FreeFem++, a tool to solve PDEs numerically

Georges Sadaka

FreeFem++, a tool to solve PDEs numerically

Georges Sadaka

LAMFA CNRS UMR 7352

Université de Picardie Jules Verne

33, rue Saint-Leu, 80039 Amiens, France

http://lamfa.u-picardie.fr/sadaka/

georges.sadaka@u-picardie.fr

May 6, 2012

Abstract

FreeFem++ is an open source platform to solve partial differential equations numerically, based on finite element methods. It was developed at the Laboratoire Jacques-Louis Lions, Université Pierre et Marie Curie, Paris by Frédéric Hecht in collaboration with Olivier Pironneau, Jacques Morice, Antoine Le Hyaric and Kohji Ohtsuka.

The FreeFem++ platform has been developed to facilitate teaching and basic research through prototyping. FreeFem++ has an advanced automatic mesh generator, capable of a posteriori mesh adaptation; it has a general purpose elliptic solver interfaced with fast algorithms such as the multi-frontal method UMFPACK, SuperLU . Hyperbolic and parabolic problems are solved by iterative algorithms prescribed by the user with the high level language of FreeFem++. It has several triangular finite elements, including discontinuous elements. For the moment this platform is restricted to the numerical simulations of problems which admit a variational formulation.

We will give in the sequel an introduction to FreeFem++ which include the basic of this software. You may find more information throw this link http://www.freefem.org/ff++.

Contents

1 Introduction

FreeFem++ is a **Free** software to solve PDE using the **F**inite **e**lement **m**ethod and it run on Mac, Unix and Window architecture.

In FreeFem++, it's used a user language to set and control the problem. This language allows for a quick specification of linear PDE's, with the variational formulation of a linear steady state problem and the user can write they own script to solve non linear problem and time depend problem.

It's a interesting tool for the problem of average size. It's also a help for the modeling in the sense where it allows to obtain quickly numerical results which is useful for modifying a physical model, to clear the avenues of Mathematical analysis investigation, etc ...

A documentation of FreeFem++ is accessible on `www.freefem.org/ff++`, on the following link `www.freefem.org/ff++/ftp/FreeFem++doc.pdf`, you may also have a documentation in spanish on the following link `http://www.freefem.org/ff++/ftp/freefem++Spanish.pdf`

You can also download an integrated environment called FreeFem++-cs, written by Antoine Le Hyaric on the following link `www.ann.jussieu.fr/~lehyaric/ffcs/install.php`

2 Characteristics of FreeFem++

Many of FreeFem++ characteristics are cited in the full documentation of FreeFem++, we cite here some of them :

- 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, with specification by the user of the intersection of boundaries.
- Automatic mesh generator, based on the Delaunay-Voronoi algorithm [LucPir98].
- load and save Mesh, solution.
- Problem description (real or complex valued) by their variational formulations, the write of the variational formulation is too close for that written on a paper.
- Metric-based anisotropic mesh adaptation.
- A large variety of triangular finite elements : linear, quadratic Lagrangian elements and more, discontinuous P1 and Raviart-Thomas elements, ...
- Automatic Building of Mass/Rigid Matrices and second member.
- Automatic interpolation of data from a mesh to an other one, so a finite element function is view as a function of (x; y) or as an array.
- LU, Cholesky, Crout, CG, GMRES, UMFPack sparse linear solver.
- Tools to define discontinuous Galerkin finite element formulations P0, P1dc, P2dc and keywords: `jump, mean, intalledges`.
- Wide range of examples : Navier-Stokes, elasticity, fluid structure, eigenvalue problem, Schwarz' domain decomposition algorithm, residual error indicator, ...
- Link with other software : modulef, emc2, medit, gnuplot, ...

– Generates Graphic/Text/File outputs.
– A parallel version using mpi.

3 How to start?

All this information here are detailed in the FreeFem++ documentation.

3.1 Install

First open the following web page

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

Choose your platform: Linux, Windows, MacOS X, or go to the end of the page to get the full list of downloads and then install by double click on the appropriate file.

