \), the basis function \( \phi_i \) in \( V_h(\mathbf{Th}, P_2) \) is given by

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

The basis function \( \phi_{k_1}(x,y) \) with the vertex \( q^{k_1} \) in Fig. 6.1(b) is defined by the barycentric coordinates:

\[ \phi_{k_1}(x,y) = \lambda_{k_1}^k(x,y)(2\lambda_{k_1}^k(x,y) - 1) \]

and for the midpoint \( q^{k_2} \)

\[ \phi_{k_2}(x,y) = 4\lambda_{k_1}^k(x,y)\lambda_{k_2}^k(x,y) \]

If we write

\[ V_h(\mathbf{Th}, P_2); \ V_h \ f_h = f(x,y); \]
then

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

See Fig. 6.5 for the projection of \( f(x,y) = \sin(\pi x) \cos(\pi y) \) into \( V_h(\mathbf{Th}, P_2) \).
6.4 P1 Nonconforming Element

Refer to [28] 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

\[ V_h(Th, P_{1nc}) \ni fh \quad \text{if} \quad f = f(x, y) \]

then

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

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 \( j = 3 \):

\[
\phi_i(x, y) = a^k_i + b^k_i x + c^k_i y \quad \text{for} \quad (x, y) \in T_k, \\
\phi_i(m^i) = 1, \quad \phi_i(m^j) = 0 \quad \text{if} \quad i \neq j
\]

Strictly speaking \( \partial \phi_i / \partial x, \partial \phi_i / \partial y \) contain Dirac distribution \( \rho \delta_{\partial T_k} \). The numerical calculations will automatically ignore them. In [28], 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

\[
a(\phi_i, \phi_i) > 0, \quad a(\phi_i, \phi_j) \leq 0 \quad \text{if} \quad i \neq j \\
\sum_{k=1}^{n_v} a(\phi_i, \phi_k) \geq 0
\]

2. \( f \geq 0 \Rightarrow u_h \geq 0 \)
6.5. OTHER FE-SPACE

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 $V_h(Th, P_{1nc})$. See Fig. 6.6 for the projection of $f(x, y) = \sin(\pi x) \cos(\pi y)$ into $V_h(Th, P_{1nc})$.

![Figure 6.6: projection to $V_h(Th, P_{1nc})$](image1)

![Figure 6.7: projection to $V_h(Th, P_{1b})$](image2)

6.5 Other FE-space

For each triangle $T_k \in T_h$, 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

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

(6.23)

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

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

2. $\beta_k(q_k) = 1$ where $q_k$ is the barycenter $\frac{q^1 + q^2 + q^3}{3}$.

If we write

$V_h(Th, P_{1b})$; $V_h \ f_0 = f(x, y)$;

then

$$f_0 = f_h(x, y) = \sum_{i=1}^{n_1} f(q^i) \phi_i(x, y) + \sum_{k=1}^{n_t} f(q^k) \beta_k(x, y)$$

See Fig. 6.7 for the projection of $f(x, y) = \sin(\pi x) \cos(\pi y)$ into $V_h(Th, P_{1b})$. 
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 \( N = 2 \)

\[
\text{fespace } Vh(Th, [P0, P1]) ;
\]

make the space

\[
V_h = \{ w = (w_1, w_2) | w_1 \in V_h(T_h, P_0), w_2 \in V_h(T_h, P_1) \}
\]

6.6.1 Raviart-Thomas element

In the Raviart-Thomas finite element \( RT_{0h} \), the degree of freedom are the fluxes across edges \( e \) of the mesh, where the flux of the function \( f : \mathbb{R}^2 \rightarrow \mathbb{R}^2 \) is \( \int_e f.n_e \), \( n_e \) is the unit normal of edge \( e \).

This implies an 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 \( (a, b, c) \). Let denote the vertices numbers by \( i_a, i_b, i_c \), and define the three edge vectors \( e_1, e_2, e_3 \) by

\[
\text{sgn}(i_b - i_c)(b - c), \quad \text{sgn}(i_c - i_a)(c - a), \quad \text{sgn}(i_a - i_b)(a - b),
\]

We get three basis functions,

\[
\phi^k_1 = \frac{\text{sgn}(i_b - i_c)}{2|T_k|}(x - a), \quad \phi^k_2 = \frac{\text{sgn}(i_c - i_a)}{2|T_k|}(x - b), \quad \phi^k_3 = \frac{\text{sgn}(i_a - i_b)}{2|T_k|}(x - c), \quad (6.24)
\]

where \( |T_k| \) is the area of the triangle \( T_k \). If we write

\[
\text{Vh}(Th, RT_0) ; \quad \text{Vh} \ f_1h, f_2h = [f1(x, y), f2(x, y)] ;
\]

then

\[
f_h = f_h(x, y) = \sum_{k=1}^{n_t} \sum_{l=1}^{6} n_{i_k j_l} [e^l_j f_jl(m^l)] \phi_{i_k j_l}
\]

where \( n_{i_k j_l} \) is the \( j_l \)-th component of the normal vector \( n_{i_k} \),

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

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

Figure 6.8: normal vectors of each edge
Example 6.1 mesh Th = square(2,2);

fespace Xh(Th,P1);
fespace Vh(Th,RT0);
Xh uh,vh;
Vh [Uxh,Uyh];

[Uxh,Uyh] = [sin(x),cos(y)]; // ok vectorial FE function
vh= x^2+y^2; // vh
Th = square(5,5);

// change the mesh
// Xh is unchange

// compute on the new Xh
Uh = x; // error: impossible to set only 1 component
// of a vector FE function.

// and now uh use the 5x5 mesh
// but the fespace of vh is alway the 2x2 mesh

plot(uh,ps="onoldmesh.ep",); // figure 6.9
uh = uh; // do a interpolation of vh (old) of 5x5 mesh
// to get the new vh on 10x10 mesh.

plot(uh,ps="onnewmesh.ep",); // figure 6.10
vh((x-1/2,y))= x^2 + y^2; // interpolate vh = ((x-1/2)^2+y^2)

Figure 6.9: vh Iso on mesh 2 × 2
Figure 6.10: vh Iso on mesh 5 × 5

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

```plaintext
real value;
value = uh(2,4); // get value= uh(2,4)
value = Uxh(2,4); // get value= Uxh(2,4)

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 6.2 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 \(T_0^h = \bigcup_k T_0^k\), \(T_1^h = \bigcup_k T_1^k\) be two triangulations of a domain \(\Omega\). Let

\[ V(T_i^h) = \{ C^0(\Omega_i^h) : f|_{T_k^i} \in P_0 \}, \quad i = 0, 1 \]

be the spaces of continuous piecewise affine functions on each triangulation.

Let \(f \in V(T_0^h)\). The problem is to find \(g \in V(T_1^h)\) such that

\[ g(q) = f(q) \quad \forall q \text{ vertex of } T_1^h \]

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 \(T_i^h\), 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 \(T_0^h\) such that in each terminal cell there are at least one, and at most 4, vertices of \(T_0^h\).

For each \(q^1\), vertex of \(T_1^h\) do:

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

Step 2 Find the the nearest vertex \(q_0^0\) to \(q^1\) in that cell.

Step 3 Choose one triangle \(T_k^0 \in T_0^h\) which has \(q_0^0\) 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.
6.7. A FAST FINITE ELEMENT INTERPOLATOR

Step 5 Calculate $g(q^{1})$ on $T_{k}^{0}$ by linear interpolation of $f$:

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

End

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

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-Voronoï mesh generators always triangulate the convex hull of the vertices of the domain. So we make sure that this information is not lost when $T_{h}^{0}, T_{h}^{1}$ 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 6.3 Some time in rare case the interpolation process miss some point, we can change the search algorithm through global variable `searchMethod`

```plaintext
searchMethod=0; // default value for fast search algorithm
searchMethod=1; // safe search algo, use brute force in case of missing point
   // (warning can be very expensive in case of lot point of outside domain)
searchMethod=2; // use always the brute force very very expensive
```

Note 6.4 Step 3 requires an array of pointers such that each vertex points to one triangle of the triangulation.
Note 6.5 The operator $=$ is the interpolation operator of FreeFem++. The continuous finite functions are extended by continuity to the outside of the domain. Try the following example

```
mesh Ths = square(10,10);
mesh Thg = square(30,30, [x*3-1, y*3-1]);
plot(Ths, Thg, ps="overlapTh.eps", wait=1);
fespace Ch(Ths,P2); fespace Dh(Ths,P2dc);
fespace Fh(Thg,P2dc);
Ch us = (x-0.5)*(y-0.5);
Dh vs = (x-0.5)*(y-0.5);
Fh ug=us,vg=vs;
plot(us,ug,wait=1, ps="us-ug.eps"); // see figure 6.12
plot(vs,vg,wait=1, ps="vs-vg.eps"); // see figure 6.13
```

Figure 6.12: Extension of a continuous FE-function
Figure 6.13: 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) - \ell(f,v) + (boundary condition);
```

Note 6.6 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 conditions. The problem is: Find $u$ a real function defined on domain $\Omega$ of $\mathbb{R}^d$ ($d = 2, 3$) such that

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

(6.25)

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 $\Gamma_d \cap \Gamma_n = \emptyset$ and $\Gamma_d \cup \Gamma_n = \partial \Omega$,
- $\kappa$ 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 6.7** This is the well known Neumann boundary condition if $a = 0$, and if $\Gamma_d$ is empty. In this case the function appears in the problem just by its derivatives, so it is defined only up to a constant (if $u$ is a solution then $u + c$ is also a solution).

Let $v$ a regular test function null on $\Gamma_d$, by integration by parts we get

$$-\int_\Omega \nabla \cdot (\kappa \nabla u) \, v \, d\omega = \int_\Omega \kappa \nabla v \cdot \nabla u \, d\omega - \int_\Gamma v \kappa \frac{\partial u}{\partial n} \, d\gamma, = \int_\Omega f \cdot v \, d\omega \quad (6.26)$$

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

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

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

The problem becomes:

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

$$\int_\Omega \kappa \nabla v \cdot \nabla u \, d\omega + \int_\Gamma auv \, d\gamma = \int_\Omega f \cdot v \, d\omega + \int_\Gamma bv \, d\gamma, \quad \forall v \in V_0$$

(6.27)

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

Except in the case of Neumann conditions everywhere, the problem (6.27) is well posed when $\kappa \geq \kappa_0 > 0$.

**Note 6.8** If we have only Neumann boundary condition, linear algebra tells us that the right hand side must be orthogonal to the kernel of the operator for the solution to exist. One way of writing the compatibility condition is:

$$\int_\Omega f \, d\omega + \int_\Gamma b \, d\gamma = 0$$
and a way to fix the constant is to solve for \( u \in H^1(\Omega) \) such that:

\[
\int_{\Omega} \varepsilon uv \, d\omega + \kappa \nabla v \cdot \nabla u \, d\omega = \int_{\Omega} fv \, d\omega + \int_{\Gamma_r} bv \, d\gamma, \quad \forall v \in H^1(\Omega)
\]

(6.28)

where \( \varepsilon \) is a small parameter (\( \sim \kappa \cdot 10^{-10} |\Omega|^\frac{3}{2} \)).

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 (6.27) becomes

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

where \( \text{Th} \) is a mesh of the the bidimensional domain \( \Omega \), and \( \text{gd} \) and \( \text{gn} \) are respectively the boundary label of boundary \( \Gamma_d \) and \( \Gamma_n \).

And the the three dimensional problem (6.27) becomes

```plaintext
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} auv \, d\gamma
    - int3d(Th) (f*v) // \int_{\Omega} fv \, d\omega
    - int2d(Th,gn) ( b * v ) // \int_{\Gamma_r} bv \, d\gamma
    + on(gd) (u= g) ; // u = g on \Gamma_d
```

where \( \text{Th} \) is a mesh of the three dimensional domain \( \Omega \), and \( \text{gd} \) and \( \text{gn} \) 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 (\( n = 2 \times k \)), the \( k \) first function parameters are unknown, and the \( k \) last are test functions.

**Note 6.9** If the functions are a part of vectoriel FE then you must give all the functions of the vectorial FE in the same order (see laplaceMixte problem for example).

**Note 6.10** 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:
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 \texttt{varf}) for possible reuse.

Remark, that the name in the "\texttt{varf}" of the unknown of test function is forgotten, we just used the order in argument list to recall name as in a \texttt{C++} function, see note 6.15.

- the terms of the bilinear form itself: if \( K \) is a given function,

- bilinear part for 3d meshes \( \mathcal{T} \)

\[
\text{int3d}(\mathcal{T})( Kv*w) = \sum_{T \in \mathcal{T}} \int_T K v w
\]
- \( \text{int3d}(\text{Th,1})(K*v*w) = \sum_{T \in \text{Th}, T \subset \Omega_1} \int_T K v w \)
- \( \text{int3d}(\text{Th,levelset}=\phi)(K*v*w) = \sum_{T \in \text{Th}} \int_{T, \phi < 0} K v w \)
- \( \text{int3d}(\text{Th,1,levelset}=\phi)(K*v*w) = \sum_{T \in \text{Th}, T \subset \Omega_1} \int_{T, \phi < 0} K v w \)
- \( \text{int2d}(\text{Th,2,5})(K*v*w) = \sum_{T \in \text{Th}} \int_{(\partial T \cup \Gamma) \cap (\Gamma_2 \cup \Gamma_5)} K v w \)
- \( \text{int2d}(\text{Th,1})(K*v*w) = \sum_{T \in \text{Th}} \int_T K v w \)
- \( \text{int2d}(\text{Th,2,5})(K*v*w) = \sum_{T \in \text{Th}} \int_{(\partial T \cup \Gamma) \cap (\Gamma_2 \cup \Gamma_5)} K v w \)
- \( \text{int2d}(\text{Th,levelset}=\phi)(K*v*w) = \sum_{T \in \text{Th}} \int_{T, \phi = 0} K v w \)
- \( \text{int2d}(\text{Th,1,levelset}=\phi)(K*v*w) = \sum_{T \in \text{Th}, T \subset \Omega_1} \int_{T, \phi = 0} K v w \)
- \( \text{intallfaces}(\text{Th})(K*v*w) = \sum_{T \in \text{Th}} \int_{\partial T} K v w \)
- \( \text{intallfaces}(\text{Th,1})(K*v*w) = \sum_{T \in \text{Th}, T \subset \Omega_1} \int_{\partial T} K v w \)

- they contribute to the sparse matrix of type \text{matrix} which, whether declared explicitly or not is contructed by \text{FreeFem++}.

- \text{bilinear part for 2d meshes Th}

- \( \text{int2d}(\text{Th})(K*v*w) = \sum_{T \in \text{Th}} \int_T K v w \)
- \( \text{int2d}(\text{Th,1})(K*v*w) = \sum_{T \in \text{Th}, T \subset \Omega_1} \int_T K v w \)
- \( \text{int2d}(\text{Th,levelset}=\phi)(K*v*w) = \sum_{T \in \text{Th}} \int_{T, \phi < 0} K v w \)
- \( \text{int2d}(\text{Th,1,levelset}=\phi)(K*v*w) = \sum_{T \in \text{Th}, T \subset \Omega_1} \int_{T, \phi < 0} K v w \)
- \( \text{int1d}(\text{Th,2,5})(K*v*w) = \sum_{T \in \text{Th}} \int_{(\partial T \cup \Gamma) \cap (\Gamma_2 \cup \Gamma_5)} K v w \)
- \( \text{int1d}(\text{Th,1})(K*v*w) = \sum_{T \in \text{Th}, T \subset \Omega_1} \int_T K v w \)
- \( \text{int1d}(\text{Th,2,5})(K*v*w) = \sum_{T \in \text{Th}} \int_{(\partial T \cup \Gamma) \cap (\Gamma_2 \cup \Gamma_5)} K v w \)
- $\text{int1d}(Th,\text{levelset} = \phi)(Kv*w) = \sum_{T \in Th} \int_{T, \phi = 0} Kvw$

- $\text{int1d}(Th,1,\text{levelset} = \phi)(Kv*w) = \sum_{T \in Th, T \subset \Omega_1} \int_{T, \phi = 0} Kvw$

- $\text{intalledges}(Th)(Kv*w) = \sum_{T \in Th} \int_{\partial T} Kvw$

- $\text{intalledges}(Th,1)(Kv*w) = \sum_{T \in Th, T \subset \Omega_1} \int_{\partial T} Kvw$

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

- the right hand-side of the PDE in 3d, the terms of the linear form: for given functions $K, f$:

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

  - $\text{int3d}(Th,1)(K*w) = \sum_{T \in Th, T \in \Omega} \int_{T} K w$

  - $\text{int3d}(Th,\text{levelset} = \phi)(K*w) = \sum_{T \in Th} \int_{T, \phi < 0} K w$

  - $\text{int3d}(Th,1,\text{levelset} = \phi)(K*w) = \sum_{T \in Th, T \subset \Omega_1} \int_{T, \phi < 0} K w$

  - $\text{int2d}(Th,2,5)(K*w) = \sum_{T \in Th} \int_{(\partial T \cup \Gamma) \cap (\Gamma_2 \cup \Gamma_5)} K w$

  - $\text{int2d}(Th,\text{levelset} = \phi)(K*w) = \sum_{T \in Th} \int_{T, \phi = 0} K w$

  - $\text{int2d}(Th,1,\text{levelset} = \phi)(K*w) = \sum_{T \in Th, T \subset \Omega_1} \int_{T, \phi = 0} K w$

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

- a vector of type real[int]

- the right hand-side of the PDE in 2d, the terms of the linear form: for given functions $K, f$:

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

  - $\text{int2d}(Th,1)(K*w) = \sum_{T \in Th, T \in \Omega} \int_{T} K w$

  - $\text{int2d}(Th,\text{levelset} = \phi)(K*w) = \sum_{T \in Th} \int_{T, \phi < 0} K w$

  - $\text{int2d}(Th,1,\text{levelset} = \phi)(K*w) = \sum_{T \in Th, T \subset \Omega_1} \int_{T, \phi < 0} K w$
- \( \text{int1d}(\text{Th}, 2, 5)( K*w) = \sum_{T \in \text{Th}} \int_{(\partial T \cup \Gamma) \cap (\Gamma_2 \cup \Gamma_5)} K w \)

- \( \text{int1d}(\text{Th}, \text{levelset}=\phi)( K*w) = \sum_{T \in \text{Th}} \int_{T, \phi=0} K w \)

- \( \text{int1d}(\text{Th}, i, \text{levelset}=\phi)( K*w) = \sum_{T \in \text{Th}, T \subset \Omega_i} \int_{T, \phi=0} K w \)

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

- a vector of type \( \text{real}[\text{int}] \)

• The boundary condition terms:

  - An ”on” scalar form (for Dirichlet): \( \text{on}(1, u = g) \)
    The meaning is for all degree of freedom \( i \) of the boundary refered by ”1”, the diagonal term of the matrix \( a_{ii} = tgv \) with the terrible geant value \( tgv = 1 \times 10^{30} \) by default and the right hand side \( b[i] = ”(\Pi_h g)[i]” \times tgv \), where the ”(\Pi_h g)[i]” is the boundary node value given by the interpolation of \( g \).
    remark, if \( tgv < 0 \) then we put to 0 all term of the line \( i \) in the matrix, except diagonal term \( a_{ii} = 1 \), and \( b[i] = ”(\Pi_h g)[i]” \). (version \( v3.10 \)).

  - An ”on” vectorial form (for Dirichlet): \( \text{on}(1, u1=g1, u2=g2) \) If you have vectorial finite element like \( \text{RT0} \), the 2 components are coupled, and so you have: \( b[i] = ”(\Pi_h (g1, g2))[i]” \times tgv \), where \( \Pi_h \) is the vectorial finite element interpolant.

  - a linear form on \( \Gamma \) (for Neumann in 2d) \( -\text{int1d}(\text{Th})( f*w) \) or \( -\text{int1d}(\text{Th}, 3))( f*w) \)

  - a bilinear form on \( \Gamma \) or \( \Gamma_2 \) (for Robin in 2d) \( \text{int1d}(\text{Th})( K*v*w) \) or \( \text{int1d}(\text{Th}, 2))( K*v*w) \).

  - a linear form on \( \Gamma \) (for Neumann in 3d) \( -\text{int2d}(\text{Th})( f*w) \) or \( -\text{int2d}(\text{Th}, 3))( f*w) \)

  - a bilinear form on \( \Gamma \) or \( \Gamma_2 \) (for Robin in 3d) \( \text{int2d}(\text{Th})( K*v*w) \) or \( \text{int2d}(\text{Th}, 2))( K*v*w) \).

Note 6.11

• 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 \( \text{int1d}(\text{Th})( K*v*w - f*w) \).
6.11 Numerical Integration

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

$$\int_D f(x) = \sum_{\ell=1}^L c_{\ell} f(\xi_{\ell}) \quad (6.29)$$

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

$$\left| \int_D f(x) - \sum_{\ell=1}^L \omega_{\ell} f(\xi_{\ell}) \right| \leq C|D|h^{r+1} \quad (6.30)$$

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

$$\{(x, y) \mid x = (1 - t)q_i^x + tq_j^x, y = (1 - t)q_i^y + tq_j^y, 0 \leq t \leq 1\}$$

For a domain $\Omega_h = \bigcup_{k=1}^n T_k$, $T_h = \{T_k\}$, we can calculate the integral over $\Gamma_h = \partial\Omega_h$ by

$$\int_{\Gamma_h} f(x)ds = \text{int1d}(Th) (f) = \text{int1d}(Th,qfe=*)(f) = \text{int1d}(Th,qforder=*)(f)$$

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

<table>
<thead>
<tr>
<th>$L$</th>
<th>$(qfe=)$</th>
<th>$qforder=$</th>
<th>point in $<a href="=t">q_i q_j</a>$</th>
<th>$\omega_{\ell}$</th>
<th>exact on $P_k$, $k =$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>qf1pE</td>
<td>2</td>
<td>$1/2$</td>
<td>$</td>
<td>q_i q_j</td>
</tr>
<tr>
<td>2</td>
<td>qf2pE</td>
<td>3</td>
<td>$(1 \pm \sqrt{1/3})/2$</td>
<td>$</td>
<td>q_i q_j</td>
</tr>
<tr>
<td>3</td>
<td>qf3pE</td>
<td>6</td>
<td>$(1 \pm \sqrt{3/5})/2$</td>
<td>$(5/18)</td>
<td>q_i q_j</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$1/2$</td>
<td>$(8/18)</td>
<td>q_i q_j</td>
</tr>
<tr>
<td>4</td>
<td>qf4pE</td>
<td>8</td>
<td>$(1 \pm \sqrt{245+14\sqrt{70}}/21)/2$, $(1 \pm \sqrt{245-14\sqrt{70}}/21)/2$,</td>
<td>$18-\sqrt{30}</td>
<td>q_i q_j</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$18+\sqrt{30}</td>
<td>q_i q_j</td>
<td>$</td>
</tr>
<tr>
<td>5</td>
<td>qf5pE</td>
<td>10</td>
<td>$(1 \pm \sqrt{245+14\sqrt{70}}/21)/2$, $(1 \pm \sqrt{245-14\sqrt{70}}/21)/2$,</td>
<td>$\frac{322-13\sqrt{70}}{1800}</td>
<td>q_i q_j</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\frac{322+13\sqrt{70}}{1800}</td>
<td>q_i q_j</td>
<td>$</td>
</tr>
<tr>
<td>2</td>
<td>qf1pElump</td>
<td>2</td>
<td>0</td>
<td>$</td>
<td>q_i q_j</td>
</tr>
</tbody>
</table>

where $|q_i q_j|$ is the length of segment $\bar{q_i q_j}$. For a part $\Gamma_1$ of $\Gamma_h$ with the label “1”, we can calculate the integral over $\Gamma_1$ by

$$\int_{\Gamma_1} f(x,y)ds = \text{int1d}(Th,1)(f) = \text{int1d}(Th,1,qfe=qf2pE)(f)$$
The integrals 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 $P(\xi, \eta)$:

\[
|T_k| = \frac{1}{2} \begin{vmatrix}
1 & q^{k_1}_x & q^{k_1}_y \\
1 & q^{k_2}_x & q^{k_2}_y \\
1 & q^{k_3}_x & q^{k_3}_y \\
\end{vmatrix}
\]

\[
D_1 = \begin{vmatrix}
x & y \\
q^{k_1}_x & q^{k_1}_y \\
1 & 1 \\
\end{vmatrix}
\]

\[
D_2 = \begin{vmatrix}
x & y \\
q^{k_2}_x & q^{k_2}_y \\
1 & 1 \\
\end{vmatrix}
\]

\[
D_3 = \begin{vmatrix}
x & y \\
q^{k_3}_x & q^{k_3}_y \\
1 & 1 \\
\end{vmatrix}
\]

\[
\xi = \frac{1}{2} D_1 / |T_k| \quad \eta = \frac{1}{2} D_2 / |T_k| \quad \text{then } 1 - \xi - \eta = \frac{1}{2} D_3 / |T_k|
\]

For a two dimensional domain or a border of three dimensional domain $\Omega_h = \sum_{k=1}^{n_h} T_k$, $\text{Th}_h = \{T_k\}$, we can calculate the integral over $\Omega_h$ by
\[
\int_{\Omega_h} f(x, y) = \int \text{int2d}(\text{Th}) (f)
\]

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

<table>
<thead>
<tr>
<th>L</th>
<th>qft=</th>
<th>qforder=</th>
<th>point in $T_k$</th>
<th>$\omega_l$</th>
<th>exact on $P_k, k =$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>q1pT</td>
<td></td>
<td>$(\frac{1}{3}, \frac{1}{3})$</td>
<td>$</td>
<td>T_k</td>
</tr>
<tr>
<td>3</td>
<td>q2pT</td>
<td></td>
<td>$(\frac{1}{3}, \frac{1}{3})$</td>
<td>$</td>
<td>T_k</td>
</tr>
<tr>
<td>7</td>
<td>q5pT</td>
<td></td>
<td>$(\frac{1}{3}, \frac{1}{3})$, $(\frac{1}{3}, \frac{1}{3})$</td>
<td>0.225$</td>
<td>T_k</td>
</tr>
<tr>
<td>15</td>
<td>q7pT</td>
<td></td>
<td>$(0, 0)$, $(1, 0)$, $(0, 1)$</td>
<td>$</td>
<td>T_k</td>
</tr>
<tr>
<td>21</td>
<td>q9pT</td>
<td></td>
<td>$(0, 0)$, $(0, 0)$, $(0, 0)$</td>
<td>$</td>
<td>T_k</td>
</tr>
</tbody>
</table>

For a two dimensional domain or a border of three dimensional domain $\Omega_h = \sum_{k=1}^{n_h} T_k$, $\text{Th}_h = \{T_k\}$, we can calculate the integral over $\Omega_h$ by
\[
\int_{\Omega_h} f(x, y) = \int \text{int2d}(\text{Th}) (f)
\]

where * stands for the name of quadrature formula or the order of the Gauss formula.
For a three dimensional domain $\Omega_h = \sum_{k=1}^{n_t} T_k$, $T_h = \{T_k\}$, we can calculate the integral over $\Omega_h$ by
\[
\int_{\Omega_h} f(x,y) = \text{int3d}(T_h)(f) = \text{int3d}(T_h, qfV=*)(f) = \text{int3d}(T_h, qforder=*)(f)
\]
where * stands for the name of quadrature formula or the order of the Gauss formula.

<table>
<thead>
<tr>
<th>$L$</th>
<th>qfV=</th>
<th>qforder=</th>
<th>point in $T_k \in \mathbb{R}^3$</th>
<th>$\omega_k$</th>
<th>exact on $P_k, k =$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>qfV1</td>
<td>2</td>
<td>$\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$</td>
<td>$</td>
<td>T_k</td>
</tr>
<tr>
<td>4</td>
<td>qfV2</td>
<td>3</td>
<td>$G(0.58, 0.13, 0.13)$</td>
<td>$</td>
<td>T_k</td>
</tr>
<tr>
<td>14</td>
<td>qfV5</td>
<td>6</td>
<td>$G(0.72, 0.092, 0.092)$</td>
<td>$0.073</td>
<td>T_k</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$G(0.067, 0.31, 0.31)$</td>
<td>$0.11</td>
<td>T_k</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$G(0.45, 0.045, 0.45)$</td>
<td>$0.042</td>
<td>T_k</td>
</tr>
<tr>
<td>4</td>
<td>qfV1lump</td>
<td>2</td>
<td>$G(1,0,0)$</td>
<td>$</td>
<td>T_k</td>
</tr>
</tbody>
</table>

Where $G_4(a, b, b)$ 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 $G_6(a, b, b)$ such that $2a + 2b = 1$ is the set of the six points in barycentric coordinate
\[
\{(a, a, b, b), (a, b, a, b), (a, b, b, a), (b, b, a, a), (b, a, a, b), (b, a, b, a)\}
\]


Note 6.13 By default, we use the formula which is exact for polynomials of degree 5 on triangles or edges (in bold in three tables).

This possible to add an own quadrature formulae with using plugin "qf11to25" on segment, triangle or Tetrahedron. The quadrature formulae in $D$ dimension is a bidimensional array of size $N_q \times (D + 1)$ such that the $D + 1$ value of on row $i = 0, ..., N_p - 1$ are $w^i, \hat{x}_1^i, ..., \hat{x}_D^i$ where $w^i$ is the weight of the quadrature point, and $1 - \sum_{k=1}^{D} \hat{x}_k^i$ is the barycentric coordinate the quadrature point.

```plaintext
// just for test ... (version 3.19-1)
// load plugin
load "qf11to25"

// Quadrature on segment
real[int,int] qql1=[[0.5,0], [0.5,1]];
QF1 qf1(1,qql1);

// def of quadrature formulae qf1 on segment
// remark:
// 1 is the order of the quadrature exact for polynome of degree < 1)
```
// Quadrature on triangle
real[int,int] qq2 = {
    [1./3,0,0],
    [1./3,1,0],
    [1./3,0,1]};
QF2 qf2(1,qq2);      // def of quadrature formulae qf2 on triangle
// remark:
// 1 is the order of the quadrature exact for polynome of degree < 1
// so must have \Rightarrow \sum w_i = 1
// Quadrature on Tetrahedron
real[int,int] qq3 = {
    [1./4,0,0,0],
    [1./4,1,0,0],
    [1./4,0,1,0],
    [1./4,0,0,1]};
QF3 qf3(1,qq3);      // def of quadrature formulae qf3 on get.
// remark:
// 1 is the order of the quadrature exact for polynome of degree < 1

real I1 = int1d(Th,qfe=qf1)(x^2) ;
real I1l = int1d(Th,qfe=qf1pElump)(x^2) ;
real I2 = int2d(Th,qft=qf2)(x^2) ;
real I2l = int2d(Th,qft=qf1pTlump)(x^2) ;

cout "<< I1 << " == " << I1l << endl;
cout "<< I2 << " == " << I2l << endl;
assert( abs(I1-I1l) < 1e-10);
assert( abs(I2-I2l) < 1e-10);

the output is

1.67 == 1.67
0.335 == 0.335

6.12 Variational Form, Sparse Matrix, PDE Data Vector

In FreeFem++ it is possible to define variational forms, and use them to build matrices and vectors and store them to speed-up the script (4 times faster here).
For example let us 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

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

where $V_0 = \{ w \in H^1(\Omega)/w|_{\Gamma_{24}} = 0 \}$. 
6.12. VARIATIONAL FORM, SPARSE MATRIX, PDE DATA VECTOR

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).

$$u^n = A^{-1}b^n, \quad b' = b_0 + Mu^{n-1}, \quad b'' = \frac{1}{\varepsilon}b_{cl}$$

$$b_i' = \begin{cases} b''_i & \text{if } i \in \Gamma_{24} \\ b'_i & \text{else if } i \not\in \Gamma_{24} \end{cases}$$

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

$$A_{ij} = \begin{cases} \frac{1}{\varepsilon} & \text{if } i \in \Gamma_{24}, and j = i \\ \int_{\Omega} w_j w_i / dt + k(\nabla w_j \cdot \nabla w_i) + \int_{\Gamma_{13}} \alpha w_j w_i & \text{else if } i \not\in \Gamma_{24}, or j \neq i \end{cases}$$

$$M_{ij} = \begin{cases} \frac{1}{\varepsilon} & \text{if } i \in \Gamma_{24}, and j = i \\ \int_{\Omega} w_j w_i / dt & \text{else if } i \not\in \Gamma_{24}, or j \neq i \end{cases}$$

$$b_0,i = \int_{\Gamma_{13}} \alpha u_{ue} w_i$$

$$b_{cl} = u^0 \text{ the initial data}$$

```cpp
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 $A,M$.

```cpp
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);

```cpp
varf vMass (u,v)= int2d(Th)( u*v/dt ) + on(2,4,u=1);

real tgv = 1e30;
matrix A= vthermic(Vh,Vh,tgv=tgv,solver=CG);
matrix M= vMass(Vh,Vh);
```

Now, to build the right hand size we need 4 vectors.

```cpp
real[int] b0 = vthermic0(0,Vh); // constant part of the RHS
real[int] bcn = vthermic0(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 6.14 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 9.6.3 Examples++-tutorial/StokesUzawa.edp gives a real example of using all this features.

And the new version of the algorithm is now:

```cpp
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 dependent 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 6.15 The functions appearing in the variational form are formal and local to the `varf` definition, the only important thing is the order in the parameter list, like in

```cpp
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 variational form $a$ of type `varf` simply write:

```cpp
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
```

Possible named parameters in " [, ... ] " are

```cpp
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.

```cpp
factorize = if true then do the matrix factorization for LU, Cholesky or Crout, the default value is false.
```

```cpp
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

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

tgv = 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 6.16** The line of the matrix corresponding to the space \( \mathcal{W}h \) and the column of the matrix corresponding to the space \( \mathcal{V}h \).

To build the dual vector \( b \) (of type real[int]) from the linear part of the variational form \( a \) do simply

```c
real b(Vh.ndof);
b = a(0,Vh);
```

A first example to compute the area of each triangle \( K \) of mesh \( Th \), just do:

```c
fespace Nh(Th,P0); // the space function constant / triangle
Nh areaK;
varf varea(unused,chiK) = int2d(Th)(chiK);
etaK[i] = 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:

\[ \eta_K[i] = \int_{1_K} = \int_{K_i} 1; \]

Now, we can use this to compute error indicators 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 \( Th \).

```c
fespace Vh(Th,P1);
Vh h ;
real[int] count(Th.nv);
varf vmeshsizen(u,v)=intalledges(Th,qfnbpE=1)(v);
varf vedgecount(u,v)=intalledges(Th,qfnbpE=1)(v/lenEdge);
    // computation of the mesh size
    // -----------------------------
count=vedgecount(0,Vh); // number of edge / vertex
```
\( h[k] = \text{vmeshsizen}(0, \text{Vh}); \)  // sum length edge / vertex
\( h[k] = h[k] / \text{count}; \)  // mean length 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 (\( h_{\text{Triangle}} \)), \( h_e \) is the size of the current edge (\( \text{lenEdge} \)), \( n \) the normal.

```plaintext
fespace Nh(Th, P0);  // the space function constant / triangle
Nh etak;
varf vetaK(unused, chiK) =
    intalledges(Th)(chiK*lenEdge*square(jump(N.x*dx(u)+N.y*dy(u))))  
    + int2d(Th)(chiK*square(hTriangle*(f+dxx(u)+dyy(u))));

etak[] = vetaK(0, Ph);
```

We add automatic expression optimization by default, if this optimization creates problems, it can be removed with the keyword `optimize` as in the following example:

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

or you can also do optimisation and remove the check by setting `optimize=2` (version 3.38).

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

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

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

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

and after some operations on sparse matrices are available for example

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

It is also possible to store the matrix of a linear interpolation operator from a finite element space $V_h$ to another $W_h$ to interpolate ($W_h, V_h, \ldots$) a function. Note that the continuous finite functions are extended by continuity outside of the domain.

The named parameters 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 $\partial_y$
  - 3 interpolate the $\partial_z$

**U2Vc=** set the which is the component of $W_h$ come in $V_h$ in interpolate process in a int array so the size of the array is number of component of $W_h$, if the put $-1$ then component is set to 0, like in the following example: (by default the component number is unchanged).

```cpp
fespace V4h(Th4,[P1,P1,P1,P1]);
fespace V3h(Th,[P1,P1,P1]);
int[int] u2vc=[1,3,-1]; // -1 => put zero on the component
matrix IV34= interpolate(V3h,V4h,inside=0,U2Vc=u2vc);          // V3h <- V4h
V4h [a1,a2,a3,a4]=[1,2,3,4];
V3h [b1,b2,b3]=[10,20,30];
b1[]=IV34*a1[];
```

So here we have: $b_1 = 2$, $b_2 = 4$, $b_3 = 0$ . .

Example 6.2 (mat_interpol.edp)

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

matrix IV= interpolate(Vh,Vh4);          // here the function is
                                             // exended by continuity

cout << " IV Vh<-Vh4 " << IV << endl;
Vh v, vv;   Vh4 v4=x*y;
v=v4;       vv[]= IV*v4[];        // here v == vv =>
real[int] diff= vv[] - v[];
cout << " || v - vv || = " << diff.linfty << endl;
assert(diff.linfty<= 1e-6);
```
matrix IV0 = interpolate(Vh, Vh4, inside=1); // here the function is
// extended by zero

cout << " IV Vh<-Vh4 (inside=1) " << IV0 << endl;

matrix IVt0 = interpolate(Vh, Vh4, inside=1, t=1);

cout << " IV Vh<-Vh^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 " << IW4V << endl;

Build interpolation matrix \( A \) at an array of points \((xx[j], yy[j])\), \(i = 0, 2\) here

\[
a_{i,j} = dop(w_i^c(xx[j], yy[j]))
\]

where \( w_i \) is the basic finite element function, \( c \) the component number, \( dop \) the type of diff operator like in op def.

real[int] xx=[.3,.4], yy=[.1,.4];
int c=0, dop=0;
matrix Ixx = interpolate(Vh, xx, yy, op=dop, composante=c);

cout << Ixx << endl;

Vh ww;
real[int] dd=[1,2];
ww[] = Ixx*dd;

Example 6.3 (schwarz2.edp)
The following shows how to implement with an interpolation matrix a domain decomposition algorithm based on Schwarz method with Robin conditions.

Given a non-overlapping partition \( \bar{\Omega} = \bar{\Omega}_0 \cup \bar{\Omega}_1 \) with \( \Omega_0 \cap \Omega_1 = \emptyset \), \( \Sigma := \bar{\Omega}_0 \cap \bar{\Omega}_1 \) the algorithm is

\[
-\Delta u_i = f \text{ in } \Omega_i, \ i = 0, 1, \quad \frac{\partial(u_1 - u_0)}{\partial n} + \alpha(u_1 - u_0) = 0 \text{ on } \Sigma.
\]

The same in variational form is:

\[
\int_{\Omega_i} \nabla u_i \cdot \nabla v + \int_{\Sigma} \alpha u_i v = \int_{\Omega_i} f v
\]

\[
- \int_{\Sigma} (\nabla u_j \cdot \nabla v - f v) + \int_{\Sigma} \alpha u_j v, \quad \forall v \in H^1_0(\Omega), \ i,j = [0,1] \cup [1,0]
\]

To discretized with the \( P^1 \) triangular Lagrangian finite element space \( V_h \) simply replace \( H^1_0(\Omega) \) by \( V_h(\Omega_0) \cup V_h(\Omega_1) \). Then difficulty is to compute \( \int_{\Omega_i} \nabla u_j \cdot \nabla v \) when \( v \) is a basis function of \( V_h(\Omega_i), \ i \neq j \). It is done as follows (with \( \Gamma = \partial \Omega \))

fespace Vh0(Th[0], P1), Vh1(Th[1], P1);
Vh0 u0=0; Vh1 u1=0;

macro grad(u) [dx(u), dy(u)] // from file schwarz2.edp
varf a(u,v)=int2d(Th[i]) (grad(u)’*grad(v))+int1d(Th[i], Interface) (alpha*u*v)
+on(Gamma, u=0);
varf b(u,v)=int2d(Th[i]) (f*v)+on(Gamma, u=0);
6.14 Finite elements connectivity

Here, we show how to get informations on a finite element space $W_h(T_n,*)$, where "*" may be P1, P2, P1nc, etc.

- $W_h.nt$ gives the number of element of $W_h$
- $W_h.ndof$ gives the number of degrees of freedom or unknown
- $W_h.ndofK$ gives the number of degrees of freedom on one element
- $W_h(k,i)$ gives the number of $i$th degrees of freedom of element $k$.

See the following example:

```cpp
Example 6.4 (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, kdf= Wh.ndofK ; // element 2
```
cout << " df of element " << k << ":" ;
for (int i=0;i<kdf;i++) cout << Wh(k,i) << " ";
cout << endl;

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 the finite element method can be a huge set of data, so it is very important to render them easy to grasp. 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 \texttt{plot} (see Section 7.1 below). For publishing purpose, FreeFem++ can store these plots as postscript files.

Another method is to use external tools, for example, gnuplot (see Section 7.2), medit (see Section 7.3) using the command \texttt{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.

\textbf{Note 7.1} The length of an arrow is always bound to be in $[5\%, 5\%]$ of the screen size, to see something (else it will only look like porcupine).

The parameters are

\begin{description}
\item[\texttt{wait} =] boolean expression to wait or not (by default no wait). If true we wait for a keyboard event or mouse event, they respond to an event by the following characters
\item[\texttt{enter}] try to show plot
\item[\texttt{p}] previous plot (10 plots saved)
\item[?] show this help
\item[+] zoom in/out around the cursor 3/2 times
\item[=] reset vue
\item[r] refresh plot
\item[\texttt{up, down, left, right}] special keys to translate
\item[3] switch 3d/2d plot keys:
\begin{itemize}
\item[\texttt{z, Z}] focal zoom unzoom
\item[H, h] switch increase or decrease the Z scale of the plot
\item[mouse motion]
\end{itemize}
\end{description}
- **left button** rotate
- **right button** zoom (ctrl+button on mac)
- **right button** +alt translate (alt+ctrl+button on mac)

**a, A** increase or decrease the arrow size

**B** switch between show border meshes or not

**i, I** update or not: the min/max bound of the functions to the window

**n, N** decrease or increase the number of iso value array

**b** switch between black and white or color plotting

**g** switch between grey or color plotting

**f** switch between filling iso or iso line

**l** switch between lighting or not

**v** switch between show or not the numerical value of colors

**m** switch between show or not meshes

**w** window dump in file ffglutXXXX.ppm

* keep/unkeep viewpoint for next plot

**k** complex data / change view type

ESC close the graphics process before version 3.22, after no way to close.

otherwise  do nothing.

**ps=** string expression for the name of the file to save the plot in postscript (sorry no save of 3d plot)

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

**fill=** fill color between iso-values (mandatory of P0 finite element).

**cmm=** string expression to write the graphic window into

**value=** to plot the value of isolines and the value of vector arrows.

**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 of isovalues (an array real[int] of increasing values)

**varrow=** sets the array of color arrows values (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.
7.1. PLOT

\texttt{hsv=} (array of float) to defined color of 3*n value in HSV color model declared for example by

\begin{verbatim}
    real[int] colors = [h1,s1,v1, ... , hn,vn,vn];
\end{verbatim}

where \(h_i,s_i,v_i\) is the \(i\)th color to defined the color table.

\texttt{boundary=} (bool) to plot or not the boundary of the domain (true by default).

\texttt{dim=} (int) sets dim of the plot 2d or 3d (2 by default)

\texttt{add=} \texttt{Not used}

\texttt{prev=} \texttt{Not used}

\texttt{ech=} \texttt{Not used}

\texttt{ZScale=} \texttt{Not used}

\texttt{WhiteBackground=} \texttt{Not used}

\texttt{OpaqueBorders=} \texttt{Not used}

\texttt{BorderAsMesh=} \texttt{Not used}

\texttt{ShowMeshes=} \texttt{Not used}

\texttt{ColorScheme=} \texttt{Not used}

\texttt{ArrowShape=} \texttt{Not used}

\texttt{ArrowSize=} \texttt{Not used}

\texttt{ComplexDisplay=} \texttt{Not used}

\texttt{LabelColors=} \texttt{Not used}

\texttt{ShowAxes=} \texttt{Not used}

\texttt{CutPlane=} \texttt{Not used}

\texttt{CameraPosition=} \texttt{Not used}

\texttt{CameraFocalPoint=} \texttt{Not used}

\texttt{CameraViewUp=} \texttt{Not used}

\texttt{CameraViewAngle=} \texttt{Not used}

\texttt{CameraClippingRange=} \texttt{Not used}

\texttt{CutPlaneOrigin=} \texttt{Not used}

\texttt{CutPlaneNormal=} \texttt{Not used}

\texttt{WindowIndex=} set glut window for display for multi windows graphics.
For example:

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

// compute a cut
for (i=0;i<10;i++)
{
    x=i/10.; y=i/10.;
    xx[i]=i;
    yy[i]=uh;
    // value of uh at point (i/10., i/10.)
}

plot(Th,uh,[uh,vh],value=true,ps="three.eps",wait=true);  // figure 7.1
// zoom on box defined by the two corner points [0.1,0.2] and [0.5,0.6]
plot(uh,[uh,vh],bb=[[0.1,0.2],[0.5,0.6]],
    wait=true,grey=1,fill=1,value=1,ps="threeg.eps");  // figure 7.2
plot([xx,yy],ps="likegnu.eps",wait=true);  // figure 7.3
```

![Figure 7.1](three.png)  ![Figure 7.2](threeg.png)

Figure 7.1: mesh, isovalue, and vector  Figure 7.2: enlargement in grey of isovalue, and vector

To change the color table and to choose the value of iso line you can do:

```c
// 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
// 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%
```
Figure 7.3: Plots a cut of uh. Note that a refinement of the same can be obtained in combination with gnuplot

```plaintext
//
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\textsuperscript{1} is installed, and only on unix computer. Add to the previous example:

\begin{verbatim}
{ 
    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");
}
\end{verbatim}

Figure 7.6: Plots a cut of uh with gnuplot

7.3 link with medit

As said above, medit\textsuperscript{2} is 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 ffmedit name. Now with version 3.2 or better

\begin{verbatim}
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",Th,u);
\end{verbatim}

\textsuperscript{1}http://www.gnuplot.info/
\textsuperscript{2}http://www-rocq.inria.fr/gamma/medit/medit.html
7.3. LINK WITH MEDIT

Before:

\begin{verbatim}
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");
\end{verbatim}
Chapter 8
Algorithms and Optimization

The complete example is in algo.edp file.

8.1 conjugate Gradient/GMRES

Suppose we want to solve the Euler problem (here $x$ has nothing to do with the reserved variable for the first coordinate in FreeFem++): find $x \in \mathbb{R}^n$ such that

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

(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 parameters are the name of that function with prototype

```cpp
func real[int] dJ(real[int] & xx);
```

which compute $\nabla J$, and a vector $x$ of type (of course the number 20 can be changed)

```cpp
real[int] x(20);
```

to initialize the process and get the result.

Given an initial value $x^{(0)}$, a maximum number $i_{\text{max}}$ of iterations, and an error tolerance $0 < \epsilon < 1$: Put $x = x^{(0)}$ and write

```cpp
NLCG(\nabla J, x, precon=M, nbiter=i_{\text{max}}, eps=\epsilon, stop=stopfunc);
```

will give the solution of $x$ of $\nabla J(x) = 0$. We can omit parameters precon, nbiter, eps, stop. 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 $\epsilon$, i.e.,

```cpp
NLCG(\nabla J, x, precon=M, nbiter=i_{\text{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 \(\epsilon\) (= 10\(^{-6}\) by default) if positive then relative test \(\|\nabla J(x)\|_P \leq \epsilon\|\nabla J(x_0)\|_P\), otherwise the absolute test is \(\|\nabla J(x)\|_P^2 \leq |\epsilon|\).

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

**stop** `stopfunc` add your test function to stop before (after version 3.31). The prototype for the function `stopfunc` is

```c
func bool stopfunc(int iter, real[int] u, real[int] g)
```

where \(u\) is the current solution, and \(g\) the current gradient not preconditioned.

**Example 8.1 (algo.edp)** For a given function \(b\), let us find the minimizer \(u\) of the functional

\[
J(u) = \frac{1}{2} \int_{\Omega} f(|\nabla u|^2) - \int_{\Omega} b u
\]

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

under the boundary condition \(u = 0\) on \(\partial \Omega\).

```c
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, uh = 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

$$\forall v_h \in V_0h, \quad \int_{\Omega} \alpha \nabla u_h \cdot \nabla v_h = \int_{\Omega} bv_h$$

where $\alpha = f'(|\nabla u|^2)$.

```c
varf alap(uh,vh) = int2d(Th)( alpha * ( dx(uh)*dx(vh) + dy(uh)*dy(vh) ) )
+ on(1,2,3,4,uh=0);
varf amass(uh) = int2d(Th)( uh*vh) + on(1,2,3,4,uh=0);
matrix amass = alap(Vh,Vh,solver=CG); //
matrix Alap= alap(Vh,Vh,solver=Cholesky,factorize=1);

// the preconditionner function
func real[int] C(real[int] & u)
{
    real[int] w = u;
    u = Alap^-1*w;
    return u; // no return of local array variable
}
/* To solve the problem, we make 10 iteration of the conjugate gradient, recompute the precondi-
tioner and restart the conjugate gradient: */

verbosity=5;
int conv=0;
real eps=1e-6;
for(int i=0;i<20;i++)
{
    conv=NLCG(dJ,u[],nbiter=10,precon=C,veps=eps);
    if (conv) break; // if converge break loop
    alpha=df( dx(u)*dx(u) + dy(u)*dy(u) ); // recompute alpha optimization
    Alap = alap(Vh,Vh,solver=Cholesky,factorize=1);
    cout << " restart with new preconditionner " << conv
    << " eps =" << eps << endl;
}
plot (u,wait=1,cmm="solution with NLCG");
```

For a given symmetric positive matrix $A$, consider the quadratic form

$$J(x) = \frac{1}{2} x^T Ax - b^T x$$

then $J(x)$ is minimized by the solution $x$ of $Ax = b$. In this case, we can use the function LinearCG

```
AffineCG(A, x, precon= M, nbiter= i_max, eps= ±\epsilon, stop= stp) ;
```

If $A$ is not symmetric, we can use GMRES(Generalized Minimum Residual) algorithm by

```
AffineGMRES(A, x, precon= M, nbiter= i_max, eps= ±\epsilon);
```

Also, we can use the non-linear version of GMRES algorithm (the functional $J$ is just convex)

```
AffineGMRES(\nabla J, x, precon= M, nbiter= i_max, eps= ±\epsilon);
```

For detail of these algorithms, refer to [19][Chapter IV, 1.3].
8.2 Algorithms for Unconstrained Optimization

Two algorithms of COOOL a package [32] are interfaced with the Newton Raphson method (call Newtonmethod) and the BFGS method. These two ones are directly available in FreeFem (no dynamical link to load). Be careful with these algorithms, because their implementation uses full matrices. We also provide several optimization algorithms from the NLopt library [47] as well as an interface for Hansen’s implementation of CMAES (a MPI version of this one is also available). These last algorithms can be found as dynamical links in the example++-load folder as the ff-NLopt and CMA_ES files (CMA_ES_MPI from the example++-mpi folder for the mpi version).

8.2.1 Example of utilization for BFGS or CMAES

```plaintext
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;
```

Using the CMA evolution strategy is almost the same, except that, as it is a derivative free optimizer, the DJ argument is omitted and there are some other named parameters to control the behaviour of the algorithm. With the same objective function as above, an example of utilization would be (see cmaes-VarIneq.edp for a complete example):

```plaintext
load "ff-cmaes"
... // define J, u and all here
real min = cmaes(J, u, stopTolFun=1e-6, stopMaxIter=3000);
cout << "minimal value is " << min << " for u = " << u << endl;
```

This algorithm works with a normal multivariate distribution in the parameters space and try to adapt its covariance matrix using the information provides by the successive function evaluations (see [48] for more details). Thus, some specific parameters can be passed to control the starting distribution, size of the sample generations etc... Named parameters for this are the following:

- **seed**: Seed for random number generator (val is an integer). No specified value will lead to a clock based seed initialization.
initialStdDev= Value for the standard deviations of the initial covariance matrix (val is a real). If the value $\sigma$ is passed, the initial covariance matrix will be set to $\sigma I$. The expected initial distance between initial $X$ and the argmin should be roughly initialStdDev. Default is 0.3.

initialStdDevs= Same as above except that the argument is an array allowing to set a value of the initial standard deviation for each parameter. Entries differing by several orders of magnitude should be avoided (if it can’t be, try rescaling the problem).

stopTolFun= Stops the algorithm if function values differences are smaller than the passed one, default is $10^{-12}$.

stopTolFunHist= Stops the algorithm if function value differences of the best values are smaller than the passed one, default is 0 (unused).

stopTolX= Stopping criteria triggered if step sizes in the parameters space are smaller than this real value, default is 0.

stopTolXFactor= Stopping criteria triggered when the standard deviation increases more than this value. The default value is $10^3$.

stopMaxFunEval= Stops the algorithm when stopMaxFunEval function evaluations have been done. Set to $900(n + 3)^2$ by default, where $n$ is the parameters space dimension.

stopMaxIter= Integer stopping the search when stopMaxIter generations has been sampled. Unused by default.

popsize= Integer value used to change the sample size. The default value is $4 + \lfloor 3 \ln(n) \rfloor$, see [48] for more details. Increasing the population size usually improves the global search capabilities at the cost of an at most linear reduction of the convergence speed with respect to popsize.

paramFile= This string type parameter allows the user to pass all the parameters using an external file as in Hansen’s original code. More parameters related to the CMA-ES algorithm can be changed with this file. A sample of it can be found in the examples++-load/ffCMAES/ folder under the name initials.par. Note that the parameters passed to the CMAES function in the FreeFem script will be ignored if an input parameters file is given.

8.3 IPOPT

The ff-Ipopt package is an interface for the IPOPT [49] optimizer. IPOPT is a software library for large scale, non-linear, constrained optimization. Detailed informations about it are in [49] and https://projects.coin-or.org/Ipopt. It implements a primal-dual interior point method along with filter method based line searches. IPOPT need a direct sparse symmetric linear solver. If your version of FreeFem has been compiled with the --enable-download tag, it will automatically be linked with a sequential version of MUMPS. An alternative to MUMPS would be to download the HSL subroutines (see http://www.coin-or.org/Ipopt/documentation/node16.html) and place them in the /ipopt/Ipopt-3.10.2/ThirdParty/HSL directory of the FreeFem++ downloads folder before compiling.
8.3.1 Short description of the algorithm

In this section, we give a very brief glimpse at the underlying mathematics of IPOPT. For a deeper introduction on interior methods for nonlinear optimization, one may consults [50], or [49] for more IPOPT specific elements. IPOPT is designed to perform optimization for both equality and inequality constrained problems. Though, nonlinear inequalities are rearranged before the beginning of the optimization process in order to restrict the panel of nonlinear constraints to those of the equality kind. Each nonlinear inequality ones are transformed into a pair of simple bound inequality and nonlinear equality constraint by the introduction of as many slack variables as is needed: $c_i(x) \leq 0$ becomes $c_i(x) + s_i = 0$ and $s_i \leq 0$, where $s_i$ is added to the initial variables of the problems $x_i$. Thus, for convenience, we will assume that the minimization problem does not contain any nonlinear inequality constraint. It means that, given a function $f : \mathbb{R}^n \mapsto \mathbb{R}$, we want to find:

$$x_0 = \arg\min_{x \in V} f(x)$$

with $V = \{ x \in \mathbb{R}^n \mid c(x) = 0 \text{ and } x_l \leq x \leq x_u \}$

Where $c : \mathbb{R}^n \to \mathbb{R}^m$ and $x_l, x_u \in \mathbb{R}^n$ and inequalities hold componentwise. The $f$ function as well as the constraints $c$ should be twice-continuously differentiable.

As a barrier method, interior points algorithms try to find a Karush-Kuhn-Tucker point for (8.2) by solving a sequence of problems, unconstrained with respect to the inequality constraints, of the form:

$$\text{for a given } \mu > 0, \text{ find } x_\mu = \arg\min_{x \in \mathbb{R}^n \mid c(x) = 0} B(x, \mu)$$

(8.3)

Where $\mu$ is a positive real number and $B(x, \mu) = f(x) - \mu \sum_{i=1}^{n} \ln(x_{u,i} - x_i) - \mu \sum_{i=1}^{m} \ln(x_i - x_{l,i})$.

The remaining equality constraints are handled with the usual Lagrange multipliers method. If the sequence of barrier parameters $\mu$ converge to 0, intuition suggests that the sequence of minimizers of (8.3) converge to a local constrained minimizer of (8.2). For a given $\mu$, (8.3) is solved by finding $(x_\mu, \lambda_\mu) \in \mathbb{R}^n \times \mathbb{R}^m$ such that:

$$\nabla B(x_\mu, \mu) + \sum_{i=1}^{m} \lambda_{\mu,i} \nabla c_i(x_\mu) = \nabla B(x_\mu, \mu) + J_c(x_\mu)^T \lambda_\mu = 0 \quad \text{and} \quad c(x_\mu) = 0$$

(8.4)

The derivations for $\nabla B$ only holds for the $x$ variables, so that:

$$\nabla B(x, \mu) = \nabla f(x) + \begin{pmatrix} \mu/(x_{u,1} - x_1) \\ \vdots \\ \mu/(x_{u,n} - x_n) \end{pmatrix} - \begin{pmatrix} \mu/(x_1 - x_{l,1}) \\ \vdots \\ \mu/(x_n - x_{l,n}) \end{pmatrix}$$

If we respectively call $z_u(x, \mu) = (\mu/(x_{u,1} - x_1), \ldots, \mu/(x_{u,n} - x_n))$ and $z_l(x, \mu)$ the other vector appearing in the above equation, then the optimum $(x_\mu, \lambda_\mu)$ satisfies:

$$\nabla f(x_\mu) + J_c(x_\mu)^T \lambda_\mu + z_u(x_\mu, \mu) - z_l(x_\mu, \mu) = 0 \quad \text{and} \quad c(x_\mu) = 0$$

(8.5)

In this equation, the $z_l$ and $z_u$ vectors seems to play the role of Lagrange multipliers for the simple bounds inequalities, and indeed, when $\mu \to 0$, they converge toward some suitable Lagrange multipliers for the KKT conditions, provided some technical assumptions are fulfilled (see [50]).

Equation (8.5) is solved by performing a Newton method in order to find a solution of (8.4) for each of the decreasing values of $\mu$. Some order 2 conditions are also taken into account to avoid convergence to non-maximizer, see [50] for precision about them. In the most classical IP algorithms, the Newton method is directly applied to (8.4). This is in most case inefficient due to frequent computation of infeasible points. These difficulties are avoided in Primal-Dual interior points methods...
where (8.4) is transformed into an extended system where \( z_u \) and \( z_l \) are treated as unknowns and the barrier problems are finding \( (x, \lambda, z_u, z_l) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}^n \) such that:

\[
\begin{align*}
\nabla f(x) + J_c(x)^T\lambda + z_u - z_l &= 0 \\
c(x) &= 0 \\
(X_u - X)z_u - \mu e &= 0 \\
(X - X_l)z_l - \mu e &= 0
\end{align*}
\]

(8.6)

Where if \( a \) is a vector of \( \mathbb{R}^n \), \( A \) denotes the diagonal matrix \( A = (a_i\delta_{ij})_{1 \leq i,j \leq n} \) and \( e \in \mathbb{R}^n = (1, 1, \ldots, 1) \). Solving this nonlinear system by the Newton methods is known as being the primal-dual interior points method. Here again, more details are available in [40]. Most actual implementations introduce features in order to globalize the convergence capability of the method, essentially by adding some line-search steps to the Newton algorithm, or by using trust regions. For the purpose of IPOPT, this is achieved by a filter line search methods, the details of which can be found in [?].

More IPOPT specific features or implementation details can be found in [49]. We will just retain that IPOPT is a smart Newton method for solving constrained optimization problem, with global convergence capabilities due to a robust line search method (in the sense that the algorithm will convergence no matter the initializer). Due to the underlying Newton method, the optimization process requires expressions of all derivatives up to the order 2 of the fitness function as well as those of the constraints. For problems whose hessian matrices are difficult to compute or lead to high dimensional dense matrices, it is possible to use a BFGS approximation of these objects at the cost of a much slower convergence rate.

### 8.3.2 IPOPT in FreeFem++

Calling the IPOPT optimizer in a FreeFem++ script is done with the IPOPT function included in the ff-Ipopt dynamic library. IPOPT is designed to solve constrained minimization problem in the form:

\[
\begin{align*}
\text{find} & \quad x_0 = \arg\min_{x \in \mathbb{R}^n} f(x) \\
\text{s.t.} & \quad \forall i \leq n, \ x_{lb}^i \leq x_i \leq x_{ub}^i \quad \text{(simple bounds)} \\
& \quad \forall i \leq m, \ c_i^l \leq c_i(x) \leq c_i^u \quad \text{(constraints functions)}
\end{align*}
\]

(8.7)

Where \( ub \) and \( lb \) stand for "upper bound" and "lower bound". If for some \( i, 1 \leq i \leq m \) we have \( c_i^l = c_i^u \), it means that \( c_i \) is an equality constraint, and an inequality one if \( c_i^l < c_i^u \).

There are different ways to pass the fitness function and constraints. The more general one is to define the functions using the keyword \texttt{func}. Any returned matrix must be a sparse one (type \texttt{matrix}, not a \texttt{real[int,int]}):

\[
\textbf{func} \ \texttt{real J(real[int] \&X) \{...\}} \quad \textit{// Fitness Function, returns a scalar} \\
\textbf{func} \ \texttt{real[int] gradJ(real[int] \&X) \{...\}} \quad \textit{// Gradient is a vector} \\
\textbf{func} \ \texttt{real[int] C(real[int] \&X) \{...\}} \quad \textit{// Constraints} \\
\textbf{func} \ \texttt{matrix jacC(real[int] \&X) \{...\}} \quad \textit{// Constraints jacobian}
\]

**Warning 1**: in the current version of FreeFem++, returning a matrix object local to a function block leads to undefined results. For each sparse matrix returning function you define, an extern \texttt{matrix} object has to be declared, whose associated function will overwrite and return on each call. Here is an example for \texttt{jacC}:

\[
\textbf{matrix} \ \texttt{jacCBuffer}; \quad \textit{// just declare, no need to define yet}
\]
Warning 2: IPOPT requires the structure of each matrix at the initialization of the algorithm. Some
errors may occur if the matrices are not constant and are built with the matrix \( A = [I,J,C] \)
syntax, or with an intermediary full matrix \( \text{real[int,int]} \), because any null coefficient is
discarded during the construction of the sparse matrix. It is also the case when making matrices
linear combinations, for which any zero coefficient will result in the suppression of the matrix from
the combination. Some controls are available to avoid such problems. Check the named parameters
descriptions (checkindex, structhess and structjac can help). We strongly advice to use
varf as much as possible for the matrix forging.

The hessian returning function is somewhat different because it has to be the hessian of the la-
grangian function : \( (x, \sigma, \lambda) \mapsto \sigma \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 c_i(x) \) where \( \lambda \in \mathbb{R}^m \) and \( \sigma \in \mathbb{R} \). Your hessian
function should then have the following prototype :

\[
\text{matrix} \ \text{hessianLBuffer}; \quad \text{func} \ \text{matrix} \ \text{hessianL(real[int]} \ \&X, \ \text{real} \ \sigma, \ \text{real[int]} \ \&\lambda) \ {\ldots}
\]

If the constraints functions are all affine, or if there are only simple bounds constraints or no
constraint at all, the lagrangian hessian is equal to the fitness function hessian, one can then omit
the sigma and lambda parameters :

\[
\text{matrix} \ \text{hessianJBuffer}; \quad \text{func} \ \text{matrix} \ \text{hessianJ(real[int]} \ \&X) \ {\ldots} \quad \text{// Hessian prototype when constraints are affine}
\]

When these functions are defined, IPOPT is called this way :

\[
\text{real[int]} \ Xi = \ldots \ ; \quad \text{// starting point} \\
\text{IPOPT(J,gradJ,hessianL,C,jacC,Xi, \text{/*some named parameters*/}) ;}
\]

If the hessian is omitted, the interface will tell IPOPT to use the (L)BFGS approximation (it can
also be enabled with a named parameter, see further). Simple bounds or unconstrained problems
do not require the constraints part, so the following expressions are valid :

\[
\text{IPOPT(J,gradJ,C,jacC,Xi, \ldots}) ; \quad \text{// IPOPT with BFGS} \\
\text{IPOPT(J,gradJ,hessianJ,Xi, \ldots}) ; \quad \text{// Newton IPOPT without constraints} \\
\text{IPOPT(J,gradJ,Xi, \ldots}) ; \quad \text{// BFGS, no constraints}
\]

Simple bounds are passed using the \( \text{lb} \) and \( \text{ub} \) named parameters, while constraints bounds are
passed with the \( \text{clb} \) and \( \text{cub} \) ones. Unboundedness in some directions can be achieved by using
the \( 1e^{19} \) and \( -1e^{19} \) values that IPOPT recognizes as \( +\infty \) and \( -\infty \) :

\[
\text{real[int]} \ xlb(n), xub(n), clb(m), cub(m); \quad \text{// fill the arrays...} \\
\text{IPOPT(J,gradJ,hessianL,C,jacC,Xi,lb=xlb,ub=xub,clb=clb,cub=cub, \text{/*some other named parameters*/}) ;}
\]
P2 fitness function and affine constraints function: In the case where the fitness function or constraints function can be expressed respectively in the following forms:

\[
\forall x \in \mathbb{R}^n, f(x) = \frac{1}{2} \langle Ax, x \rangle + \langle b, x \rangle \quad (A, b) \in \mathcal{M}_{n,n}(\mathbb{R}) \times \mathbb{R}^n
\]

or

\[
C(x) = Ax + b \quad (A, b) \in \mathcal{M}_{n,m}(\mathbb{R}) \times \mathbb{R}^m
\]

where \( A \) and \( b \) are constant, it is possible to directly pass the \((A, b)\) pair instead of defining 3 (or 2) functions. It also indicates to IPOPT that some objects are constant and that they have to be evaluated only once, thus avoiding multiple copies of the same matrix. The syntax is:

```cpp
// Affine constraints with "standard" fitness function
matrix A= ... ;
real[int] b = ... ;  // Linear part of the constraints
IPOPT(J,gradJ,hessianJ, [A,b] ,Xi, /*bounds and named params*/);
// [b,A] would work as well... Scatterbrains pampering...
```

Note that if you define the constraints in this way, they doesn’t contribute to the hessian, so the hessian should only take one \texttt{real[int]} as argument.

```cpp
// Affine constraints and P2 fitness func:
matrix A= ... ;  // Bilinear form matrix
real[int] b = ... ;  // Linear contribution to f
matrix Ac= ... ;  // Linear part of the constraints
real[int] bc= ... ;  // Constant part of constraints
IPOPT([A,b], [Ac,bc] ,Xi, /*bounds and named params*/);
```

If both objective and constraints functions are given this way, it automatically activates the IPOPT mehrotra-algorithm option (better for linear and quadratic programming according to the documentation). Otherwise, this option can only be set through the option file (see the named parameters section).

A spurious case is the one of defining \( f \) in this manner while using standard functions for the constraints:

```cpp
// Pure quadratic f - A is a matrix:
IPOPT(A, /*constraints args*/, Xi, /*bounds and named params*/);
```

Indeed, when passing \([A,b]\) in order to define \( f \), the lagrangian hessian is automatically build has the constant \( x \mapsto A \) function, with no way to add possible constraints contributions, leading to incorrect second order derivatives. So, a problem should be defined like that in only two cases:

1) constraints are nonlinear but you want to use the BFGS mode (then add \texttt{bfgs=1} to the named parameter),
2) constraints are affine, but in this case, why not passing them in the same way?

Here are some other valid definitions of the problem (cases when \( f \) is a pure quadratic or linear form, or \( C \) a pure linear function, etc...):

```cpp
// Pure linear f - b i a real[int] :
IPOPT(b, /*constraints args*/, Xi, /*bounds and named params*/);
```

```cpp
// linear constraints - Ac is a matrix
IPOPT(/*fitness func args*/, Ac, Xi, /*bounds and named params*/);
```
**Returned Value:** The IPOPT function returns an error code of type int. A zero value is obtained when the algorithm succeeds and positive values reflects the fact that IPOPT encounters minor troubles. Negative values reveals more problematic cases. The associated IPOPT return tags are listed in the table below. The [IPOPT pdf documentation](http://www.coin-or.org/Ipopt/documentation/node59.html) provides more accurate description of these return status:

<table>
<thead>
<tr>
<th>Success</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solve_Succeeded</td>
</tr>
<tr>
<td>1</td>
<td>Solved_To_Acceptable_Level</td>
</tr>
<tr>
<td>2</td>
<td>Infeasible_Problem_Detected</td>
</tr>
<tr>
<td>3</td>
<td>Search_Direction_Becomes_Too_Small</td>
</tr>
<tr>
<td>4</td>
<td>Diverging_Iterates</td>
</tr>
<tr>
<td>5</td>
<td>User_Requested_Stop</td>
</tr>
<tr>
<td>6</td>
<td>Feasible_Point_Found</td>
</tr>
<tr>
<td></td>
<td>-1 Maximum_Iterations_Exceeded</td>
</tr>
<tr>
<td></td>
<td>-2 Restoration_Failed</td>
</tr>
<tr>
<td></td>
<td>-3 Error_In_Step_Computation</td>
</tr>
<tr>
<td></td>
<td>-4 Maximum_CpuTime_Exceeded</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem definition issues</th>
<th>Critical errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10 Not_Enough_Degrees_Of_Freedom</td>
<td>-100 Unrecoverable_Exception</td>
</tr>
<tr>
<td>-11 Invalid_Problem_Definition</td>
<td>-101 NonIpopt_Exception_Thrown</td>
</tr>
<tr>
<td>-12 Invalid_Option</td>
<td>-102 Insufficient_Memory</td>
</tr>
<tr>
<td>-13 Invalid_Number_Detected</td>
<td>-199 Internal_Error</td>
</tr>
</tbody>
</table>

**Named Parameters:** The available named parameters in this interface are those we thought to be the most subject to variations from one optimization to another, plus a few ones that are interface specific. Though, as one could see at [http://www.coin-or.org/Ipopt/documentation/node59.html](http://www.coin-or.org/Ipopt/documentation/node59.html) there are many parameters that can be changed within IPOPT, affecting the algorithm behaviour. These parameters can still be controlled by placing an option file in the execution directory. Note that IPOPT’s [pdf documentation](http://www.coin-or.org/Ipopt/documentation/node59.html) may provides more informations than the previously mentioned online version for certain parameters. The in-script available parameters are:

- \( lb, \) \( ub \): real[int] for lower and upper simple bounds upon the search variables, must be of size \( n \) (search space dimension). If two components of same index in these arrays are equal then the corresponding search variable is fixed. By default IPOPT will remove any fixed variable from the optimization process and always use the fixed value when calling functions. It can be changed using the `fixedvar` parameter.

- \( clb, \) \( cub \): real[int] of size \( m \) (number of constraints) for lower and upper constraints bounds. Equality between two components of same index \( i \) in \( clb \) and \( cub \) reflect an equality constraint.

- \( structjacc \): To pass the greatest possible structure (indexes of non null coefficients) of the constraints jacobian under the form \([I, J]\) where \( I \) and \( J \) are two integer arrays. If not defined, the structure of the constraints jacobian, evaluated in \( X_i \), is used (no issue if the jacobian is constant or always defined with the same \( varf \), hazardous if it is with triplet array or if a full matrix is involved).

- \( structhess \): Same as above but for the hessian function (unused if \( f \) is P2 or less and constraints are affine). Here again, keep in mind that it is the hessian of the lagrangian function (which is equal to the hessian of \( f \) only if constraints are affine). If no structure is given with this parameter, the lagrangian hessian is evaluated on the starting point, with \( \sigma = 1 \) and \( \lambda = (1, 1, \ldots, 1) \) (it is safe if all the constraints and fitness function hessians are constant or build with \( varf \), and here again less reliable if built with triplet array or full matrix).
checkindex: A bool that triggers an index dichotomic search when matrices are copied from FreeFem functions to IPOPT arrays. It is used to avoid wrong index matching when some null coefficients are removed from the matrices by FreeFem. It will not solve the problems arising when a too small structure has been given at the initialization of the algorithm. Enabled by default (except in cases where all matrices are obviously constant).

warmstart: If set to true, the constraints dual variables $\lambda$, and simple bounds dual variables are initialized with the values of the arrays passed to lm, lz and uz named parameters (see below).

lm: real[int] of size $m$, which is used to get the final values of the constraints dual variables $\lambda$ and/or initialize them in case of a warm start (the passed array is also updated to the last dual variables values at the end of the algorithm).

lz, uz: real[int] of size $n$ to get the final values and/or initialize (in case of warm start) the dual variables associated to simple bounds.

tol: real, convergence tolerance for the algorithm, the default value is $10^{-8}$.

maxiter: int, maximum number of iterations with 3000 as default value.

maxcputime: real value, maximum runtime duration. Default is $10^6$ (almost 11 days and a half).

bfgs: bool enabling or not the (low-storage) BFGS approximation of the lagrangian hessian. It is set to false by default, unless there is no way to compute the hessian with the functions that have been passed to IPOPT.

derivativetest: Used to perform a comparison of the derivatives given to IPOPT with finite differences computation. The possible string values are: "none" (default), "first-order", "second-order" and "only-second-order". The associated derivative error tolerance can be changed via the option file. One should not care about any error given by it before having tried, and failed, to perform a first optimization.

dth: Perturbation parameter for the derivative test computations with finite differences. Set by default to $10^{-8}$.

dttol: Tolerance value for the derivative test error detection (default value unknown yet, maybe $10^{-5}$).

optfile: string parameter to specify the IPOPT option file name. IPOPT will look for a ipopt.opt file by default. Options set in the file will overwrite those defined in the FreeFem script.

printlevel: An int to control IPOPT output print level, set to 5 by default, the possible values are from 0 to 12. A description of the output informations is available in the pdf documentation of IPOPT.

fixedvar: string for the definition of simple bounds equality constraints treatment: use "make_parameter" (default value) to simply remove them from the optimization process (the functions will always be evaluated with the fixed value for those variables), "make_constraint" to treat them as any other constraint or "relax_bounds" to relax fixing bound constraints.

mustrategy: a string to choose the update strategy for the barrier parameter $\mu$. The two possible tags are "monotone", to use the monotone (Fiacco-McCormick) strategy, or "adaptive" (default setting).
\textbf{muinit}: real positive value for the barrier parameter initialization. It is only relevant when \texttt{mustrategy} has been set to \texttt{monotone}.

\textbf{pivtol}: real value to set the pivot tolerance for the linear solver. A smaller number pivots for sparsity, a larger number pivots for stability. The value has to be in the $[0, 1]$ interval and is set to $10^{-6}$ by default.

\textbf{brf}: Bounds relax factor: before starting the optimization, the bounds given by the user are relaxed. This option sets the factor for this relaxation. If it is set to zero, then the bounds relaxation is disabled. This real has to be positive and its default value is $10^{-8}$.

\textbf{objvalue}: An identifier to a \texttt{real} type variable to get the last value of the objective function (best value in case of success).

\textbf{mumin}: Minimal value for the barrier parameter $\mu$, a \texttt{real} with $10^{-11}$ as default value.

\textbf{linesearch}: A boolean which disables the line search when set to \texttt{false}. The line search is activated by default. When disabled, the method becomes a standard Newton algorithm other the primal-dual system. The global convergence is then no longer assured, meaning that many initializers could lead to diverging iterates. But on the other hand, it can be useful when trying to catch a precise local minimum without having some out of control process making the iterate caught by some other near optimum.

### 8.4 Some short examples using IPOPT

**Example 8.2 (IpoptVI.edp)** A very simple example consisting in, given two functions $f$ and $g$ (defined on $\Omega \subset \mathbb{R}^2$), minimizing $J(u) = \frac{1}{2} \int_{\Omega} |\nabla u|^2 - \int_{\Omega} fu$, with $u \leq g$ almost everywhere:

```plaintext
code
load "ff-Ipopt";
// load the interface
int nn=20;
// mesh quality
mesh Th=square(nn,nn);
// build a square mesh
fespace Vh(Th,P1);
// finite element space
func f = 1.;
// rhs function
real r=0.03,s=0.1;
// some parameters for g
func g = r - r/2*exp(-0.5*(\texttt{square}(x-0.5)+\texttt{square}(y-0.5))/\texttt{square}(s));
// g is constant minus a gaussian

macro Grad(u) [dx(u),dy(u)] // the gradient operator
varf vP(u,v) = int2d(Th) (Grad(u)'*Grad(v)) - int2d(Th) (f*v);

Here we build the matrix and second member associated to the functional to minimize once and for all. The $[A, b]$ syntax for the fitness function is then used to pass it to IPOPT.

matrix A = vP(Vh,Vh,solver=CG);
real[int] b = vP(0,Vh);

We use simple bounds to impose the boundary condition $u = 0$ on $\partial \Omega$, as well as the $u \leq g$ condition.

Vh lb=-1.e19;
// lower-unbounded in the interior
Vh ub=g;
// upper-bounded by g in the interior
varf vGamma(u,v) = on(1,2,3,4,u=1);
```
8.4. SOME SHORT EXAMPLES USING IPOPT

```plaintext
real[int] onGamma=vGamma(0,Vh);
ub[] = onGamma ? 0. : ub[];  // enforcing the boundary condition
lb[] = onGamma ? 0. : lb[];

Vh u=0;    // starting point
IPOPT([[A,b],u[]],lb=lb[],ub=ub[]);  // solve the problem
plot(u,wait=1);
```

Example 8.3 (IpoptVI2.edp) Let $\Omega$ be a domain of $\mathbb{R}^2$, $f_1, f_2 \in L^2(\Omega)$ and $g_1, g_2 \in L^2(\partial \Omega)$ four given functions with $g_1 \leq g_2$ almost everywhere. We define the space:

$$V = \{(v_1, v_2) \in H^1(\Omega)^2; v_1|_{\partial \Omega} = g_1, v_2|_{\partial \Omega} = g_2, v_1 \leq v_2 \ a.e. \}$$

as well as the functional $J: H^1(\Omega)^2 \rightarrow \mathbb{R}$:

$$J(v_1, v_2) = \frac{1}{2} \int_{\Omega} |\nabla v_1|^2 - \int_{\Omega} f_1 v_1 + \frac{1}{2} \int_{\Omega} |\nabla v_2|^2 - \int_{\Omega} f_2 v_2$$

The problem consists in finding (numerically) two functions $(u_1, u_2) = \arg\min_{(v_1, v_2) \in V} J(v_1, v_2)$.

```plaintext
load "ff-IpOpt";

mesh Th=square(10,10);
fespace Vh(Th,[P1,P1] );
fespace Wh(Th,[P1] );
int iter=0;

func f1 = 10;  // right hand sides
func f2 = -15;
func g1 = -0.1;  // Boundary conditions functions
func g2 = 0.1;

while(++iter)  // mesh adaptation loop
{
    macro Grad(u) [dx(u), dy(u)]  // gradient macro
    varf vP([u1,u2],[v1,v2]) = int2d(Th)(Grad(u1)ʼ*Grad(v1) + Grad(u2)ʼ*Grad(v2))
                          - int2d(Th)(f1*v1+f2*v2);

    matrix A = vP(Vh,Vh);  // Fitness function matrix...
    real[int] b = vP(0,Vh);  // and linear form

    int[int] II1=[0], II2=[1];  // Constraints matrix
    matrix C1 = interpolate (Wh,Vh, U2Vc=II1);
    matrix C2 = interpolate (Wh,Vh, U2Vc=II2);
    matrix CC = -1*C1 + C2;  // u2 - u1 >0
    Wh cl=0;  // constraints lower bounds (no upper bounds)

    // Boundary conditions
    varf vGamma([u1,u2],[v1,v2]) = on(1,2,3,4,u1=1,u2=1);
    real[int] onGamma=vGamma(0,Vh);
    Vh [ub1,ub2]=[g1,g2];
    Vh [lb1,lb2]=[g1,g2];
    ub1[] = onGamma ? ub1[] : 1e19 ;  // Unbounded in interior
    lb1[] = onGamma ? lb1[] : -1e19 ;
```
Figure 8.1: Numerical Approximation of the Variational Inequality

\[ V_h [u_1, u_2] = [0, 0]; \] // starting point

\[
\text{IPOPT}([b, A], C, u_1[], lb=lb1[], ub=ub1[], clb=cl[]);
\]

\[
\text{plot}(u_1, u_2, \text{wait}=1, \text{nbiso}=60, \text{dim}=3);
\]

\[
\text{if}(\text{iter} > 1) \text{ break;}
\]

\[
\text{Th = adaptmesh(Th, [u_1, u_2], err=0.004, nbvx=100000);} \]

\[ \text{8.5 3D constrained minimal surface with IPOPT} \]

\[ \text{8.5.1 Area and volume expressions} \]

This example aimed at numerically solving some constrained minimal surface problems with the IPOPT algorithm. We restrain to \( C^k \) \((k \geq 1)\), closed, spherically parametrizable surfaces, i.e. surfaces \( S \) such that:

\[ \exists \rho \in C^k([0, 2\pi] \times [0, \pi]) \quad S = \left\{ X = \begin{pmatrix} \rho(\theta, \phi) \\ 0 \\ 0 \end{pmatrix}, (\theta, \phi) \in [0, 2\pi] \times [0, \pi] \right\} \]

Where the components are expressed in the spherical coordinate system. Let’s call \( \Omega \) the \([0, 2\pi] \times [0, \pi]\) angular parameters set. In order to exclude self crossing and opened shapes, the following assumptions upon \( \rho \) are made:

\[ \rho \geq 0 \quad \text{and} \quad \forall \phi, \rho(0, \phi) = \rho(2\pi, \phi) \]

For a given function \( \rho \) the first fundamental form (the metric) of the defined surface has the following matrix representation:

\[
G = \begin{pmatrix}
\rho^2 \sin^2(\phi) + (\partial_\theta \rho)^2 & \partial_\theta \rho \partial_\phi \rho \\
\partial_\theta \rho \partial_\phi \rho & \rho^2 + (\partial_\phi \rho)^2
\end{pmatrix}
\] \hspace{1cm} (8.8)

This metric is used to express the area of the surface. Let \( g = \det(G) \), then we have:

\[
\mathcal{A}(\rho) = \int_{\Omega} ||\partial_\theta X \wedge \partial_\phi X|| = \int_{\Omega} \sqrt{g} = \int_{\Omega} \sqrt{\rho^2(\partial_\theta \rho)^2 + \rho^4 \sin^2(\phi) + \rho^2 (\partial_\phi \rho)^2 \sin^2(\phi)} d\theta d\phi
\] \hspace{1cm} (8.9)
The volume of the space enclosed within the shape is easier to express:

\[ V(\rho) = \int_{\Omega} \int_{0}^{\rho(\theta, \phi)} r^2 \sin(\phi) d\theta d\phi = \frac{1}{3} \int_{\Omega} \rho^3 \sin(\phi) d\theta d\phi \]  
(8.10)

### 8.5.2 Derivatives

In order to use a newton based interior point optimization algorithm, one must be able to evaluate the derivatives of \( A \) and \( V \) with respect to \( \rho \). Concerning the area we have the following result:

\[ \forall v \in C^1(\Omega), \langle dA(\rho), v \rangle = \int_{\Omega} \left( \frac{1}{2} \frac{d\bar{g}(\rho)(v)}{\sqrt{\bar{g}}} \right) d\theta d\phi \]

Where \( \bar{g} \) is the application mapping the \((\theta, \phi) \mapsto g(\theta, \phi)\) scalar field to \( \rho \). This leads to the following expression, easy to transpose in a freefem script using:

\[ \forall v \in C^1(\Omega), \langle dA(\rho), v \rangle = \int_{\Omega} \left( 2\rho^3 \sin^2(\phi) + \rho(\partial_{\theta}\rho)^2 + \rho(\partial_{\phi}\rho)^2 \sin^2(\phi) \right) v + \int_{\Omega} \rho^2 \partial_{\theta} \rho \partial_{\theta} v + \rho^2 \partial_{\phi} \rho \sin^2(\phi) \partial_{\phi} v \]  
(8.11)

With a similar approach, one can derive an expression for second order derivatives. Though comporting no specific difficulties, the detailed calculus are tedious, the result is that these derivatives can be written using a \( 3 \times 3 \) matrix \( B \) whose coefficients are expressed in term of \( \rho \) and its derivatives with respect to \( \theta \) and \( \phi \), such that:

\[ \forall (w, v) \in C^1(\Omega), d^2A(\rho)(w, v) = \int_{\Omega} \left( w \partial_{\theta} w \partial_{\phi} w \right) B \begin{pmatrix} v \\ \partial_{\theta} v \\ \partial_{\phi} v \end{pmatrix} d\theta d\phi \]  
(8.12)

Deriving the volume function derivatives is again an easier task. We immediately get the following expressions:

\[ \forall v, \langle dV(\rho), v \rangle = \int_{\Omega} \rho^2 \sin(\phi) v d\theta d\phi \]

\[ \forall w, v, d^2V(\rho)(w, v) = \int_{\Omega} 2\rho \sin(\phi) w v d\theta d\phi \]  
(8.13)

### 8.5.3 The problem and its script:

The whole code is available in \texttt{IpoptMinSurfVol.edp}. We propose to solve the following problem:

**Example 8.4** Given a positive function \( \rho_{\text{object}} \) piecewise continuous, and a scalar \( V_{\text{max}} > V(\rho_{\text{object}}) \), find \( \rho_0 \) such that:

\[ \rho_0 = \arg\min_{\rho \in C^1(\Omega)} A(\rho), \text{ s.t. } \rho_0 \geq \rho_{\text{object}} \text{ and } V(\rho_0) \leq V_{\text{max}} \]

If \( \rho_{\text{object}} \) is the spherical parametrization of the surface of a 3-dimensional object (domain) \( O \), it can be interpreted as finding the surface with minimal area enclosing the object with a given maximal volume. If \( V_{\text{max}} \) is close to \( V(\rho_{\text{object}}) \), so should be \( \rho_0 \) and \( \rho_{\text{object}} \). With higher values of \( V_{\text{max}} \), \( \rho \) should be closer to the unconstrained minimal surface surrounding \( O \) which is obtained as soon as
\[ V_{\text{max}} \geq \frac{4}{3} \pi \| \rho_{\text{object}} \|^3_{\infty} \text{ (sufficient but not necessary).} \]

It also could be interesting to solve the same problem with the constraint \( V(\rho_0) \geq V_{\text{min}} \) which lead to a sphere when \( V_{\text{min}} \geq \frac{1}{6} \pi \text{diam}(O)^3 \) and move toward the solution of the unconstrained problem as \( V_{\text{min}} \) decreases.

We start by meshing the domain \([0, 2\pi] \times [0, \pi]\), then a periodic P1 finite elements space is defined.

```plaintext
load "msh3";
load "medit";
load "ff-Ipopt";

int np=40; // initial mesh quality parameter
mesh Th = square(2*np,np,[2*pi*x,pi*y]);

fespace Vh(Th,P1,periodic=[[2,y],[4,y]]);
Vh startshape=5; // initial shape

We create some finite element functions whose underlying arrays will be used to store the values of dual variables associated to all the constraints in order to reinitialize the algorithm with it in the case where we use mesh adaptation. Doing so, the algorithm will almost restart at the accuracy level it reached before mesh adaptation, thus saving many iterations.

Vh uz=1.,lz=1.; // Simple bounds dual variable
real[int] lm=[1]; // dual variable for volume constraint

Then, follows the mesh adaptation loop, and a rendering function, Plot3D, using 3D mesh to display the shape it is passed with medit (the movemesh23 procedure often crashes when called with ragged shapes).

```plaintext
int nadapt=1;
for(int kkk=0;kkk<nadapt;++kkk) // Mesh adaptation loop
{
  int iter=0; // iterations count
  func sin2 = square(sin(y)); // a function that will be often used
  func int Plot3D(real[int] &rho,string cmm,bool ffplot) {...} // see the .edp file

Here are the functions related to the area computation and its shape derivative, according to equations 8.9 and 8.11:

```plaintext
func real Area(real[int] &X)
{
  Vh rho;
  rho[] = X;
  Vh rho2 = square(rho);
  Vh rho4 = square(rho2);
  real res = int2d(Th)( sqrt( rho4*sin2 + rho2*square(dx(rho)) + rho2*sin2*square(dy(rho))) )
}
```
plot(rho, ... /*some parameters*/ ... );
return res;
}

func real[int] GradArea(real[int] &X)  // The gradient
{
Vh rho,rho2;
 rho[] = X;
rho2[] = square(X);
Vh sqrtPsi, alpha;
 // Psi is actually det(G)
{
 Vh ddxrho2 = dx(rho),ddyrho2 = dy(rho);
 sqrtPsi = sqrt( rho2*rho2*sin2 + rho2*ddxrho2 + rho2*ddyrho2*sin2 );
 alpha = 2.*rho2*rho*sin2 + rho*ddxrho2 + rho*ddyrho2*sin2;
}
varf dSurface(u,v) =
int2d(Th)(1./sqrtPsi*(alpha*v+rho2*ddx(u)*dx(v)+rho2*ddy(u)*sin2*dy(v)));
real[int] grad = dSurface(0,Vh);
return grad;
}
The function returning the hessian of the area for a given shape is a bit blurry, thus we won’t show here all of equation [5.12] coefficients definition, they can be found in the edp file.

matrix hessianA;  // The global matrix buffer

func matrix HessianArea(real[int] &X)
{
Vh rho,rho2;
 rho[] = X;
rho2 = square(rho);
Vh sqrtPsi,sqrtPsi3,C00,C01,C02,C11,C12,C22,A;
{
 ...  // definition of the above functions
}
varf d2Area(w,v) =
int2d(Th)(
  1./sqrtPsi * (A*w*v + 2*rho*ddx(u)*dx(v)+rho2*ddy(u)*sin2*dy(v))
  + 2*rho*ddy(u)*sin2*dwdx(v) + 2*rho*ddy(u)*sin2*w*dvy(v)
  + rho2*ddx(u)*dx(v) + rho2*sin2*dy(u)*sin2*dy(v))
  + 1./sqrtPsi3 * (C00*w*v + C01*ddx(u)*dx(v) + C01*w*dwdx(v)
  + C02*w*dy(v) + C02*w*dy(v) + C11*ddx(u)*dx(v)
  + C12*ddx(u)*dy(v) + C12*ddx(u)*dy(v) + C22*ddx(u)*dy(v))
);
// end of int2d and varf
hessianA = d2Area(Vh,Vh);
return hessianA;
}

And the volume related functions :

func real Volume(real[int] &X)
{
Vh rho;
rho[]=X;
Vh rho3=rho*rho*rho;
real res = 1./3.*int2d(Th)(rho3*sin(y));
```c
    return res;
}

func real[int] GradVolume(real[int] &X)
{
    Vh rho;
    rho[]=X;
    varf dVolume(u,v) = int2d(Th)(rho*rho*sin(y)*v);
    real[int] grad = dVolume(0,Vh);
    return grad;
}

matrix hessianV;       // buffer
func matrix HessianVolume(real[int] &X)
{
    Vh rho;
    rho[]=X;
    varf d2Volume(w,v) = int2d(Th)(2*rho*sin(y)*v*w);
    hessianV = d2Volume(Vh,Vh);
    return hessianV;
}

If we want to use the volume as a constraint function we must wrap it and its derivatives in some
FreeFem++ functions returning the appropriate types. It is not done in the above functions in
case where one wants to use it as fitness function. The lagrangian hessian also have to be wrapped
since the Volume is not linear with respect to $\rho$, it has some non-null second order derivatives.

func real[int] ipVolume(real[int] &X) {
    real[int] vol = [Volume(X)];
    return vol;
}

matrix mdV;           // buffer
func matrix ipGradVolume(real[int] &X)
{
    // transforms a vector into a sparse matrix
    real[int,int] dvol(1,Vh.ndof);
    dvol(0,:) = GradVolume(X);
    mdV = dvol;
    return mdV;
}

matrix HLagrangian;    // buffer
func matrix ipHessianLag(real[int] &X, real objfact, real[int] &lambda)
{
    HLagrangian = objfact*HessianArea(X) + lambda[0]*HessianVolume(X);
    return HLagrangian;
}

The ipGradVolume function could bring some troubles during the optimization process because
the gradient vector is transformed in a sparse matrix, so any null coefficient will be discarded. We
are here obliged to give IPOPT the structure by hand and use the checkindex named-parameter
to avoid bad indexing during copies. This gradient is actually dense, there is no reason for some
components to be constantly zero :

    // sparse structure of a dense vector
int[int] gvi(Vh.ndof),gvj=0:Vh.ndof-1;
gvi=0;
    // only one line
These two arrays will be passed to IPOPT with `structjacc=[gvi,gvj]`. The last remaining things are the bounds definition. The simple lower bounds must be equal to the components of the P1 projection of $\rho_{\text{object}}$. And we choose $\alpha \in [0,1]$ to set $V_{\text{max}}$ to $(1-\alpha)V(\rho_{\text{object}}) + \alpha \frac{4}{3}\pi \|\rho_{\text{object}}\|_\infty$

```plaintext
real e=0.1, r0=0.25, rr=2-r0;
real E=1./(e*e), RR=1./(rr*rr);
```

// An indented disc
```plaintext
func disc1 = sqrt(1./(RR+(E-RR)*cos(y)*cos(y)))*(1+0.1*cos(9*x));
```

// Almost a standard disc
```plaintext
func disc2 = sqrt(1./(RR+(E-RR)*cos(x)*cos(x)*sin2));
```

$V_h$ lb = max(disc1, disc2);

// glue the object parts
```plaintext
real Vobj = Volume(lb[]);
```

// object volume
```plaintext
real Vnvc = 4./3.*pi_pow(lb[].linfty,3);
```

Calling IPOPT:

```plaintext
IPOPT(Area,GradArea,ipHessianLag,
ipVolume,ipGradVolume,rc[]),
ub=ub[], lb=lb[], clb=clb, cub=cub, // simple bounds and volume bounds
checkindex=1, structjacc=[gvi,gvj], // for safe matrices copies
maxiter=kkk<nadapt-1 ? 40:150, // accurate optim only for last mesh
adaptation iteration
warmstart=kkk, lm=lm, uz=uz[], lz=lz[], // warmstart handling
tol=0.00001);
```

// displays current solution
```plaintext
Plot3D(rc[],"Shape_at_"+kkk,0);
```

At last, before closing the mesh adaptation loop, we have to perform the said adaptation. The mesh is adaptated with respect to the $X = (\rho,0,0)$ (in spherical coordinates) vector field, not directly with respect to $\rho$, otherwise the true curvature of the 3D-shape would not be well taken into account.

```plaintext
if(kkk<nadapt-1)
{
    Th = adaptmesh(Th,
    rc*cos(x)*sin(y), // X
    rc*sin(x)*sin(y), // Y
    rc*cos(y), // Z
    nbvx=50000,
    periodic=[[2,y],[4,y]]);
    // keeps mesh peridicity
plot(Th);
startshape = rc; // shape interpolation on the new mesh
uz=uz;
// dual variables interpolation
lz=lz;
}
```

// end if
```

Here are some pictures of the resulting surfaces obtained for decreasing values of $\alpha$ (and a slightly more complicated object than two orthogonal discs). We get back the enclosed object when $\alpha = 0$
8.6 The nlOpt optimizers

The \texttt{ff-NLopt} package provides a FreeFem interface to the free/open-source library for nonlinear optimization, thus easing the use of several different free optimization (constrained or not) routines available online along with the PDE solver. All the algorithms are well documented in \cite{nlOpt}, thus no exhaustive informations concerning their mathematical specificities will be found here and we will focus on the way they are called in a FreeFem script. One needing detailed informations about these algorithms should visit the said cite where a description of each of them is given, as well as many bibliographical links. Most of the gradient based algorithm of nlOpt uses a full matrix approximation of the hessian, so if you’re planing to solve a large scale problem, our advise would be to use the IPOPT optimizer which definitely surpass them. Finally, an examples of use can be found in the \texttt{examples++-load/} directory under the name \texttt{VarIneq2.edp}

All the nlOpt features are called that way:

\begin{verbatim}
load "ff-NLopt"

... // define J, u, and maybe grad(J), some constraints etc...
real min = nloptXXXXXX(J,u, // unavoidable part
grad = <name of grad(J)>, // if needed
lb = // lower bounds array
ub= // upper bounds array
...
// some optional arguments :
// constraints functions names,
\end{verbatim}
8.6. THE NLOPT OPTIMIZERS

// stopping criterions,
// algo. specific parameters,
// etc...

);  

XXXXXX refers to the algorithm tag (not necessarily 6 characters long). u is the starting position (a real[int] type array) which will be overwritten by the algorithm, the value at the end being the found argmin. And as usual, J is a function taking a real[int] type array as argument and returning a real. grad, lb and ub are "half-optionnal" arguments, in the sense that they are obligatory for some routines but not all.

The possible optional named parameters are the following, note that they are not used by all algorithm s (some does not supports constraints, or a type of constraints, some are gradient-based and other are derivative free, etc...). One can refer to the table after the parameters description to check which are the named parameters supported by a specific algorithm. Using an unsupported parameter will not stop the compiler work and seldom breaks runtime, it will just be ignored. That said, when it is obvious you are missusing a routine, you will get a warning message at runtime (for example if you pass a gradient to a derivative free algorithm, or set the population of a non-genetic one, etc...). In the following description, n stands for the dimension of the search space.

Half-optional parameters :

grad= The name of the function which computes the gradient of the cost function (prototype should be real[int] → real[int], both argument and result should have the size n). This is needed as soon s a gradient-based method is involved, ignored if defined in a derivative free context.

lb/ub = Lower and upper bounds arrays (real[int] type) of size n. Used to define the bounds within which the search variable is allowed to move. Needed for some algorithms, optional or unsupported for others.

subOpt : Only enabled for the Augmented Lagrangian and MLSL method who need a sub-optimizer in order to work. Just pass the tag of the desired local algorithm with a string.

Constraints related parameters (optional - unused if not specified):

IConst/EConst : Allows to pass the name of a function implementing some inequality (resp. equality) constraints on the search space. The function type must be real[int] → real[int] where the size of the returned array is equal to the number of constraints (of the same type - it means that all the constraints are computed in one vectorial function). In order to mix inequality and equality constraints in a same minimization attempt, two vectorial functions have to be defined and passed. See example ?? for more details about how these constraints have to be implemented.

gradIConst/gradEConst : Use to provide the inequality (resp. equality) constraints gradient. These are real[int] → real[int,int] type functions. Assuming we have defined a constraint function (either inequality or equality) with p constraints, the size of the matrix returned by its associated gradient must be p × n (the i-th line of the matrix is the gradient of the i-th constraint). It is needed in a gradient-based context as soon as an inequality or equality constraint function is passed to the optimizer and ignored in all other cases.
tolIConst/tolEConst: Tolerance values for each constraint. This is an array of size equal to the number of inequality (resp. equality) constraints. Default value os set to $10^{-12}$ for each constraint of any type.

Stopping criteria:

stopFuncValue: Makes the algorithm end when the objective function reaches this real value.

stopRelXTol: Stops the algorithm when the relative moves in each direction of the search space is smaller than this real value.

stopAbsXTol: Stops the algorithm when the moves in each direction of the search space is smaller than the corresponding value in this real[int] array.

stopRelFTol: Stops the algorithm when the relative variation of the objective function is smaller than this real value.

stopAbsFTol: Stops the algorithm when the variation of the objective function is smaller than this real value.

stopMaxFEval: Stops the algorithm when the number of fitness evaluations reaches this integer value.

stopTime: Stops the algorithm when the optimization time in second exceeds this real value. This is not a strict maximum: the time may exceed it slightly, depending upon the algorithm and on how slow your function evaluation is.

Note that when an AUGLAG or MLSL method is used, the meta-algorithm and the sub-algorithm may have different termination criteria. Thus, for algorithms of this kind, the following named parameters has been defined (just adding the SO prefix - for Sub-Optimizer) to set the ending condition of the sub-algorithm (the meta one uses the ones above) : SOSTopFuncValue, SOSTopRelXTol, and so on... If these ones are not used, the sub-optimizer will use those of the master routine.

Other named parameters:

popSize: integer used to change the size of the sample for stochastic search methods. Default value is a peculiar heuristic to the chosen algorithm.

SOPopSize: Same as above, but when the stochastic search is passed to a meta-algorithm.

nGradStored: The number (integer type) of gradients to "remember" from previous optimization steps: increasing this increases the memory requirements but may speed convergence. It is set to a heuristic value by default. If used with AUGLAG or MLSL, it will only affect the given subsidiary algorithm.

The following table sums up the main characteristics of each algorithm, providing the more important information about which features are supported by which algorithm and what are the unavoidable arguments they need. More details can be found in [47].
## 8.6. THE NLOPT OPTIMIZERS

<table>
<thead>
<tr>
<th>Id Tag</th>
<th>Full Name</th>
<th>Bounds</th>
<th>Gradient-Based</th>
<th>Stochastic</th>
<th>Constraints</th>
<th>Sub-Opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIRECT</td>
<td>Dividing rectangles</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>DIRECTL</td>
<td>Locally biased dividing rectangles</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>DIRECTLRand</td>
<td>Randomized locally biased dividing rectangles</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>DIRECTNoScal</td>
<td>Dividing rectangles - no scaling</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>DIRECTRandNoScal</td>
<td>Randomized locally biased dividing rectangles - no scaling</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OrigDIRECT</td>
<td>Original Gablonsky’s dividing rectangles</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>OrigDIRECTL</td>
<td>Original Gablonsky’s locally biased dividing rectangles</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>StoGO</td>
<td>Stochastic(?) Global Optimization</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>StoGORand</td>
<td>Randomized Stochastic(?) Global Optimization</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>LBFGS</td>
<td>Low-storage BFGS</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>PRAXIS</td>
<td>Principal AXIS</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Var1</td>
<td>Rank-1 shifted limited-memory variable-metric</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Var2</td>
<td>Rank-2 shifted limited-memory variable-metric</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>TNewton</td>
<td>Truncated Newton</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>TNewtonRestart</td>
<td>Steepest descent restarting truncated Newton</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>TNewtonPrecond</td>
<td>BFGS preconditioned truncated Newton</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>TNewtonRestartPrecond</td>
<td>BFGS preconditioned truncated Newton with steepest descent restarting</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRS2</td>
<td>Controlled random search with local mutation</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>MMA</td>
<td>Method of moving asymptotes</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>COBYLA</td>
<td>Constrained optimization by linear approximations</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>NEWUOA</td>
<td>NEWUOA</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>NEWUOAbound</td>
<td>NEWUOA for bounded optimization</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>NelderMead</td>
<td>Nelder-Mead simplex</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Subplex</td>
<td>Subplex</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>BOBYQA</td>
<td>BOBYQA</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>ISRES</td>
<td>Improved stochastic ranking evolution strategy</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>SLSQP</td>
<td>Sequential least-square quadratic programming</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>MLSL</td>
<td>Multi-level single-linkage</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>MLSLLDS</td>
<td>Low discrepancy multi-level single-linkage</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>AUGLAG</td>
<td>Constraints augmented lagrangian</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>AUGLAGEQ</td>
<td>Equality constraints augmented lagrangian</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

**Legend:**

- ✔: Supported and optional
- ✔️: Should be supported and optional, may lead to weird behaviour though.
- ✗: Intrinsic characteristic of the algorithm which then need one or more unavoidable parameter to work (for stochastic algorithm, the population size always have a default value, they will then work if it is ommitted)
- •/✔️: For routines with subsidiary algorithms only, indicates that the corresponding feature will depend on the chosen sub-optimizer.
8.7 Optimization with MPI

The only quick way to use the previously presented algorithms on a parallel architecture lies in parallelizing the used cost function (which is in most real life case, the expensive part of the algorithm). Somehow, we provide a parallel version of the CMA-ES algorithm. The parallelization principle is the trivial one of evolving/genetic algorithms: at each iteration the cost function has to be evaluated \( N \) times without any dependence at all, these \( N \) calculus are then equally distributed to each processes. Calling the MPI version of CMA-ES is nearly the same as calling its sequential version (a complete example of use can be found in the cmaes-mpi-VarIneq.edp file):

```cpp
load "mpi-cmaes"
...
  // define J, u and all here
  real min = cmaesMPI(J,u,stopTolFun=1e-6,stopMaxIter=3000);
  cout << "minimal value is " << min << " for u = " << u << endl;
```

If the population size is not changed using the popsize parameter, it will use the heuristic value slightly changed to be equal to the closest greater multiple of the size of the communicator used by the optimizer. The FreeFem mpicommworld is used by default. The user can specify his own MPI communicator with the named parameter " comm=", see the MPI section of this manual for more informations about communicators in FreeFem++.
Chapter 9
Mathematical Models

Summary  This chapter goes deeper 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 problems 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 by 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 the “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: \((x, y, u(x, y + \delta y))\). Denote by \(n(x, y) = (n_x(x, y), n_y(x, y), n_z(x, y))\) 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 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

\[
-\mu n_y(x, y) \delta x, \quad \mu \left(n_y(x, y) + \frac{\partial n_y}{\partial y}\right) (x, y) \delta x.
\]
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
\[
\nu_x = \left( \frac{\partial u/\partial x}{\sqrt{1 + (\partial u/\partial x)^2 + (\partial u/\partial y)^2}} \right) \simeq \partial u/\partial x,
\]
\[
\nu_y = \left( \frac{\partial u/\partial y}{\sqrt{1 + (\partial u/\partial x)^2 + (\partial u/\partial y)^2}} \right) \simeq \partial u/\partial y.
\]

Letting \( \delta x \rightarrow dx \), \( \delta y \rightarrow dy \), we have the equilibrium of the vertical displacement of soap film on ABCD by \( p \)
\[
\mu dxdy \frac{\partial^2 u}{\partial x^2} + \mu dxdy \frac{\partial^2 u}{\partial y^2} + pdxdy = 0.
\]

Using the Laplace operator \( \Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \), we can find the virtual displacement write the following
\[
- \Delta u = f \quad \text{in } \Omega \quad \text{(9.1)}
\]
where \( f = p/\mu \), \( \Omega = \{(x,y); x^2 + y^2 < 1\} \). Poisson’s equation (2.1) appears 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
\[
u = 0 \quad \text{on } \partial \Omega \quad \text{(9.2)}
\]
If the force is gravity, for simplify, we assume that \( f = -1 \).

**Example 9.1 (a_tutorial.edp)**

```plaintext
1 : border a(t=0,2*pi){ x = cos(t); y = sin(t); label=1;};
2 : mesh disk = buildmesh(a(50));
3 : plot(disk);
4 : fespace fempl(disk,P1);
5 : fempl u,v;
6 : func f = -1;
7 : problem laplace(u,v) =
8 :     int2d(disk)(dx(u)*dx(v) + dy(u)*dy(v)) // bilinear form
9 :     - int2d(disk)( f*v ) // linear form
10 :     + on(1,u=0) ; // boundary condition
11 : func ue = (xˆ2+yˆ2-1)/4; // ue: exact solution
12 : laplace;
13 : fempl err = u - ue;
14 : plot(u,ps="aTutorial.eps",value=true,wait=true);
15 : plot(err,value=true,wait=true);
16 : cout << "error L2=" << sqrt(int2d(disk)( errˆ2 ))<< endl;
17 : cout << "error H10=" << sqrt(int2d(disk)((dx(u)-x/2)ˆ2))<< endl;
18 : disk = adaptmesh(disk,u,err=0.01);
19 : plot(disk,wait=1);
20 : laplace;
21 : plot(u,value=true,wait=true);
22 : err = u - ue; // become FE-function on adapted mesh
```
9.1. STATIC PROBLEMS

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 $E$ satisfies

$$\text{div}E = \rho/\epsilon, \quad \text{curl}E = 0 \quad (9.3)$$

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 $E = -\nabla \phi$. Then we have Poisson’s 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 $\{C_1,\ldots,K\}$. Let us assume $K$ conductors $C_1,\cdots,C_K$ within an enclosure $C_0$. Each one is held at an electrostatic potential $\phi_i$. We assume that the enclosure $C0$ is held at potential 0. In order to know $\phi(x)$ 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 \( \sum_{i=0}^{N} C_i \). 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:

\[
\varphi = \varphi_i \text{ on } C_i, \quad i = 1...N, \varphi = 0 \text{ on } C_0.
\]

(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.

```plaintext
// a circle with center at (0,0) and radius 5
border C0(t=0,2*pi) { x = 5 * cos(t); y = 5 * sin(t); }
border C1(t=0,2*pi) { x = 2+0.3 * cos(t); y = 3*sin(t); }
border C2(t=0,2*pi) { x = -2+0.3 * cos(t); y = 3*sin(t); }
mesh Th = buildmesh(C0(60)+C1(-50)+C2(-50));
plot(Th,ps="electroMesh"); // figure 9.3
fespace Vh(Th,P1); // P1 FE-space
Vh uh,vh; // unknown and test function.
problem Electro(uh,vh) = // definition of the problem
  int2d(Th) ( dx(uh)*dx(vh) + dy(uh)*dy(vh) ) // bilinear
  + on(C0,uh=0) // boundary condition on C0
  + on(C1,uh=1) // +1 volt on C1
  + on(C2,uh=-1) ; // -1 volt on C2
Electro; // solve the problem, see figure 9.4 for the solution
plot(uh,ps="electro.eps",wait=true); // figure 9.4
```

Figure 9.3: Disk with two elliptical holes  
Figure 9.4: Equipotential lines, where \( C_1 \) is located in right hand side
9.1.3 Aerodynamics

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

$$\Delta \varphi = 0 \text{ in } \Omega, \quad \varphi|_S = c, \quad \varphi|_{\Gamma_\infty} = u_\infty x - u_\infty 2x \quad (9.6)$$

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

$$-\Delta \varphi = 0 \text{ in } \Omega, \quad \varphi|_{\Gamma_1} = y - 0.1x, \quad \varphi|_{\Gamma_2} = c, \quad (9.9)$$

where $\Gamma_2$ is the wing profile and $\Gamma_1$ is an approximation of infinity. One finds $c$ by solving:

$$-\Delta \varphi_0 = 0 \text{ in } \Omega, \quad \varphi_0|_{\Gamma_1} = y - 0.1x, \quad \varphi_0|_{\Gamma_2} = 0, \quad (9.10)$$
$$-\Delta \varphi_1 = 0 \text{ in } \Omega, \quad \varphi_1|_{\Gamma_1} = 0, \quad \varphi_1|_{\Gamma_2} = 1. \quad (9.11)$$

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

```plaintext
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; }
```
\[ y = -(0.17735 \times \sqrt{t} - 0.075597 \times t - 0.212836 \times t^2 + 0.17363 \times t^3 - 0.06254 \times t^4); \]

\begin{verbatim}
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;

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);
\end{verbatim
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 \( \{ T_h \} \) is regular (see Section 5.4), then we have the estimates

\[
\| \nabla u - \nabla u_h \|_{0,\Omega} \leq C_1 h \tag{9.14}
\]
\[
\| u - u_h \|_{0,\Omega} \leq C_2 h^2 \tag{9.15}
\]

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

\[
\int_\Omega | \nabla u - \nabla u_h |^2 \, dx \, dy = \int_\Omega \nabla u \cdot \nabla (u - 2u_h) \, dx \, dy + \int_\Omega \nabla u_h \cdot \nabla u_h \, dx \, dy
\]
\[
= \int_\Omega f(u - 2u_h) \, dx \, dy + \int_\Omega f(u_h) \, dx \, dy
\]

The constants \( C_1, C_2 \) are depend on \( T_h \) and \( f \), so we will find them by FreeFem++. In general, we cannot get the solution \( u \) as a elementary functions (see Section 4.8) even if spetical functions are added. Instead of the exact solution, here we use the approximate solution \( u_0 \) in \( V_h(T_h, P_2) \), \( h \sim 0 \).

**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 Boundary Conditions

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 fespace with 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) =
// definition of the problem
  int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) )
  // bilinear form
+ int2d(Th)( -f*vh )
  // linear form
;
laplace;
// solve the problem
plot(uh,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 boundaries. Example 9.6 give such example.

Example 9.6 (periodic4.edp)

real r=0.25;
// a diamond with a hole
9.1. STATIC PROBLEMS

```c
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 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 isovalue 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.

An other example with no equal border, just to see if the code works.
Example 9.7 (periodic4bis.edp)

```plaintext
// irregular boundary condition.
// to build border AB
macro LINEBORDER(A,B,lab) border A#B(t=0,1) {real t1=1.-t;
  // EOM
  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 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; // absisse on AB = ||AX||/||AB||
func s2=dist(Bx,By,x,y)/l2; // absisse on BC = ||BX||/||BC||
func s3=dist(Cx,Cy,x,y)/l3; // absisse on CD = ||CX||/||CD||
func s4=dist(Dx,Dy,x,y)/l4; // absisse on DA = ||DX||/||DA||

mesh Th=buildmesh(AB(n)+BC(n)+CD(n)+DA(n),fixedborder=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 Poisson(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)

```plaintext
verbosity=1;
load "msh3"
load "tetgen"
load "medit"
bool buildTh=0;
```
9.1. STATIC PROBLEMS

mesh3 Th;
try {
   // a way to build one time the mesh an read if the file exist.
   Th = readmesh3("Th-hex-sph.mesh");
}
catch(...) { buildTh=1; }
if( buildTh ){
   ...
   put the code example page
   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 = 1.0;
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 Problems with mixed boundary condition

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

$$
-\Delta u = f \quad \text{in } \Omega \\
u = g \quad \text{on } \Gamma_D, \quad \partial u/\partial n = 0 \quad \text{on } \Gamma_N
$$

(9.16)

where $\Gamma_D$ is a part of the boundary $\Gamma$ and $\Gamma_N = \Gamma \setminus \Gamma_D$. The solution $u$ has the singularity at the points $(\gamma_1, \gamma_2) = \Gamma_D \cap \Gamma_N$. 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

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

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 $r = \sqrt{x^2+y^2}$ and $\theta = \tan(y,x)$ in FreeFem++.
CHAPTER 9. MATHEMATICAL MODELS

Example 9.9 Assume that \( f = -2 \times 30(x^2 + y^2) \) and \( g = u_e = 10(x^2 + y^2)^{1/4} \sin \left( \frac{\tan^{-1}(y/x)}{2} \right) + 30(x^2y^2) \), where \( u_e \) is the exact solution.

```plaintext
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: mesh T0h = buildmesh(N(10)+D1(10)+D2(10)+D3(20)+D4(10));
7: plot(T0h,wait=true);
8: fespace V0h(T0h,P1);
9: V0h u0, v0;
10: func f=-2*30*(x^2+y^2); // given function
11: func us = sin(atan2(y,x)/2)*sqrt( sqrt(x^2+y^2) ); // the singular term of the solution is K*us (K: constant)
12: real K=10.;
13: func ue = K*us + 30*(x^2*y^2);
14: solve Poisson0(u0,v0) = int2d(T0h)( dx(u0)*dx(v0) + dy(u0)*dy(v0) ) // bilinear form
15: - int2d(T0h)( f*v0 ) // linear form
16: + on(2,u0=ue); // boundary condition
17: for (int i=0;i<5;i++)
18: { mesh Th=adaptmesh(Th,us);
19: mesh Th = adaptmesh(Th,us);
```
9.1. STATIC PROBLEMS

29:
30: \textbf{fespace} Vh(Th, P1);
31: Vh u, v;
32: \textbf{solve} Poisson(u,v) =
33: \int2d(Th) \left( dx(u) \times dx(v) + dy(u) \times dy(v) \right) \quad // \text{bilinear form}
34: - \int2d(Th) \left( f \times v \right) \quad \quad // \text{linear form}
35: + \ \text{on}(2, u=ue) \quad ; \quad // \text{boundary condition}
36:
37: /* plot the solution */
38: \textbf{plot}(Th,ps="adaptDNmix.ps");
39: \textbf{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: \text{cout} \text{ "Relative error in first mesh \} \int2d(Th)(H1err0)/H1e\)" << endl;
50: \text{cout} \text{ "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,

\[
\frac{\|u_0^h - u_e\|_1,\Omega}{H1e} = 0.120421
\]
\[
\frac{\|u^h - u_e\|_1,\Omega}{H1e} = 0.0150581
\]

where $u_0^h$ is the numerical solution in $T0h$ and $u^h$ is $u$ in this program.

9.1.7 Poisson with mixed 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

\[
-\Delta p = 1 \quad \text{in } \Omega
\]
\[
p = g_d \quad \text{on } \Gamma_D, \quad \partial p / \partial n = g_n \quad \text{on } \Gamma_N
\]  

(9.17)

where $\Gamma_D$ is a part of the boundary $\Gamma$ and $\Gamma_N = \Gamma \setminus \Gamma_D$.

The mixte formulation is: find $p$ and $u$ such that

\[
\nabla p + u = 0 \quad \text{in } \Omega
\]
\[
\nabla \cdot u = f \quad \text{in } \Omega
\]
\[
p = g_d \quad \text{on } \Gamma_D, \quad \partial u \cdot n = g_n, n \quad \text{on } \Gamma_N
\]  

(9.18)

where $g_n$ is a vector such that $g_n, n = g_n$.

The variationnal formulation is,

\[
\forall v \in \mathcal{V}_0, \quad \int_{\Omega} p \nabla v + vv = \int_{\Gamma_d} g_d v n
\]
\[
\forall q \in \mathcal{P} \quad \int_{\Omega} q \nabla u = \int_{\Omega} q f
\]
\[
\partial u n = g_n, n \quad \text{on } \Gamma_N
\]  

(9.19)
where the functional space are:

\[ P = L^2(\Omega), \quad V = H(\text{div}) = \{ \mathbf{v} \in L^2(\Omega)^2, \nabla \cdot \mathbf{v} \in L^2(\Omega) \}\]

and

\[ V_0 = \{ \mathbf{v} \in V; \quad \mathbf{v} \cdot n = 0 \text{ on } \Gamma_N \}. \]

To write, the FreeFem++ example, we have just to choose the finite elements spaces. Here \( V \)
space is discretized with Raviart-Thomas finite element \( \text{RT0} \) and \( P \) is discretized by constant finite element \( P0 \).

**Example 9.10 (LaplaceRT.edp)**

```plaintext
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 sure
                  // 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.

```plaintext
1 : border ba(t=0,1.0){x=t; y=0; label=1;};  // see Fig.5.15
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;};
```
9.1. STATIC PROBLEMS

7: mesh Th = buildmesh (ba(6) + bb(4) + bc(4) + bd(4) + be(4) + bf(6));
8: savemesh(Th,"th.msh");
9: fespace Vh(Th,P2);
10: fespace Nh(Th,P0);
11: Vh u,v;
12: Nh rho;
13: real[int] viso(21);
14: for (int i=0;i<viso.n;i++)
15: viso[i]=10.^(+(i-16.)/2.);
16: real error=0.01;
17: func f=(x-y);
18: problem Probem1(u,v,solver=CG,eps=1.0e-6) =
19: int2d(Th,qforder=5)( u*v*1.0e-10+ dx(u)*dx(v) + dy(u)*dy(v))
20: + int2d(Th,qforder=5)( -f*v);
21: ************

Now, the local error indicator $\eta_T$ is:

$$\eta_T = \left( h_T^2 ||f + \Delta u_h||^2_{L^2(T)} + \sum_{e\in E_T} h_e \left( \|\frac{\partial u_h}{\partial n}\|_{L^2(e)}^2 \right) \right)^{1/2}$$

where $h_T$ is the longest's edge of $T$, $E_T$ 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 course, 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: intalledges(Th)(chiK*lenEdge*square(jump(N.x*dx(u)+N.y*dy(u))))
33: + int2d(Th)(chiK*square(hTriangle*(f+dxx(u)+dyy(u))))
34: for (int i=0;i< 4;i++)
35: {
36: Probem1;
37: cout << u[].min << " " << u[].max << endl;
38: plot(u,wait=1);
39: cout << " indicator2 " << endl;
40: rho[] = indicator2(0,Nh);
41: rho=sqrt(rho);
42: cout << "rho = min " << rho[].min << " max=" << rho[].max << endl;
43: plot(rho,fill=1,wait=1,cmm="indicator density ",ps="rhoP2.eps", value=1,viso=viso.nbiso=viso.n);
44: plot(Th,wait=1,cmm="Mesh ",ps="ThrhoP2.eps");
45: Th=adaptmesh(Th,[dx(u),dy(u)],err=error,anisomax=1);
46: plot(Th,wait=1);
47: u=u;
48: rho=rho;
49: error = error/2;
50: };

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]
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 defined 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 \texttt{MeshSizecomputation} to get a \( P_1 \) mesh size as the average of edge length.

```cpp
// 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 = \text{integral}(e) 1 ds) */
  varf vmeshsizen(u,v)=intalledges(Th,qfnbpE=1)(v);
  /* number of edge / par vertex */
  varf vedgecount(u,v)=intalledges(Th,qfnbpE=1)(v/lenEdge);
  /*
   computation of the mesh size
   ----------------------------- */
```

Figure 9.11: Density of the error indicator with isotropic \( P_2 \) metric
count = vedgecount(0, Vh);

h[0] = 0.;

h[0] = vmeshsizen(0, Vh);

\texttt{cout} << " count min = " << count.min << " " << count.max << endl;

h[0] = h[0] / count;

\texttt{cout} << " -- bound meshsize = " << h[0].min << " " << h[0].max << endl;

// end of macro \texttt{MeshSizecomputation}

A second macro to remesh according to the new mesh size.

\texttt{macro ReMeshIndicator(Th, Ph, Vh, vindicator, coef)} \{
\}

\texttt{Vh h=0;}

//evalutate the mesh size */

\texttt{MeshSizecomputation(Th, Vh, h);}

\texttt{Ph etak;}

etak[] = vindicator(0, Ph);

etak[] = sqrt(etak[]);

\texttt{real etastar = coef * (etak[].sum/etak[].n);}

\texttt{cout} << " etastar = " << etastar << " sum=" << etak[].sum << " " << endl;

/* here \texttt{etaK} is discontinous
we use the \texttt{P1 L2} projection with mass lumping . */

\texttt{Vh fn, sigma;}

\texttt{varf veta(unused, v)=int2d(Th)(etak*v);} \texttt{varf vun(unused, v)=int2d(Th)(1*v);} \texttt{fn[]} = veta(0, Vh); \texttt{sigma[]} = vun(0, Vh);

\texttt{fn[]} = fn[0] / sigma[];

\texttt{fn = max(min(fn/etastar, 3.), 0.3333)};

/* new mesh size */

\texttt{h = h / fn ;}

/* plot(h, wait=1); */

/* build the new mesh */

\texttt{Th = adaptmesh(Th, IsMetric=1, h, splitpbedge=1, nbvx=10000);} \}

\texttt{We skip the mesh construction, see the previous example,}

\texttt{fespace Vh(Th, P1); // for the mesh size and solution}

\texttt{fespace Ph(Th, P0); // for the error indicator}

\texttt{real hinit=0.2; // initial mesh size}

\texttt{Vh h=hinit; // the FE function for the mesh size}

\texttt{ // to build a mesh with a given mesh size : meshsize}
Th=adaptmesh(Th,h,IsMetric=1,splitpbedge=1,nbvx=10000);
plot(Th,wait=1,ps="RRI-Th-init.eps");
Vh u,v;

func f=(x-y);

problem Poisson(u,v) =
   int2d(Th,qforder=5) ( u*v*1.0e-10+ dx(u)*dx(v) + dy(u)*dy(v))
   - int2d(Th,qforder=5) ( f*v);

varf indicator2(unused,chiK) =
   intalledges(Th) (chiK*lenEdge*square(jump(N.x*dx(u)+N.y*dy(u))))
   +int2d(Th) (chiK*square(hTriangle*(f+dxx(u)+dyy(u)))) ;

for (int i=0;i< 10;i++)
{
   u=u;
Poisson;
   plot(Th,u,wait=1);
   real cc=0.8;
   if(i>5) cc=1;
   ReMeshIndicator(Th,Ph,Vh,indicator2,cc);
   plot(Th,wait=1);
}

Figure 9.12: the error indicator with isotropic $P_1$, the mesh and isovalue of the solution

9.2 Elasticity

Consider an elastic plate with undeformed shape $\Omega \times ]-h,h[ \times ]-h,h[ \subset \mathbb{R}^3$, $\Omega \subset \mathbb{R}^2$. 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
9.2. ELASTICITY

\[ \mathbf{u} = (u_1, u_2, u_3) = (\xi_1 - x_1, \xi_2 - x_2, \xi_3 - x_3) \] is called the displacement vector. By the deformation, the line segment \( \mathbf{x}, \mathbf{x} + \tau \Delta \mathbf{x} \) moves approximately to \( \mathbf{x} + \mathbf{u}(\mathbf{x}), \mathbf{x} + \tau \Delta \mathbf{x} + \mathbf{u}(\mathbf{x} + \tau \Delta \mathbf{x}) \) 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. [21, p.32])

\[ \lim_{\tau \to 0} \eta(\tau) = (1 + 2\epsilon_{ij}(u_i u_j))^{1/2} - 1, \quad 2\epsilon_{ij} = \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \]

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

\[ (\partial u_k / \partial x_i)(\partial u_k / \partial x_i) \approx 0 \]

and the following is called small strain tensor

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

The tensor \( \epsilon_{ij} \) is called finite strain tensor.

Consider the small plane \( \Delta \mathbf{\Pi}(\mathbf{x}) \) centered at \( \mathbf{x} \) with the unit normal direction \( \mathbf{n} = (n_1, n_2, n_3) \), then the surface on \( \Delta \mathbf{\Pi}(\mathbf{x}) \) at \( \mathbf{x} \)

\[ (\sigma_{ij}(\mathbf{x})n_j, \sigma_{2j}(\mathbf{x})n_j, \sigma_{3j}(\mathbf{x})n_j) \]

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

\[ \sigma_{ij}(\mathbf{x}) = \epsilon_{ij}(\mathbf{x}) \varepsilon_{ij}(\mathbf{x}) \]

with the symmetry \( c_{ijkl} = c_{jikl}, c_{ijkl} = c_{jikl}, c_{ijkl} = c_{klij} \).

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. [21, p.43]) satisfying

\[ \sigma_{ij} = \lambda \delta_{ij} \text{div}\mathbf{u} + 2\mu \varepsilon_{ij} \quad (9.20) \]

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 \( \Gamma_N \times] - h, h[, \Gamma_N = \partial \Omega \setminus \Gamma_D \), 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 \( u_3 = 0, g_3 = 0 \) is satisfied. In this case, we can suppose that \( f_3 = g_3 = u_3 = 0 \) and \( \mathbf{u}(x_1, x_2, x_3) = \mathbf{n}(x_1, x_2) \) 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 \( \mathbf{u}(x_1, x_2, x_3) = \mathbf{n}(x_1, x_2) \) for all \( -h < x_3 < h \).
Generalized plain stress: The cylinder is subjected to no load at \( x_3 = \pm h \). Introducing the mean values with respect to thickness,
\[
\bar{\pi}_i(x_1, x_2) = \frac{1}{2h} \int_{-h}^{h} u(x_1, x_2, x_3) dx_3
\]
and we derive \( \bar{\pi}_3 \equiv 0 \). Similarly we define the mean values \( \bar{f}, \bar{g}, \bar{\varepsilon}_{ij} \) of the body force and surface force as well as the mean values \( \bar{\sigma}_{ij} \) of the components of stress and strain, respectively.

In what follows we omit the overlines of \( u, f, g, \varepsilon_{ij} \) and \( \sigma_{ij} \). 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, \( \sigma_{ij} = \lambda^* \delta_{ij} \text{div} u + 2 \mu \varepsilon_{ij}, \lambda^* = (2 \lambda \mu) / (\lambda + \mu) \).

The equations of elasticity are naturally written in variational form for the displacement vector \( u(x) \in V \) as
\[
\int_{\Omega} [2 \mu \varepsilon_{ij}(u) \varepsilon_{ij}(v) + \lambda \varepsilon_{ii}(u) \varepsilon_{jj}(v)] = \int_{\Omega} f \cdot v + \int_{\Gamma} g \cdot v, \forall v \in V
\]
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 \( f \) and the boundary force \( g \) is zero on lower and upper side. On the two vertical sides of the beam are fixed.

```plaintext
// 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;}; // right 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));

macro epsilon(u1,u2) [dx(u1),dy(u2),(dy(u1)+dx(u2))/sqrt2]

macro div(u,v) ( dx(u)+dy(v) )

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 3.8
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);
```
Example 9.14 (beam-3d.edp)  Consider elastic box with the undeformed parallelepiped shape $[0.5, 0.1, 0.1]$. 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.

```cpp
#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] // EOM
macro div(u1,u2,u3) ( dx(u1)+dy(u2)+dz(u3) ) // EOM

solve Lame([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 ); // see fig ??
```
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 \( L = (f, g) \in L^2(\Omega)^2 \times L^2(\Gamma_N)^2 \) is given, then the displacement \( u \) is the minimizer of the potential energy functional

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

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

\[
V(\Omega_\Sigma) = \{ v \in H^1(\Omega_\Sigma)^2; \ v = 0 \text{ on } \Gamma_D = \partial \Omega \setminus \overline{\Gamma_N}\},
\]

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

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

If the elasticity is homogeneous isotropic, then the displacement \( u(x) \) is decomposed in an open neighborhood \( U_k \) of \( \gamma_k \) as in (see e.g. [22])

\[
u(x) = \sum_{k=1}^{2} K_i(\gamma_k) r_k^{1/2} S_{ki}^C(\theta_k) + u_{k,R}(x) \quad \text{for } x \in \Omega_\Sigma \cap U_k, \ k = 1, 2 \quad (9.23)
\]

with \( u_{k,R} \in H^2(\Omega_\Sigma \cap U_k)^2 \), 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

\[
S_{ki}^C(\theta_k) = \frac{1}{4\mu(2\pi)^{1/2}} \begin{bmatrix} 2\kappa - 1 \cos(\theta_k/2) - \cos(3\theta_k/2) \\ -2\kappa + 1 \sin(\theta_k/2) + \sin(3\theta_k/2) \end{bmatrix}, \quad (9.24)
\]

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 \( K_i(\gamma) \) and \( K_2(\gamma) \), 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. 5.30) with \( d = 0.0001 \). The undeformed crack \( \Sigma \) is approximated by

\[
\Sigma_d = \{(x, y) : -1 \leq x \leq -10 \ast d, -d \leq y \leq d\} \\
\cup \{(x, y) : -10 \ast d \leq x \leq 0, -d + 0.1 \ast x \leq y \leq d - 0.1 \ast x\}
\]

and \( \Gamma_D = \mathbb{R} \) in Fig. 5.30. In this example, we use three technique:

- Fast Finite Element Interpolator from the mesh \( \mathbb{Q} \) to \( \mathbb{Q} \) 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,

\[
\text{mesh } \text{Plate} = \text{movemesh}(\text{Zoom}, [x+u, y+v]);
\]

\[
\text{plot(Plate);}
\]

- Adaptivity is an important technique here, because a large singularity occurs at \( \gamma \) as shown in \([9,23]\).

The first example creates 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 engineers use photoelasticity to make stress field visible, which shows the principal stress difference

\[
\sigma_1 - \sigma_2 = \sqrt{(\sigma_{11} - \sigma_{22})^2 + 4\sigma_{12}^2}
\]

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, \( K_2(\gamma) = 0 \))** \( \{ \text{CrackOpen.edp} \) \)

```plaintext
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=cb; y=cb*(t-1); }
border T(t=0,2) { x=cb*(1-t); y=ca; }

\text{mesh } \text{Th} = \text{buildmesh}(\text{L1(n/2)}+\text{L2(n/2)}+\text{B(n)}\
+\text{C1(n)}+\text{C21(3)}+\text{C22(3)}+\text{C3(n)}+\text{R(n)}+\text{T(n)});\n\text{cb}=0.1; \text{ca}=0.1;
\text{plot}(\text{Th},\text{wait}=1);\n\text{mesh } \text{Zoom} = \text{buildmesh}(\text{L1(n/2)}+\text{L2(n/2)}+\text{B(n)}+\text{C1(n)}\
+\text{C21(3)}+\text{C22(3)}+\text{C3(n)}+\text{R(n)}+\text{T(n)});\n\text{plot}(\text{Zoom},\text{wait}=1);
\text{real } E = 21.5;
\text{real } \text{sigma} = 0.29;
\text{real } \text{mu} = E/(2*(1+\text{sigma}));
\text{real } \text{lambda} = E*\text{sigma}/((1+\text{sigma})*(1-2*\text{sigma}));
\text{fespace } Vh(\text{Th},[P2,P2]);
\text{fespace } zVh(\text{Zoom,P2});
\text{Vh } \{u,v\}, \{w,s\};
\text{solve } \text{Problem}([u,v],[w,s]) =\n\text{int2d}(\text{Th})\{
2*\mu*(dx(u)*dx(w) + ((dx(v)+dy(u))*(dx(s)+dy(w)))/4 )
+ \lambda*\mu*(dx(u)+dy(v))*(dx(w)+dy(s))/2
\}
-\text{int1d}(\text{Th},T)(0.1*(4-x)*s)+\text{int1d}(\text{Th},B)(0.1*(4-x)*s)
+\text{on}(\text{R},u=0)+\text{on}(\text{R},v=0); // \text{fixed}
```


zVh Sx, Sy, Sxy, N;
for (int i=1; i<=5; i++)
{
    mesh Plate = movemesh(Zoom, [x+u, y+v]);  // deformation near γ
    Sx = lambda*(dx(u)+dy(v)) + 2*mu*dx(u);
    Sy = lambda*(dx(u)+dy(v)) + 2*mu*dy(v);
    Sxy = mu*(dy(u) + dx(v));
    N = 0.1*1*sqrt((Sx-Sy)^2 + 4*Sxy^2);  // principal stress difference
    if (i==1) {
        plot(Plate, ps="1stCOD.eps", bw=1);  // Fig. 9.13
        plot(N, ps="1stPhoto.eps", bw=1);  // Fig. 9.13
    }
    else if (i==5) {
        plot(Plate, ps="LastCOD.eps", bw=1);  // Fig. 9.14
        plot(N, ps="LastPhoto.eps", bw=1);  // Fig. 9.14
        break;
    }
}
Th=adaptmesh(Th, [u, v]);
Problem;

Figure 9.13: Crack open displacement (COD) and Principal stress difference in the first mesh
Figure 9.14: COD and Principal stress difference in the last adaptive mesh

It is difficult to create mode II deformation by the opposed shear force on B and T that is observed in a laboratory. So we use the body shear force along Σ, that is, the x-component \( f_1 \) of the body force \( f \) is given by

\[
f_1(x, y) = H(y - 0.001) \ast H(0.1 - y) - H(-y - 0.001) \ast H(y + 0.1)
\]

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));
(fuse 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)(

```plaintext
-- (continued)
```
2*μ*(dx(u)*dx(w) + ((dx(v)+dy(u))*(dx(s)+dy(w)))/4 ) 
+ λ*(dx(u)+dy(v))*(dx(w)+dy(s))/2 
) 
-int2d(Th)(fx*w) 
+ on(R,u=0)+on(R,v=0); // fixed 

for (int i=1; i<=3; i++) 
{
    mesh Plate = movemesh(Zoom,[x+u,y+v]); // deformation near γ 
    Sx = λ*(dx(u)+dy(v)) + 2*μ*dx(u); 
    Sy = λ*(dx(u)+dy(v)) + 2*μ*dy(v); 
    Sxy = μ*(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. 9.15 
        plot(N,ps="1stPhoto2.eps",bw=1); // Fig. 9.15 
    } else if (i==3) { 
        plot(Plate,ps="LastCOD2.eps",bw=1); // Fig. 9.16 
        plot(N,ps="LastPhoto2.eps",bw=1); // Fig. 9.16 
        break; 
    } 
    Th=adaptmesh(Th,[u,v]); 
    Problem; 
}

Figure 9.15: (COD) and Principal stress difference in the first mesh 

Figure 9.16: COD and Principal stress difference in the last adaptive mesh 

### 9.3 Nonlinear Static Problems

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

\[ J(u) = \int_{\Omega} \frac{1}{2} f(|\nabla u|^2) - u \ast b \]

where \( u \) is function of \( H^1_0(\Omega) \) and \( f \) defined by 

\[ f(x) = a \ast x + x - \ln(1 + x), \quad f'(x) = a + \frac{x}{1 + x}, \quad f''(x) = \frac{1}{(1 + x)^2} \]
9.3.1 Newton-Raphson algorithm

Now, we solve the Euler problem $\nabla J(u) = 0$ with Newton-Raphson algorithm, that is,

$$u^{n+1} = u^n - (\nabla^2 J(u^n))^{-1} \nabla J(u^n)$$

First we introduce the two variational form $vdJ$ and $vhJ$ to compute respectively $\nabla J$ and $\nabla^2 J$

```cpp
// 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
// --------------------------------------
// ddJ = f''(|\nabla u|^2) optimisation
varf vhJ(uh,vh) = int2d(Th)( alpha*( dx(uh)*dx(vh) + dy(uh)*dy(vh) ) + dx(u)*dx(uh) + dy(u)*dx(uh) ) * (dx(uh)*dx(vh) + dy(uh)*dy(vh));

// 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);
    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 depends on your installation of FreeFem++; you need to have compiled (see README_arpack), ARPACK. This tool is available in FreeFem++ if the word “eigenvalue” appears 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 tool is based on the arpack++[^1] the object-oriented version of ARPACK eigenvalue package.[^2]

The function EigenValue computes the generalized eigenvalue of $Au = \lambda Bu$. The Shift-invert method is used by default, with $\sigma$ the shift of the method. The matrix $OP$ is defined with $A - \sigma B$. The return value is the number of converged eigenvalues (can be greater than the number of requested eigenvalues $nev$).

```c
int k=EigenValue(OP,B,nev=, sigma= );
```

where the matrix $OP = A - \sigma B$ with a solver and boundary condition, and the matrix $B$.

From version 3.31 there is also a functional interface:

```c
int k=EigenValue(n,FOP1,FB,nev=, sigma= );
```

where $n$ is the size of the problem, and the operators are now defined through functions, defining respectively the matrix product of $OP^{-1}$ and $B$, as in

```c
int n = OP1.n;
func real[int] FOP1(real[int] & u) { real[int] Au=OP^-1*u;return Au;}
func real[int] FB(real[int] & u) { real[int] Au=B*u;return Au;}
```

If you want finer control over the method employed in ARPACK, you can specify which mode ARPACK will work with (mode=, see ARPACK documentation). The operators necessary for the chosen mode can be passed through the optional parameters $A=, A1=, B=, B1=$ (see below).

- **mode=1**: Regular mode for solving $Au = \lambda u$
  ```c
  int k=EigenValue(n,A=FOP,mode=1,nev= );
  ```
  where the function $FOP$ defines the matrix product of $A$

- **mode=2**: Regular inverse mode for solving $Au = \lambda Bu$
  ```c
  int k=EigenValue(n,A=FOP,B=FB,B1=FB1,mode=2, nev= );
  ```
  where the functions $FOP$, $FB$ and $FB1$ define respectively the matrix product of $A$, $B$ and $B^{-1}$

- **mode=3**: Shift-invert mode for solving $Au = \lambda Bu$
  ```c
  int k=EigenValue(n,A1=FOP1,B=FB,mode=3,sigma=sigma, nev= );
  ```
  where the functions $FOP1$ and $FB$ define respectively the matrix product of $OP^{-1} = (A - \sigma B)^{-1}$ and $B$

[^1]: [http://www.caam.rice.edu/software/ARPACK/](http://www.caam.rice.edu/software/ARPACK/)
You can also specify which subset of eigenvalues you want to compute (which= ). The default value is which= “LM”, for eigenvalues with largest magnitude. “SM” is for smallest magnitude, “LA” for largest algebraic value, “SA” for smallest algebraic value, and “BE” for both ends of the spectrum.

Remark: For complex problems, you need to use the keyword complexEigenValue instead of EigenValue when passing operators through functions.

**Note 9.1 Boundary condition and Eigenvalue Problems**
The locking (Dirichlet ) boundary condition is made with exact penalization so we put $1e30=tgv$ on the diagonal term of the locked degree of freedom (see equation (6.31)). So take Dirichlet boundary condition just on $A$ and not on $B$. because we solve $w = OP^{-1} * B * v$.

If you put locking (Dirichlet ) boundary condition on $B$ matrix (with key work on) you get small spurious modes ($10^{-30}$), due to boundary condition, but if you forget the locking boundary condition on $B$ matrix (no key work ”on”) you get huge spurious ($10^{30}$) modes associated to these boundary conditions. We compute only small mode, so we get the good one in this case.

```plaintext
sym=  the problem is symmetric (all the eigenvalue 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 non symmetric 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;
mode= the computational mode used by ARPACK (see above);
which= the requested subset of eigenvalues (see above).
```

**Example 9.17 (lapEignenValue.edp)** In the first example, we compute the eigenvalues and the eigenvectors of the Dirichlet problem on square $\Omega = ]0, \pi[^2$.
The problem is to 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 $u_{m,n} = \sin(nx) \ast \sin(my)$.

We use the generalized inverse shift mode of the arpack++ library, to find 20 eigenvalues and eigenvectors close to the shift value $\sigma = 20$.

```plaintext
// Computation of the eigen value and eigen vector of the Dirichlet problem on square $[0,\pi]^2$
// we use the inverse shift mode
// the shift is given with the real sigma
// find $\lambda$ and $u_{\lambda} \in H_0^1(\Omega)$ such that:
// $\int_{\Omega} \nabla u_{\lambda} \nabla v = \lambda \int_{\Omega} u_{\lambda} v, \forall v \in H_0^1(\Omega)$

verbosity=10;

mesh Th=square(20,20,[pi*x,pi*y]);
fespace Vh(Th,P2);
Vh u1,u2;

real sigma = 20;

// 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 [..]

matrix OP= op(Vh,Vh,solver=Crout,factorize=1); // crout solver because the matrix in not positive
matrix B= b(Vh,Vh,solver=CG,eps=1e-20);

// important remark:
// the boundary condition is make with exact penalization:
// we put 1e30=tgv on the diagonal term of the lock degree of freedom.
// So take Dirichlet boundary condition just on a variational form
// and not on b variational form.
// because we solve $w = OP^{-1} \ast B \ast v$

int nev=20; // number of computed eigen value close to sigma
real[int] ev(nev); // to store the nev eigenvalue
Vh[int] eV(nev); // to store the nev eigenvector

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

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

for (int i=0;i<k;i++)
{
    u1=eV[i];
    real gg = int2d(Th) (dx(u1)*dx(u1) + dy(u1)*dy(u1));
    real mm= int2d(Th) (u1*u1) ;
}
```
cout << " ---- " << i<< " " << ev[i]<< " err= " <<int2d(Th)(dx(u1)*dx(u1) + dy(u1)*dy(u1) - (ev[i])*u1*u1) << " --- "<<endl;
plot(eV[i],cmm="Eigen Vector "+i+" valeur =" + ev[i] ,wait=1,value=1);
}

The output of this example is:

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

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

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

Eigenvalues:
lambda[1]: 5.0002
lambda[2]: 8.00074
lambda[3]: 10.0011
lambda[4]: 10.0011
lambda[5]: 13.002
lambda[6]: 13.0039
lambda[7]: 17.0046
lambda[8]: 17.0048
lambda[9]: 18.0083
lambda[10]: 20.0096
lambda[11]: 20.0096
lambda[12]: 25.014
lambda[13]: 25.0283
lambda[14]: 26.0159
lambda[15]: 26.0159
lambda[16]: 29.0258
lambda[17]: 29.0273
lambda[18]: 32.0449
lambda[19]: 34.049
lambda[20]: 34.0492

---- 0 5.0002 err= -0.000225891 ----
---- 1 8.00074 err= -0.000787446 ----
---- 2 10.0011 err= -0.00134596 ----
---- 3 10.0011 err= -0.00134619 ----
---- 4 13.002 err= -0.00227747 ----
---- 5 13.0039 err= -0.004179 ----
---- 6 17.0046 err= -0.00623649 ----
---- 7 17.0048 err= -0.00639952 ----
9.5. Evolution Problems

FreeFem++ also solves evolution problems such as the heat equation:

\[
\frac{\partial u}{\partial t} - \mu \Delta u = f \quad \text{in } \Omega \times [0,T],
\]

\[
u(x,0) = u_0(x) \quad \text{in } \Omega; \quad (\partial u/\partial n)(x,t) = 0 \quad \text{on } \partial \Omega \times [0,T].
\]

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,

\[
\frac{\partial u}{\partial t}(x,y,t) = \lim_{\tau \to 0} \frac{u(x,y,t) - u(x,y,t - \tau)}{\tau}
\]
which indicates that \( u^m(x,y) = u(x,y,m\tau) \) will satisfy approximatively
\[
\frac{\partial u}{\partial t}(x,y,m\tau) \simeq \frac{u^m(x,y) - u^{m-1}(x,y)}{\tau}
\]

The time discretization of heat equation (9.27) is as follows:
\[
\frac{u^{m+1} - u^m}{\tau} - \mu \Delta u^{m+1} = f^{m+1} \quad \text{in } \Omega \\
u^0(x) = u_0(x) \quad \text{in } \Omega; \quad \partial u^{m+1}/\partial n(x) = 0 \quad \text{on } \partial \Omega, \quad \text{for all } m = 0, \ldots, [T/\tau],
\]

which is so-called **backward Euler method** for (9.27). To obtain the variational formulation, multiply with the test function \( v \) both sides of the equation:
\[
\int_{\Omega} \{u^{m+1}v - \tau \Delta u^{m+1}v\} = \int_{\Omega} \{u^m + \tau f^{m+1}\} v.
\]

By the divergence theorem, we have
\[
\int_{\Omega} \{u^{m+1}v + \tau \nabla u^{m+1} \cdot \nabla v\} - \int_{\partial \Omega} \tau \left( \frac{\partial u^{m+1}}{\partial n} \right) v = \int_{\Omega} \{u^m v + \tau f^{m+1} v\}.
\]

By the boundary condition \( \partial u^{m+1}/\partial n = 0 \), it follows that
\[
\int_{\Omega} \{u^{m+1}v + \tau \nabla u^{m+1} \cdot \nabla v\} - \int_{\Omega} \{u^m v + \tau f^{m+1} v\} = 0.
\] (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 \( u(x,y,t) = tx^4 \).
\[
\frac{\partial u}{\partial t} - \mu \Delta u = x^4 - \mu 12 tx^2 \quad \text{in } \Omega \times ]0,3[, \quad \Omega = ]0,1[^2 \\
u(x,y,0) = 0 \quad \text{on } \Omega, \quad u|_{\partial \Omega} = t \cdot x^4
\]

\[
\text{mesh Th=square(16,16);} \\
\text{fespace Vh(Th,P1);} \\
\text{Vh u,v,uu,f,g;} \\
\text{real dt = 0.1, mu = 0.01;} \\
\text{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));} \\
\text{real t = 0; \quad // start from t=0} \\
\text{uu = 0; \quad // u(x,y,0)=0} \\
\text{for (int m=0;m<=3/dt;m++)} \\
\{
\text{t=t+dt;} \\
\text{f = x^4-mu*t*12*x^2;} \\
\text{g = t*x^4;}
\}
9.5. EVOLUTION PROBLEMS

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 4.11).

Note 9.2 The stiffness matrix in the 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 $f \in L^2(0, T; V')$, $u^0 \in H$

$$\frac{d}{dt}(u(t), v) + a(u(t), v) = (f(t), v) \quad \forall v \in V, \quad a.e. t \in [0, T]$$ (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 $\theta \in [0, 1]$

$$\frac{1}{\tau} (u_{h}^{n+1} - u_{h}^{n}, \phi_i) + a (u_{h}^{n+\theta}, \phi_i) = \langle f^{n+\theta}, \phi_i \rangle$$ (9.30)

$$u_{h}^{n+\theta} = \theta u_{h}^{n+1} + (1-\theta)u_h^n, \quad f^{n+\theta} = \theta f^{n+1} + (1-\theta)f^n$$

**Note [9.30]** is the forward Euler scheme if $\theta = 0$, Crank-Nicolson scheme if $\theta = 1/2$, the backward Euler scheme if $\theta = 1$.

Unknown vectors $u^n = (u_{h}^1, \cdots, u_{h}^M)^T$ in

$$u^n_i(x) = u^n_1 \phi_1(x) + \cdots + u^n_m \phi_m(x), \quad u^n_1, \cdots, u^n_m \in \mathbb{R}$$

are obtained from solving the matrix

$$(M + \theta \tau A)u^{n+1} = \{ M - (1-\theta)\tau A \} u^n + \tau \{ \theta f^{n+1} + (1-\theta)f^n \}$$ (9.31)

$$M = (m_{ij}), \quad m_{ij} = \langle \phi_j, \phi_i \rangle, \quad A = (a_{ij}), \quad a_{ij} = a(\phi_j, \phi_i)$$

Refer [27] pp.70–75 for solvability of [9.31]. The stability of [9.31] is in [27] Theorem 2.13:

Let $\{T_h\}_{h>0}$ be regular triangulations (see Section 5.4). Then there is a number $\epsilon_0 > 0$ independent of $h$ such that,

$$\begin{aligned}
|u^n_h|^2 &\leq \left\{ \begin{array}{ll}
\frac{1}{3} \|u_0^n\|^2 + \tau \sum_{k=0}^{n-1} \|f^{k+\theta}\|^2_{V_h} & \theta \in [0, 1/2) \\
\|u_0^n\|^2 + \tau \sum_{k=0}^{n-1} \|f^{k+\theta}\|^2_{V_h} & \theta \in [1/2, 1]
\end{array} \right. \\
\theta \in [0,1/2)
\end{aligned}$$ (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

\[
\tau < \frac{2(1-\delta)}{(1-2\theta)c_0^2} h^2
\]  

(9.33)

for arbitrary \( \delta \in (0, 1) \).

2. When \( 1/2 \leq \theta \leq 1 \), we can take \( \tau \) arbitrary.

Example 9.19

```csharp
mesh Th=square(12,12);
fespace Vh(Th,P1);
fespace Ph(Th,P0);
Ph h = hTriangle; // mesh sizes for each triangle
real tau = 0.1, theta=0.;
func real f(real t) {
    return x^2*(x-1)^2 + t*(-2 + 12*x - 11*x^2 - 2*x^3 + x^4);
}
ofstream out("err02.csv"); // file to store calculations
out << "mesh size = "<<h[].max<<", time step = "<<tau<<endl;
for (int n=0;n<5/tau;n++)
    out<<n*tau<<",";
out << endl;
Vh u,v,oldU;
Vh f1, f0;
problem aTau(u,v) =
    int2d(Th)( u*v + theta*tau*(dx(u)*dx(v) + dy(u)*dy(v) + u*v))
    - int2d(Th)(oldU*v - (1-theta)*tau*(dx(oldU)*dx(v)+dy(oldU)*dy(v)+oldU*v))
    - int2d(Th)(tau*( theta*f1+(1-theta)*f0 )*v );
while (theta <= 1.0) {
    real t = 0, T=3; // from t=0 to T
    oldU = 0; // u(x,y,0)=0
    out <<theta<<",";
    for (int n=0;n<T/tau;n++)
        t = t+tau;
        f0 = f(n*tau); f1 = f((n+1)*tau);
        aTau;
        oldU = u;
        plot (u);
        Vh uex = t*x^2*(1-x)^2; // exact sol. = tr^2(1-x)^2
        Vh err = u - uex; // err = FE-sol - exact
        out<< abs(err[].max)/abs(uex[].max) <<","; // \|err\|_{L^\infty(\Omega)}/\|u_{ex}\|_{L^\infty(\Omega)}
    }
    out << endl;
    theta = theta + 0.1;
}
```

We can see in Fig. 9.19 that \( u_h^n(\theta) \) become unstable at \( \theta = 0.4 \), and figures are omitted in the case \( \theta < 0.4 \).
9.5. EVOLUTION PROBLEMS

Figure 9.19: $\max_{x \in \Omega} |u_n^\theta(\theta) - u_{ex}(n\tau)| / \max_{x \in \Omega} |u_{ex}(n\tau)|$ at $n = 0, 1, \cdots, 29$

9.5.2 Convection

The hyperbolic equation
\[ \partial_t u + \alpha \cdot \nabla u = f; \quad \text{for a vector-valued function } \alpha, \tag{9.34} \]
appears frequently in scientific problems, for example in the Navier-Stokes equations, in the 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,
\[ \partial_t u + \alpha \partial_x u = 0, \quad u(x, 0) = u^0(x), \tag{9.35} \]
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, the construction works on similar principles. One begins with the ordinary differential equation (with the convention that $\alpha$ 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 $x, t$ are parameters, and we denote the solution by $X_{x,t}(\tau)$. Then it is noticed that $(x, t) \mapsto v(X(\tau), \tau)$ in $\tau = t$ satisfies the equation
\[ \partial_t v + \alpha \partial_x v = \partial_t \dot{X} \dot{v} + a \partial_x \dot{X} \dot{v} = 0 \]
and by the definition $\partial_t X = \dot{X} = +\alpha$ and $\partial_x X = \partial_x x = t$, because if $\tau = t$ we have $X(\tau) = x$.

The general solution of (9.35) is thus the value of the boundary condition in $X_{x,t}(0)$, that is to say $u(x, t) = u^0(X_{x,t}(0))$ where $X_{x,t}(0)$ 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 $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, the equation for the convection is written
\[ \partial_t u + \alpha \cdot \nabla u = 0 \quad \text{in } \Omega \times (0, T) \]
where $\alpha(x, t) \in \mathbb{R}^d$. FreeFem++ implements the Characteristic-Galerkin method for convection operators. Recall that the equation (9.34) can be discretized as
\[ \frac{Du}{Dt} = f \quad \text{i.e. } \frac{du}{dt}(X(t), t) = f(X(t), t) \quad \text{where } \frac{dX}{dt}(t) = \alpha(X(t), t) \]
where $D$ is the total derivative operator. So a good scheme is one step of backward convection by the method of Characteristics-Galerkin

$$
\frac{1}{\tau} (u^{m+1}(x) - u^m(X^m(x))) = f^m(x) \tag{9.36}
$$

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

$$
u^m(X(m\tau)) = u^m(X((m+1)\tau)) - \tau \sum_{i=1}^d \frac{\partial u^m}{\partial x_i}(X((m+1)\tau)) \frac{\partial X_i}{\partial t}((m+1)\tau) + o(\tau)$$

$$= u^m(x) - \tau \alpha^m(x) \cdot \nabla u^m(x) + o(\tau) \tag{9.37}$$

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

$$u^m(X^m(x)) = u^m(x) - \tau \alpha^m(x) \cdot \nabla u^m(x) + o(\tau). \tag{9.38}$$

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

$$\text{convect} (\alpha, -\tau, u^m) \approx u^m(x - \alpha\tau),$$

we can get the approximation

$$u^m(X^m(x)) \approx \text{convect} ([a^m_1, a^m_2], -\tau, u^m) \text{ by } X^m \approx x \mapsto x - \tau[a^m_1(x), a^m_2(x)].$$

A classical convection problem is that of the “rotating bell” (quoted from [19][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)**

```plaintext
border C(t=0, 2*pi) { x=cos(t); y=sin(t); } // the unit circle
mesh Th = buildmesh(C(30)); // triangulates the disk
fespace Vh(Th,P1);
Vh u0 = exp(-10*((x-0.3)^2 + (y-0.3)^2)); // give u^0

real dt = 0.17, t=0; // time step
Vh a1 = -y, a2 = x; // rotation velocity
Vh u;
for (int m=0; m<2*pi/dt; m++) {
    // time step
    Vh u_ = u;
    u0 = exp(-10*((x-0.3)^2 + (y-0.3)^2));
    u = convect(a1, -dt, u0);
    u = convect(a2, -dt, u);
}
```

```plaintext
}```
9.5. EVOLUTION PROBLEMS

\[ t \leftarrow t + dt; \]
\[ u = \text{convect}([a1, a2], -dt, u0); \]
\[ u0 = u; \]
\[ \text{plot}(u, \text{cmm}=" t = " + t + ", \text{min}=" + u[].\text{min} + ", \text{max}=" + u[].\text{max}, \text{wait}=0); \]

Note 9.3 The scheme \text{convect} is unconditionally stable, then the bell become lower and lower (the maximum of \( u^{37} \) is 0.406 as shown in Fig. 9.21).

Figure 9.20: \( u^0 = e^{-10((x-0.3)^2+(y-0.3)^2)} \)

Figure 9.21: 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 equation in two space variables, (see for example Wilmott et al[44] or Achdou et al [3]).

\[
\partial_t u + \frac{(\sigma_1 x)^2}{2} \frac{\partial^2 u}{\partial x^2} + \frac{(\sigma_2 y)^2}{2} \frac{\partial^2 u}{\partial y^2} + \rho xy \frac{\partial^2 u}{\partial x \partial y} + rS_1 \frac{\partial u}{\partial x} + rS_2 \frac{\partial u}{\partial y} - rP = 0
\]

which is to be integrated in \((0, T) \times \mathbb{R}^+ \times \mathbb{R}^+\) subject to, in the case of a put

\[
u(x, y, T) = (K - \max(x, y))^+. \]

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

\[
\sigma_1 = 0.3, \quad \sigma_2 = 0.3, \quad \rho = 0.3, \quad r = 0.05, \quad K = 40, \quad T = 0.5
\]
An implicit Euler scheme is used and a mesh adaptation is done every 10 time steps. To have an unconditionally stable scheme, the first order terms are treated by the Characteristic Galerkin method, which, roughly, approximates
\[
\frac{\partial u}{\partial t} + a_1 \frac{\partial u}{\partial x} + a_2 \frac{\partial u}{\partial y} \approx \frac{1}{\tau} \left( u^{n+1}(x) - u^n(x - \alpha \tau) \right) 
\] (9.42)

Example 9.21 \([\text{BlackScholes2D.edp}]\)

```c
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].

9.6 Navier-Stokes Equation

9.6.1 Stokes and Navier-Stokes

The Stokes equations are: for a given \( f \in L^2(\Omega)^2 \),
\[
\begin{align*}
-\Delta u + \nabla p &= f \\
\nabla \cdot u &= 0
\end{align*}
\] (9.43)

where \( u = (u_1, u_2) \) is the velocity vector and \( p \) the pressure. For simplicity, let us choose Dirichlet boundary conditions on the velocity, \( u = u_\Gamma \) on \( \Gamma \).
In Temam [Theorem 2.2], there is a weak form of (9.43): Find $\mathbf{v} = (v_1, v_2) \in \mathbf{V}(\Omega)$

$$\mathbf{V}(\Omega) = \{ \mathbf{w} \in H^1_0(\Omega)^2 \mid \text{div}\mathbf{w} = 0 \}$$

which satisfy

$$\sum_{i=1}^2 \int_\Omega \nabla u_i \cdot \nabla v_i = \int_\Omega f \cdot \mathbf{w} \quad \text{for all } \mathbf{v} \in \mathbf{V}$$

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

$$\mathbf{V} = H^1_0(\Omega)^2; \quad \mathbf{W} = \left\{ q \in L^2(\Omega) \mid \int_\Omega q = 0 \right\}$$

By multiplying the first equation in (9.43) with $\mathbf{v} \in \mathbf{V}$ and the second with $q \in \mathbf{W}$, subsequent integration over $\Omega$, and an application of Green’s formula, we have

$$\int_\Omega \nabla \mathbf{u} \cdot \nabla \mathbf{v} - \int_\Omega \text{div} \mathbf{v} p = \int_\Omega f \cdot \mathbf{v}$$

$$\int_\Omega \text{div} \mathbf{u} q = 0$$

This yields the weak form of (9.43): Find $(\mathbf{u}, p) \in \mathbf{V} \times \mathbf{W}$ such that

$$a(\mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) = (f, \mathbf{v})$$

$$b(\mathbf{u}, q) = 0$$

(9.44) (9.45)
for all \((v, q) \in V \times W\), where
\[
a(u, v) = \int_{\Omega} \nabla u \cdot \nabla v = \sum_{i=1}^{2} \int_{\Omega} \nabla u_i \cdot \nabla v_i \quad (9.46)
\]
\[
b(u, q) = -\int_{\Omega} \text{div} u q \quad (9.47)
\]
Now, we consider finite element spaces \(V_h \subset V\) and \(W_h \subset W\), and we assume the following basis functions
\[
V_h = V_h \times V_h, \quad V_h = \{ v_h | v_h = v_1 \varphi_1 + \cdots + v_M \varphi_M \},
\]
\[
W_h = \{ q_h | q_h = q_1 \psi_1 + \cdots + q_M \psi_M \}
\]
The discrete weak form is: Find \((u_h, p_h) \in V_h \times W_h\) such that
\[
a(u_h, v_h) + b(v_h, p_h) = (f, v_h), \quad \forall v_h \in V_h
\]
\[
b(u_h, q_h) = 0, \quad \forall q_h \in W_h
\]
(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
\[
Z_h = \{ v_h \in V_h | b(w_h, q_h) = 0 \quad \text{for all } q_h \in W_h \}
\]

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 \((u_h, p_h)\) of (9.48) satisfying
\[
\|u - u_h\|_{1, \Omega} + \|p - p_h\|_{0, \Omega} \leq C \left( \inf_{v_h \in V_h} \|u - v_h\|_{1, \Omega} + \inf_{q_h \in W_h} \|p - q_h\|_{0, \Omega} \right)
\]
with a constant \(C > 0\) (see e.g. [25, Theorem 10.4]).

Let us denote that
\[
A = (A_{ij}), \quad A_{ij} = \int_{\Omega} \nabla \phi_j \cdot \nabla \phi_i \quad i, j = 1, \cdots, M_V
\]
\[
B = (Bx_{ij}, By_{ij}), \quad Bx_{ij} = -\int_{\Omega} \partial \phi_j / \partial x \varphi_i \quad By_{ij} = -\int_{\Omega} \partial \phi_j / \partial y \varphi_i \\
i = 1, \cdots, M_W; j = 1, \cdots, M_V
\]
then (9.48) is written by
\[
\begin{pmatrix}
A & B^* \\
B & 0
\end{pmatrix}
\begin{pmatrix}
U_h \\
p_h
\end{pmatrix}
= 
\begin{pmatrix}
F_h \\
0
\end{pmatrix}
\]
(9.50)
9.6. NAVIER-STOKES EQUATION

where

\[
A = \begin{pmatrix} A & 0 \\ 0 & A \end{pmatrix} \quad B^* = \begin{bmatrix} Bx^T \\ By^T \end{bmatrix} \quad U_h = \begin{bmatrix} \{u_{1,h}\} \\ \{u_{2,h}\} \end{bmatrix} \quad F_h = \begin{bmatrix} \{\int_{\Omega} f_1 \phi_i\} \\ \{\int_{\Omega} f_2 \phi_i\} \end{bmatrix}
\]

**Penalty method:** This method consists of replacing (9.48) by a more regular problem: Find \((v_h, p_h) \in V_h \times \tilde{W}_h\) satisfying

\[
a(u_h, v_h) + b(v_h, p_h) = (f, v_h), \quad \forall v_h \in V_h
\]

\[
b(u_h, q_h) - \epsilon(p_h, q_h) = 0, \quad \forall q_h \in \tilde{W}_h
\]

where \(\tilde{W}_h \subset L^2(\Omega)\). Formally, we have

\[
\text{div} u_h = \epsilon p_h
\]

and the corresponding algebraic problem

\[
\begin{pmatrix} A & B^* \\ B & -\epsilon I \end{pmatrix} \begin{bmatrix} U_h^e \\ \{p_h^e\} \end{bmatrix} = \begin{bmatrix} F_h \\ 0 \end{bmatrix}
\]

**Note 9.5** We can eliminate \(p_h^e = (1/\epsilon)B u_h^e\) to obtain

\[
(A + (1/\epsilon)B^* B) U_h^e = F_h^e
\]

Since the matrix \(A + (1/\epsilon)B^* B\) is symmetric, positive-definite, and sparse, \((9.52)\) can be solved by known technique. There is a constant \(C > 0\) independent of \(\epsilon\) such that

\[
\|u_h - u_h^e\|_{1,\Omega} + \|p_h - p_h^e\|_{0,\Omega} \leq C \epsilon
\]

(see e.g. [22, 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 \(u_{\Gamma} \cdot n = 0\) and \(u_{\Gamma} \cdot s = 1\) on the top boundary and zero elsewhere (\(n\) is the unit normal to \(\Gamma\), and \(s\) 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 pressure is approximated with the \(P_1\) FE (\(M_h\) space),

where

\[
X_h = \{ v \in H^1([0,1]^2) \mid \forall K \in T_h \quad v|_K \in P_2 \}
\]

and

\[
M_h = \{ v \in H^1([0,1]^2) \mid \forall K \in T_h \quad v|_K \in P_1 \}
\]

The FE spaces and functions are constructed by
The Stokes operator is implemented as a system-solve for the velocity \((u_1,u_2)\) and the pressure \(p\). The test function for the velocity is \((v_1,v_2)\) and \(q\) for the pressure, so the variational form (9.48) in freefem language is:

```freefem
solve Stokes (u1,u2,p,v1,v2,q,solver=Crout) =
  int2d(Th)( dx(u1)*dx(v1) + dy(u1)*dy(v1) + dx(u2)*dx(v2) + dy(u2)*dy(v2) )
  - p*q*(0.000001)
  - p*dx(v1) - p*dy(v2)
  - dx(u1)*q - dy(u2)*q
  + on(3,u1=1,u2=0)
  + on(1,2,4,u1=0,u2=0);
```

Each unknown has its own boundary conditions.

If the streamlines are required, they can be computed by finding \(\psi\) such that \(\text{rot} \psi = u\) or better,

\[-\Delta \psi = \nabla \times u\]

```freefem
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

\[
\frac{1}{\tau}(u^{n+1} - u^n \circ X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} = 0, \\
\nabla \cdot u^{n+1} = 0
\]

(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

```freefem
int i=0;
real nu=1./100.;
real dt=0.1;
real alpha=1/dt;
```
9.6. NAVIER-STOKES EQUATION

\[
\begin{align*}
Xh & \text{ up1, up2;}
\end{align*}
\]

\textbf{problem} \quad NS (u1,u2,p,v1,v2,q,solver=Crout,init=i) =

\[
\begin{align*}
\text{int2d}(Th)(
\alpha & ( u1*v1 + u2*v2 ) \\
& + nu * ( dx(u1)*dx(v1) + dy(u1)*dy(v1) \\
& + dx(u2)*dx(v2) + dy(u2)*dy(v2) ) \\
& - p*q*(0.000001) \\
& - p*dx(v1) - p*dy(v2) \\
& - dx(u1)*q - dy(u2)*q \\
) \\
+ \text{int2d}(Th) ( -alpha* \\
& \text{convect([up1,up2],-dt,up1)*v1} -alpha*\text{convect([up1,up2],-dt,up2)*v2} )
\end{align*}
\]

\textbf{on} (3,u1=1,u2=0)

\textbf{on} (1,2,4,u1=0,u2=0)

\textbf{for} (i=0;i<=10;i++)

\{
  \text{up1=u1;}
  \text{up2=u2;}
  \text{NS;}
  \text{if} ( (i \% 10) ) \text{// plot every 10 iteration}
    \text{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 Algorithm and Conjugate Gradients

We solve Stokes problem without penalty. The classical iterative method of Uzawa is described by
the algorithm (see e.g.\[24, 17.3\], \[34, 13\] or \[35, 13\]):

\textbf{Initialize:} \quad Let \( p_h^0 \) be an arbitrary chosen element of \( L^2(\Omega) \).

\textbf{Calculate} \( u_h \): \quad Once \( p_h^n \) is known, \( v_h^n \) is the solution of

\[
\begin{align*}
  u_h^n = A^{-1}(f - B^*p_h^n)
\end{align*}
\]

\textbf{Advance} \( p_h \): \quad Let \( p_h^{n+1} \) be defined by

\[
\begin{align*}
p_h^{n+1} = p_h^n + \rho_n B u_h^n
\end{align*}
\]

There is a constant \( \alpha > 0 \) such that \( \alpha \leq \rho_n \leq 2 \) for each \( n \), then \( u_h^n \) converges to the solution \( u_h \),
and then \( B v_h^n \rightarrow 0 \) as \( n \rightarrow \infty \) from the \textbf{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.

\textbf{Example 9.23 (StokesUzawa.edp)}

\textbf{mesh} \quad Th=square(10,10);
\textbf{fespace} \quad Xh(Th,P2),Mh(Th,P1);
\textbf{Xh} \quad u1,u2,v1,v2;
\textbf{Mh} \quad p,q,ppp; \quad \text{// ppp is a working pressure}
278  

\[ \text{varf } bx(u_1,q) = \int_2d(Th)( -(dx(u_1)*q)); \]
\[ \text{varf } by(u_1,q) = \int_2d(Th)( -(dy(u_1)*q)); \]
\[ \text{varf } a(u_1,u_2) = \int_2d(Th)((dx(u_1)*dx(u_2) + dy(u_1)*dy(u_2)) \]
\[ + \text{ on(3,u_1=1) } + \text{ on(1,2,4,u_1=0) ;} \]
\[ \text{// remark: put the on(3,u_1=1) before on(1,2,4,u_1=0) } \]
\[ \text{// because we want zero on intersection } \%
\]
\[ \text{matrix } A= a(Xh,Xh,solver=CG); \]
\[ \text{matrix } Bx= bx(Xh,Mh); \]
\[ \text{matrix } By= by(Xh,Mh); \]
\[ \text{Xh bc1; bc1[]} = a(0,Xh); \]
\[ \text{// boundary condition contribution on } u_1 \]
\[ \text{Xh bc2; bc2 = 0 ;} \]
\[ \text{// no boundary condition contribution on } u_2 \]
\[ \text{Xh b; } \]
\[ p_h^n \to BA^{-1}(-B^*p_h^n) = -\text{div}_h \text{ is realized as the function } \text{divup}. \]

\[ \text{func real[\int] divup(real[\int] & pp) } \]
\[ \{ \]
\[ \text{b[]} = Bx'*pp; b[] *= -1; b[] += bc1[] ; \]
\[ \text{u1[]} = A^{-1}b[]; \]
\[ \text{// compute } u_1(pp) \]
\[ \text{b[]} = By'*pp; b[] *= -1; b[] += bc2[] ; \]
\[ \text{u2[]} = A^{-1}b[]; \]
\[ \text{// compute } u_2(pp) \]
\[ \text{// } u^n = A^{-1}(Bx^T p^n ) By^T p^n)^T \]
\[ \text{ppp[]} = \text{Bx*}\text{u1[]} ; \]
\[ \text{// } ppp = Bxu_1 \]
\[ \text{ppp[]} += \text{By*}\text{u2[]} ; \]
\[ \text{// } +Byu_2 \]
\[ \text{return ppp[] ;} \]
\[ \}

Call now the conjugate gradient algorithm:

\[ p=0; q=0; \]
\[ \text{LinearCG(divup,p[],eps=1.e-6,nbiter=50); } \]
\[ \text{// } p_{h}^{n+1} = p_{h}^{n} + Bu_{h}^{n} \]
\[ \text{// if } n > 50 \text{ or } |p_{h}^{n+1} - p_{h}^{n}| \leq 10^{-6}, \text{ then the loop end.} \]
\[ \text{divup(p[]);} \]
\[ \text{// compute the final solution} \]
\[ \text{plot([u1,u2],p,wait=1,value=true,coef=0.1);} \]

9.6.3  \text{NSUzawaCahouetChabart.edp}

In this example we solve the Navier-Stokes equation past a cylinder with the Uzawa algorithm
preconditioned by the Cahouet-Chabart method (see [36] for all the details). The idea of the
preconditioner is that in a periodic domain, all differential operators commute and the Uzawa
algorithm comes to solving the linear operator \[ \nabla.((\alpha Id + \nu \Delta)^{-1}\nabla, \] where \( Id \) is the
identity operator. So the preconditioner suggested is \( \alpha \Delta^{-1} + \nu Id \).

To implement this, we do

Example 9.24 (NSUzawaCahouetChabart.edp)

\[ \text{real } D=0.1, H=0.41; \]
\[ \text{real } cx0 = 0.2, cy0 = 0.2; \]
\[ \text{// center of cyl.} \]
\[ \text{real } xa = 0.15, ya=0.2, xe = 0.25, ye =0.2; \]
9.6. **NAVIER-STOKES EQUATION**

\* border \* fr1(t=0,2.2){x=t; y=0; label=1;}
\* border \* fr2(t=0,H){x=2.2; y=t; label=2;}
\* border \* fr3(t=2.2,0){x=t; y=H; label=1;}
\* border \* fr4(t=H,0){x=0; y=t; label=1;}
\* border \* fr5(t=2*pi,0){x=cx0+D*sin(t)/2; y=cy0+D*cos(t)/2; label=3;}
\* int \* nn=15;

\* mesh \* Th=buildmesh(fr1(5*nn)+fr2(nn)+fr3(5*nn)+fr4(nn)+fr5(-nn*3));
\* real \* Um= 1.5; \* // max velocity (Rey 100)
\* func \* Ub = Um*.2./3.;
\* real \* nu = 1e-3;
\* real \* Rey = Ub*D/nu;
\* func \* U1 = 4.*Um*y*(H-y)/(H*H) ;
\* func \* U2 = 0. ;
\* real \* T=2,t=0;
\* real \* dt = D/nn/Um; \* // CFL = 1
\* cout << " dt = " << dt <<endl;
\* real \* alpha=1/dt,epsq=1e-10;

\* fespace \* Mh(Th,[P1]);
\* fespace \* Xh(Th,[P2]);
\* fespace \* Wh(Th,[P1dc]);
\* macro \* grad(u) [dx(u),dy(u)]
\* macro \* div(u1,u2) (dx(u1)+dy(u2))

\* varf \* von1([u1,u2,p],[v1,v2,q]) = on(3,u1=0,u2=0) + on(1,u1=U1,u2=U2);

\* varf \* vA(p,q) =int2d(Th)((grad( p ) ’*grad(q)) ) + int1d(Th,2)(100*p*q) ;
\* varf \* vM(p,q) =int2d(Th,qft=qf2pT)( p*q )+ on(2,p=0);
\* varf \* vu([u1],[v1]) = int2d(Th)(alpha*(u1*v1)+nu*(grad(u1)’*grad(v1) )
\* + on(1,3,u1=0) ;
\* varf \* vu1([p],[v1]) = int2d(Th)(p*dx(v1)) ;
\* varf \* vu2([p],[v1]) = int2d(Th)(p*dy(v1)) ;

\* matrix \* pAM=vM(Mh,Mh,solver=UMFPACK);
\* matrix \* pAA=vA(Mh,Mh,solver=UMFPACK);
\* matrix \* AU=vu(Xh,Xh,solver=UMFPACK);
\* matrix \* B1=vu1(Mh,Xh);
\* matrix \* B2=vu2(Mh,Xh);
\* Xh \* u1,u2;
\* Mh \* p;
\* varf \* vonu1([u1],[v1]) = on(1,u1=U1) + on(3,u1=0);
\* varf \* vonu2([u1],[v1]) = on(1,u1=U2) + on(3,u1=0);

\* real[ int ] \* brhs1 = vonu1(0,Xh);
The functions to define Uzawa and the preconditioner part.

```cpp
func real[int] JUzawa(real[int] & pp)
{
    real[int] b1=brhs1; b1 += B1*pp;
    real[int] b2=brhs2; b2 += B2*pp;
    u1[] = AU^-1 * b1;
    u2[] = AU^-1 * b2;
    pp = B1'*u1[];
    pp += B2'*u2[];
    pp = -pp;
    return pp;
}
func real[int] Precon(real[int] & p)
{
    real[int] pa= pAA^-1*p;
    real[int] pm= pAM^-1*p;
    real[int] pp= alpha*pa+nu*pm;
    return pp;
}
```

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)

```cpp
verbosity = 0;
p=0;
Wh w; // to store vorticity ..
real eps=1e-6;
int ndt = T/dt;
for(int i=0; i<ndt;++i)
{
    brhs1 = vrhs1(0,Xh);
    brhs2 = vrhs2(0,Xh);
    int res=LinearCG(JUzawa,p[],precon=Precon,nbiter=100,verbosity=10,veps=eps);
    assert(res==1);
    eps = -abs(eps);
    w = -dy(u1)+dx(u2);
    plot(w,fill=1,wait=0, nbiso=40);
    dt = min(dt,T-t);
    t += dt;
    if( dt < 1e-10*T) break;
}
plot(w,fill=1,wait=0, nbiso=40,ps="NScahouetChabart"); // see fig. 9.24
```
9.7 Variational Inequality

We present a classical example of variational inequality.
Let us denote $C = \{ u \in H^1_0(\Omega), u \leq g \}$

The problem is:

$$u = \arg \min_{u \in C} J(u) = \frac{1}{2} \int_{\Omega} \nabla u . \nabla u - \int_{\Omega} f u$$

where $f$ and $g$ are given function.

The solution is a projection on the convex $C$ of $f^*$ for the scalar product $((v, w)) = \int_{\Omega} \nabla v . \nabla w$ of $H^1_0(\Omega)$ where $f^*$ is solution of $((f^*, v)) = \int_{\Omega} f v, \forall v \in H^1_0(\Omega)$. The projection on a convex satisfy clearly $\forall v \in C, \ (u - v, u - f) \leq 0$, and after expanding, we get the classical inequality

$$\forall v \in C, \ \int_{\Omega} \nabla(u - v) \nabla u \leq \int_{\Omega} (u - v) f.$$

We can also rewrite the problem as a saddle point problem

Find $\lambda, u$ such that:

$$\max_{\lambda \in L^2(\Omega), \lambda \geq 0} \min_{u \in H^1_0(\Omega)} L(u, \lambda) = \frac{1}{2} \int_{\Omega} \nabla u . \nabla u - \int_{\Omega} f u + \int_{\Omega} \lambda(u - g)^+$$

where $((u - g)^+) = \max(0, u - g)$

This saddle point problem is equivalent to find $u, \lambda$ such that:

$$\begin{align*}
\int_{\Omega} \nabla u . \nabla v + \lambda v^+ d\omega &= \int_{\Omega} f u, \quad \forall v \in H^1_0(\Omega) \\
\int_{\Omega} \mu(u - g)^+ &= 0, \quad \forall \mu \in L^2(\Omega), \mu \geq 0, \lambda \geq 0,
\end{align*}$$

(9.54)

A algorithm to solve the previous problem is:

1. k=0, and choose, $\lambda_0$ belong $H^{-1}(\Omega)$
2. loop on $k = 0, ...$
(a) set $I_k = \{ x \in \Omega / \lambda_k + c \ast (u_{k+1} - g) \leq 0 \}$
(b) $V_{g,k+1} = \{ v \in H^1_0(\Omega) / v = g \text{ on } I_k \}$,
(c) $V_{0,k+1} = \{ v \in H^1_0(\Omega) / v = 0 \text{ on } I_k \}$,
(d) Find $u_{k+1} \in V_{g,k+1}$ and $\lambda_{k+1} \in H^{-1}(\Omega)$ such that

$$\begin{align*}
\int_{\Omega} \nabla u_{k+1} \cdot \nabla v_{k+1} \, d\omega &= \int_{\Omega} f v_{k+1}, \quad \forall v_{k+1} \in V_{0,k+1} \\
< \lambda_{k+1}, v > &= \int_{\Omega} \nabla u_{k+1} \cdot \nabla v - fv \, d\omega
\end{align*}$$

where $<,>$ is the duality bracket between $H^1_0(\Omega)$ and $H^{-1}(\Omega)$, and $c$ is a penalty constant (large enough).

You can find all the mathematics about this algorithm in [38].

Now how to do that in FreeFem++

The full example is:

Example 9.25 (VI.edp)

```c
mesh Th=square(20,20);
real eps=1e-5;

fespace Vh(Th,P1); // P1 FE space
int n = Vh.ndof; // number of Degree of freedom
Vh uh,uhp; // solution and previous one
Vh Ik; // to def the set where the contain is reached.
real[int] rhs(n); // to store the right and side of the equation
real c=1000; // the penalty parameter of the algorithm
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

// the variatonal form of the problem:
varf a(uh,vh) =
  int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) ) // bilinear form
  - int2d(Th)( f*vh ) // linear form
  + on(1,2,3,4,uh=fd) ; // boundary condition form

// two version of the matrix of the problem
matrix A=a(Vh,Vh,tgv=tgv,solver=CG); // one changing
matrix AA=a(Vh,Vh,solver:GC); // one for computing residual

// the mass Matrix construction:
varf vM(uh,vh) = int2d(Th)(uh*vh);
matrix M=vM(Vh,Vh); // to do a fast computing of $L^2$ norm : sqrt( u'*w=M*u )

Aii=A.diag; // get the diagonal of the matrix (appear in version 1.46-1)
rhs = a(0,Vh,tgv=tgv);
```

9.8 Domain decomposition

We present three classic examples of domain decomposition technique: first, Schwarz algorithm with overlapping, second Schwarz algorithm without overlapping (also call Shur complement), and last we show to use the conjugate gradient to solve the boundary problem of the Shur complement.

9.8.1 Schwarz Overlap Scheme

To solve

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

the Schwarz algorithm runs like this

\[-\Delta u^{n+1}_1 = f \text{ in } \Omega_1 \quad u^{n+1}_1|_{\Gamma_1} = u^n_2 \]
\[-\Delta u^{n+1}_2 = f \text{ in } \Omega_2 \quad u^{n+1}_2|_{\Gamma_2} = u^n_1 \]

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 9.25: The 2 overlapping mesh $TH$ and $th$

Example 9.26 (Schwarz-overlap.edp)

```plaintext
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;
```
9.8. DOMAIN DECOMPOSITION

plot(U,u,wait=true);
);

Figure 9.26: Isovalues of the solution at iteration 0 and iteration 9

9.8.2 Schwarz non Overlap Scheme

To solve
\[-\Delta u = f \text{ in } \Omega = \Omega_1 \cup \Omega_2 \quad u|_{\Gamma} = 0,\]
the Schwarz algorithm for domain decomposition without overlapping runs like this

Let introduce \( \Gamma_i \) is common the boundary of \( \Omega_1 \) and \( \Omega_2 \) and \( \Gamma^i_c = \partial \Omega_i \setminus \Gamma_i \).
The problem find \( \lambda \) such that \((u_1|_{\Gamma_i} = u_2|_{\Gamma_i})\) where \( u_i \) is solution of the following Laplace problem:
\[-\Delta u_i = f \text{ in } \Omega_i \quad u_i|_{\Gamma_i} = \lambda \quad u_i|_{\Gamma^i_c} = 0\]

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)**

```cpp
// 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");
```

**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 $\Gamma^i_e = \partial \Omega_i \setminus \Gamma_i$. 
Figure 9.28: Isovalues of the solution at iteration 0 and iteration 9 without overlapping

The problem find $\lambda$ such that $(u_1|_{\Gamma_i} = u_2|_{\Gamma_i})$ where $u_i$ is solution of the following Laplace problem:

$$-\Delta u_i = f \text{ in } \Omega_i \quad u_i|_{\Gamma_i} = \lambda \quad u_i|_{\Gamma_i}^e = 0$$

The version of this example uses the Shur complement. The problem on the border is solved by a conjugate gradient method.

First, we construct the two domain

**Example 9.28 (Schwarz-gc.edp)**

```plaintext
// Schwarz without overlapping (Shur complement for Neumann to Dirichlet)
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;

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);

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:

\[
\begin{align*}
\text{Vh1 } u_1, v_1; \\
\text{Vh2 } u_2, v_2;
\end{align*}
\]

\[
\begin{align*}
\text{int } i=0; \\
\text{problem Pb2}(u_2, v_2, init=i, solver=\text{Cholesky}) = \\
\int_\Omega (dx(u_2) dx(v_2) + dy(u_2) dy(v_2)) \\
\quad + \int_\Gamma (-v_2) \\
\quad + \int_{\Omega_2} (-\lambda v_2) + \text{on(outside, } u_2 = 0 \text{);}
\end{align*}
\]

\[
\begin{align*}
\text{problem Pb1}(u_1, v_1, init=i, solver=\text{Cholesky}) = \\
\int_\Omega (dx(u_1) dx(v_1) + dy(u_1) dy(v_1)) \\
\quad + \int_\Gamma (-v_1) \\
\quad + \int_{\Omega_1} (+\lambda v_1) + \text{on(outside, } u_1 = 0 \text{);}
\end{align*}
\]

or, we define a border matrix, because the \( \lambda \) function is none zero inside the domain \( \Omega_1 \):

\[
\begin{align*}
\text{varf } b(u_2, v_2, solver=\text{CG}) = \int_{\Omega_1} (u_2 v_2); \\
\text{matrix } B = b(\text{Vh1, Vh1, solver=CG});
\end{align*}
\]

The boundary problem function,

\[
\lambda \rightarrow \int_{\Gamma_i} (u_1 - u_2) v_1
\]

\[
\begin{align*}
\text{func real[\text{int}] BoundaryProblem(real[\text{int}] &l)} \\
\quad \{ \\
\quad \text{lambda[]} = l; \quad \text{\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \ariant
9.9 Fluid/Structures Coupled Problem

This problem involves the Lamé system of elasticity and the Stokes system for viscous fluids with velocity \( u \) and pressure \( p \):

\[
-\Delta u + \nabla p = 0, \quad \nabla \cdot u = 0, \quad \text{in } \Omega, \quad u = u_\Gamma \quad \text{on } \Gamma = \partial \Omega
\]

where \( u_\Gamma \) is the velocity of the boundaries. The force that the fluid applies to the boundaries is the normal stress

\[
h = (\nabla u + \nabla u^T)n - pn
\]

Elastic solids subject to forces deform: a point in the solid, at \( (x,y) \) goes to \( (X,Y) \) after. When the displacement vector \( v = (v_1,v_2) = (X-x, Y-y) \) is small, Hooke’s law relates the stress tensor \( \sigma \) inside the solid to the deformation tensor \( \epsilon \):

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

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 \( v(x) \in V \) as

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

The data are the gravity force \( g \) and the boundary stress \( h \).

**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.

```cpp
// 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;}; // right 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,ww,vv,s,fluidforce=0;
```
cout << "lambda,mu,gravity ="<<lambda<< " " << mu << " " << gravity << endl;

solve bb(([uu,vv],[w,s])) =

    int2d(th)
        lambda*div(w,s)*div(uu,vv)
        +2.*mu*( epsilon(w,s)’*epsilon(uu,vv) ) )
    + int2d(th) (-gravity*s)
    + on(1,uu=0,vv=0)
    + fluidforce[];

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

Then Stokes equation for fluids ast low speed are solved in the box below the beam, but the beam has deformed the box (see border h):

.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
                                          + sigma12*(N.y*w+N.x*s) )
    )
    + on(1,uu=0,vv=0);
    plot([uu,vv],wait=1);
    real err = sqrt(int2d(th)( (uu-uul1)^2 + (vv-vvl1)^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);
}
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

$$-\mu_i \Delta u = f \quad \text{in } \Omega_i$$

$$\mu_i \frac{\partial u}{\partial n} |_{\Gamma_i} = -\mu_j \frac{\partial u}{\partial n} |_{\Gamma_j} \quad \text{on } \Omega_i \cap \Omega_j \quad \text{if } 1 \leq i < j \leq 3 \quad (9.56)$$

where $\frac{\partial u}{\partial n} |_{\Gamma_i}$ denotes the value of the normal derivative $\frac{\partial u}{\partial n}$ on the boundary $\Gamma_i$ of the domain $\Omega_i$. By introducing the characteristic function $\chi_i$ of $\Omega_i$, that is,

$$\chi_i(x) = 1 \quad \text{if } x \in \Omega_i; \quad \chi_i(x) = 0 \quad \text{if } x \notin \Omega_i$$

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

$$a(u, v) = \ell(f, v) \quad \text{for all } v \in H^1_0(\Omega) \quad (9.58)$$

$$a(u, v) = \int_{\Omega} \mu \nabla u \cdot \nabla v, \quad \ell(f, v) = \int_{\Omega} f v$$

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:

```c
// 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=0.5;};
border e(t=0.5,1){x=1-t;y=1;};
border f(t=0,1){x=0;y=1-t;};
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;};

// internal boundary

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);
// $P_1$ continuous functions / element

Ph reg=region;
// defined the $P_0$ function associated to region number
plot(reg,fill=1,wait=1,value=1);

Figure 9.30: the function \textit{reg}

Figure 9.31: the function \textit{nu}

region is a keyword of FreeFem++ which is in fact a variable depending of the current position (is not a function today, use Ph reg=region; to set a function). This variable value returned is the number of the subdomain of the current position. This number is defined by "buildmesh" which scans while building the mesh all its connected component. So to get the number of a region containing a particular point one does:

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

```plaintext
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 9.32: the isovalue of the solution $u$
9.11 Free Boundary Problem

The domain $\Omega$ is defined with:

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

int n=4;

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

Figure 9.33: The mesh of the domain $\Omega$

The free boundary problem is:

Find $u$ and $\Omega$ such that:

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

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

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 than $u = y$ on $\Gamma_b^n$ and $\Gamma_a^n$

$$
\int_{\Omega^n} \nabla u \nabla u' = 0, \quad \forall u' \in V \text{ with } u' = 0 \text{ on } \Gamma_b^n \cup \Gamma_f^n
$$

and secondly to construct a domain deformation $F(x,y) = [x, y - v(x,y)]$
where \( v \) is solution of the following problem:

\[
\begin{align*}
-\Delta v &= 0 \quad \text{in } \Omega^n, \\
v &= 0 \quad \text{on } \Gamma^n_a, \\
\frac{\partial v}{\partial n} &= 0 \quad \text{on } \Gamma^n_b \cup \Gamma^n_d, \\
\frac{\partial v}{\partial n} &= \frac{\partial u}{\partial n} - \frac{q}{K} n_x \quad \text{on } \Gamma^n_f.
\end{align*}
\]

The variational formulation is:

find \( v \) on \( V \), such that \( v = 0 \) on \( \Gamma_a^n \)

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

Finally the new domain \( \Omega^{n+1} = F(\Omega^n) \)

**Example 9.30 (freeboundary.edp)** The FreeFem++ implementation is:

```plaintext
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,v,f,u=y) ;

problem Pv(v,vv,solver=CG) = int2d(Th)( dx(v)*dx(vv)+dy(v)*dy(vv))
  + on (a, v=0) + int1d(Th,f)(vv*((q/K)*N.y- (dx(u)*N.x+dy(u)*N.y)));

real errv=1;
real erradap=0.001;
verbosity=1;
while(errv>1e-6)
{
  j++;
  Pu;
  Pv;
  plot(Th,u,v ,wait=0);
  errv=int1d(Th,f)(v*v);
  real coef=1;

  real mintcc = checkmovemesh(Th,[x,y])/5.;
  real mint = checkmovemesh(Th,[x,y-v*coef]);

  if (mint<mintcc || j%10==0) { // mesh to bad => remeshing
    Th=adaptmesh(Th,u,err=erradap ) ;
    mintcc = checkmovemesh(Th,[x,y])/5.;
  }

  while (1)

```
9.12. NON LINEAR ELASTICITY (NLINEAR-ELAS.EDP)

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

\[
\min J(u_1, u_2) = \int_\Omega f(F_2) - \int_{\Gamma_p} P_n u_2
\]

where \(F_2(u_1, u_2) = A(E[u_1, u_2], E[u_1, u_2])\) and \(A(X, Y)\) is bilinear sym. positive form with respect to 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 $u = (u_1, u_2)$, $v = (v_1, v_2)$, $w = (w_1, w_2)$.
So, the differential of $J$ is

$$DJ(u)(v) = \int DF^2(u)(v) f'(F^2(u))) - \int_{\Gamma_p} P_\alpha v_2$$

where $DF^2(u)(v) = 2 A( DE[u](v), E[u] )$ and $DE$ is the first differential of $E$.
The second order differential is

$$D^2J(u)((v), (w)) = \int DF^2(u)(v) DF^2(u)(w) f''(F^2(u))) + \int D^2F^2(u)(v, w) f'(F^2(u)))$$

where

$$D^2F^2(u)(v, w) = 2 A( D^2E[u](v, w), E[u] ) + 2 A( DE[u](v), DE[u](w) ).$$

and $D^2E$ is the second differential of $E$.

So all notations can be define with macros:

- **macro** EL(u,v) [dx(u), (dx(v)*dy(u)), dy(v)] // is $[\epsilon_{11}, 2\epsilon_{12}, \epsilon_{22}]$
- **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 $[E_{11}, 2E_{12}, E_{22}]$
- **macro** dE(u,v,uu,vv) (EL(uu,vv)+dENL(u,v,uu,vv)) //
- **macro** ddE(u,v,uu,vv) (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) (dE(u,v)’*A*ddE(u,v,uu,vv)) *2. ) + (E(u,v)’*A*ddE(u,v,uu,vv)) *2. ) // EOM

The Newton Method is
choose $n = 0$,and $u_0, v_0$ the initial displacement

- loop:
  - find $(du, dv)$ : solution of
    $$D^2J(u_n, v_n)((w, s), (du, dv)) = DJ(u_n, v_n)(w, s), \forall w, s$$
  - $u_n = u_n - du, v_n = v_n - dv$
  - until $(du, dv)$ small is enough
The way to implement this algorithm in FreeFem++ is use a macro tool to implement $A$ and $F^2, f, f', f''$.

A macro is like in ccp 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.

\[ \sigma = 2\mu E + \lambda \text{tr}(E)\text{Id} \]

\[ A(u, v) = \sigma(u) : E(v) \]

\[ \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} \]

real $a_{11} = 2\mu + \lambda$ ;
real $a_{22} = \mu$ ;
real $a_{33} = 2\mu + \lambda$ ;
real $a_{12} = 0$ ;
real $a_{13} = \lambda$ ;
real $a_{23} = 0$ ;
real $a_{21} = a_{12}$ ;
real $a_{31} = a_{13}$ ;
real $a_{32} = a_{23}$ ;
func A = [ [ a11,a12,a13], [ a21,a22,a23], [ a31,a32,a33] ];

real Pa=1e2; // a pressure of 100 Pa

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

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

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

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

varf vmass([uu,vv],[w,s],solver=CG) = int2d(Th)( uu*w + vv*s );
matrix M=vmass(Vh,Vh);
problem NonLin([uu,vv],[w,s],solver=LU)=
    int2d(Th,qforder=1)( // (D^2 J(un)) part
        dF2(un,vn,uu,vv)*dF2(un,vn,w,s)*ddfe2 + ddF2(un,vn,w,s,uu,vv)*ddfe2
    )
    - 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);

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 << " dfe2 max " << dfe2[].max << " , min " << dfe2[].min << endl;
    cout << " ddfe2 max " << ddfe2[].max << " , min " << ddfe2[].min << endl;
    NonLin; // compute [uu,vv] = (D^2 J(un))^{-1}(DJ(un))
w[] = M*uu[];
real res = sqrt(w[]’ * uu[]);  // norme $L^2_{[uu,vv]}
ul = uu;
v1 = vv;
cout << " L^2 residual = " << res << endl;
cout << " ul min = " << ul[].min << " ul max = " << ul[].max << endl;
cout << " vl min = " << vl[].min << " vl max = " << vl[].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 9.36

9.13 Compressible Neo-Hookean Materials: Computational Solutions

Author: Alex Sadovsky mailsashas@gmail.com

9.13.1 Notation

In what follows, the symbols $u, F, B, C, \sigma$ denote, respectively, the displacement field, the deformation gradient, the left Cauchy-Green strain tensor $B = FF^T$, the right Cauchy-Green strain tensor $C = F^T F$, and the Cauchy stress tensor. We also introduce the symbols $I_1 := \text{tr} C$ and $J := \det F$. Use will be made of the identity

$$\frac{\partial J}{\partial C} = JC^{-1}$$

(9.59)

The symbol $I$ 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 $T$ to be rewritten as an integral of the Kirchhoff stress $\kappa = JT$ over $\Omega_0$.

Recommended References

For an exposition of nonlinear elasticity and of the underlying linear- and tensor algebra, see [39]. For an advanced mathematical analysis of the Finite Element Method, see [40]. An explanation of the Finite Element formulation of a nonlinear elastostatic boundary value problem, see http://www.engin.brown.edu/courses/en222/Notes/FEMfinitestrain/FEMfinitestrain.htm.
9.13.2 A Neo-Hookean Compressible Material

Constitutive Theory and Tangent Stress Measures The strain energy density function is given by

\[ W = \frac{\mu}{2} (I_1 - \text{tr} I - 2 \ln J) \] (9.60)

(see [37], formula (12)). The corresponding 2nd Piola-Kirchoff stress tensor is given by

\[ S_n := \frac{\partial W}{\partial E}(F_n) = \mu(I - C^{-1}) \] (9.61)

The Kirchhoff stress, then, is

\[ \kappa = FSF^T = \mu(B - I) \] (9.62)

The tangent Kirchhoff stress tensor at \( F_n \) acting on \( \delta F_{n+1} \) is, consequently,

\[ \frac{\partial \kappa}{\partial F}(F_n)\delta F_{n+1} = \mu \left[ F_n(\delta F_{n+1})^T + \delta F_{n+1}(F_n)^T \right] \] (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 \( \partial \Omega_t \) (= 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

\[ 0 = \int_{\Omega_0} \kappa[F] : \{(\nabla \otimes w)(F)^{-1}\} - \int_{\partial \Omega_0} P \cdot \hat{N}_0 \] for all test functions \( w \),

(9.64)

where we have taken

\[ \delta F_{n+1} = \nabla \otimes \delta u_{n+1} \]

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 \( w \).
9.14. WHISPERING GALLERY MODES

9.13.3 An Approach to Implementation in FreeFem++

The associated file is examples++-tutorial/nl-elast-neo-Hookean.edp.

Introducing the code-like notation, where a string in ‘<’’s is to be read as one symbol, the individual components of the tensor

\[
<TanK> := \frac{\partial \kappa}{\partial F}[F_n] \delta F_{n+1}
\]  

(9.65)

will be implemented as the macros \(<TanK11>, <TanK12>, \ldots\).

The individual components of the tensor quantities

\[
D_1 := F_n(\delta F_{n+1})^T + \delta F_{n+1}(F_n)^T,
\]

\[
D_2 := F_n^{-T} \delta F_{n+1},
\]

\[
D_3 := (\nabla \otimes w)F_n^{-2} \delta F_{n+1},
\]

and

\[
D_4 := (\nabla \otimes w)F_n^{-1},
\]

will be implemented as the macros

\[
<d1Aux11>, <d1Aux12>, \ldots, <d1Aux22>, \ldots, <d2Aux22>, \ldots, <d3Aux22>, \ldots, <d4Aux22>, \ldots
\]

(9.66)

respectively.

In the above notation, the tangent Kirchhoff stress term becomes

\[
\frac{\partial \kappa}{\partial F}(F_n) \delta F_{n+1} = \mu D_1
\]  

(9.67)

while the weak BVP formulation acquires the form

\[
0 = \int_{\Omega_0} \kappa[F_n] : D_4 - \int_{\Omega_0} \kappa[F_n] : D_3 + \int_{\Omega_0} \left\{ \frac{\partial \kappa}{\partial F}[F_n] \delta F_{n+1} \right\} : D_4 \text{ for all test functions } w
\]

(9.68)

9.14 Whispering gallery modes

Written by I. S. Grudinin.

In whispering gallery mode (WGM) resonators, which are typically spheres or disks, electromagnetic field is trapped by total internal reflections from the boundary. Modes of such resonators are distinguished by compact volume and record high quality factors (Q) in a broad range of frequencies. Modern applications of such resonators include microwave and optical cavities for atomic clocks, cavity optomechanics, nonlinear and quantum optics. Analytical solutions for WG modes are only available for a limited number of idealized geometries, such as sphere or ellipsoid. Since resonator dimensions are typically much larger than optical wavelength, direct application of numerical 3D finite difference time domain (FDTD) or finite element methods (FEM) is not practical. It’s possible to solve the vectorial wave equation by reducing it to a two dimensional case by taking axial symmetry into account. Such reduction leads to a system of 3 equations to be solved in a 2D “\( \rho - z \)” section of a resonator. Please refer to [51] for a detailed derivation and to [52] for an example of using FreeFem++ to compute WGMs.


9.14.1 Wave equation for the WGMs

Since electric field is discontinuous on the surface of a dielectric and magnetic field is typically not, we derive our equations for the magnetic field. The electric field can be easily derived at a later stage from

\[ E = \frac{1}{\omega \varepsilon_0} \varepsilon - \nabla \times H. \]

Following a standard procedure starting with Maxwell equations we derive a wave equation in a single-axis anisotropic medium such as an optical crystal:

\[ \nabla \times (\varepsilon^{-1} \nabla \times H) - k_0^2 H - \alpha \nabla (\nabla \cdot H) = 0 \tag{9.69} \]

Here \( k_0 = \omega / c \) is the wavenumber, \( \alpha \) is the penalty term added to fight spurious FEM solutions. For anisotropic single-axis medium with \( \partial \varepsilon / \partial \phi = 0 \) in cylindrical system of coordinates we have:

\[ \varepsilon = \begin{pmatrix} \varepsilon_\rho & 0 & 0 \\ 0 & \varepsilon_\rho & 0 \\ 0 & 0 & \varepsilon_z \end{pmatrix}. \]

We now assume axial symmetry of our electromagnetic fields and insert an imaginary unity in front of the \( H_\phi \) to allow all field components to be real numbers and also to account for the phase shift of this component \( H(\rho, \phi, z) = \{H_\rho(\rho, z), iH_\phi(\rho, z), H_z(\rho, z)\} \times e^{im\phi}. \)

We write the wave equation (9.69) explicitly in cylindrical coordinates, thus obtaining a set of three differential equations for the domain \( \Omega \) given by the resonator’s cross section and some space outside:

\[ A_1\{H_\rho^l, H_\phi^l, H_z^l\} = 0 \quad (9.70) \]

\[ A_2\{H_\rho^l, H_\phi^l, H_z^l\} = 0 \]

\[ A_3\{H_\rho^l, H_\phi^l, H_z^l\} = 0 \]

The numerical solutions of these equations and boundary conditions can be found with FreeFem++ if we write the system in the weak, or integral form.

9.15 Weak formulation

In general, to obtain the integral or “weak” statements equivalent to system (9.70) and boundary conditions we form a scalar dot product between an arbitrary magnetic field test function \( H^t = \{H_\rho^t, H_\phi^t, H_z^t\} \) and the components of our vectorial equation \( A_1, A_2, A_3 \), and integrate over the resonator’s cross section domain \( \Omega \) (and its boundary for the boundary conditions):

\[ \int_\Omega (H_\rho^t A_1 + H_\phi^t A_2 + H_z^t A_3) d\Omega \tag{9.71} \]

We can reduce the order of partial derivatives in this integral by using the Green’s formula for integration by parts. For example:

\[ \int_\Omega H_z^t \frac{\partial^2 H_z}{\partial \rho^2} d\Omega = - \int_\Omega \frac{\partial H_z^t}{\partial \rho} \frac{\partial H_z}{\partial \rho} d\Omega + \oint H_z^t \frac{\partial H_z}{\partial \rho} n_\rho d\Gamma \tag{9.72} \]

Thus converting equations (9.70) we obtain a large expression for the weak form (see [1])

9.15.1 A dielectric sphere example with FreeFem++

We now compute the fundamental mode frequency for a fused silica sphere. The sphere is 36 micrometer in diameter, the refractive index is 1.46, the boundary condition is the magnetic wall (which can actually be omitted as it holds automatically). The example can be found in examples++-eigenvalue/WGM-sphere.edp in the distribution.
Chapter 10

MPI Parallel version

A first attempt of parallelization of FreeFem++ is made here with mpi. An extended interface with MPI has been added to FreeFem++ version 3.5, (see the MPI documentation for the functionality of the language at http://www.mpi-forum.org/docs/mpi21-report.pdf).

10.1 MPI keywords

The following keywords and concepts are used:

- mpiComm to defined a communication world
- mpiGroup to defined a group of processors in the communication world
- mpiRequest to defined a equest to wait for the end of the communication

10.2 MPI constants

- mpisize The total number of processes,
- mpirank the id-number of my current process in \{0,...,mpisize − 1\},
- mpiUndefined The MPI_Undefined constant,
- mpiAnySource The MPI_ANY_SOURCE constant,
- mpiCommWorld The MPI_COMM_WORLD constant,

... and all the keywords of MPI_Op for the reduce operator:
  - mpiMAX, mpiMIN, mpiSUM, mpiPROD, mpiLAND, mpiLOR, mpiLXOR, mpiBAND, mpiBXOR.

10.3 MPI Constructor

```cpp
int[int] proc1=[1,2,3],proc2=[0,4];
mpiGroup grp(procs);  // set MPI_Group to proc 1,2,3 in MPI_COMM_WORLD
```


\begin{verbatim}
mpiGroup grp1(comm,proc1);  // set MPI_Group to proc 1,2,3 in comm
mpiGroup grp2(grp,proc2);  // set MPI_Group to grp union proc1

mpiComm comm=mpiCommWorld;  // set a MPI_Comm to MPI_COMM_WORLD
mpiComm ncomm(mpiCommWorld,grp);  // set the MPI_Comm form grp

mpiComm ncomm(comm,color,key);  // MPI_Comm_split(MPI_Comm comm,
   // int color, int key, MPI_Comm *ncomm)
mpiComm ncomm(processor(local_comm,local_leader),
   processor(peer_comm,peer_leader),tag);
// build MPI_INTERCOMM_CREATE(local_comm, local_leader, peer_comm,
   // remote_leader, tag, &nicomm)
mpiComm ncomm(intercomm,hight) ;  // build using
   // MPI_Intercomm_merge( intercomm, high, &ncomm)
mpiRequest rq;  // defined an MPI_Request

mpiRequest[int] arq(10);  // defined an array of 10 MPI_Request

10.4 MPI functions

mpiSize(comm);  // return the size of comm (int)
mpiRank(comm);  // return the rank in comm (int)

processor(i)  // return processor i with no Request in MPI_COMM_WORLD
processor(mpiAnySource)  // return processor any source
   // with no Request in MPI_COMM_WORLD

processor(i,comm)  // return processor i with no Request in comm
processor(comm,i)  // return processor i with no Request in comm
processor(i,rq,comm)  // return processor i with Request rq in comm
processor(i,rq)  // return processor i with Request rq in
   // MPI_COMM_WORLD

processorblock(i)  // return processor i in MPI_COMM_WORLD
   // in block mode for synchronously communication
processorblock(mpiAnySource)  // return processor any source
   // in MPI_COMM_WORLD in block mode for synchronously communication

processorblock(i,comm)  // return processor i in comm in block mode

mpiBarrier(comm);  // do a MPI_Barrier on communicator comm,
mpiWait(rq);  // wait on of Request,
mpiWaitAll(arq);  // wait add of Request array,
mpiWtime();  // return MPIWtime in second (real),
mpiWtick();  // return MPIWTick in second (real),

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
\end{verbatim}
10.6. SCHWARZ EXAMPLE IN PARALLEL

```cpp
// of com to other comm processor
status=Send( processor(10,comm) , a); // send synchronously
Recv)
// 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], mesh, mesh3, mesh3[int], matrix, matrix<complex>

```cpp
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:

```cpp
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 all this functionality and Thank, to Guy-Antoine Atenekeng Kahou, for his help to code this interface.

10.6 Schwarz example in parallel

This example is a rewriting of example schwarz-overlap in section 9.8.1

```cpp
[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
\[ // \text{add the stop test} \]
\[ // \text{---------------------------------} \]

\[
\text{if ( mpisize != 2 ) } \{
\text{cout } \llg \text{ " sorry, number of processors !=2 " } \llg \text{ endl;} \\
\text{exit(1);}\}
\]
\text{verbosity} = 3;
\text{int interior} = 2;
\text{int exterior} = 1;
\text{border a(t=1,2)\{x=t;y=0;label=exterior;\};}
\text{border b(t=0,1)\{x=2;y=t;label=exterior;\};}
\text{border c(t=2,0)\{x=t;y=1;label=exterior;\};}
\text{border d(t=1,0)\{x = 1-t; y = t;label=interior;\};}
\text{border e(t=0, pi/2)\{ x= cos(t); y = sin(t);label=interior;\};}
\text{border e1(t=pi/2, 2*pi)\{ x= cos(t); y = sin(t);label=exterior;\};}
\text{int n=4;}
\text{mesh[int] Th(mpisize);} \\
\text{if (mpirank == 0) } \{
\text{Th[0] = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));} \\
\text{else } \{
\text{Th[1] = buildmesh ( e(5*n) + e1(25*n) );}
\}
\}
\text{broadcast (processor(0),Th[0]);}
\text{broadcast (processor(1),Th[1]);}
\]
\text{fespace Vh(Th[mpirank],P1);} \\
\text{fespace Vhother(Th[1-mpirank],P1);} \\
\]
\text{Vh u=0,v;} \\
\text{Vhother U=0;} \\
\text{int i=0;}
\text{problem pb(u,v,init=i,solver=Cholesky) =}
\text{int2d(Th[mpirank]) ( dx(u)*dx(v)+dy(u)*dy(v) )} \\
- \text{int2d(Th[mpirank]) ( v) } \\
+ \text{on(interior,u = U) } + \text{on(exterior,u= 0 );}
\]
\text{for ( i=0 ;i< 20; i++) } \{ \\
\text{cout } \llg mpirank \llg " looP " \llg i \llg \text{ endl;} \\
\}
\text{pb;} \\
\text{processor(1-mpirank) } \llg u[]; \text{ processor(1-mpirank) } >> U[]; \\
\text{real err0, err1;}
\text{err0 = int1d(Th[mpirank],interior)(square(U-u));}
\]
\text{processor(1-mpirank)<<err0; processor(1-mpirank)>>err1;}
\text{real err= sqrt(err0+err1);} \\
\text{cout } \llg \text{ err = " } \llg \text{ err } \llg \text{ err0 = " } \llg \text{ err0 } \llg \text{ err1 } \llg \text{ err1 } \llg \text{ endl;} \\
\text{if (err<1e-3) break;} \\
\}
\text{if (mpirank==0) } \{
\text{plot (u,U,ps="uU.eps");} \\
\}
10.6.1 True parallel Schwarz example

This is an explanation of the two script examples++-mpi/MPIGMRES[2]D.edp, a Schwarz parallel with a complexity almost independent of the number of process (with a coarse grid preconditioner).

Thank to F. Nataf.

To solve the following Poisson problem on domain Ω with boundary Γ in $L^2(Ω)$:

$$-\Delta u = f, \text{ in } Ω, \text{ and } u = g \text{ on } Γ,$$

where $f$ and $g$ are two given functions of $L^2(Ω)$ and of $H^1(Γ)$,

Let introduce $(π_i)_{i=1,...,N_p}$, a regular partition of the unity of Ω, q-e-d:

$$π_i \in C^0(Ω): \quad π_i ≥ 0 \text{ and } \sum_{i=1}^{N_p} π_i = 1.$$

Denote $Ω_i$ the sub domain which is the support of $π_i$ function and also denote $Γ_i$ the boundary of $Ω_i$.

The parallel Schwarz method is Let $ℓ = 0$ the iterator and a initial guest $u^0$ respecting the boundary condition (i.e. $u^0|_Γ = g$).

$$∀ i = 1..,N_p : \quad -\Delta u^ℓ_i = f, \text{ in } Ω_i, \quad \text{and } u^ℓ_i = u^ℓ \text{ on } Γ_i \setminus Γ, \quad u^ℓ_i = g \text{ on } Γ_i \cap Γ \quad (10.1)$$

$$u^{ℓ+1} = \sum_{i=1}^{N_p} π_i u^ℓ_i \quad (10.2)$$

After discretization with the Lagrange finite element method, with a compatible mesh $T_{hi}$ of $Ω_i$, i.e., the exist a global mesh $T_h$ such that $T_{hi}$ is include in $T_h$. Let us denote:

- $V_{hi}$ the finite element space corresponding to domain $Ω_i$,
- $N_{hi}$ is the set of the degree of freedom $σ^k_i$,
- $N_{hi}^{Γ_i}$ is the set of the degree of freedom of $V_{hi}$ on the boundary $Γ_i$ of $Ω_i$,
- $σ^k_i(v_h)$ is the value the degree of freedom $k$,
- $V_{0hi} = \{v_h \in V_{hi} : \forall k ∈ N_{hi}^{Γ_i}, \quad σ^k_i(v_h) = 0\}$,
- the conditional expression $a \? b : c$ is defined like in C of C++ language by

  $$a?b:c \equiv \begin{cases} 
  b & \text{if } a \text{ is true} \\
  c & \text{else} 
  \end{cases}.$$

Remark we never use finite element space associated to the full domain Ω because it to expensive.

We have to defined to operator to build the previous algorithm:

We denote $u^ℓ_{hi}$ the restriction of $u^ℓ$ on $V_{hi}$, so the discrete problem on $Ω_i$ of problem (10.1) is find $u^ℓ_{hi} \in V_{hi}$ such that: where $g^k_i$ is the value of $g$ associated to the degree of freedom $k ∈ N_{hi}^{Γ_i}$.

In FreeFem++, it can be written has with $U$ is the vector corresponding to $u^ℓ_{hi}$ and the vector $U1$ is the vector corresponding to $u^ℓ_{hi}$ is the solution of:

```cpp
real[int] U1(Ui.n);
```
real[int] b = onG .* U;
b = onG ? b : Bi ;
U1 = Ai⁻¹*b;

where onG[i] = (i ∈ Γ₁ \ Γ)?1 : 0, and Bi the right of side of the problem, are defined by

```cpp
fespace Whi(Thi,P2);       // def of the Finite element space.
varf vPb(U,V)= int3d(Thi)(grad(U)'*grad(V)) + int3d(Thi)(F*V) + on(1,U=g) + on(10,U=G);
varf vPbon(U,V)= on(10,U=1)+on(1,U=0);
matrix Ai = vPb(Whi,Whi,solver=sparsesolver);
real[int] onG = vPbon(0,Whi);
real[int] Bi=vPb(0,Whi);
```

where the FreeFem++ label of Γ is 1 and the label of Γ₁ \ Γ is 10.

To build the transfer/update part corresponding to (10.2) equation on process _i_, let us call njpart the number the neighborhood of domain of Ω _i_ (i.e: π _j_ is none 0 of Ω _i_), we store in an array jpart all this neighborhood. Let us introduce two array of matrix, Smj[j] to defined the vector to send from _i_ to _j_ a neighborhood process, and the matrix rMj[j] to after to reduce with neighborhood _j_ domain.

So the tranfert and update part compute \( v_i = \pi_i u_i + \sum_{j \in J_i} \pi_j u_j \) and can be write the FreeFem++ function Update:

```cpp
func bool Update(real[int] &ui, real[int] &vi) {
    int n= jpart.n;
    for(int j=0;j<njpart;++j) Usend[j][]=sMj[j]*ui;
    mpiRequest[int] rq(n*2);
    for (int j=0;j<n;++j) Irecv(processor(jpart[j],comm,rq[j ]), Ri[j][]);
    for (int j=0;j<n;++j) Isend(processor(jpart[j],comm,rq[j+n]), Si[j][]);
    for (int j=0;j<n*2;++j)
    
    int k= mpiWaitAny(rq);
    // apply the unity local partition .
    vi = Pii*ui;
    for(int j=0;j<njpart;++j) vi += rMj[j]*Vrecv[j][];   // add \( \pi_j u_j \)
    return true;
}
```

where the buffer are defined by:

```cpp
InitU(njpart,Whij,Thij,aThij,Usend)     // defined the send buffer
InitU(njpart,Whij,Thij,aThij,Vrecv)    // defined the recv buffer
```

with the following macro definition:

```cpp
macro InitU(n,Vh,Th,aTh,U) Vh[int] U(n); for(int j=0;j<n;++j) (Th=aTh[j]; U[j]=0); //
```

**First gmres algorithm:** you can easily accelerate the fixe point algorithm by using a parallel GMRES algorithm after the introduction the following affine \( \mathcal{A}_i \) operator sub domain \( \Omega_i \).

```cpp
func real[int] DJ0(real[int]& U) {
    real[int] V(U.n) , b= onG .* U;
    b = onG ? b : Bi ;
    V = Ai⁻¹*b;
    Update(V,U);
}
```
Where the parallel MPIGMRES or MPICG algorithm is just a simple way to solve in parallel the following \( A_i x_i = b_i, i = 1, \ldots, N_p \) by just changing the dot product by reduce the local dot product of all process with the following MPI code:

```cpp
template<class R> R ReduceSum1(R s, MPI_Comm * comm)
{    R r=0;
    MPI_Allreduce( &s, &r, 1 ,MPI_TYPE<R>::TYPE(), MPI_SUM, *comm );
    return r; }
```

This is done in MPICG dynamics library tool.

**Second gmres algorithm:** Use scharwz algorithm as a preconditioner of basic GMRES method to solving the parallel problem.

```cpp
func real[int] DJ(real[int]& U) // the original problem
{    ++kiter;
    real[int] V(U.n);
    V = Ai*U;
    V = onGi ? 0.: V;  // remove boundary term ...
    return V;
}

func real[int] PDJ(real[int]& U) // the preconditioner
{    real[int] V(U.n);
    real[int] b= onG ? 0. : U;
    V = Ai^-1*b;
    Update(V,U);
    return U;
}
```

**Third gmres algorithm:** Add a coarse solver to the previous algorithm

First build a coarse grid on processor 0, and the matrix \( AC,Rci,Pci; \)

if(mpiRank(comm)==0)
\[
AC = vPbC(VhC,VhC,solver=sparsesolver); \quad \text{// the corase problem}
\]

\[ \text{Pci}= \text{interpolate}(\text{Whi},VhC); \quad \text{// the projection on coarse grid.} \]

\[ \text{Rci} = \text{Pci}^*\text{Pii}; \quad \text{// the Restiction on Process i grid with the partition } \pi_i \]

```cpp
func bool CoarseSolve(real[int]& V,real[int]& U, mpiComm & comm)
{
    \text{// solving the coarse probleme}

    real[int] Uc(Rci.n),Bc(Uc.n);
    Uc= Rci*U;
    mpiReduce(Uc,Bc.processor(0,comm),mpiSUM);
    if(mpiRank(comm)==0)
        Uc = AC^-1*Bc;
    broadcast(processor(0,comm),Uc);
    V = Pci*Uc;
}
```
The New preconditionner

```c

CoarseSolve(V,U,comm);
U = A^-1*b; // ( C1( I -A C2) Uo V = U -V; // Update(V,U);
return U;
}
```

The code to the 4 algorithms:

```c
real epss=1e-6;
int rgmres=0;
if(gmres==1) {
    rgmres=MPIAffineGMRES(DJ0,u[],veps=epss,nbiter=300,comm=comm,
    dimKrylov=100,verbosity=ipart ? 0: 50);
    real[int] b= onG .* u[];
b = onG ? b : Bi ;
v[] = A^-1*b;
Update(v[],u[]);
}
else if(gmres==2) rgmres= MPILinearGMRES(DJ,precon=PDJ,u[],Bi,veps=epss,nbiter=300,comm=comm
    ,dimKrylov=100,verbosity=ipart ? 0: 50);
else if(gmres==3) rgmres= MPILinearGMRES(DJ,precon=PDJC,u[],Bi,veps=epss,nbiter=300,comm=comm,
    dimKrylov=100,verbosity=ipart ? 0: 50);
else // algo Shwarz for demo ... for(int iter=0;iter <10; ++iter) ....
```

We have all ingredient to solve in parallel if we have et the partitions of the unity. To build this partition we do: the initial step on process 1 tp build a coarse mesh, $T_h^*$ of the full domain, and build the partition $\pi$ function constant equal to $i$ on each sub domain $O_i$, $i=1,..,N_p$, of the grid with the Metis graph partitioner [?] and on each process $i$ in 1..,$N_p$ do

1. Broadcast from process 1, the mesh $T_h^*$ (call Thii in FreeFem++ script), and $\pi$ function,
2. remark that the characteristic function $I_{O_i}$ of domain $O_i$, is defined by $(\pi = i)?1 : 0,$
3. let us call $\Pi^2_P$ (resp. $\Pi^2_V$) the $L^2$ on $P^*_h$ the space of the constant finite element function per element on $T^*_h$ (resp. $V^*_h$ the space of the affine continuous finite element per element on $T^*_h$). and build in parallel the $\pi_i$ and $\Omega_i$ such that $O_i \subset \Omega_i$ where $O_i = supp((\Pi^2_P,\Pi^2_V)^m I_{O_i})$.

The set $J_i$ of neighborhood of the domain $\Omega_i$, and the local version on $V_{h_i}$ can be defined the array jpart and njpart with:

$$\pi_i = \frac{\pi^*_i}{\sum_{j=1}^{N_p} \pi^*_j}$$ (10.3)

4. We call $T^*_h$ the sub mesh part of $T^*_h$ where $\pi_j$ are none zero. and tank to the function trunc to build this array,

$$\text{for} (\text{int} \ jp=0; jp<njpart;++jp)$$
$$aThij[jp] = \text{trunc}(Thi,pij[jp]>1e-10,label=10);$$

5. At this step we have all on the coarse mesh , so we can build the fine final mesh by splitting all meshes: $Th_i, Th_j[j], Th_j[j]$ with FreeFem++ trunc mesh function which do restriction and slipping.

6. The construction of the send/recv matrices $sM_j$ and $rM_j$: can done with this code:

```plaintext
mesh3 Thij=Thi;          // variable meshes
fespace Whij(Thij,Pk);   // variable fespace..
matrix Pii; Whi wpii=pii; Pii = wpii[];      // Diagonal matrix corresponding $\pi_i$

matrix[int] sMj(njpart), rMj(njpart);  // M send/rend case..
for(int jp=0; jp<njpart;++jp)
{
  int j=jpart[jp];
  Thij = aThij[jp];                  // change mesh to change Whij,Whij
  matrix I = interpolate(Whij,Whi);  // Whij <- Whi
  sMj[jp] = I*Pii;                   // Whi -> s Whij
  rMj[jp] = interpolate(Whij,Whi,t=1); }
```

7. We choose a function $\pi^*_i = (\Pi^2_P\Pi^2_V)^m I_{O_i}$, so the partition of the unity is simply defined by
To build a not to bad application, I have added code to change variables from parameter values with the following code:

```c++
#include "getARGV.idp"
int verbosity=getARGV("-vv",0);
int vdebug=getARGV("-d",1);
int ksplit=getARGV("-k",10);
int nloc = getARGV("-n",25);
string sff=getARGV("-p"," ");
int gmres=getARGV("-gmres",3);
bool dplot=getARGV("-dp",0);
int nC = getARGV("-N",max(nloc/10,4));
```

And small include to make graphic in parallel of distribute solution of vector $u$ on mesh $T_h$ with the following interface:

```c++
#include "MPIplot.idp"
func bool plotMPIall(mesh &Th,real[int] & u,string cm)
{ PLOTMPIALL(mesh,Pk, Th, u, { cmm=cm, nbiso=20, fill=1, dim=3, value=1 }); return 1; }
```

remark the {cmm=cm, ... =1} in the macro argument is a way to quote macro argument so the argument is cmm=cm, ... =1.
Chapter 11

Parallel sparse solvers

Parallel sparse solvers use several processors to solve linear systems of equation. Like sequential, parallel linear solvers can be direct or iterative. In FreeFem++ both are available.

11.1 Using parallel sparse solvers in FreeFem++

We recall that the solver parameters are defined in the following commands: solve, problem, set (setting parameter of a matrix) and in the construction of the matrix corresponding to a bilinear form. In these commands, the parameter solver must be set to sparsesolver for parallel sparse solver. We have added specify parameters to these command lines for parallel sparse solvers. These 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 solvers and are vectors. These parameters allow the user to preprocess the matrix (see the section on sparse direct solver above 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 are two possibilities to control solver parameters. The first method defines parameters with lparams, dparams and sparams in .edp file. The second one reads the solver parameters from a data file. The name of this file is specified by datafilename. If lparams, dparams, sparams or datafilename is not provided by the user, the solver’s default value is used.

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 libraries 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.
CHAPTER 11. PARALLEL SPARSE SOLVERS

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).

Example 11.1 (testdirectsolvers.edp)

```plaintext
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]],
```
11.1. USING PARALLEL SPARSE SOLVERS IN FREEFEM++

real[int] xx = [1, 32, 45, 7, 2], x(5), b(5), di(5);
b = A * xx;
cout << "b=\n" << b << endl;
cout << "xx=\n" << xx << endl;

set(A, solver=sparsesolver, datafilename="ffpastix_iparm_dparm.txt");
cout << "solving solution" << endl;
x = A^{-1} * b;
cout << "b=\n" << b << endl;
cout << "x=\n" << x << endl;
di = xx - x;
if(mpirank==0){
cout << "x-xx=\n" << endl; cout << "Linf \n" << di.linfty << " L2 " << di.l2 << endl;
}

real defaulttoSuperLUdist();

// 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=\n" << b << endl;
cout << "xx=\n" << xx << endl;

set(A, solver=sparsesolver, datafilename="ffsuperlu_dist_fileparam.txt");
cout << "solving solution" << endl;
x = A^{-1} * b;
cout << "b=\n" << b << endl;
cout << "x=\n" << x << endl;
di = xx - x;
if(mpirank==0){
cout << "x-xx=\n" << endl; cout << "Linf \n" << di.linfty << " 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=sparsesolver,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.linfty << " L2 " << di.l2 << endl;
}

11.2 Sparse direct solver

In this section, we present the sparse direct solvers interfaced with FreeFem++.

11.2.1 MUMPS solver

MUltifrontal Massively Parallel Solver (MUMPS) is a free library [?, ?, ?]. This package solves linear system of the form $Ax = b$ 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 constructs a direct factorization $A = LU$, $A = L'DL$ depending of the symmetry of the matrix $A$. MUMPS uses the following libraries: BLAS[?], BLACS and ScaLAPACK[?].

Remark 7 MUMPS does not solve linear system with a rectangular matrix.

Installation of MUMPS To used MUMPS in FreeFem++, you have to install the MUMPS package into your computer. MUMPS is written in Fortran 90. The parallel version is constructed using MPI [?] for message passing and BLAS [?], BLACS and ScaLAPACK[?]. Therefore, a fortran compiler is needed, and MPI, BLAS, BLACS and ScaLAPACK. 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 works 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 then type make mumps in a terminal window.

Now we give a short description of MUMPS parameters before describing the method to call MUMPS in FreeFem++. 
11.2. SPARSE DIRECT SOLVER

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): INCLTL(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 call MUMPS in FreeFem++, we need to load the dynamic library MUMPS_freeem.dylib (MacOSX), MUMPS_freeem.so (Unix) or MUMPS_freeem.dll (Windows). This is done in typing load "MUMPS_freeem" 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

- For lparams:

  $$lparams[0] = SYM,$$
  $$lparams[1] = PAR,$$
  $$\forall i = 1, \ldots, 40, \quad lparams[i+1] = ICNTL(i).$$

- For dparams:

  $$\forall i = 1, \ldots, 15, \quad dparams[i-1] = CNTL(i).$$
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 non symmetric 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 the 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++
*/
```
11.2. SPARSE DIRECT SOLVER

If no solver parameter is given, we used default option of MUMPS solver.

example A simple example of calling MUMPS in FreeFem++ with these 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 [?, ?] solves linear systems 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 functions 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 includes a parallel symbolic factorization [?]. This scientific library is written in C and MPI for communications.

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 uses a 2D-logical process group. This process grid is specified by nprow (process row) and npcol (process column) such that \( N_p = nprow \times npcol \) where \( N_p \) is the number of all processes allocated for SuperLU_DIST.

The input matrix parameters are controlled by "matrix= " in sparams for internal parameter or in the third line of parameters file. The different values are:
- matrix = assembled: global matrix are available on all processes
- matrix = distributedglobal: the global matrix is distributed among all the processes
- 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 documentation. 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 \( A_p \).

\[
A_p = P_r P_c D_r A D_c P_r^t
\]

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 call 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 call SuperLU_DIST with internal parameter, we used the parameters sparams. The value of parameters of SuperLU_DIST in sparams is defined by
11.2. SPARSE DIRECT SOLVER

sparams ="nprow=1, npcol=1, matrix= distributedgloba, Fact= DOFACT, Equil=NO, ParSymbFact=NO, ColPerm= MMD_AT_PLUS_A, RowPerm= LargeDiag, DiagPivotThresh=1.0, IterRefine=DOUBLE, Trans=NOTRANS, ReplaceTinyPivot=NO, SolveInitialized=NO, PrintStat=NO, DiagScale=NOEQUIL "

This value 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 : NOREFIN, 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 matrices. 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-sparse solver.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.

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

\[
\forall i = 0, \ldots, 63, \quad lparams[i] = iparm[i].
\]
\[
\forall i = 0, \ldots, 63, \quad dparams[i] = dparm[i].
\]

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.

```c
assembled // matrix input :: assembled, distributed global and distributed */
iparm[0]
...
...  
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
### 11.3 Parallel sparse iterative solver

Concerning iterative solvers, we have chosen \textit{pARMS} [?], \textit{HIPS} [?] and \textit{Hypre} [?]. Each software implements a different type of parallel preconditioner. So, \textit{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 \( b - Ax \perp L_m \), where \( K_m \) is Krylov subspace of dimension \( m \) defined by \( K_m = \{ r_0, Ar_0, A^2r_0, ..., A^{m-1}r_0 \} \) and \( L_m \) is another subspace of dimension \( m \) which depends on type of Krylov subspace. For example in GMRES, \( L_m = AK_m \).

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

\textit{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 \( A_i(i = 1, ..., p) \) 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 \( \text{ilut} \) (Incomplete LU with threshold), \( \text{iluk} \) (Incomplete LU with level of fill in) and \( \text{ARMS} \) (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 \( \text{libparms.a} \).

Using pARMS as interface to FreeFem++ Before calling pARMS solver inside FreeFem++, you must first compile file \( \text{parms_FreeFem.cpp} \) to create a dynamic library \( \text{parms_FreeFem.so} \). To do this, move to the directory \( \text{src/solver} \) of FreeFem++, edit the file \( \text{makefileparms.inc} \) to specify the following variables:

- \( \text{PARMS(DIR)} \): Directory of pARMS
- \( \text{PARMS(INCLUDE)} \): Directory for header of pARMS
- \( \text{METIS} \): METIS directory
- \( \text{METIS(LIB)} \): METIS librairy
- \( \text{MPI} \): MPI directory
- \( \text{MPI(INCLUDE)} \): MPI headers
- \( \text{FREEFEM} \): FreeFem++ directory
- \( \text{FREEFEM(INCLUDE)} \): FreeFem++ header for sparse linear solver
- \( \text{LIBBLAS} \): Blas library

After that, in the command line type \( \text{make parms} \) to create \( \text{parms_FreeFem.so} \).

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: \texttt{load \text{parms_freefem} // Tell FreeFem that you will use pARMS}
2: \texttt{border C(t=0,2*\pi)(x=\cos(t); y=\sin(t);label=1;)}
3: \texttt{mesh Th = \text{buildmesh}(C(50));}
4: \texttt{fespace Vh(Th,P2);}
5: \texttt{Vh u,v;}
6: \texttt{func f= x*y;}
7: \texttt{problem Poisson(u,v,solver=\text{sparsesolver}) = \text{int2d}(\text{Th})(dx(u)*dx(v) + dy(u)*dy(v)) // a sparse solver, in this case pARMS}
8: \texttt{- \text{int2d}(\text{Th})( f*v) \text{ // right hand side}}
9: \texttt{+ on(1,u=0) ; \text{ // Dirichlet boundary condition}}
10: \texttt{real cpu=clock();}
11: \texttt{Poisson; \text{ // SOLVE THE PDE}}
12: \texttt{plot(u);}
13: \texttt{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:

\[ \text{solver} = \text{FGMRES}, \quad \text{Krylov dimension} = 30, \quad \text{Maximum of Krylov dimension} = 1000, \quad \text{Tolerance for convergence} = 1 \times 10^{-08}. \]

(see book of Saad [?] to understand all this parameters.)

\[ \text{preconditioner} = \text{Restricted Additive Schwarz}, \quad \text{Inner Krylov dimension} = 5, \quad \text{Maximum of inner Krylov dimension} = 5, \quad \text{Inner preconditioner} = \text{ILUK}. \]

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)**

```cpp
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++) d parm[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
               - (dx(u)+dy(v))*psh // psh div(u)
   + on(cylinder,infwall,supwall,u=0.,v=0.)+on(inlet,u=uc,v=0); // Bdy 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 paradent of Grid5000 and report results in table 11.8.
### Entries of iparm

<table>
<thead>
<tr>
<th>Entries of iparm</th>
<th>Significations of each entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>iparm[0]</td>
<td>Krylov subspace methods. Different values for this parameters are specify on table 11.6</td>
</tr>
<tr>
<td>iparm[1]</td>
<td>Preconditionner. Different preconditionners for this parameters are specify on table 11.7</td>
</tr>
<tr>
<td>iparm[4]</td>
<td>Number of level in arms when used.</td>
</tr>
<tr>
<td>iparm[7]</td>
<td>Symmetric (=1 for symmetric) or unsymmetric matrix: default value 0 (unsymmetric matrix)</td>
</tr>
<tr>
<td>iparm[8]</td>
<td>Overlap size between different subdomain: default value 0 (no overlap)</td>
</tr>
<tr>
<td>iparm[9]</td>
<td>Scale the input matrix or not: Default value 1 (Matrix should be scale)</td>
</tr>
<tr>
<td>iparm[10]</td>
<td>Block size in arms when used: default value 20</td>
</tr>
<tr>
<td>iparm[12]</td>
<td>ilfil for Schur complement const: default value 20</td>
</tr>
<tr>
<td>iparm[14]</td>
<td>Multicoloring or not in ILU when used: default value 1</td>
</tr>
<tr>
<td>iparm[15]</td>
<td>Inner iteration: default value 0</td>
</tr>
<tr>
<td>iparm[16]</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 11.4: Meaning of iparams corresponding variables for example 11.4

### Entries of dparm

<table>
<thead>
<tr>
<th>Entries of dparm</th>
<th>Significations of each entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>dparm[0]</td>
<td>precision for outer iteration: default value 1e-08</td>
</tr>
<tr>
<td>dparm[1]</td>
<td>precision for inner iteration: default value 1e-2</td>
</tr>
<tr>
<td>dparm[2]</td>
<td>tolerance used for diagonal domain: default value 0.1</td>
</tr>
<tr>
<td>dparm[3]</td>
<td>drop tolerance droptol0 (ilut, iluk, and arms): default value 1e-2</td>
</tr>
</tbody>
</table>

Table 11.5: Significations of dparams corresponding variables for example 11.4

### Values of iparm[0]

<table>
<thead>
<tr>
<th>Values of iparm[0]</th>
<th>Krylov subspace methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>FGMRES (Flexible GMRES)</td>
</tr>
<tr>
<td>1</td>
<td>DGMRES (Deflated GMRES)</td>
</tr>
<tr>
<td>2</td>
<td>BICGSTAB</td>
</tr>
</tbody>
</table>

Table 11.6: Krylov Solvers in pARMS
11.3. PARALLEL SPARSE ITERATIVE SOLVER

<table>
<thead>
<tr>
<th>Values of iparm[1]</th>
<th>Preconditioners</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Preconditioners type is \textit{additive Schwarz preconditioner with ilu0 as local preconditioner},</td>
</tr>
<tr>
<td>1</td>
<td>Preconditioners type is \textit{additive Schwarz preconditioner with iluk as local preconditioner},</td>
</tr>
<tr>
<td>2</td>
<td>Preconditioners type is \textit{additive Schwarz preconditioner with ilut as local preconditioner},</td>
</tr>
<tr>
<td>3</td>
<td>Preconditioners type is \textit{additive Schwarz preconditioner with arms as local preconditioner},</td>
</tr>
<tr>
<td>4</td>
<td>Preconditioners type is \textit{Left Schur complement preconditioner with ilu0 as local preconditioner},</td>
</tr>
<tr>
<td>5</td>
<td>Preconditioners type is \textit{Left Schur complement preconditioner with ilut as local preconditioner},</td>
</tr>
<tr>
<td>6</td>
<td>Preconditioners type is \textit{Left Schur complement preconditioner with iluk as local preconditioner},</td>
</tr>
<tr>
<td>7</td>
<td>Preconditioners type is \textit{Left Schur complement preconditioner with arms as local preconditioner},</td>
</tr>
<tr>
<td>8</td>
<td>Preconditioners type is \textit{Right Schur complement preconditioner with ilu0 as local preconditioner},</td>
</tr>
<tr>
<td>9</td>
<td>Preconditioners type is \textit{Right Schur complement preconditioner with ilut as local preconditioner},</td>
</tr>
<tr>
<td>10</td>
<td>Preconditioners type is \textit{Right Schur complement preconditioner with iluk as local preconditioner},</td>
</tr>
<tr>
<td>11</td>
<td>Preconditioners type is \textit{Right Schur complement preconditioner with arms as local preconditioner},</td>
</tr>
<tr>
<td>12</td>
<td>Preconditioners type is \textit{sch_gilu0}, Schur complement preconditioner with global ilu0</td>
</tr>
<tr>
<td>13</td>
<td>Preconditioners type is \textit{SchurSymmetric GS preconditioner}</td>
</tr>
</tbody>
</table>

Table 11.7: Preconditioners in pARMS

<table>
<thead>
<tr>
<th>np</th>
<th>add(iluk)</th>
<th>time</th>
<th>schur(iluk)</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>230</td>
<td>637.57</td>
<td>21</td>
<td>557.8</td>
</tr>
<tr>
<td>8</td>
<td>240</td>
<td>364.12</td>
<td>22</td>
<td>302.25</td>
</tr>
<tr>
<td>16</td>
<td>247</td>
<td>212.07</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>261</td>
<td>111.16</td>
<td>25</td>
<td>81.5</td>
</tr>
</tbody>
</table>

Table 11.8: Convergence and time for solving linear system from example 11.4
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 preconditioners, HIPS implements a type of multilevel ILU. For further informations on those preconditioners 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:

- `HIPS_DIR`: Directory of HIPS
- `HIPS_INCLUDE`: -I$(HIPS_DIR)/SRC/INCLUDE : Directory for HIPS headers
- `LIB_DIR`: -L$(HIPS_DIR)/LIB : Libraries directory
- `LIBHIPSSERIAL`: $(HIPS_DIR)/LIB/libhipsserial.a: HIPS utilities library
- `LIBHIPSS`: $(HIPS_DIR)/LIB/libhips.a: HIPS library
- `FREEFEM`: FreeFem++ directory
- `FREEFEM_INCLUDE`: FreeFem headers for sparse linear solver
- `METIS` : METIS directory
- `METIS_LIB`: METIS library
- `MPI` : MPI directory
- `MPI_INCLUDE`: MPI headers

After specifies all the variables, in the command line in the directory `src/solver` type `make hips` to create `hips_FreeFem.so`. 

<table>
<thead>
<tr>
<th>n</th>
<th>matrix size</th>
</tr>
</thead>
<tbody>
<tr>
<td>nnz</td>
<td>number of non null entries inside matrix</td>
</tr>
<tr>
<td>nit</td>
<td>number of iteration for convergence</td>
</tr>
<tr>
<td>time</td>
<td>Time for convergence</td>
</tr>
<tr>
<td>Te</td>
<td>Time for constructing finite element matrix</td>
</tr>
<tr>
<td>np</td>
<td>number of processor</td>
</tr>
</tbody>
</table>

Table 11.9: Legend of table 11.8
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"
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;
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)) // ’) for emacs
   + int2d(Th,2) (u+v) // ) for emacs
   - int3d(Th) (f*v)
   - int2d(Th,2) ( ue*v + (uex*N.x +uey*N.y +uez*N.z)*v )
   + on(1,u=ue);
<table>
<thead>
<tr>
<th>$n$</th>
<th>$nnz$</th>
<th>$Te$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4 \times 10^6$</td>
<td>$118 \times 10^6$</td>
<td>$221.34$</td>
</tr>
<tr>
<td>8</td>
<td>190</td>
<td>120.34</td>
</tr>
<tr>
<td>16</td>
<td>189</td>
<td>61.08</td>
</tr>
<tr>
<td>32</td>
<td>186</td>
<td>31.70</td>
</tr>
<tr>
<td>64</td>
<td>183</td>
<td>23.44</td>
</tr>
</tbody>
</table>

Table 11.10: Iterations and Timing of solving linear system from example

```c
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=sparsesolver, dparams=dparm, lparams=iparm); // Set hips as linear solver
44: u[]=Aa^-1*F[];
```

Legend of table 11.10 are give in table 11.9

### 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) $Ac = r_c$, 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`. 
11.3. PARALLEL SPARSE ITERATIVE SOLVER

<table>
<thead>
<tr>
<th>Entries of iparm</th>
<th>Significations of each entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>iparm[0]</td>
<td>Strategy use for solving (Iterative=0 or Hybrid=1 or Direct=2). Defaults values are: Iterative</td>
</tr>
<tr>
<td>iparm[1]</td>
<td>Krylov methods. If iparm[0]=0, give type of Krylov methods: 0 for GMRES, 1 for PCG</td>
</tr>
<tr>
<td>iparm[4]</td>
<td>Symmetric(=0 for symmetric) and 1 for unsymmetric matrix: default value 1(unsymmetric matrix)</td>
</tr>
<tr>
<td>iparm[5]</td>
<td>Pattern of matrix are symmetric or not: default value 0</td>
</tr>
<tr>
<td>iparm[6]</td>
<td>Partition type of input matrix: default value 0</td>
</tr>
<tr>
<td>iparm[7]</td>
<td>Number of level that use the HIPS locally consistent fill-in: Default value 2</td>
</tr>
<tr>
<td>iparm[8]</td>
<td>Numbering in indices array will start at 0 or 1: Default value 0</td>
</tr>
<tr>
<td>iparm[9]</td>
<td>Scale matrix. Default value 1</td>
</tr>
<tr>
<td>iparm[10]</td>
<td>Reordering use inside subdomains for reducing fill-in: Only use for iterative. Default value 1</td>
</tr>
<tr>
<td>iparm[11]</td>
<td>Number of unknowns per node in the matrix non-zero pattern graph: Default value 1</td>
</tr>
<tr>
<td>iparm[12]</td>
<td>This value is used to set the number of time the normalization is applied to the matrix: Default 2.</td>
</tr>
<tr>
<td>iparm[14]</td>
<td>HIPS_DOMSIZE Subdomain size</td>
</tr>
</tbody>
</table>

Table 11.11: Significations of iparams corresponding to HIPS interface

<table>
<thead>
<tr>
<th>dparm[0]</th>
<th>HIPS_PREC: Relative residual norm: Default=1e-9</th>
</tr>
</thead>
<tbody>
<tr>
<td>dparm[1]</td>
<td>HIPS_DROPTOL: Numerical threshold in ILUT for interior domain</td>
</tr>
<tr>
<td>dparm[2]</td>
<td>HIPS_DROPTOL1: Numerical threshold in ILUT for Schur preconditioner: Default=0.005</td>
</tr>
<tr>
<td>dparm[3]</td>
<td>HIPS_DROPTOLE: Numerical threshold for coupling between the interior level and Schur: Default 0.005</td>
</tr>
<tr>
<td>dparm[4]</td>
<td>HIPS_AMALG: Numerical threshold for coupling between the interior level and Schur: Default=0.005</td>
</tr>
<tr>
<td>dparm[5]</td>
<td>HIPS_DROPSCHEUR: Numerical threshold for coupling between the interior level and Schur: Default=0.005</td>
</tr>
</tbody>
</table>

Table 11.12: Significations of dparams corresponding to HIPS interface
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:

\[ \begin{align*}
HYPRE_DIR : & \text{ Directory of HYPRE} \\
HYPRE_INCLUDE = & -I$(HYPRE_DIR)/src/hypre/include/ : \\
HYPRE_LIB = & -L$(HIPS_DIR)/src/lib/ -lHYPRE : Hypre Library \\
FREEFEM : & \text{ FreeFem++ directory} \\
FREEFEM_INCLUDE : & \text{ FreeFem header for sparse linear solver} \\
METIS : & \text{ METIS directory} \\
METIS_LIB : & \text{ METIS library} \\
MPI : & \text{ MPI directory} \\
MPI_INCLUDE : & \text{ MPI headers}
\end{align*} \]

Like with pARMS, the calling of HYPRE 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)

```cpp
1: load "msh3" // load librairie
2: load "hipre_FreeFem"
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
```

Like with pARMS, the calling of HYPRE 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.
11.4. Domain Decomposition

In the previous section, we saw that the phases to construct a matrix are sequential. One strategy to construct the matrix in parallel is to divide geometrically the domain into subdomains. In every subdomain we construct a local submatrix and after that we assemble every submatrix to form the global matrix.

We can use this technique to solve pde directly in domain $\Omega$. In this case, in every subdomains you have to define artificial boundary conditions to form consistent equations in every subdomains. After this, you solve equation in every subdomains and define a strategy to obtain the global solution.

For the implementation, we use set:

```plaintext
set(Aa,solver=sparesolver,dparams=dparm, lparams=iparm);
```

### Table 11.13: Definitions of common entries of `iparms` and `dparams` vectors for every preconditioner in HYPRE

<table>
<thead>
<tr>
<th><code>iparms[0]</code></th>
<th>Solver identification: 0: BiCGStab, 1: GMRES, 2: PCG. By default=1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iparms[4]</code></td>
<td>Solver print info level: Default=2</td>
</tr>
<tr>
<td><code>iparms[5]</code></td>
<td>Solver log : Default=1</td>
</tr>
<tr>
<td><code>iparms[6]</code></td>
<td>Solver stopping criteria only for BiCGStab : Default=1</td>
</tr>
<tr>
<td><code>dparams[0]</code></td>
<td>Tolerance for convergence : Default = 1.0e−11</td>
</tr>
</tbody>
</table>

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 for the simulations. We saw that in every case the time for the construction of the finite element matrix is constant. This is normal because until now this phase is sequential in FreeFem++. In contrast, phases for solving the linear system are parallel. We saw on several examples presented here that when we increase the number of processors, in general we decrease the time used for solving the linear systems. 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 exchanged data across processors consequently increasing the time for solving the linear systems. Furthermore, in decomposition domain type preconditioners, 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 increase the time used for solving linear equations.

To end this, we should note that good use of the preconditioners 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 preconditioners sometimes depends on how its parameters are set. For this reason we advise the user to pay attention to the meaning of the parameters in the user guide of the iterative solvers interfaced in FreeFem++.
<table>
<thead>
<tr>
<th>iparms[7]</th>
<th>AMG interpolation type: Default=6</th>
</tr>
</thead>
<tbody>
<tr>
<td>iparms[8]</td>
<td>Specifies the use of GSMG - geometrically</td>
</tr>
<tr>
<td></td>
<td>smooth coarsening and interpolation: Default=1</td>
</tr>
<tr>
<td>iparms[9]</td>
<td>AMG coarsen type: Default=6</td>
</tr>
<tr>
<td>iparms[10]</td>
<td>Defines whether local or global measures</td>
</tr>
<tr>
<td></td>
<td>are used: Default=1</td>
</tr>
<tr>
<td>iparms[11]</td>
<td>AMG cycle type: Default=1</td>
</tr>
<tr>
<td>iparms[12]</td>
<td>AMG Smoother type: Default=1</td>
</tr>
<tr>
<td>iparms[13]</td>
<td>AMG number of levels for smoothers: Default=3</td>
</tr>
<tr>
<td>iparms[14]</td>
<td>AMG number of sweeps for smoothers: Default=2</td>
</tr>
<tr>
<td>iparms[15]</td>
<td>Maximum number of multigrid levels: Default=25</td>
</tr>
<tr>
<td>iparms[16]</td>
<td>Defines which variant of the Schwartz method is</td>
</tr>
<tr>
<td></td>
<td>used:</td>
</tr>
<tr>
<td></td>
<td>0: hybrid multiplicative Schwartz method (no</td>
</tr>
<tr>
<td></td>
<td>overlap across processor boundaries)</td>
</tr>
<tr>
<td></td>
<td>1: hybrid additive Schwartz method (no overlap</td>
</tr>
<tr>
<td></td>
<td>across processor boundaries)</td>
</tr>
<tr>
<td></td>
<td>2: additive Schwartz method</td>
</tr>
<tr>
<td></td>
<td>3: hybrid multiplicative Schwartz method (with</td>
</tr>
<tr>
<td></td>
<td>overlap across processor boundaries)</td>
</tr>
<tr>
<td></td>
<td>Default=1</td>
</tr>
<tr>
<td>iparms[17]</td>
<td>Size of the system of PDEs: Default=1</td>
</tr>
<tr>
<td>iparms[18]</td>
<td>Overlap for the Schwarz method: Default=1</td>
</tr>
<tr>
<td>iparms[19]</td>
<td>Type of domain used for the Schwarz method</td>
</tr>
<tr>
<td></td>
<td>0: each point is a domain</td>
</tr>
<tr>
<td></td>
<td>1: each node is a domain (only of interest in</td>
</tr>
<tr>
<td></td>
<td>“systems” AMG)</td>
</tr>
<tr>
<td></td>
<td>2: each domain is generated by agglomeration</td>
</tr>
<tr>
<td></td>
<td>(default)</td>
</tr>
<tr>
<td>dparms[1]</td>
<td>AMG strength threshold: Default=0.25</td>
</tr>
<tr>
<td>dparms[2]</td>
<td>Truncation factor for the interpolation:</td>
</tr>
<tr>
<td></td>
<td>Default=1e-2</td>
</tr>
<tr>
<td>dparms[3]</td>
<td>Sets a parameter to modify the definition</td>
</tr>
<tr>
<td></td>
<td>of strength for diagonal dominant portions of</td>
</tr>
<tr>
<td></td>
<td>the matrix: Default=0.9</td>
</tr>
<tr>
<td>dparms[3]</td>
<td>Defines a smoothing parameter for the additive</td>
</tr>
<tr>
<td></td>
<td>Schwartz method Default=1.</td>
</tr>
</tbody>
</table>

Table 11.14: Definitions of other entries of iparms and dparms if preconditioner is BOOMER AMG

<table>
<thead>
<tr>
<th>iparms[7]</th>
<th>Row size in Parallel ILUT: Default=1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>iparms[8]</td>
<td>Set maximum number of iterations: Default=30</td>
</tr>
<tr>
<td>dparms[1]</td>
<td>Drop tolerance in Parallel ILUT: Default=1e-5</td>
</tr>
</tbody>
</table>

Table 11.15: Definitions of other entries of iparms and dparms if preconditioner is PILUT
11.4. DOMAIN DECOMPOSITION

<table>
<thead>
<tr>
<th>iparms[7]</th>
<th>Number of levels in Parallel Sparse Approximate inverse: Default=1</th>
</tr>
</thead>
</table>
| 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 Default=0 |
| dparms[1] | Filters parameters: The filter parameter is used to drop small nonzeros in the preconditioner, to reduce the cost of applying the preconditioner: Default=0.1 |
| dparms[2] | Threshold parameter: Default=0.1 |

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) Default=1 |
| iparms[8] | Overlap for the Schwartz method: Default=1 |
| 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

<table>
<thead>
<tr>
<th>n=4 × 10^6</th>
<th>nnz=13 × 10^6</th>
<th>Te=571,29</th>
</tr>
</thead>
<tbody>
<tr>
<td>np</td>
<td>AMG</td>
<td>time</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>1491.83</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>708.49</td>
</tr>
<tr>
<td>32</td>
<td>4</td>
<td>296.22</td>
</tr>
<tr>
<td>64</td>
<td>4</td>
<td>145.64</td>
</tr>
</tbody>
</table>

Table 11.18: Convergence and time for solving linear system from example 11.4
In terms of parallel programming for FreeFem++, with MPI, this means that the user must be able to divide processors available for computation into subgroups of processors and also must be able to realize different type of communications in FreeFem++ script. Here is a wrapper of some MPI functions.

### 11.4.1 Communicators and groups

#### Groups

```cpp
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

```cpp
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
```

```cpp
mpiComm cc(mpiGroup gp): Same as previous constructor but default comm here is MPI.COMM_WORLD.
```

```cpp
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.
```

```cpp
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)

```cpp
1: int color=mpiRank(comm)%2;
2: mpiComm ccc(processor(color,comm),0);
3: mpiComm qpp(comm,);
4: mpiComm cp(cc,color,0);
```

#### Example 11.8 (merge.edp)

```cpp
1: mpiComm comm,cc;
2: int color=mpiRank(comm)%2;
3: int rk=mpiRank(comm);
4: int size=mpiSize(comm);
5: cout << "Color values " << color << endl;
6: mpiComm ccc(processor((rk<size/2),comm),rk);
7: mpiComm cp(cc,color,0);
8: int rleader;
9: if (rk == 0) { rleader = size/2; }
10: else if (rk == size/2) { rleader = 0; }
```
11.4. DOMAIN DECOMPOSITION

10:  else { rleader = 3; }
11:  mpiComm qqq(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++.

11.5 HPDDM solvers

The corresponding examples are in the directory examples++-hpddm. Real valued problems (diffusion, heat, elasticity and Stokes) and complex valued problems (Maxwell and Helmholtz) are given in both 2D and 3D. We detail here the 3D elasticity problem and the 3D time-dependent heat problem.

Example 11.9 (elasticity-3d.edp) A three dimensional elasticity problem is defined. The solver is a domain decomposition method. Domain decomposition methods are a natural framework for parallel computers. The scripts in the directory examples++-hpddm run on multicores computers (from 2 to tens of thousands of cores). Recall that like in any MPI code the number of MPI processes, mpiSize, is given in the command line via the option -np. We focus on the script elasticity-3d.edp but the other scripts have the same structure. The command line to run the example on four processes with ffglut visualization is:

```
ff-mpirun -np 4 elasticity-3d.edp -glut ffglut
```

```cpp
39  int[int] arrayIntersection; // ranks of neighboring subdomains
40  int[int][int] restrictionIntersection(0); // local-to-neighbors renumbering
41  real[int] D; // partition of unity

43  meshN ThGlobal = cube(10 * getARGV("-global", 5), getARGV("-global", 5),
44  getARGV("-global", 5), [10 * x, y, z], label = LL); // global mesh
45  build(Th, ThBorder, ThGlobal, fakeInterface, s, overlap, D, arrayIntersection, restrictionIntersection, Wh, Pi, comm, excluded, 3)
```
11.5. HPDDM SOLVERS

```cpp
real f = -9000.0;
real strain = 100.0;
real Young = 2.0e11; // steel
real poisson = 0.35;
real tmp = 1.0 + poisson;
real mu = Young / (2.0 * tmp);
real lambda = Young * poisson / (tmp * (1.0 - 2.0 * poisson));
```

```cpp
real[int] rhs; // local right-hand side
```

```cpp
matrix<real> Mat; // local operator
```

```cpp
meshN ThAugmented = Th + ThBorder;
varf vPb(def(u), def(v)) = intN(ThAugmented)(lambda * div(u) * div(v) + 2.0 * mu * \(\epsilon(u)^T \epsilon(v)\)) + intN(ThAugmented)(f * vC) + on(1, u = 0.0, uB = 0.0, uC = 0.0);
```

```cpp
fespace WhAugmented(ThAugmented, Pk);
Mat = vPb(WhAugmented, WhAugmented, tgv = -1);
real[int] rhsFull = vPb(0, WhAugmented, tgv = -1);
matrix R = interpolate(Wh, WhAugmented);
```

```cpp
renumbering(Mat, R, rhsFull, rhs);
```

```cpp
ThBorder = cube(1, 1, 1, [x, y, z]);
```

```cpp
dschwarz A(Mat, arrayIntersection, restrictionIntersection, scaling = D);
```

Listing 11.1: ../../../examples++-hpddm/elasticity-3d.edp

The macro `build` is of particular interest since it handles the data distribution among the `mpisize` MPI processes with the following steps:

- The initial mesh `ThGlobal` is partitioned by process 0 into `mpisize` submeshes.
- The partition is broadcasted to every process `i` for `0 < i < mpisize`. From then on, all tasks are parallel.
- Each process creates the local submesh `Th` (if the refinement factor `s` defined via the option `-split` is larger than 1, each local edge is split into `s` subedges, resulting in each element being split into `s^2` elements in 2D and `s^3` elements in 3D) so that the collection of these submeshes is an overlapping domain decomposition of a refined mesh. The number of extra layers added to the initial partition is monitored by the option `-overlap`.
- Connectivity structures are created:
  - `D` is the diagonal of the local partition of unity (see below §11.5.2 for more details)
  - `arrayIntersection` is the list of neighbors of the current subdomain
  - For `j` in `arrayIntersection`, `restrictionIntersection[j]` is the list of the degrees of freedom that belong to the intersection of the current subdomain with its neighbor `j`.

Then, the variational formulation `vPb` of a three dimensional elasticity problem is used to assemble a local matrix `Mat`. This matrix along with `D`, `arrayIntersection` and `restrictionIntersection` are arguments for the constructor of the distributed matrix `A`. This is enough to solve the problem with a one-level additive Schwarz method which can be either ASM or RAS.
For some problems it is interesting to use optimized interface conditions. When there are many subdomains, it is usually profitable to add a second level to the solver. Options are set in the sequel of the script:

```
set(A, sparams = "−hpddm_schwarz_method ras −hpddm_schwarz_coarse_correction balanced ↦ −hpddm_variant right −hpddm_verbosity 1 −hpddm_geneo_nu 10");
```

Listing 11.2: ../examples++-hpddm/elasticity-3d.edp

In the above line, the first option selects the one-level preconditioner ras (possible choices are ras, oras, soras, asm, osm or none), the second option selects the correction formula for the second level here balanced (possible options are deflated, additive or balanced), the third option selects right preconditioning, the fourth one is verbosity level of HPDDM (different from the one of FreeFem++), the fifth one prints all possible options of HPPDM and the last one specifies the number of coarse degrees of freedom per subdomain of the GENEO coarse space. All other options of cheatsheet of the HPDDM [6] library can be selected via the FreeFem++ function set.

In the last part of the script, the global linear system is solved by the domain decomposition method defined above.

```
Wh<real> def(u); // local solution
if(Opt.n > 0) // optimized Schwarz methods
  DDM(A, u[], rhs, excluded = excluded, ret = ret, O = Opt);
else
  u[] = A⁻¹ * rhs;

real[int] err(u[].n);
err = A * u[]; // global matrix−vector product
err -= rhs;

plotMPI(Th, u[], "Global solution", Pk, def, real, 3, 1)
plotMPI(Th, err, "Global residual", Pk, def, real, 3, 1)
real alpha = 2000.0;
meshN ThMoved = movemesh3(Th, transfo = [x + alpha * u, y + alpha * uB, z + alpha * uC]);

u[] = mpirank;
plotMPI(ThMoved, u[], "Global moved solution", Pk, def, real, 3, 1)
```

Listing 11.3: ../examples++-hpddm/elasticity-3d.edp

11.5.1 Time dependent problem

Example 11.10 (heat-3d.edp) A three dimensional heat problem

\[
\frac{\partial u}{\partial t} - \Delta u = 1, \quad u(0, \cdot) := 0 \text{ in } \Omega.
\]

is discretized by an implicit Euler scheme. At each time step n, we shall seek \(u^n(x, y, z)\) satisfying for all \(w \in H^1(\Omega)\):

\[
\int_{\Omega} u^n - u^{n-1} \frac{\delta t}{\partial t} w + \nabla u^n \nabla w = \int_{\Omega} w, \quad u^0 := 0 \text{ in } \Omega.
\]
So that at each time step a linear system

\[(M + dt \ast K)u^n \] \[= M \ast u^{n-1} + \delta t \ast F\]

is solved by a domain decomposition method where \(M\) is the mass matrix and \(K\) is the rigidity matrix. In order to save computational efforts, the domain decomposition method preconditioner is built only once and then reused for all subsequent solves with matrix \(A := M + dt \ast K\). The distributed matrix vector product with matrix \(M\) is made through the call to the function \(dmv\) using the partition of unity associated to matrix \(A\).

```
set(A, sparsms = "-hpddm_reuse.preconditioner=1");
Wh<real> def(u) = init(0.0); // local solution
for(int i = 0; i < iMax; ++i) {
    real[int] newRhs(rhs.n);
    dmv(A, M, u[], newRhs); // newRhs = M \ast u
    newRhs += rhs;
    if(Opt.n > 0) // optimized Schwarz methods
        DDM(A, u[], newRhs, excluded = excluded, ret = ret, O = Opt);
    else
        u[] = A^{-1} \ast newRhs;
}
plotMPI(Th, u[], "Global solution", Pk, def, real, 3, 0)
```

Listing 11.4: ./examples++-hpddm/heat-3d.edp

### 11.5.2 Distributed vectors in HPDDM

We give here some hints on the way vectors are distributed among \(np\) processes when using FreeFem++ interfaced with HPDDM. The set of degrees of freedom \(\mathcal{N}\) is decomposed into \(np\) overlapping sets \((\mathcal{N}_i)_{1 \leq i \leq np}\). A MPI-process is in charge of each subset. Let \(n := \#\mathcal{N}\) be the number of degrees of freedom of the global finite element space. Let \(R_i\) denote the restriction operator from \(\mathbb{R}^n\) onto \(\mathbb{R}^{\#\mathcal{N}_i}\). We have also defined local diagonal matrices \(D_i \in \mathbb{R}^{\#\mathcal{N}_i \times \#\mathcal{N}_i}\) so that we have a partition of unity at the algebraic level:

\[
U = \sum_{i=1}^{np} R_i^T D_i R_i U \quad \forall U \in \mathbb{R}^n.
\] (11.1)

A global vector \(U \in \mathbb{R}^n\) is actually not stored. Rather, it is stored in a distributed way. Each process \(i, 1 \leq i \leq N\), stores the local vector \(U_i := R_i U \in \mathbb{R}^{\#\mathcal{N}_i}\).

It is important to ensure that the result of all linear algebra operators applied to this representation are coherent.

As an example, consider the scalar product of two distributed vectors \(U, V \in \mathbb{R}^n\). Using the partition of unity [11.1], we have:

\[
(U, V) = \left(U, \sum_{i=1}^{np} R_i^T D_i R_i V\right) = \sum_{i=1}^{np} (R_i U, D_i R_i V) \\
= \sum_{i=1}^{np} (U_i, D_i V_i).
\]
Thus, the formula for the scalar product is:

$$(U, V) = \sum_{i=1}^{np} (R_i U, D_i R_i V).$$

Local scalar products are performed concurrently. Thus, the implementation is parallel except for the sum which corresponds to a \texttt{MPI Reduce} call across the $np$ MPI processes. Note also that the implementation relies on the knowledge of a partition of unity so that the FreeFem++ syntax is \texttt{dscalprod(D, u, v)}.

A \texttt{axpy} procedure $y \leftarrow \alpha x + y$ for $x, y \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$ is easily implemented concurrently for distributed vectors in the form:

$$y_i \leftarrow \alpha x_i + y_i, \forall 1 \leq i \leq np.$$ 

The matrix vector product is more involved and details are given in the SIAM book An Introduction to Domain Decomposition Methods: algorithms, theory and parallel implementation and even more details are given in P. Jolivet’s PhD manuscr.
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
  \((\mathbb{R}^j, j=1,\dim), (I) \, \text{Ref}φ^v_i, \, i=1,\text{NbOfVertices}\)
- Edges (I) NbOfEdges
  \(@\text{Vertex}_1^i, @\text{Vertex}_2^i, (I) \, \text{Ref}φ^e_i, \, i=1,\text{NbOfEdges}\)
- Triangles (I) NbOfTriangles
  \((@\text{Vertex}_1^j, j=1,3), (I) \, \text{Ref}φ^t_i, \, i=1,\text{NbOfTriangles}\)
- Quadrilaterals (I) NbOfQuadrilaterals
  \((@\text{Vertex}_1^j, j=1,4), (I) \, \text{Ref}φ^q_i, \, i=1,\text{NbOfQuadrilaterals}\)
- Geometry
  \((C^*) \, \text{FileNameOfGeometricSupport}\)

  - VertexOnGeometricVertex
    \((I) \, \text{NbOfVertexOnGeometricVertex}\)
    \(@\text{Vertex}_i, @\text{Vertex}_geo^i, \, i=1,\text{NbOfVertexOnGeometricVertex}\)
  - EdgeOnGeometricEdge
    \((I) \, \text{NbOfEdgeOnGeometricEdge}\)
    \(@\text{Edge}_i, @\text{Edge}_geo^i, \, i=1,\text{NbOfEdgeOnGeometricEdge}\)
  - CrackedEdges (I) NbOfCrackedEdges
    \(@\text{Edge}_1^i, @\text{Edge}_2^i, \, i=1,\text{NbOfCrackedEdges}\)
When the current mesh refers to a previous mesh, we have in addition

- **MeshSupportOfVertices**
  - (C*) FileNameOfMeshSupport

  - **VertexOnSupportVertex**
    - (I) NbOfVertexOnSupportVertex
    - (@@Vertex$_i$, @@Vertex$_i^{supp}$, $i=1$, NbOfVertexOnSupportVertex)

- **VertexOnSupportEdge**
  - (I) NbOfVertexOnSupportEdge
  - (@@Vertex$_i$, @@Edge$_i^{supp}$, (R) $u_i^{supp}$, $i=1$, NbOfVertexOnSupportVertex)

- **VertexOnSupportTriangle**
  - (I) NbOfVertexOnSupportTriangle
  - (@@Vertex$_i$, @@Tria$_i^{supp}$, (R) $u_i^{supp}$, (R) $v_i^{supp}$, $i=1$, NbOfVertexOnSupportTriangle)

- **VertexOnSupportQuadrilaterals**
  - (I) NbOfVertexOnSupportQuadrilaterals
  - (@@Vertex$_i$, @@Quad$_i^{supp}$, (R) $u_i^{supp}$, (R) $v_i^{supp}$, (R) $w_i^{supp}$, $i=1$, NbOfVertexOnSupportQuadrilaterals)

### 12.2 bb File type for Store Solutions

The file is formatted such that:

```
2 nbsol nbv 2
((U$_{ij}$, $\forall i \in \{1, ..., nbsol\}$, $\forall j \in \{1, ..., nbv\})
```

where

- **nbsol** is an integer equal to the number of solutions.
- **nbv** is an 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$^1$ ... typesol$^n$ nbv 2
((U$_{ij}^k$, $\forall i \in \{1, ..., typesol^k\}$, $\forall k \in \{1, ..., n\}$, $\forall j \in \{1, ..., nbv\})
```

where

- **n** is a integer equal to the number of solutions
- **typesol$^k$**, type of the solution number $k$, is
12.4 Metric File

A metric file can be of two types, isotropic or anisotropic. The isotropic file is such that

\( \text{nbv} \) 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 \( M_i = h_i^{-2} \text{Id} \), where \( \text{Id} \) is the identity matrix.

The metric anisotropic

\( \text{nbv} \) is a integer equal to the number of vertices,

\( a_{11}, a_{21}, a_{22} \) is metric \( M_i = \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix} \) 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 \( |u|/\sqrt{u \cdot M_i u} \), 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.
c(1:2,nbv) is a real array giving the two coordinates of each vertex.
refs(nbv) is an integer array giving the reference numbers of the vertices.
reft(nbv) is an integer array giving the reference numbers of the triangles.

**AM_FMT Files**  In fortran the am_fmt files are read as follows:

```
open(1, file='xxx.am_fmt', form='formatted', status='old')
read (1,*) nbv, nbt
read (1,*) ((nu(i,j),i=1,3),j=1,nbt),
read (1,*) ((c(i,j),i=1,2),j=1,nbv),
read (1,*) (reft(i),i=1,nbt),
read (1,*) (refs(i),i=1,nbv)
close(1)
```

**AM Files**  In fortran the am files are read as follows:

```
open(1, file='xxx.am', form='unformatted', status='old')
read (1,*) nbv, nbt
read (1) ((nu(i,j),i=1,3),j=1,nbt),
& ((c(i,j),i=1,2),j=1,nbv),
& (reft(i),i=1,nbt),
& (refs(i),i=1,nbv)
close(1)
```

**AMDBA Files**  In fortran the amdba files are read as follows:

```
open(1, file='xxx.amdba', form='formatted', status='old')
read (1,*) nbv, nbt
read (1,*) (k,(c(i,k),i=1,2),refs(k),j=1,nbv)
read (1,*) (k,(nu(i,k),i=1,3),reft(k),j=1,nbt)
close(1)
```

**msh Files**  First, we add the notions of boundary edges

nbbe is the number of boundary edge.
nube(1:2,1:nbbe) is an integer array giving the two vertex numbers
refbe(1:nbbe) is an integer array giving the two vertex numbers

In fortran the msh files are read as follows:

```
open(1, file='xxx.msh', form='formatted', status='old')
read (1,*) nbv, nbt, nbbe
read (1,*) ((c(i,k),i=1,2),refs(k),j=1,nbv)
read (1,*) ((nu(i,k),i=1,3),reft(k),j=1,nbt)
read (1,*) ((ne(i,k),i=1,2), refbe(k),j=1,nbbe)
close(1)
```
ftq Files  In fortran the ftq files are read as follows:

```fortran
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 element \( j \) is a quadrilateral.
Chapter 13

Addition of a new finite element

13.1 Some notations

For a function $f$ taking value in $\mathbb{R}^N$, $N = 1, 2, \cdots$, 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 $\omega^K_i$ ($i = 0, \cdots, NbDoF - 1$) of the finite element space has the $j$-th component $\omega^K_j$ for $j = 0, \cdots, N - 1$.

The operator $\Pi_h$ is called the interpolator of the finite element. We have the identity $\omega^K_i = \Pi_h \omega^K_i$.

Formally, the interpolator $\Pi_h$ is constructed by the following formula:

$$\Pi_h f = \sum_{k=0}^{\kappa Pi-1} \alpha_k f_{j_k}(P_{p_k}) \omega^K_{i_k} \quad (13.1)$$

where $P_{p_k}$ 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: with the classical scalar Lagrange finite element, we have $\kappa Pi = npPi = NbOfNode$ and

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

Example 2: The Raviart-Thomas finite element:

$$RT0_h = \{ v \in H(div) / \forall K \in T_h \quad v|_K(x, y) = |^{\alpha_K}_{\beta_K} \gamma_K |^{\tilde{y}} \} \quad (13.2)$$

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

To compute this flux, we use a quadrature formula with one point, the middle point of the edge. Consider a triangle $T$ with three vertices $(a, b, c)$. Let denote the vertices numbers by $i_a, i_b, i_c$, and define the three edge vectors $e^0, e^1, e^2$ by $sgn(i_b - i_c)(b - c), sgn(i_c - i_a)(c - a), sgn(i_a - i_b)(a - b)$.

The three basis functions are:

$$\omega^K_0 = sgn(i_b - i_c)2|T|(x - a), \quad \omega^K_1 = sgn(i_c - i_a)2|T|(x - b), \quad \omega^K_2 = sgn(i_a - i_b)2|T|(x - c), \quad (13.3)$$
where $|T|$ is the area of the triangle $T$.

So we have $N = 2$, $k\Pi = 6$; $np\Pi = 3$; and:

- $P_p = \{ \begin{array}{c} b+c \frac{a+c}{2}, b+a \frac{a+c}{2} \end{array} \}$
- $\alpha_0 = -e_0^0$, $\alpha_1 = e_0^1$, $\alpha_2 = -e_2^1$, $\alpha_3 = e_1^1$, $\alpha_4 = -e_2^2$, $\alpha_5 = e_1^2$ (effectively, the vector $(-e_2^m,e_1^m)$ is orthogonal to the edge $e^m = (e_1^m,e_2^m)$ with a length equal to the side of the edge or equal to $\int_{e^m} 1$).
- $i_k = \{0,0,1,1,2,2\}$.
- $p_k = \{0,0,1,1,2,2\}$, $j_k = \{0,1,0,1,0,1,1\}$.

### 13.2 Which class to add?

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

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

namespace Fem2D {

    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 $k\Pi$ of coef to build the interpolator [13.1]
                    3, // number $np\Pi$ 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 $\bar{K}$
                    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 \( \omega^K_i \), this function return the value of all the basis function
or this derivatives in array val, computed at point \( \hat{P} \) on the reference triangle corresponding
to point \( R^2 \) \( P=K(\hat{P}) \); on the current triangle \( K \).
The index \( i,j,k \) of the array \( val(i,j,k) \) corresponding to:

\( i \) is basic function number on finite element \( i \in [0, NoF] \)
\( j \) is the value of component \( j \in [0, N[ \)
\( k \) is the type of computed value \( f(P), dx(f)(P), dy(f)(P), ... i \in [0, last_operatortype] \). Remark
for optimization, this value is computed only if \( whatd[k] \) is true, and the numbering is defined
with

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

const int last_operatortype=10;

The shape function:

void TypeOfFE_RTortho::FB(const bool *whatd, const Mesh & Th, const Triangle & K,
    const R2 & PHat, RNMK_ & val) const
{
    //
    R2 P(K(PHat));
    R2 A(K[0]), B(K[1]), C(K[2]);
    R 10=1-P.x-P.y,11=P.x,12=P.y;
    assert(val.N() >=3);
    assert(val.M()==2 );
val=0;
R a=1./(2*K.area);
R a0= K.EdgeOrientation(0) * a ;
R a1= K.EdgeOrientation(1) * a ;
R a2= K.EdgeOrientation(2) * a ;

if (whatd[op_id])   // value of the function
{
  assert(val.K()>op_id);
  RN_ f0(val('.',0,0));    // value first component
  RN_ f1(val('.',1,0));    // value second component
  f1[0] = (P.x-A.x)*a0;
  f0[0] = -(P.y-A.y)*a0;
  f1[1] = (P.x-B.x)*a1;
  f0[1] = -(P.y-B.y)*a1;
  f1[2] = (P.x-C.x)*a2;
  f0[2] = -(P.y-C.y)*a2;
}

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

if (whatd[op_dy])
{
  assert(val.K()>op_dy);
  val(0,0,op_dy) = -a0;
  val(1,0,op_dy) = -a1;
  val(2,0,op_dy) = -a2;
}

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

The function to defined the coefficient $\alpha_k$:

void TypeOfFE_RT::Pi_h_alpha(const baseFElement & K, KN_<double> & v) const
{
  const Triangle & T(K.T);
  for (int i=0,k=0;i<3;i++)
  {
    R2 E(T.Edge(i));
    R signe = T.EdgeOrientation(i);
    v[k++]= signe*E.y;
    v[k++]= -signe*E.x;
  }
}
13.2. WHICH CLASS TO ADD?

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:

```cpp
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

```cpp
static TypeOfFE_RTortho The_TypeOfFE_RTortho; //
static ListOfTFE typefemRTOrtho("RT0Ortho", & The_TypeOfFE_RTortho); //
// let the 2 globals variables
static TypeOfFE_RTortho The_TypeOfFE_RTortho; //
static ListOfTFE typefemRTOrtho("RT0Ortho", & The_TypeOfFE_RTortho); //
// link with FreeFem++ do not work with static library .a
// FH so add a extern name to call in init_static_FE
// (see end of FESpace.cpp)
void init_FE_ADD() { }
// --- end --
} // FEM2d namespace

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

```cpp
// correct Problem of static library link with new make file
void init_static_FE()
{
    extern void init_FE_P2h() ;
    init_FE_P2h() ;
    extern void init_FE_ADD() ;
    init_FE_ADD() ;
    // new line 1
    // new line 2
}
```

and now you have to change the makefile. First, create a file FE_ADD.cpp containing all this code, like in file src/femlib/Element_P2h.cpp, after modifying the Makefile.am by adding the name of your file to the variable EXTRA_DIST like:

```sh
# Makefile using Automake + Autoconf
# ----------------------------------
# $Id$
#
# This is not compiled as a separate library because its
# interconnections with other libraries have not been solved.
```
and do in the freefem++ root directory

    autoreconf
    ./reconfigure
    make

For codewarrior compilation add the file in the project an remove the flag in panal PPC linker FreeFEm++ Setting Dead-strip Static Initialization Code Flag.
Appendix A

Table of Notations

Here mathematical expressions and corresponding FreeFem++ commands are explained.

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 ($a \in \mathbb{N} \iff \text{int } a$); “int” means long integer inside FreeFem++

$\mathbb{R}$ the set of real numbers ($a \in \mathbb{R} \iff \text{real } a$); double inside FreeFem++

$\mathbb{C}$ the set of complex numbers ($a \in \mathbb{C} \iff \text{complex } a$); 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 $E$ ($E^2 = E \times E$, $E^n = E \times E^{n-1}$)

$f : E \to F$ 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., $I(x) = x \ \forall x \in E$

$f \circ g$ for $f : F \to G$ and $g : E \to F$, $E \ni x \mapsto (f \circ g)(x) = f(g(x)) \in G$ (see Section ??)

$f|_A$ the restriction of $f : E \to F$ to the subset $A$ of $E$

$\{a_k\}$ column vector with components $a_k$

$(a_k)$ row vector with components $a_k$

$(a_k)^T$ denotes the transpose of a matrix $(a_k)$, and is $\{a_k\}$

$\{a_{ij}\}$ matrix with components $a_{ij}$, and $(a_{ij})^T = (a_{ji})$

### A.3 Numbers

For two real numbers $a, b$

$[a, b]$ is the interval $\{x \in \mathbb{R} | a \leq x \leq b\}$

$]a, b]$ is the interval $\{x \in \mathbb{R} | a < x \leq b\}$

$[a, b]$ is the interval $\{x \in \mathbb{R} | a \leq x < b\}$

$]a, b]$ is the interval $\{x \in \mathbb{R} | a < x < b\}$

### A.4 Differential Calculus

$\partial f / \partial x$ the partial derivative of $f : \mathbb{R}^d \to \mathbb{R}$ with respect to $x$ ( $\text{d}x$ (ε) )

$\nabla f$ the gradient of $f : \Omega \to \mathbb{R}$, i.e., $\nabla f = (\partial f / \partial x, \partial f / \partial y)$

$\text{div} \ f$ or $\nabla \cdot f$ the divergence of $f : \Omega \to \mathbb{R}^d$, i.e., $\text{div} \ f = \partial f_1 / \partial x + \partial f_2 / \partial y$

$\Delta f$ the Laplacian of $f : \Omega \to \mathbb{R}$, i.e., $\Delta f = \partial^2 f / \partial x^2 + \partial^2 f / \partial y^2$
A.5 Meshes

Ω usually denotes a domain on which PDE is defined

Γ denotes the boundary of Ω, i.e., \( \Gamma = \partial \Omega \) (keyword \texttt{border}, see Section 5.1.2)

\( \mathcal{T}_h \) the triangulation of Ω, i.e., the set of triangles \( T_k \), where \( h \) stands for mesh size (keyword \texttt{mesh}, \texttt{buildmesh}, see Section 5)

\( n_t \) the number of triangles in \( \mathcal{T}_h \) (get by \texttt{Th.nt}, see “mesh.edp”)

\( \Omega_h \) denotes the approximated domain \( \Omega_h = \bigcup_{k=1}^{n_t} T_k \) of Ω. If Ω 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 \texttt{Th.nv})

\( n_{be} \) the number of boundary element in \( \mathcal{T}_h \) (get by \texttt{Th.nbe})

\( |\Omega_h| \) the measure (area or volume) in \( \mathcal{T}_h \) (get by \texttt{Th.measure})

\( |\partial \Omega_h| \) the measure of the border (length or area) in \( \mathcal{T}_h \) (get by \texttt{Th.bordermeasure})

\( h_{\min} \) the minimal edge size of \( \mathcal{T}_h \) (get by \texttt{Th.hmin})

\( h_{\max} \) the maximal edge size of \( \mathcal{T}_h \) (get by \texttt{Th.hmax})

\([q^i,q^j] \) 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 \texttt{(Th[k-1][j-1].x, Th[k-1][j-1].y)})

\( I_\Omega \) the set \( \{i \in \mathbb{N} | q^i \notin \Gamma_h \} \)

A.6 Finite Element Spaces

\( L^2(\Omega) \) the set \( \left\{ w(x,y) \left| \int_{\Omega} |w(x,y)|^2 \, dx \, dy < \infty \right. \right\} \)

norm: \( \|w\|_{0,\Omega} = \left( \int_{\Omega} |w(x,y)|^2 \, dx \, dy \right)^{1/2} \)

scalar product: \( (v,w) = \int_{\Omega} v \, w \)

\( H^1(\Omega) \) the set \( \left\{ w \in L^2(\Omega) \left| \int_{\Omega} (|\partial w/\partial x|^2 + |\partial w/\partial y|^2) \, dx \, dy < \infty \right. \right\} \)

norm: \( \|w\|_{1,\Omega} = \left( \|w\|_{0,\Omega}^2 + \|\nabla w\|_{0,\Omega}^2 \right)^{1/2} \)
\( H^m(\Omega) \) the set \( \{ w \in L^2(\Omega) \mid \int_{\Omega} \frac{\partial^{|\alpha|} w}{\partial x^{\alpha_1} \partial y^{\alpha_2}} \in L^2(\Omega) \quad \forall \alpha = (\alpha_1, \alpha_2) \in \mathbb{N}^2, |\alpha| = \alpha_1 + \alpha_2 \} \)

scalar product: \( (v, w)_{1,\Omega} = \sum_{|\alpha| \leq m} \int_{\Omega} D^\alpha v D^\alpha w \)

\( H^1_0(\Omega) \) 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 “\texttt{fespace Vh(Th,*)}” in FreeFem++ (see Section 6 for “\texttt{*}”)

\( \Pi_h f \) the projection of the function \( f \) into \( V_h \) (“\texttt{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 \( (v_1, \cdots, v_M)^T \) if \( v = v_1 \phi_1 + \cdots + v_M \phi_M \), which is shown by “\texttt{fespace Vh(Th,P2); Vh v; cout << v[] << endl;}”
Appendix B

Grammar

B.1 The bison grammar

start: input ENDOFILE;

input: instructions ;

instructions: instruction
           | instructions instruction ;

list_of_id_args:
    | id
    | id '=' no_comma_expr
    | FSPACE 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 ',' FSPACE 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 | FSPACE;

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

parameters_list:
    no_set_expr
    | FSPACE ID
    | ID '=' no_set_expr

361
APPENDIX B. GRAMMAR

| 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 ;

defspace_def: ID '(' parameters_list ')' ;

defspace_def_list: defspace_def
            | defspace_def_list ',', defspace_def ;

declaration:  type_of_dcl list_of_dcls ';
                | 'fespace' defspace_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
B.1. THE BISON GRAMMAR

| 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_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_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

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

sub_script_expr:
  no_set_expr
| ':
| no_set_expr ':=' no_set_expr
| no_set_expr ':=' no_set_expr ':=' no_set_expr

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

array: no_comma_expr
  | array ',' no_comma_expr

unary_expr:
pow_expr
  | unop pow_expr $prec UNARY

pow_expr: primary
  | primary '' unary_expr
  | primary '_' unary_expr
  | primary ''  // transpose

primary:
  ID
  | LNUM
  | DNUM
  | CNUM
  | STRING
  | primary '(' parameters ')''
  | primary '[' Expr ']
  | primary '[' ']''
  | primary '.' ID
  | primary().' ID
  | primary '++'
  | primary '--'
  | TYPE '(' Expr ')' '
  | '(' Expr ')''
  | '[' array ']';
B.2 The Types of the languages, and cast

B.3 All the operators

- CG, type :<TypeSolveMat>
- Cholesky, type :<TypeSolveMat>
- Crout, type :<TypeSolveMat>
- GMRES, type :<TypeSolveMat>
- LU, type :<TypeSolveMat>
- LinearCG, type :<Polymorphic> operator() :
  ( <long> : <Polymorphic>, <KN<double> *>, <KN<double> *> )

- N, type :<Fem2D::R3>
- NoUseOfWait, type :<bool *>
- P, type :<Fem2D::R3>
- P0, type :<Fem2D::TypeOfFE>
- P1, type :<Fem2D::TypeOfFE>
- P1nc, type :<Fem2D::TypeOfFE>
- P2, type :<Fem2D::TypeOfFE>
- RT0, type :<Fem2D::TypeOfFE>
- RTmodif, type :<Fem2D::TypeOfFE>
- abs, type :<Polymorphic> operator() :
  ( <double> : <double> )

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

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

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

- append, type :<std::ios_base::openmode>
- asin, type :<Polymorphic> operator() :
  ( <double> : <double> )

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

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

- atan2, type :<Polymorphic> operator() :
  ( <double> : <double>, <double> )

- atanh, type :<Polymorphic> operator() :
  ( <double> : <double> )
- buildmesh, type :<Polymorphic> operator() :
  ( <Fem2D::Mesh> : <E_BorderN> )
- buildmeshborder, type :<Polymorphic> operator() :
  ( <Fem2D::Mesh> : <E_BorderN> )
- cin, type :<istream>
- clock, type :<Polymorphic>
  ( <double> : )
- conj, type :<Polymorphic> operator() :
  ( <complex> : <complex> )
- convect, type :<Polymorphic> operator() :
  ( <double> : <E_Array>, <double>, <double> )
- cos, type :<Polymorphic> operator() :
  ( <double> : <double> )
  ( <complex> : <complex> )
- cosh, type :<Polymorphic> operator() :
  ( <double> : <double> )
  ( <complex> : <complex> )
- cout, type :<ostream>
- dumptable, type :<Polymorphic> operator() :
  ( <ostream> : <ostream> )
- dx, type :<Polymorphic> operator() :
  ( <LinearComb<MDroit, C_F0>> : <LinearComb<MDroit, C_F0>> )
  ( <double> : std::pair<FEbase<double> *, int>> )
  ( <LinearComb<MGauche, C_F0>> : <LinearComb<MGauche, C_F0>> )
- dy, type :<Polymorphic> operator() :
  ( <LinearComb<MDroit, C_F0>> : <LinearComb<MDroit, C_F0>> )
  ( <double> : std::pair<FEbase<double> *, int>> )
  ( <LinearComb<MGauche, C_F0>> : <LinearComb<MGauche, C_F0>> )
- endl, type :<char>
- exec, type :<Polymorphic> operator() :
  ( <long> : <string> )
- exit, type :<Polymorphic> operator() :
  ( <long> : <long> )
- exp, type :<Polymorphic> operator() :
  ( <double> : <double> )
  ( <complex> : <complex> )
- false, type :<bool>
- imag, type :<Polymorphic> operator() :
  ( <double> : <complex> )

- int1d, type :<Polymorphic> operator() :
  ( <CDomainOfIntegration> : <Fem2D::Mesh>... )

- int2d, type :<Polymorphic> operator() :
  ( <CDomainOfIntegration> : <Fem2D::Mesh>... )

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- `square`, type :<Polymorphic> operator() :
  (  <Fem2D::Mesh> :  <long>, <long> )
B.3. ALL THE OPERATORS

( <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 functionalities in FreeFem++ under the three environments 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. You will 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 that 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 will be loaded so it must be either 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 will be loaded so it 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 known if you try to compile for a pure Windows environment or for a cygwin environment so to build the load module under cygwin you must add the -cygwin parameter.

C.1 A first example myfunction.cpp

The following defines a new function call myfunction with no parameter, but using the $x,y$ current value.
#include <iostream>
#include <cfloat>
using namespace std;
#include "error.hpp"
#include "AFunction.hpp"
#include "rgraph.hpp"
#include "RNM.hpp"
#include "fem.hpp"
#include "FESpace.hpp"
#include "MeshPoint.hpp"

using namespace Fem2D;

double myfunction(Stack stack)
{
    // to get FreeFem++ data
    MeshPoint &mp = *MeshPointStack(stack); // the struct to get x,y, normal , value
    double x = mp.P.x; // get the current x value
    double y = mp.P.y; // get the current y value
    // cout << "x = " << x << " y=" << y << endl;
    return sin(x)*cos(y);
}

Now the Problem is to build the link with FreeFem++ , to do that we need two classes, one to call the function myfunction
All FreeFem++ evaluable expression must be a struct/class C++ which derive from E_F0.
By default this expression does not depend of the mesh position, but if they derive from E_F0mps the expression depends of the mesh position, and for more details see [17].

    // A class build the link with FreeFem++
    // generaly 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 fntion 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
C.1. A FIRST EXAMPLE MYFUNCTION.CPP

```cpp
// 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:

```cpp
void init(){
    Global.Add("myfunction", "(" , new OneOperator0s<double>(myfunction));
}
LOADFUNC(init);
```

It will be called automatically at load module time.

To compile and link, use the ff-c++ script:

```bash
% 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

```bash
% FreeFem++-nw load.edp
```

---

Under Windows, launch FreeFem++ with the mouse (or ctrl O) on the example.
C.2 Example: Discrete Fast Fourier Transform

This will add FFT to FreeFem++ , taken from [http://www.fftw.org/](http://www.fftw.org/). To download and install under download/include just go in download/fftw and trymake.

The 1D fft (fast discret fourier transform) for a simple array \( f \) of size \( n \) is defined by the following formula

\[
\text{dfft}(f, \varepsilon)_k = \sum_{j=0}^{n-1} f_j \varepsilon^{2\pi ij k/n}
\]

The 2D DFFT for an array of size \( N = n \times m \) is

\[
\text{dfft}(f, m, \varepsilon)_{k+l} = \sum_{j'=0}^{m-1} \sum_{j=0}^{n-1} f_i + nj \varepsilon^{2\pi i(jk/n + lj'/m)}
\]

Remark: the value \( n \) is given by \( \text{size}(f)/m \), and the numbering is row-major order.

So the classical discrete DFFT is \( \hat{f} = \text{dfft}(f, -1)/\sqrt{n} \) and the reverse dFFT \( f = \text{dfft}(\hat{f}, 1)/\sqrt{n} \)

Remark: the 2D Laplace operator is

\[
f(x, y) = 1/\sqrt{N} \sum_{j'=0}^{m-1} \sum_{j=0}^{n-1} \hat{f}_i + nj \varepsilon^{2\pi i(xj+yj')}
\]

and we have

\[
f_{k+l} = f(k/n, l/m)
\]

So

\[
\Delta \hat{f}_{kl} = -((2\pi)^2((\hat{k})^2 + (\hat{l})^2))\hat{f}_{kl}
\]

where \( \hat{k} = k \) if \( k \leq n/2 \) else \( \hat{k} = k - n \) and \( \hat{l} = l \) if \( l \leq m/2 \) else \( \hat{l} = l - m \).

And to have a real function we need all modes to be symmetric around zero, so \( n \) and \( m \) must be odd.

Compile to build a new library

```bash
% ff-c++ dfft.cpp ../download/install/lib/libfftw3.a -I../download/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,

```
file : dfft.edp
Load: lg_fem cadna lg_mesh eigenvalue UMFPACK
1 : // Example of dynamic function load
2 : // -------------------------------
3 : // Id: freefem ++ doc.tex,v.1.1102010/06/0411:27:2HechtExp
4 : // Discret Fast Fourier Transform
5 : // -------------------------------
6 : load "dfft" load: init dfft
```
C.2. EXAMPLE: DISCRETE FAST FOURIER TRANSFORM

```cpp
load: dlopen(dfft.dylib) = 0x2b0c700

7: int nx=32, ny=16, N=nx*ny;
9: // warning the Fourier space is not exactly the unite square due to periodic conditions
10: mesh Th=square(nx-1, ny-1, [(nx-1)*x/nx, (ny-1)*y/ny]);
11: // waring the numbering is of the vertices (x,y) is given by i=x/nx+nx*y/ny
12: mesh Th=

14: fespace Vh(Th, P1);
16: func f1 = cos(2*x*2*pi)*cos(3*y*2*pi);
17: Vh<complex> u=f1,v;
18: Vh w=f1;
19: 
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: 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));// to remove div / 0
43: fhat[] = fhat[]./ wij[]; //
45: u[] = dfft(fhat[],ny,1);
46: u[] /= complex(N);
47: ur = real(u);
48: w = real(ue); // the 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
APPENDIX C. DYNAMICAL LINK

-- 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

```cpp
#include <iostream>
#include <cfloat>
#include <cmath>
using namespace std;
#include "error.hpp"
#include "AFunction.hpp"
#include "rgraph.hpp"
#include "RNM.hpp"
#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;
```

// Implementation of P1-P0 FVM-FEM

```cpp
#include <iostream>
#include <cfloat>
#include <cmath>
using namespace std;
#include "error.hpp"
#include "AFunction.hpp"
#include "rgraph.hpp"
#include "RNM.hpp"
#include "MatriceCreuse_tpl.hpp"
#include "MeshPoint.hpp"
#include "lgfem.hpp"
#include "lgsolver.hpp"
#include "problem.hpp"
```

```cpp
class MatrixUpWind0 : public E_F0mps {
public:
    typedef Matrice_Creuse<R> * Result;
```
Expression emat, expTh, expc, expu1, expu2;
MatrixUpWind0(const basicAC_F0 & args)
{
    args.SetNameParam();
    // the matrix expression
    emat = args[0];
    expTh = to<pmesh>(args[1]); // a the expression to get the mesh
    expc = CastTo<double>(args[2]); // the expression to get c (must be a double)
    // a array expression [a, b]
    // a array expression [a, b]
    const E_Array * a = dynamic_cast<const E_Array*>(Expression(args[3]));
    if (a->size() != 2) CompileError("syntax: MatrixUpWind0(Th, rhi, [u1,u2])");
    int err = 0;
    expu1 = CastTo<double>((*a)[0]); // fist exp of the array (must be a double)
    expu2 = CastTo<double>((*a)[1]); // second exp of the array (must be a double)
}

~MatrixUpWind0();

static ArrayOfaType typeargs()
{
    return ArrayOfaType(atype<Matrice_Creuse<R>>(),
atype<pmesh>(), atype<double>(), atype<E_Array>());
}

static E_F0 * f(const basicAC_F0 & args) { return new MatrixUpWind0(args);}

AnyType operator()(Stack s) const;

};

int fvmP1P0(double q[3][2], double u[2], double c[3], double a[3][3], double where[3])
{
    // computes matrix a on a triangle for the Dervieux FVM
    for(int i=0; i<3; i++) for(int j=0; j<3; j++) a[i][j]=0;

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

AnyType MatrixUpWind0::operator()(Stack stack) const
{
    Matrice_Creuse<R> * sparse_mat = GetAny<Matrice_Creuse<R>>(*(emat)(stack));
    MatriceMorse<R> * amorse = 0;
    MeshPoint * mp = MeshPointStack(stack), mps = mp;
}
Mesh * pTh = GetAny<pmesh>((*expTh)(stack));
ffassert(pTh);
Mesh & Th (*pTh);
{
  map< pair<int,int>, R> Aij;
  KN<double> cc(Th.nv);
  double infini=DBL_MAX;
  cc=infini;
  for (int it=0;it<Th.nt;it++)
    for (int iv=0;iv<3;iv++)
    {
      int i=Th(it,iv);
      if ( cc[i]==infini) { // if nuset the set
        mp->setP(&Th,it,iv);
        cc[i]=GetAny<double>((*expc)(stack));
      }
    }

    for (int k=0;k<Th.nt;k++)
    {
      const Triangle & K(Th[k]);
      const Vertex & A(K[0]), &B(K[1]),&C(K[2]);
      R2 Pt(1./3.,1./3.);
      R u[2];
      MeshPointStack(stack)->set(Th,K(Pt),Pt,K,K.lab);
      u[0] = GetAny< R>( (*expu1)(stack) ) ;
      u[1] = GetAny< R>( (*expu2)(stack) ) ;

      int ii[3] ={ Th(A), Th(B),Th(C)};
      double q[3][2]= { { A.x,A.y} ,{B.x,B.y},{C.x,C.y} } ; // coordinates of 3 vertices (input)
      double c[3]={cc[ii[0]],cc[ii[1]],cc[ii[2]]};
      double a[3][3], where[3]={A.lab,B.lab,C.lab};
      if (fvmP1P0(q,u,c,a,where) )
      {
        for (int i=0;i<3;i++)
          for (int j=0;j<3;j++)
            if (fabs(a[i][j]) >= 1e-30)
              { Aij[make_pair(ii[i],ii[j])]+=a[i][j];
            }
        }
    amorse= new MatriceMorse<R>(Th.nv,Th.nv,Aij,false);
}

sparse_mat->pUh=0;
sparse_mat->pVh=0;
sparse_mat->A.master(amorse);
sparse_mat->typemat=(amorse->n == amorse->m) ? TypeSolveMat(TypeSolveMat::GMRES) :
                      TypeSolveMat(TypeSolveMat::NONEQUARE); // none square matrice (morse)
*mmp=mps;

if(verbosity>3) { cout << " End Build MatrixUpWind : " << endl;}

return sparse_mat;
C.4. More on Adding a New Finite Element

First read the section 13 of the appendix, we add two new finite elements examples in the directory examples++-load.

The Bernardi-Raugel Element  The Bernardi-Raugel finite element is meant to solve the Navier Stokes equations in \( u, p \) formulation; the velocity space \( P_{br}^K \) is minimal to prove the inf-sup condition with piecewise constant pressure by triangle. The finite element space \( V_h \) is

\[
V_h = \{ u \in H^1(\Omega)^2; \quad \forall K \in T_h, u|_K \in P_{br}^K \}
\]

where

\[
P_{br}^K = \text{span}\{ \lambda^K_i e_i \}_{i=1,2,3} \cup \{ \lambda^K_i \lambda^K_{i+1} n^K_{i+2} \}_{i=1,2,3}
\]

with notation \( 4 = 1, 5 = 2 \) and where \( \lambda^K_i \) are the barycentric coordinates of the triangle \( K \), \((e_i)_{i=1,2}\) 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 Bernardi Raugel Finite Element
// F. Hecht, decembre 2005
// ----------------------- related files:
// to check and validate : testFE.edp
// to get a real example : NSP2BRP0.edp

#include "error.hpp"
#include "AFfunction.hpp"
#include "rgraph.hpp"
using namespace std;
#include "RNM.hpp"
#include "fem.hpp"
#include "FESpace.hpp"
#include "AddNewFE.h"

namespace Fem2D {

class TypeOfFE_P2BRLagrange : public TypeOfFE {
public:
    static int Data[];

    TypeOfFE_P2BRLagrange(): TypeOfFE(6+3+0,
        2,
        Data,
        4,
        1,
        6+3*(2+2),   // nb coef to build interpolation
        9,           // np point to build interpolation
        0)
    {
        // to long see the source
    }

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

    void TypeOfFE_P2BRLagrange::Pi_h_alpha(const baseFElement & K,KN_<double> & v) const;
} ;

int TypeOfFE_P2BRLagrange::Data[]={
    0,0, 1,1, 2,2, 3,4,5,
    0,1, 0,1, 0,1, 0,0,0,
    0,0, 1,1, 2,2, 3,4,5,
    0,0, 0,0, 0,0, 0,0,0,
    0,1, 2,3, 4,5, 6,7,8,
    0,0
};

void TypeOfFE_P2BRLagrange::Pi_h_alpha(const baseFElement & K,KN_<double> & v) const {
    const Triangle & T(K.T);
    int k=0;
    // coef pour les 3 sommets fois le 2 composantes
    for (int i=0;i<6;i++)
        v[k++] = 1;
    // integration sur les aretes
    for (int i=0;i<3;i++)
        {
            R2 N(T.Edge(i).perp());
            N *= T.EdgeOrientation(i)*0.5 ;
            v[k++] = N.x;
            v[k++] = N.y;
            v[k++] = N.x;
            v[k++] = N.y;
        }
    }
}
void TypeOfFE_P2BRLagrange::FB(const bool * whatd, const Mesh & , const Triangle & K, const R2 & P, RNMK_ & val) const
{
    .... // to long see the source
}

// ---- cooking to add the finite element to freefem table --------
// a static variable to def the finite element
static TypeOfFE_P2BRLagrange P2LagrangeP2BR;

// now adding FE in FreeFem++ table
static AddNewFE P2BR("P2BR", &P2LagrangeP2BR);

// --- end cooking
// end FEM2d namespace

A way to check the finite element

load "BernadiRaugel" // a macro the compute numerical derivative

macro DD(f,hx,hy) ( (f(x1+hx,y1+hy)-f(x1-hx,y1-hy))/(2*(hx+hy))) //

mesh Th=square(1,1,[10*(x+y/3),10*(y-x/3)]);

real x1=0.7, y1=0.9, h=1e-7;
int it1=Th(x1,y1).nuTriangle;

fespace Vh(Th,P2BR);

Vh [a1,a2],[b1,b2],[c1,c2];

for (int i=0;i<Vh.ndofK;++i)
cout << i << " " << Vh(0,i) << endl;
for (int i=0;i<Vh.ndofK;++i)
{
    a1[]=0;
    int j=Vh(it1,i);
    a1[][j]=1; // a bascis functions
    plot([a1,a2], wait=1);

    [b1,b2]=[a1,a2]; // do the interpolation
    c1[] = a1[] - b1[];

    cout << " " << j << " " << 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 beginning 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 changed because the pressure is constant, and we cannot plot isovalues of piecewise constant functions.

The Morley Element  See the example bilapMorley.edp.

C.5 Add a new sparse solver

Warning the sparse solver interface as been completely rewritten in version 3.2 , so the section is obsolete, the example in are correct/ Only a fast sketch of the code is given here; for details see the .cpp 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_tpl1.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>
{
C.5. **ADD A NEW SPARSE SOLVER**

.... double version

```cpp
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.

```cpp
n=AA.n;
m=AA.m;
nnz=AA.nbcoef;
a=AA.a;
asub=AA.cl;
x=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 solver the linear solver \( x = A^{-1}b \)

```cpp
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`

```cpp
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.sparams,ds.perm_r,ds.perm_c);
}
```

```cpp
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.sparams,ds.perm_r,ds.perm_c);
}
```
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 defaultvalue=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 =TypeSolveMat::defaultvalue;
}

To add new function/name defaultsolver, defaulttoSuperLU in FreeFem++ , and set the default solver to the new solver, just do:

void 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));
}
LOADFUNC(init);

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.tar.gz
tar xvfz superlu_3.0.tar.gz
go SuperLU_3.0 directory
$EDITOR make.inc
make
```

2. In directory 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, 2, 0],
   [ 0, 0, 0, 3],
   [ 4,0 , 0, 0]];
```

\begin{verbatim}
[ 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=sparesolver);
xx = A^-1*b;
cout << xx << endl;
if(i==0)defaulttoGMRES();
if(i==1)default solver();
\end{verbatim}

To Test do for exemple:

FreeFem++ SuperLu.edp
Appendix D

Plugin

D.1 gsl

The interface with de gsl spline is available from version 3.47 the seven kind of spline are

0 gslinterpcspline (default type of spline)
1 gslinterpakima
2 gslinterpsteffen
3 gslinterplinear
4 gslinterppolynomial
5 gslinterpcsplineperiodic
6 gslinterpakimaperiodic

A brief wing example given all the syntaxe:

```c
int n=10;
real[int,int] dspline(2,n+1); // data points to def the spline
for(int i=0; i<= n; ++i) // set data poin
{
    real xx = square(real(i)/n);
    real yy = sin(xx*pi*2);
    dspline(0,i)= xx;
    dspline(1,i)= yy;
}
gslspline spline1(gslinterpcspline,dspline); // def the spline1

gslspline spline11(dspline); // def the spline11

gslspline spline2(gslinterpsteffen,dspline); // def the spline2

gslspline spline3(gslinterpcspline,dspline(0,:),dspline(1,:)); // def

gslspline spline33(dspline(0,:),dspline(1,:)); // def the spline3

gslspline spline33(dspline(0,:),dspline(1,:)); // def the spline3

spline1=spline2; // copie et change in spline1 in spline2
real t= 1;
real s1= spline1(t); // eval the function spline1 at t
real ds1= spline1.d(t); // eval the derive of function spline1 at t
real ddss1= spline1.dd(t); // eval the 2 derive of function spline1 at t
```
This can be useful to build functions from data values.

The list of all GSL functions and the FreeFem name without `_`

- GSLcdfugaussian$	ext{P}(a)$  $\rightarrow$  gslcdfugaussian$	ext{P}(a)$
- GSLcdfugaussian$	ext{Q}(a)$  $\rightarrow$  gslcdfugaussian$	ext{Q}(a)$
- GSLcdfugaussian$	ext{Pinv}(a)$  $\rightarrow$  gslcdfugaussian$	ext{Pinv}(a)$
- GSLcdfugaussian$	ext{Qinv}(a)$  $\rightarrow$  gslcdfugaussian$	ext{Qinv}(a)$
- GSLcdfgaussian$	ext{P}(a,b)$  $\rightarrow$  gslcdfgaussian$	ext{P}(a,b)$
- GSLcdfgaussian$	ext{Q}(a,b)$  $\rightarrow$  gslcdfgaussian$	ext{Q}(a,b)$
- GSLcdfgaussian$	ext{Pinv}(a,b)$  $\rightarrow$  gslcdfgaussian$	ext{Pinv}(a,b)$
- GSLcdfgaussian$	ext{Qinv}(a,b)$  $\rightarrow$  gslcdfgaussian$	ext{Qinv}(a,b)$
- GSLcdfgamma$	ext{P}(a,b,c)$  $\rightarrow$  gslcdfgamma$	ext{P}(a,b,c)$
- GSLcdfgamma$	ext{Q}(a,b,c)$  $\rightarrow$  gslcdfgamma$	ext{Q}(a,b,c)$
- GSLcdfgamma$	ext{Pinv}(a,b,c)$  $\rightarrow$  gslcdfgamma$	ext{Pinv}(a,b,c)$
- GSLcdfgamma$	ext{Qinv}(a,b,c)$  $\rightarrow$  gslcdfgamma$	ext{Qinv}(a,b,c)$
- GSLcdfcauchy$	ext{P}(a,b)$  $\rightarrow$  gslcdfcauchy$	ext{P}(a,b)$
- GSLcdfcauchy$	ext{Q}(a,b)$  $\rightarrow$  gslcdfcauchy$	ext{Q}(a,b)$
- GSLcdfcauchy$	ext{Pinv}(a,b)$  $\rightarrow$  gslcdfcauchy$	ext{Pinv}(a,b)$
- GSLcdfcauchy$	ext{Qinv}(a,b)$  $\rightarrow$  gslcdfcauchy$	ext{Qinv}(a,b)$
- GSLcdflaplace$	ext{P}(a,b)$  $\rightarrow$  gslcdflaplace$	ext{P}(a,b)$
- GSLcdflaplace$	ext{Q}(a,b)$  $\rightarrow$  gslcdflaplace$	ext{Q}(a,b)$
- GSLcdflaplace$	ext{Pinv}(a,b)$  $\rightarrow$  gslcdflaplace$	ext{Pinv}(a,b)$
- GSLcdflaplace$	ext{Qinv}(a,b)$  $\rightarrow$  gslcdflaplace$	ext{Qinv}(a,b)$
- GSLcdfrayleigh$	ext{P}(a,b)$  $\rightarrow$  gslcdfrayleigh$	ext{P}(a,b)$
- GSLcdfrayleigh$	ext{Q}(a,b)$  $\rightarrow$  gslcdfrayleigh$	ext{Q}(a,b)$
- GSLcdfrayleigh$	ext{Pinv}(a,b)$  $\rightarrow$  gslcdfrayleigh$	ext{Pinv}(a,b)$
- GSLcdfrayleigh$	ext{Qinv}(a,b)$  $\rightarrow$  gslcdfrayleigh$	ext{Qinv}(a,b)$
- GSLcdfexponential$	ext{P}(a,b)$  $\rightarrow$  gslcdfexponential$	ext{P}(a,b)$
- GSLcdfexponential$	ext{Q}(a,b)$  $\rightarrow$  gslcdfexponential$	ext{Q}(a,b)$
- GSLcdfexponential$	ext{Pinv}(a,b)$  $\rightarrow$  gslcdfexponential$	ext{Pinv}(a,b)$
- GSLcdfexponential$	ext{Qinv}(a,b)$  $\rightarrow$  gslcdfexponential$	ext{Qinv}(a,b)$
- GSLcdfexppow$	ext{P}(a,b,c)$  $\rightarrow$  gslcdfexppow$	ext{P}(a,b,c)$
- GSLcdfexppow$	ext{Q}(a,b,c)$  $\rightarrow$  gslcdfexppow$	ext{Q}(a,b,c)$
- GSLcdfbeta$	ext{P}(a,b)$  $\rightarrow$  gslcdfbeta$	ext{P}(a,b)$
- GSLcdfbeta$	ext{Q}(a,b)$  $\rightarrow$  gslcdfbeta$	ext{Q}(a,b)$
- GSLcdfbeta$	ext{Pinv}(a,b,c)$  $\rightarrow$  gslcdfbeta$	ext{Pinv}(a,b,c)$
- GSLcdfbeta$	ext{Qinv}(a,b,c)$  $\rightarrow$  gslcdfbeta$	ext{Qinv}(a,b,c)$
- GSLcdfbeta$	ext{Pinv}(a,b,c)$  $\rightarrow$  gslcdfbeta$	ext{Pinv}(a,b,c)$
- GSLcdfbeta$	ext{Qinv}(a,b,c)$  $\rightarrow$  gslcdfbeta$	ext{Qinv}(a,b,c)$
• \(gsl\text{-cdf}\_flat(P(a,b,c)) \rightarrow gsl\text{-cdfflat}(P(a,b,c))\)
• \(gsl\text{-cdf}\_flat(Q(a,b,c)) \rightarrow gsl\text{-cdfflatQ}(a,b,c)\)
• \(gsl\text{-cdf}\_flat\_Pinv(a,b,c) \rightarrow gsl\text{-cdfflatPinv}(a,b,c)\)
• \(gsl\text{-cdf}\_flat\_Qinv(a,b,c) \rightarrow gsl\text{-cdfflatQinv}(a,b,c)\)
• \(gsl\text{-cdf}\_lognormal\_P(a,b,c) \rightarrow gsl\text{-cdflognormalP}(a,b,c)\)
• \(gsl\text{-cdf}\_lognormal\_Q(a,b,c) \rightarrow gsl\text{-cdflognormalQ}(a,b,c)\)
• \(gsl\text{-cdf}\_lognormal\_Pinv(a,b,c) \rightarrow gsl\text{-cdflognormalPinv}(a,b,c)\)
• \(gsl\text{-cdf}\_lognormal\_Qinv(a,b,c) \rightarrow gsl\text{-cdflognormalQinv}(a,b,c)\)
• \(gsl\text{-cdf}\_gumbel1\_P(a,b,c) \rightarrow gsl\text{-cdfgumbel1P}(a,b,c)\)
• \(gsl\text{-cdf}\_gumbel1\_Q(a,b,c) \rightarrow gsl\text{-cdfgumbel1Q}(a,b,c)\)
• \(gsl\text{-cdf}\_gumbel1\_Pinv(a,b,c) \rightarrow gsl\text{-cdfgumbel1Pinv}(a,b,c)\)
• \(gsl\text{-cdf}\_gumbel1\_Qinv(a,b,c) \rightarrow gsl\text{-cdfgumbel1Qinv}(a,b,c)\)
• \(gsl\text{-cdf}\_gumbel2\_P(a,b,c) \rightarrow gsl\text{-cdfgumbel2P}(a,b,c)\)
• \(gsl\text{-cdf}\_gumbel2\_Q(a,b,c) \rightarrow gsl\text{-cdfgumbel2Q}(a,b,c)\)
• \(gsl\text{-cdf}\_gumbel2\_Pinv(a,b,c) \rightarrow gsl\text{-cdfgumbel2Pinv}(a,b,c)\)
• \(gsl\text{-cdf}\_gumbel2\_Qinv(a,b,c) \rightarrow gsl\text{-cdfgumbel2Qinv}(a,b,c)\)
• \(gsl\text{-cdf}\_weibull\_P(a,b,c) \rightarrow gsl\text{-cdfweibullP}(a,b,c)\)
• \(gsl\text{-cdf}\_weibull\_Q(a,b,c) \rightarrow gsl\text{-cdfweibullQ}(a,b,c)\)
• \(gsl\text{-cdf}\_weibull\_Pinv(a,b,c) \rightarrow gsl\text{-cdfweibullPinv}(a,b,c)\)
• \(gsl\text{-cdf}\_weibull\_Qinv(a,b,c) \rightarrow gsl\text{-cdfweibullQinv}(a,b,c)\)
• \(gsl\text{-cdf}\_pareto\_P(a,b,c) \rightarrow gsl\text{-cdfparetoP}(a,b,c)\)
• \(gsl\text{-cdf}\_pareto\_Q(a,b,c) \rightarrow gsl\text{-cdfparetoQ}(a,b,c)\)
• \(gsl\text{-cdf}\_pareto\_Pinv(a,b,c) \rightarrow gsl\text{-cdfparetoPinv}(a,b,c)\)
• \(gsl\text{-cdf}\_pareto\_Qinv(a,b,c) \rightarrow gsl\text{-cdfparetoQinv}(a,b,c)\)
• \(gsl\text{-cdf}\_logistic\_P(a,b) \rightarrow gsl\text{-cdflogisticP}(a,b)\)
• \(gsl\text{-cdf}\_logistic\_Q(a,b) \rightarrow gsl\text{-cdflogisticQ}(a,b)\)
• \(gsl\text{-cdf}\_logistic\_Pinv(a,b) \rightarrow gsl\text{-cdflogisticPinv}(a,b)\)
• \(gsl\text{-cdf}\_logistic\_Qinv(a,b) \rightarrow gsl\text{-cdflogisticQinv}(a,b)\)
• \(gsl\text{-cdf}\_binomial\_P(a,b,c) \rightarrow gsl\text{-cdfbinomialP}(a,b,c)\)
• \(gsl\text{-cdf}\_binomial\_Q(a,b,c) \rightarrow gsl\text{-cdfbinomialQ}(a,b,c)\)
• \(gsl\text{-cdf}\_poisson\_P(a,b) \rightarrow gsl\text{-cdfpoissonP}(a,b)\)
• \(gsl\text{-cdf}\_poisson\_Q(a,b) \rightarrow gsl\text{-cdfpoissonQ}(a,b)\)
• \(gsl\text{-cdf}\_geometric\_P(a,b) \rightarrow gsl\text{-cdfgeometricP}(a,b)\)
• \(gsl\text{-cdf}\_geometric\_Q(a,b) \rightarrow gsl\text{-cdfgeometricQ}(a,b)\)
• \(gsl\text{-cdf}\_negative\_binomial\_P(a,b,c) \rightarrow gsl\text{-cdfnegativebinomialP}(a,b,c)\)
• \(gsl\text{-cdf}\_negative\_binomial\_Q(a,b,c) \rightarrow gsl\text{-cdfnegativebinomialQ}(a,b,c)\)
• \(gsl\text{-cdf}\_pascal\_P(a,b,c) \rightarrow gsl\text{-cdfpascalP}(a,b,c)\)
• \(gsl\text{-cdf}\_pascal\_Q(a,b,c) \rightarrow gsl\text{-cdfpascalQ}(a,b,c)\)
• \(gsl\text{-ran}\_bernoulli\_pdf(a,b) \rightarrow gsl\text{ranbernoullipdf}(a,b)\)
• \(gsl\text{-ran}\_beta(a,b,c) \rightarrow gsl\text{ranbeta}(a,b,c)\)
• \(gsl\text{-ran}\_beta\_pdf(a,b,c) \rightarrow gsl\text{ranbetapdf}(a,b,c)\)
• \(gsl\text{-ran}\_binomial\_pdf(a,b,c) \rightarrow gsl\text{ranbinomialpdf}(a,b,c)\)
• \(gsl\text{-ran}\_exponential\_pdf(a,b) \rightarrow gsl\text{ranexponentialpdf}(a,b)\)
• \(gsl\text{-ran}\_exppow\_pdf(a,b,c) \rightarrow gsl\text{ranexppowpdf}(a,b,c)\)
• \(gsl\text{-ran}\_cauchy(a,b) \rightarrow gsl\text{rancauchy}(a,b)\)
• \(gsl\text{-ran}\_cauchy\_pdf(a,b) \rightarrow gsl\text{rancauchypdf}(a,b)\)
• \(gsl\text{-ran}\_chisq\_pdf(a,b) \rightarrow gsl\text{ranchisqpdf}(a,b)\)
• `gsl_ran_erlang(a,b,c)` → `gslranerlang(a,b,c)`
• `gsl_ran_erlang_pdf(a,b,c)` → `gslranerlangpdf(a,b,c)`
• `gsl_ran_fdist(a,b,c)` → `gslranfdist(a,b,c)`
• `gsl_ran_fdist_pdf(a,b,c)` → `gslranfdistpdf(a,b,c)`
• `gsl_ran_flat(a,b,c)` → `gslranflat(a,b,c)`
• `gsl_ran_flat_pdf(a,b,c)` → `gslranflatpdf(a,b,c)`
• `gsl_ran_gamma(a,b,c)` → `gslrangamma(a,b,c)`
• `gsl_ran_gamma_int(a,b)` → `gslrangammaint(a,b)`
• `gsl_ran_gamma_pdf(a,b,c)` → `gslrangammapdf(a,b,c)`
• `gsl_ran_gamma_mt(a,b,c)` → `gslrangammamt(a,b,c)`
• `gsl_ran_gamma_knuth(a,b,c)` → `gslrangammaknuth(a,b,c)`
• `gsl_ran_gaussian(a,b)` → `gslrangaussian(a,b)`
• `gsl_ran_gaussian_ratio_method(a,b)` → `gslrangaussianratimethod(a,b)`
• `gsl_ran_gaussian_ziggurat(a,b)` → `gslrangaussianziggurat(a,b)`
• `gsl_ran_gaussian_pdf(a,b)` → `gslrangaussianpdf(a,b)`
• `gsl_ran_ugaussian(a)` → `gslranugaussian(a)`
• `gsl_ran_ugaussian_ratio_method(a)` → `gslranugaussianratimethod(a)`
• `gsl_ran_ugaussian_pdf(a)` → `gslranugaussianpdf(a)`
• `gsl_ran_ugaussian_tail(a,b,c)` → `gslrangaussiantail(a,b,c)`
• `gsl_ran_ugaussian_tail_pdf(a,b,c)` → `gslrangaussiantailpdf(a,b,c)`
• `gsl_ran_landau(a)` → `gslranlandau(a)`
• `gsl_ran_landau_pdf(a)` → `gslranlandaupdf(a)`
• `gsl_ran_geometric_pdf(a,b)` → `gslrangeometricpdf(a,b)`
• `gsl_ran_gumbel1(a,b,c)` → `gslrangumbell1(a,b,c)`
• `gsl_ran_gumbel1_pdf(a,b,c)` → `gslrangumbell1pdf(a,b,c)`
• `gsl_ran_gumbel2(a,b,c)` → `gslrangumbell2(a,b,c)`
• `gsl_ran_gumbel2_pdf(a,b,c)` → `gslrangumbell2pdf(a,b,c)`
• `gsl_ran_logistic(a,b)` → `gslranlogistic(a,b)`
• `gsl_ran_logistic_pdf(a,b)` → `gslranlogisticpdf(a,b)`
• `gsl_ran_lognormal(a,b,c)` → `gslranlognormal(a,b,c)`
• `gsl_ran_lognormal_pdf(a,b,c)` → `gslranlognormalpdf(a,b,c)`
• `gsl_ran_logarithmic_pdf(a,b)` → `gslranlogarithmicpdf(a,b)`
• `gsl_ran_negative_binomial_pdf(a,b,c)` → `gslrannegativebinomialpdf(a,b,c)`
• `gsl_ran_pascal_pdf(a,b,c)` → `gslrangepascalpdf(a,b,c)`
• `gsl_ran_pareto(a,b,c)` → `gslranpareto(a,b,c)`
• `gsl_ran_pareto_pdf(a,b,c)` → `gslranparetopdf(a,b,c)`
• `gsl_ran_poisson_pdf(a,b)` → `gslranpoissonpdf(a,b)`
• `gsl_ran_rayleigh(a,b)` → `gslranrayleigh(a,b)`
• `gsl_ran_rayleigh_pdf(a,b)` → `gslranrayleighpdf(a,b)`
• `gsl_ran_rayleigh_tail(a,b,c)` → `gslranrayleightail(a,b,c)`
• `gsl_ran_rayleigh_tail_pdf(a,b,c)` → `gslranrayleightailpdf(a,b,c)`
• `gsl_ran_tdist(a,b)` → `gslrantdist(a,b)`
• `gsl_ran_tdist_pdf(a,b)` → `gslrantdistpdf(a,b)`
• `gsl_ran_laplace(a,b)` → `gslranlaplace(a,b)`
• `gsl_ran_laplace_pdf(a,b)` → `gslranlaplacepdf(a,b)`
• `gsl_ran_levy(a,b,c)` → `gslranlevy(a,b,c)`
• `gsl_ran_weibull(a,b,c)` → `gslranweibull(a,b,c)`
• `gsl_ran_weibull_pdf(a,b,c)` → `gslranweibullpdf(a,b,c)`
• gsl_sfnairy_Ai(a, b) → gslnairyAi(a, b)
• gsl_sfnairy_Bi(a, b) → gslnairyBi(a, b)
• gsl_sfnairy_Aiscaled(a, b) → gslnairyAiscaled(a, b)
• gsl_sfnairy_Bismcaled(a, b) → gslnairyBismcaled(a, b)
• gsl_sfnairy_Aideriv(a, b) → gslnairyAideriv(a, b)
• gsl_sfnairy_Bideriv(a, b) → gslnairyBideriv(a, b)
• gsl_sfnairy_Aiderivscaled(a, b) → gslnairyAiderivscaled(a, b)
• gsl_sfnairy_Biderivscaled(a, b) → gslnairyBiderivscaled(a, b)
• gsl_sfnairy_zero_Ai(a) → gslnairyzeroAi(a)
• gsl_sfnairy_zero_Bi(a) → gslnairyzeroBi(a)
• gsl_sfnairy_zero_Aideriv(a) → gslnairyzeroAideriv(a)
• gsl_sfnairy_zero_Bideriv(a) → gslnairyzeroBideriv(a)
• gsl_sfnbessel_J0(a) → gslnbesselJ0(a)
• gsl_sfnbessel_J1(a) → gslnbesselJ1(a)
• gsl_sfnbessel_Jn(a, b) → gslnbesselJn(a, b)
• gsl_sfnbessel_Y0(a) → gslnbesselY0(a)
• gsl_sfnbessel_Y1(a) → gslnbesselY1(a)
• gsl_sfnbessel_Yn(a, b) → gslnbesselYn(a, b)
• gsl_sfnbessel_I0(a) → gslnbesselI0(a)
• gsl_sfnbessel_I1(a) → gslnbesselI1(a)
• gsl_sfnbessel_In(a, b) → gslnbesselIn(a, b)
• gsl_sfnbessel_I0scaled(a) → gslnbesselI0scaled(a)
• gsl_sfnbessel_I1scaled(a) → gslnbesselI1scaled(a)
• gsl_sfnbessel_Inscaled(a, b) → gslnbesselInscaled(a, b)
• gsl_sfnbessel_K0(a) → gslnbesselK0(a)
• gsl_sfnbessel_K1(a) → gslnbesselK1(a)
• gsl_sfnbessel_Kn(a, b) → gslnbesselKn(a, b)
• gsl_sfnbessel_K0scaled(a) → gslnbesselK0scaled(a)
• gsl_sfnbessel_K1scaled(a) → gslnbesselK1scaled(a)
• gsl_sfnbessel_Knscaled(a, b) → gslnbesselKnscaled(a, b)
• gsl_sfnbessel_j0(a) → gslnbesselj0(a)
• gsl_sfnbessel_j1(a) → gslnbesselj1(a)
• gsl_sfnbessel_j2(a) → gslnbesselj2(a)
• gsl_sfnbessel_jl(a, b) → gslnbesseljl(a, b)
• gsl_sfnbessel_y0(a) → gslnbessely0(a)
• gsl_sfnbessel_y1(a) → gslnbessely1(a)
• gsl_sfnbessel_y2(a) → gslnbessely2(a)
• gsl_sfnbessel_yl(a, b) → gslnbesselyl(a, b)
• gsl_sfnbessel_I0scaled(a) → gslnbesselI0scaled(a)
• gsl_sfnbessel_II_scaled(a) → gslnbesselII_scaled(a)
• gsl_sfnbessel_ILscaled(a) → gslnbesselILscaled(a)
• gsl_sfnbessel_K0scaled(a) → gslnbesselK0scaled(a)
• gsl_sfnbessel_K1scaled(a) → gslnbesselK1scaled(a)
• gsl_sfnbessel_Knscaled(a, b) → gslnbesselKnscaled(a, b)
• gsl_sf_bessel_Knu_scaled(a,b) \mapsto \text{gslsfbesselKnuscaled}(a,b)
• gsl_sf_bessel_Knu(a,b) \mapsto \text{gslsfbesselKnu}(a,b)
• gsl_sf_bessel_InKnu(a,b) \mapsto \text{gslsfbesselInKnu}(a,b)
• gsl_sf_bessel_zero_J0(a) \mapsto \text{gslsfbesselzeroJ0}(a)
• gsl_sf_bessel_zero_J1(a) \mapsto \text{gslsfbesselzeroJ1}(a)
• gsl_sf_bessel_zero_Jnu(a,b) \mapsto \text{gslsfbesselzeroJnu}(a,b)
• gsl_sf_clausen(a) \mapsto \text{gslsfclausen}(a)
• gsl_sf_hydrogenicR_1(a,b) \mapsto \text{gslsfhydrogenicR1}(a,b)
• gsl_sf_dawson(a) \mapsto \text{gslsfdawson}(a)
• gsl_sf_debye_1(a) \mapsto \text{gslsfdebye1}(a)
• gsl_sf_debye_2(a) \mapsto \text{gslsfdebye2}(a)
• gsl_sf_debye_3(a) \mapsto \text{gslsfdebye3}(a)
• gsl_sf_debye_4(a) \mapsto \text{gslsfdebye4}(a)
• gsl_sf_debye_5(a) \mapsto \text{gslsfdebye5}(a)
• gsl_sf_debye_6(a) \mapsto \text{glsfdebye6}(a)
• gsl_sf_dilog(a) \mapsto \text{gslsfexp}(a)
• gsl_sf_multiply(a,b) \mapsto \text{gslsfmultiply}(a,b)
• gsl_sf_ellint_Kcomp(a,b) \mapsto \text{gslsfellintKcomp}(a,b)
• gsl_sf_ellint_Ecomp(a,b) \mapsto \text{gslsfellintEcomp}(a,b)
• gsl_sf_ellint_Pcomp(a,b,c) \mapsto \text{gslsfellintPcomp}(a,b,c)
• gsl_sf_ellint_Dcomp(a,b) \mapsto \text{gslsfellintDcomp}(a,b)
• gsl_sf_ellint_F(a,b,c) \mapsto \text{gslsfellintF}(a,b,c)
• gsl_sf_ellint_E(a,b,c) \mapsto \text{gslsfellintE}(a,b,c)
• gsl_sf_ellint_RC(a,b,c) \mapsto \text{gslsfellintRC}(a,b,c)
• gsl_sf_erd(a) \mapsto \text{gslsferfc}(a)
• gsl_sf_log_erfc(a) \mapsto \text{gslsflogerfc}(a)
• gsl_sf_erf(a) \mapsto \text{gslsfexp}(a)
• gsl_sf_exp(a) \mapsto \text{gslsfexp}(a)
• gsl_sf_exp_mult(a,b) \mapsto \text{glsfexpmult}(a,b)
• gsl_sf_expml(a) \mapsto \text{glsfexpml}(a)
• gsl_sf_exprel(a) \mapsto \text{glsfexprel}(a)
• gsl_sf_exprel_2(a) \mapsto \text{glsfexprel2}(a)
• gsl_sf_exprel_n(a,b) \mapsto \text{glsfexpreln}(a,b)
• gsl_sf_expint_E1(a) \mapsto \text{glsfexpintE1}(a)
• gsl_sf_expint_E2(a) \mapsto \text{glsfexpintE2}(a)
• gsl_sf_expint_En(a,b) \mapsto \text{glsfexpintEn}(a,b)
• gsl_sf_expint_E1_scaled(a) \mapsto \text{glsfexpintE1scaled}(a)
• gsl_sf_expint_E2_scaled(a) \mapsto \text{glsfexpintE2scaled}(a)
• gsl_sf_expint_En_scaled(a,b) \mapsto \text{glsfexpintEnscaled}(a,b)
• gsl_sf_expint_Ei(a) \mapsto \text{glsfexpintEi}(a)
• gsl_sf_expint_Ei_scaled(a) \mapsto \text{glsfexpintEiscaled}(a)
• gsl_sf_Shii(a) \mapsto \text{glsfShii}(a)
• gsl_sf_Chi(a) \mapsto \text{glsfChi}(a)
• gsl_sf_expint_3(a) \mapsto \text{glsfexpint3}(a)
• gsl_sf_Si(a) \mapsto \text{glsfSi}(a)
• gsl_sf_Ci(a) \mapsto \text{glsfCi}(a)
• gsl_sf_atanint(a) \mapsto \text{glsfatanint}(a)
• $gsl\text{_sf\_fermi\_dirac\_m1}(a) \rightarrow gsl\text{sf\_fermi\_dirac\_m1}(a)$
• $gsl\text{_sf\_fermi\_dirac\_0}(a) \rightarrow gsl\text{sf\_fermi\_dirac\_0}(a)$
• $gsl\text{_sf\_fermi\_dirac\_1}(a) \rightarrow gsl\text{sf\_fermi\_dirac\_1}(a)$
• $gsl\text{_sf\_fermi\_dirac\_2}(a) \rightarrow gsl\text{sf\_fermi\_dirac\_2}(a)$
• $gsl\text{_sf\_fermi\_dirac\_int}(a,b) \rightarrow gsl\text{sf\_fermi\_dirac\_int}(a,b)$
• $gsl\text{_sf\_fermi\_dirac\_mhalf}(a) \rightarrow gsl\text{sf\_fermi\_dirac\_mhalf}(a)$
• $gsl\text{_sf\_fermi\_dirac\_half}(a) \rightarrow gsl\text{sf\_fermi\_dirac\_half}(a)$
• $gsl\text{_sf\_fermi\_dirac\_3half}(a) \rightarrow gsl\text{sf\_fermi\_dirac\_3half}(a)$
• $gsl\text{_sf\_fermi\_dirac\_inc\_0}(a,b) \rightarrow gsl\text{sf\_fermi\_dirac\_inc\_0}(a,b)$
• $gsl\text{_sf\_lngamma}(a) \rightarrow gsl\text{sf\_lngamma}(a)$
• $gsl\text{_sf\_gamma}(a) \rightarrow gsl\text{sf\_gamma}(a)$
• $gsl\text{_sf\_gammastar}(a) \rightarrow gsl\text{sf\_gammastar}(a)$
• $gsl\text{_sf\_taylorcoeff}(a,b) \rightarrow gsl\text{sf\_taylorcoeff}(a,b)$
• $gsl\text{_sf\_fact}(a) \rightarrow gsl\text{sf\_fact}(a)$
• $gsl\text{_sf\_doublefact}(a) \rightarrow gsl\text{sf\_doublefact}(a)$
• $gsl\text{_sf\_lnfact}(a) \rightarrow gsl\text{sf\_lnfact}(a)$
• $gsl\text{_sf\_lnchoose}(a,b) \rightarrow gsl\text{sf\_lnchoose}(a,b)$
• $gsl\text{_sf\_pochrel}(a,b) \rightarrow gsl\text{sf\_pochrel}(a,b)$
• $gsl\text{_sf\_lngamma\_inc\_Q}(a,b) \rightarrow gsl\text{sf\_lngamma\_inc\_Q}(a,b)$
• $gsl\text{_sf\_lngamma\_inc\_P}(a,b) \rightarrow gsl\text{sf\_lngamma\_inc\_P}(a,b)$
• $gsl\text{_sf\_lngamma\_inc}(a,b) \rightarrow gsl\text{sf\_lngamma\_inc}(a,b)$
• $gsl\text{_sf\_lnbeta}(a,b) \rightarrow gsl\text{sf\_lnbeta}(a,b)$
• $gsl\text{_sf\_beta}(a,b) \rightarrow gsl\text{sf\_beta}(a,b)$
• $gsl\text{_sf\_beta\_inc}(a,b,c) \rightarrow gsl\text{sf\_beta\_inc}(a,b,c)$
• $gsl\text{_sf\_gegenpoly\_1}(a,b) \rightarrow gsl\text{sf\_gegenpoly\_1}(a,b)$
• $gsl\text{_sf\_gegenpoly\_2}(a,b) \rightarrow gsl\text{sf\_gegenpoly\_2}(a,b)$
• $gsl\text{_sf\_gegenpoly\_3}(a,b) \rightarrow gsl\text{sf\_gegenpoly\_3}(a,b)$
• $gsl\text{_sf\_gegenpoly\_n}(a,b,c) \rightarrow gsl\text{sf\_gegenpoly\_n}(a,b,c)$
• $gsl\text{_sf\_hyperg\_0F1}(a,b) \rightarrow gsl\text{sf\_hyperg\_0F1}(a,b)$
• $gsl\text{_sf\_hyperg\_1F1\_int}(a,b,c) \rightarrow gsl\text{sf\_hyperg\_1F1\_int}(a,b,c)$
• $gsl\text{_sf\_hyperg\_1F1}(a,b,c) \rightarrow gsl\text{sf\_hyperg\_1F1}(a,b,c)$
• $gsl\text{_sf\_hyperg\_U\_int}(a,b,c) \rightarrow gsl\text{sf\_hyperg\_U\_int}(a,b,c)$
• $gsl\text{_sf\_hyperg\_U}(a,b,c) \rightarrow gsl\text{sf\_hyperg\_U}(a,b,c)$
• $gsl\text{_sf\_hyperg\_2F0}(a,b,c) \rightarrow gsl\text{sf\_hyperg\_2F0}(a,b,c)$
• $gsl\text{_sf\_laguerre\_1}(a,b) \rightarrow gsl\text{sf\_laguerre\_1}(a,b)$
• $gsl\text{_sf\_laguerre\_2}(a,b) \rightarrow gsl\text{sf\_laguerre\_2}(a,b)$
• $gsl\text{_sf\_laguerre\_3}(a,b) \rightarrow gsl\text{sf\_laguerre\_3}(a,b)$
• $gsl\text{_sf\_laguerre\_n}(a,b,c) \rightarrow gsl\text{sf\_laguerre\_n}(a,b,c)$
• $gsl\text{_sf\_lambert\_W0}(a) \rightarrow gsl\text{sf\_lambert\_W0}(a)$
• $gsl\text{_sf\_lambert\_Wm1}(a) \rightarrow gsl\text{sf\_lambert\_Wm1}(a)$
• $gsl\text{_sf\_legendre\_P\_1}(a,b) \rightarrow gsl\text{sf\_legendre\_P\_1}(a,b)$
• $gsl\text{_sf\_legendre\_P\_2}(a) \rightarrow gsl\text{sf\_legendre\_P\_2}(a)$
• $gsl\text{_sf\_legendre\_P\_3}(a) \rightarrow gsl\text{sf\_legendre\_P\_3}(a)$
• $gsl\text{_sf\_legendre\_Q\_0}(a) \rightarrow gsl\text{sf\_legendre\_Q\_0}(a)$
D.2 ffrandom
plugin to linux random functions the range is random generator in the range from 0 to $(2^{31}) - 1$

```cpp
load "ffrandom"
srandomdev(); // set a true random seed ..
// warning under window this
// change the seed by randinit(random()) ) so all
// freefem++ random function are change..

int maxrang = 2^31-1;
cout << " max range " << maxrang << endl;
cout << random() << endl;
cout << random() << endl;
cout << random() << endl;
srandom(10);
cout << random() << endl;
cout << random() << endl;
cout << random() << endl;
```
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 sofware (files in directory src/Algo) and the file mt19937ar.cpp which may be redistributed under the terms of the GNU LESSER GENERAL PUBLIC LICENSE Version 2.1, February 1999

GNU LESSER GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".) "Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library’s complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

397
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 (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 modifi-
cation 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 mech-
anism 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, IN-
CIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR IN-
ABILITY 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 E

Keywords

Main Keywords

adaptmesh
Cmatrix
R3
bool
border
break
buildmesh
catch
cin
complex
continue
cout
element
else
df
fespace
for
func
if
ifstream
include
int
intalledge
load
macro
matrix
mesh
movemesh
ofstream
plot
problem
plot
plot
problem
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


[12] F. HECHT, Outils et algorithmes pour la méthode des éléments finis, HdR, Université Pierre et Marie Curie, France, 1992


Index

<<, 99
   matrix, 95
>> , 99
.* , 92
.* , 93
= , 184
?: , 77
[ ] , 28, 65, 124, 126
', 92, 93
3d
   beam-3d.edp, 255
   periodic, 244
time dependent, 39

accuracy, 31
acos, 80
acosh, 80
adaptation, 124, 143, 164
adaptmesh, 53, 65, 124, 126
   absererror=, 127
   anisomax=, 249
cutoff=, 127
err=, 126
errg=, 126
hmax=, 126
hmin=, 126
inquire=, 127
IsMetric=, 127
iso=, 126
keepbackvertices=, 127
maxsubdiv=, 127
metric=, 128
nbjacobian=, 126
nbsmooth=, 126
nbvx=, 126
nomeshgeneration=, 128
omega=, 126
   periodic=, 128
   powerin=, 128
   ratio=, 126
   rescaling=, 127
   splitin2=, 127
   splitpbedge=, 127
   thetamax=, 127
   uniform, 127
   verbosity=, 127
adj, 116
AffineCG, 211
AffineGMRES, 211
alphanumeric, 73
append, 99
area, 76
area coordinate, 192
argument, 78
ARGV, 101
-D, 101
array, 74, 84, 96, 194
   .imax, 87
   .imin, 87
   .l1, 87
   .l2, 87
   .linfty, 87
   .max, 87
   .min, 87
   .sum, 87
   ::, 85
   ?:, 85
   = + - * / .* ./ += -= /= *=
array, 96
column, 90
dot, 94
dot product, 87
FE function, 96
fespace, 174, 175
full matrix, 96

409
INDEX

im, 85, 86
line, 90
mapping, 87
matrix, 96
max, 85
mesh, 307
min, 83, 97
quantile, 87
re, 85
renumbering, 89
resize, 85
sort, 85
sum, 85
varf, 197
asin, 80
asinh, 80
assert(), 76
atan, 80
atan2, 80
atanh, 80
axisymmetric, 39

backward Euler method, 266
barycentric coordinates, 25
BDM1, 173
BDM1Ortho, 173
be, 116
whoinElement, 117
bessel, 82
BFGS, 214
BiLaplacien, 172
binary, 99
block matrix, 91
bool, 74
border, 110, 112
multi, 113
boundary condition, 196
BoundaryEdge, 75
boundingbox, 118
break, 97
broadcast, 307
bubble, 179
buildlayers, 148
buildmesh, 34
fixedborder, 111
fixedborder=, 111
fixedborder=1, 244

nbvx=, 111
catch, 105
Cauchy, 69
ceil, 80
CG, 186
change, 135
Characteristics-Galerkin, 45
checkmovemesh, 121
chi, 119
Cholesky, 186, 213
cin, 76, 99
clock, 23
CMAES, 214
initialStdDev=, 215
seed=, 214
column, 90
compatibility condition, 185
compiler, 73
Complex, 74
complex, 62, 78, 93
Complex geometry, 42
concatenation, 42
cone, 147
cconj, 78
connectivity, 201
continue, 97
convect, 46, 276
cos, 80
cosh, 80
cout, 76, 99
Crout, 186

cube, 136, 138, 147
flags=, 137
label=, 137
region=, 137
Curve, 166
de Moivre’s formula, 78
default, 100
defaultsolver, 384
defaulttoGMRES, 385
defaulttoSuperLU, 384
degree of freedom, 25
DFFT, 374
diag, 93, 282
diagonal matrix, 94
diffnp, 79
diffpos, 79
dimKrylov=, 248
Dirichlet, 31, 37, 69, 185
discontinuous functions, 294
Discontinuous-Galerkin, 45
displacement vector, 253
dist, 79
divide
term to term, 92
domain decomposition, 285, 286
dot, 94
dot product, 93, 94, 97
dumptable, 76

Edge03d, 173
Edge13d, 173
Edge23d, 173
degreeOrientation, 75

EigenValue, 263
ivalue=, 262
maxit=, 262
mode=, 262
ncv=, 262
nev=, 262
rawvector=, 262
sigma=, 262
sym=, 262
tol=, 262
value=, 262
vector=, 262
which=, 262
eigenvalue problems, 38
Element, 116
elements, 25, 26
emptymesh, 120
ENDIFMACRO, 104
endl, 99
erf, 82
erfc, 82
exception, 105
exec, 76, 208, 209
exit, 307
exp, 80
expression optimization, 198
external C++ function, 49

factorize=, 213
false, 74, 76
fdom, 80

FE function
[[], 181
complex, 62, 83, 93
n, 181
value, 181
FE space, 174, 175
FE-function, 83, 174, 175
FEQF, 173
FEQF1, 173
FEQF13d, 174
FEQF2, 173
FEQF23d, 174
FEQF5, 173
FEQF53d, 174
FEQF7, 174
FEQF9, 174
FEspace
(int , int ), 201
ndof, 201
nt, 201
defspace, 169

BDM1, 25, 173
BDM1Ortho, 25, 173
Edge03d, 26, 173
Edge13d, 26, 173
Edge23d, 26, 173
FEQF, 173
FEQF1, 173
FEQF13d, 174
FEQF2, 173
FEQF23d, 174
FEQF5, 173
FEQF53d, 174
FEQF7, 174
FEQF9, 174
HCT, 25, 172
Morley, 25
P0, 25, 170
P03d, 26
P0Edge, 171
P1, 25, 170
P13d, 26, 170
P1b, 25, 170
P1b3d, 26, 170
INDEX

P1bl, 170
P1bl3d, 170
P1dc, 25, 170
P1Edge, 171
P1nc, 25, 173
P2, 25, 170
P23d, 26
P2b, 25
P2BR, 25, 172, 379
P2dc, 25, 171
P2Edge, 171
P2h, 25
P2Morley, 171
P3, 25, 171
P3dc, 25, 171
P3Edge, 171
P4, 25, 171
P4dc, 25, 171
P4Edge, 171
periodic=, 186, 242
RT0, 25, 172
RT03d, 26
RT0Ortho, 25, 173
RT1, 25, 173
RT1Ortho, 25, 173
ff-mpirun, 16
ffmedit, 208
ffNLOpt, 231
FFT, 374
file
  am, 347, 348
  am_fmt, 115, 347, 348
  amdba, 347, 348
  bang, 115, 345
  data base, 345
  ftq, 349
  mesh, 115
  msh, 348
  nopo, 115
finite element space, 169
Finite Volume Methods, 48
fixed, 100
flabel
  change, 135
floor, 80
fluid, 275
flush, 100
fmax, 80
fmin, 80
fmod, 80
for, 97
Fourier, 40
region
  change, 135
func, 75
function
  tables, 119
functions, 80
gamma, 82
geometry input, 36
geline, 79
global
  area, 76
  BoundaryEdge, 75
cin, 76
  edgeOrientation, 75
  endl, 76
  false, 76
  hTriangle, 76
  InternalEdge, 76
  label, 75
  lenEdge, 75
  N, 75
  notaregion, 75
  nTonEgde, 76
  nuEdge, 76
  nuTriangle, 76
  pi, 76
  region, 75
  searchMethod, 183
  true, 76
  volume, 76
  x, 75
  y, 75
  z, 75
GMRES, 186
gnuplot, 208
good, 100
graphics packages, 31
gslinterpakima, 387
gslinterpakimaperiodic, 387
gslinterpcspline, 388
gslinterpcsplineperiodic, 387
gsliplinear, 387
gsliinterp, 387
gsliinterp, 387
gsliinterpol, 387
gsliinterpol, 387
hat function, 25
HCT, 172
Helmholtz, 62
hmax, 359
hmin, 359
hTriangle, 76, 249
IFMACRO, 104
ifstream, 99
    binary, 99
ill posed problems, 38
im, 85, 86, 92
imag, 78
include, 20, 102
includepath, 20
init, 48
init=, 187
initial condition, 69
inside=, 199
int, 74
int1d, 187
int2d, 187
int3d, 186, 187
intalledges, 187, 249
InternalEdge, 76
interpolate, 198
    inside=, 199
    op=, 199
t=, 199
interpolation, 184
invdiffnp, 79
invdiffpos, 79
Irecv, 307
Isend, 307
isoline, 165
isotropic, 253
j0, 82
j1, 82
ju, 82
jump, 249
j1, 86
j2, 86
label, 75, 109, 110, 113
    change, 135
label on the boundaries, 37
label=, 128
lagrangian, 122
Laplace operator, 31
lenEdge, 75, 249
level line, 47
levelset, 187
lgamma, 82
line, 90
LinearCG, 211
    eps=, 211
    nbiiter=, 211
    precon=, 211
    veps=, 211
LinearGMRES, 211
    eps=, 211
    nbiiter=, 211
    precon=, 211
    veps=, 211
linfty, 86
load, 20
loadpath, 20
log, 80
log10, 80
loop
    implicite, 98
lrint, 80
lround, 80
LU, 186
m, 383
macro, 47, 102, 299
    parameter, 102
    quote, 314
    quoting, 102, 103
    with parameter, 50
    without parameter, 47
mass lumping, 49
matrix, 28, 75, 213
    array, 90
    block, 91
mpiGroup, 305
mpiLAND, 305
mpiLOR, 305
mpiLXOR, 305
mpiMAX, 305
mpiMIN, 305
mpiPROD, 305
mpiRank, 306
mpirank, 305
mpiReduce, 307
mpiReduceScatter, 307
mpiRequest, 305
mpiScatter, 307
mpiSize, 306
mpisize, 305
mpiSUM, 305
mpiUndefined, 305
mpiWait, 306
mpiWtick, 306
mpiWtime, 306
multi-physics system, 42
multiple meshes, 42

N, 75, 190
n, 181, 383
Navier-Stokes, 276, 278
nbcoef, 383
nbe, 116
ndof, 201
ndofK, 201
Neumann, 69, 190
Newton, 214, 260
Newton Algorithm, 56
NLCG, 213
eps=, 211
nbiter=, 211
veps=, 211
nodes, 25
non-homogeneous Dirichlet, 31
nonlinear, 42
nonlinear problem, 39
norm, 374
normal, 47, 190
noshowbase, 100
noshowpos, 100
notaregion, 75
nt, 201, 359
nTonEdge, 47, 76
nuEdge, 76
number of degrees of freedom, 201
number of element, 201
nuTriangle, 76
nv, 359

ofstream, 99
append, 99
binary, 99
flush, 100
good, 100
seekg, 100
teepl, 100
on, 190
intersection, 278
scalar, 190
optimize=, 198
outer product, 90, 93

P, 75
P0, 170
P0Edge, 171
P1, 170
P1b, 170
P1bl, 170
P1dc, 170
P1Edge, 171
P1nc, 173
P2, 170
P2BR, 172
P2dc, 171
P2Edge, 171
P2Morley, 171
P3, 171
P3dc, 171
P4, 171
P4dc, 171
P4Edge, 171
P5Edge, 171
parabolic, 39
periodic, 169, 186, 242
3d, 244
pi, 76
plot, 203
aspectratio=, 204
bb=, 204
INDEX

border, 112
boundary=, 205
bw=, 204
cmm=, 204
coeff=, 204
cut, 206
dim=, 205
grey=, 204
mesh, 112
nbarrow=, 204
nbiso=, 204
ps=, 204
value=, 204
varrow=, 204
viso=, 116

plug
ffrandom, 82, 394

gls, 82
gsl, 82, 387

point
region, 116
triangle, 116

polar, 78
pow, 80

ppm2rmn, 167
precision, 100

precon=, 187, 197
problem, 48, 75, 184
eps=, 187
init=, 187
precon=, 187
solver=, 187
strategy =, 187, 197
tgv=, 187
tolpivot =, 187
tolpivotsym =, 187, 197

processor, 306, 307
processorblock, 306

product
Hermitian dot, 93
dot, 93, 94
outer, 93
term to term, 93
projection, 79

qfnbpE=, 197, 250

qft=, 191
qV=, 193
quadrature: q5pT, 192
quadrature: q5pV, 193
quadrature: default, 193
quadrature: q1pE, 191
quadrature: q1pElump, 191
quadrature: q1pT, 192
quadrature: q1pTlump, 192
quadrature: q2pE, 191
quadrature: q2pT, 192
quadrature: q2pT4P1, 192
quadrature: q3pE, 191
quadrature: q4pE, 190
quadrature: q5pE, 190
quadrature: q5pT, 192
quadrature: q7pT, 192
quadrature: q9pT, 192
quadrature: qfe=, 192, 193
quadrature: qforder=, 192, 193
quadrature: qf=, 192, 193
quadrature: qfV, 193
quadrature: qfV1, 193
quadrature: qfV1lump, 193
quadrature: qfV2, 193
quadrature: qfV5, 193
quantile, 89

radiation, 42
rand, 81
randinit, 81
randint31, 81
randint32, 81
random, 81
randow, 82
randreal1, 81
randreal2, 81
randreal3, 81
randreal53, 81
re., 85, 92
read files, 115
readmesh, 114, 137
readmesh3, 244
real, 74, 78
rectangle, 109
region, 75, 116, 293
change, 135
INDEX

region indicator, 42
renumbering, 89
resize, 83, 94
Reusable matrices, 277
rint, 80
Robin, 40, 185, 190
round, 80
RT0, 172
RT0Ortho, 173
RT1, 173
RT1Ortho, 173
savemesh, 114, 137
savesol
  order=, 154
schwarz, 307
scientific, 100
searchMethod, 183
sec:Plot, 203
Secv, 307
seekg, 100
Send, 307
set, 48
  matrix, 91
setw, 99, 100
showbase, 100
showpos, 100
shurr, 285, 286
sin, 80
singularity, 125
sinh, 80
solve, 75, 184
  eps=, 187
  init=, 187
  linear system, 93
  precon=, 187
  solver=, 187
  strategy=, 187
  tgv=, 28, 187
  tolpivot=, 187
  tolpivotSYM=, 187
  solver=, 213
CG, 126, 186
Cholesky, 186
Crout, 186
GMRES, 186
LU, 186
sparesolver, 186
UMFPACK, 186
sort, 85, 175
sparesolver, 186
split=, 128
splitmesh, 129
square, 109, 249
  flags=, 110
  label=, 110
  region=, 110
srandom, 82
srandomdev, 82
Stokes, 275
stokes, 272
stop test, 187
  absolute, 280
strain tensor, 253
strategy=, 187
streamlines, 276
stress tensor, 253
string, 79
  concatenation, 79
  find, 79
Stringification, 104
subdomains, 292
sum, 85
tan, 80
tanh, 80
Taylor-Hood, 277
teelp, 100
tetg, 139
  facetcl=, 139
  holelist=, 139
  nboffacetcl=, 139
  nbofholes=, 139
  nbofregions=, 139
  regionlist=, 139
  switch=, 139
  tetgconvexhull, 142
tetgreconstruction, 140
tetgtransfo, 142
  facetcl=, 142
  nboffacetcl=, 142
  ptmerge=, 142
  refface=, 142
  regionlist=, 142
INDEX

switch=, 142
tgamma, 82
tgv= < 0, 190
tolpivot=, 187
tolpivotsym=, 187
transpose, 92, 94, 97, 364
triangle [[]], 116
    area, 116
    label, 116
    region, 116
triangulate, 119
triangulation files, as well as read and write, 36
true, 74, 76
trunc, 128
    label=, 128
    split=, 128
try, 105
tutorial
    LaplaceRT.edp, 248
    adapt.edp, 125
    adaptindicatorP2.edp, 248
    AdaptResidualErrorIndicator.edp, 250
    aTutorial.edp, 236
    beam.edp, 254
    BlackSchol.edp, 272
    convect.edp, 270
    fluidStruct.edp, 289
    freeboundary.edp, 296
    movemesh.edp, 122
    NSUzawaCahouetChabart.edp, 278
    periodic.edp, 242
    periodic4.edp, 242
    periodic4bis.edp, 244
    readmesh.edp, 115
    Schwarz-gc.edp, 287
    Schwarz-no-overlap.edp, 286
    Schwarz-overlap.edp, 284
    StokesUzawa.edp, 277
tutorial
    VI.edp, 282
type of finite element, 170

UMFPACK, 186
upwinding, 45

varf, 28, 75, 187, 194
    array, 196
    matrix, 196
    optimize=, 198
variable, 73
variational formulation, 32
veps=, 213
verbosity, 17, 20
version, 76
vertex
    label, 116
    x, 116
    y, 116
viso, 47
volume, 76
weak form, 32
while, 97
whoinelement, 116
write files, 115
x, 75
y, 75
y0, 82
y1, 82
yn, 82
z, 79
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.