3.2 Text editor

1. For Windows :
   Install **notepad++** which is available at `http://notepad-plus.sourceforge.net/uk/site.htm`
   – Open Notepad++ and Enter F5
   – In the new window enter the command `FreeFem++ "$(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 P1dc P2dc P3dc P4dc RT0 RT1 RT2 RT3 RT4 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 asin 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.
2. For MacOS :
   Install **Smultron** which is available at `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
```

3. For Linux :
   Install **Kate** which is available at `ftp://ftp.kde.org/pub/kde/stable/3.5.10/src/kdebase-3.5.10.tar.bz2`

   To personalize with color syntax for .edp file, it suffices to take those given by **Kate** for c++ and to add the keywords of FreeFem++. Then, download edp.xml and save it in the directory ".kde/share/apps/katepart/syntax".

   We may find other description for other text editor in the full documentation of FreeFem++.

3.3 Save and run

All FreeFem++ code must be saved with file extension .edp and to run them you may double click on the file on MacOS or Windows otherwise we note that this can also be done in terminal mode by : `FreeFem++ mycode.edp`

4 Syntax and some operators

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 ";".
Another trick is to *comment in and out* by using the "`//`" as in C++. We note that, we can also comment a paragraph by using "`/* paragraph */`" and in order to make a break during the computation, we can use "`exit(0);`".
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 and the variable `clock()` gives the computer clock.
The language allows the manipulation of basic types :

- current coordinates : `x`, `y` and `z`;
- current differentials operators : `dx`= $\frac{\partial}{\partial x}$, `dy`= $\frac{\partial}{\partial y}$, `dz`= $\frac{\partial}{\partial z}$, `dxy`= $\frac{\partial}{\partial xy}$, `dxz`= $\frac{\partial}{\partial xz}$, `dyz`= $\frac{\partial}{\partial yz}$, `dxx`= $\frac{\partial}{\partial xx}$, `dyy`= $\frac{\partial}{\partial yy}$ and `dzz`= $\frac{\partial}{\partial zz}$;
- integers, example : `int a=1;`
- reals, example : `real b=1.;` (don't forget to put a point after the integer number)
- complex, example : `complex c=1.+3i;`
- strings, example : `string test="toto";`
- arrays with real component, example: `real[int] V(n);` where `n` is the size of `V`,
- arrays with complex component, example: `complex[int] V(n);`
- matrix with real component, example: `real[int,int] A(m,n);`
- matrix with complex component, example: `complex[int,int] C(m,n);`

- bidimensional (2D) finite element meshes, example : `mesh Th;`
- 2D finite element spaces, example : `fespace Vh(Th,P1); // where Vh is the Id space`
- threedimensional (3D) finite element meshes, example : `mesh3 Th3;`
- 3D finite element spaces, example : `fespace Vh3(Th3,P13d);`
- `int1d(Th,Γ)( u*v )` $= \int_\Gamma u \cdot v \, dx$ where $\Gamma \subset \mathbb{R}$;
- `int2d(Th)( u*v )` $= \int_\Omega u \cdot v \, dxdy$ where $\Omega \subset \mathbb{R}^2$;
- `int3d(Th)( u*v )` $= \int_\Omega u \cdot v \, dxdydz$ where $\Omega \subset \mathbb{R}^3$.

4.2 Some operators

We cite here some of the operator that are defined in FreeFem++:

```
+, -, *, /, ^,
<, >, <=, >=,
&, |, // where a & b = a and b, a | b = a or b
=, +=, -=, /=, *=, !=, ==.
```

4.3 Manipulation of functions

We can define a function as :

- an analytical function, example : `func u0=exp(-x^2-y^2),u1=1.*(x>=-2 & x<=2);`
- a finite element function or array, example : `Vh u0=exp(-x^2-y^2);`.
  We note that, in this case `u0` is a finite element, thus `u0[]` return the values of `u0` at each degree of freedom and to have access to the i^{th} element of `u0[]` we may use `u0[][i]`.
  We can also have an access to the value of `u0` at the point `(a,b)` by using `u0(a,b)`;
- a complex value of finite element function or array, example : `Vh<complex> u0=x+1i*y;`
- a formal line function, example : `func real g(int a, real b) { .....; return a+b;}` and to call this function for example we can use `g(1,2)`.
  We can also put an array inside this function as :

```
func real f(int a, real[int] U){
    Vh NU;
    NU[]=U;
    return a*NU;
}
Vh U=x,FNU=f(5,U[]);
```

- `macro` function, example : `macro F(t,u,v)[t*dx(u),t*dy(v)]//`, notice that every `macro` must end by "`//`", it simply replaces the name `F(t,u,v)` by `[t*dx(u),t*dy(v)]` and to have access only to the first element of `F`, we can use `F(t,u,v)[0]`.
  In fact, we note that the best way to define a function is to use `macro` function since in this example `t,u` and `v` could be integer, real, complex, array or finite element, ...
  For example, here is the most used function defined by a `macro` :

```
macro Grad(u)[dx(u),dy(u)]// in 2D
```

```
macro Grad(u)[dx(u),dy(u),dz(u)]// in 3D
macro div(u,v)[dx(u)+dy(v)]// in 2D
macro div(u,v,w)[dx(u)+dy(v)+dz(w)]// in 3D
```

4.4 Manipulation of arrays and matrices

Like in matlab, we can define an array such as : `real[int] U=1:2:10;` which is an array of 5 values `U[i]=2*i+1; i=0 to 4` and to have access to the i^{th} element of `U` we may use `U(i)`.

Also we can define a matrix such as `real[int,int] A=[ [1,2,3] , [2,3,4] ];` which is a matrix of size 2×3 and to have access to the $(i,j)^{\text{th}}$ element of `A` we may use `A(i,j)`.

We will give here some of manipulation of array and matrix that we can do with FreeFem++:

```
real[int] u1=[1,2,3],u2=2:4; // defining u1 and u2
real u1pu2=u1'*u2; // give the scalar product of u1 and u2, here
   ➥u1' is the transpose of u1;
real[int] u1du2=u1./u2; // divided term by term
real[int] u1mu2=u1.*u2; // multiplied term by term
matrix A=u1*u2'; // product of u1 and the transpose of u2
matrix<complex> C=[ [1,1i],[1+2i,.5*1i] ];
real trA=trace([1,2,3]*[2,3,4]'); // trace of the matrix
real detA=det([ [1,2],[-2,1] ]); // just for matrix 1x1 and 2x2
```

4.5 Loops and conditions

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.

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

An example below shows a sum from 1 to 10 with result is in `sum`,

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

The while-loop

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

is executed repeatedly until CONDITION become false. The sum from 1 to 5 can also be computed by `while`, in this example, we want to show how we can exit from a loop in midstream by `break` and how the `continue` statement will pass the part from `continue` to the end of the loop :

```
int i=1, sum=0;
while (i<=10) {
   sum += i; i++;
   if (sum>0) continue;
   if (i==5) break;
}
```

4.6 Input and output data

The syntax of input/output statements is similar to C++ syntax. It uses `cout`, `cin`, `endl`, << and >> :

```
int i;
cout << " std-out" << endl;
cout << " enter i= ? ";
cin >> i ;
Vh uh=x+y;
ofstream f("toto.txt"); f << uh[]; // to save the solution
ifstream f("toto.txt"); f >> uh[]; // to read the solution
```

We will present in the sequel, some useful script to use the FreeFem++ data with other software such as **ffglut**, **Gnuplot** [1], **Medit** [2], **Matlab** [3], **Mathematica** [4], **Visit** [5] when we save data with extension as **.eps**, **.gnu**, **.gp**, **.mesh**, **.sol**, **.bb**, **.txt** and **.vtu**.
For **ffglut** which is the visualization tools through a pipe of FreeFem++, we can plot the solution and save it with a .eps format such as :

```
plot(uh,cmm="t="+t+"  ;||u||_L^2="+NORML2[kk], fill=true,value=
   ➥true,dim=2);
```

For **Gnuplot**, we can save the data with extension **.gnu** or **.gp** such as :

```
{ ofstream gnu("plot.gnu"); // or plot.gp
//ofstream gnu("plot."+1000+k".gnu");  // to save the data
for (int i=0;i<=n;i++)
   gnu<<xx[i]<<" "<<yy[i]<<endl; // to plot yy[i] vs xx[i]
}
exec("echo 'plot \"plot.gnu\" w lp \
pause 5 \
quit' | gnuplot");
```

1. http://www.gnuplot.info/
2. http://www.ann.jussieu.fr/~frey/software.html
3. http://www.mathworks.fr/products/matlab/
4. http://www.wolfram.com/mathematica/
5. https://wci.llnl.gov/codes/visit/

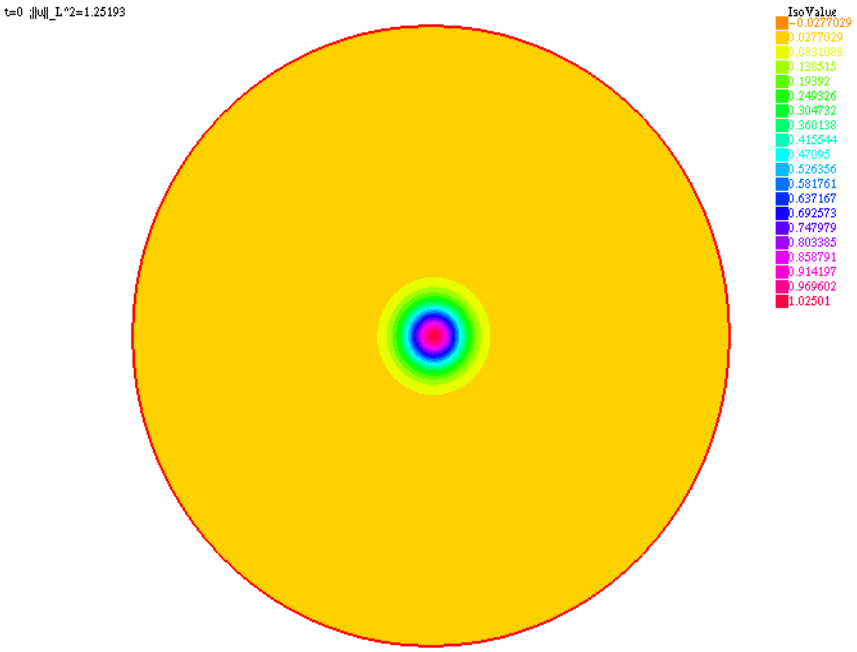

Figure 1: Visualising of the solution using ffglut

For **Medit**, we can save the data with extension **.mesh** and **.sol** such as :

```
load "medit"
int k=0;
savemesh(Th,"solution."+(1000+k)+".mesh");
savesol("solution."+(1000+k)+".sol",Th,uh);
medit("solution",Th,uh); // to plot the solution here
k+=1;
```

And then throw a terminal, in order to visualize the movie of the first 11 saved data, we can type this line :

```
ffmedit -a 1000 1010 solution.1000.sol
```

Don't forget in the window of **Medit**, to click on "m" to visualize the solution!

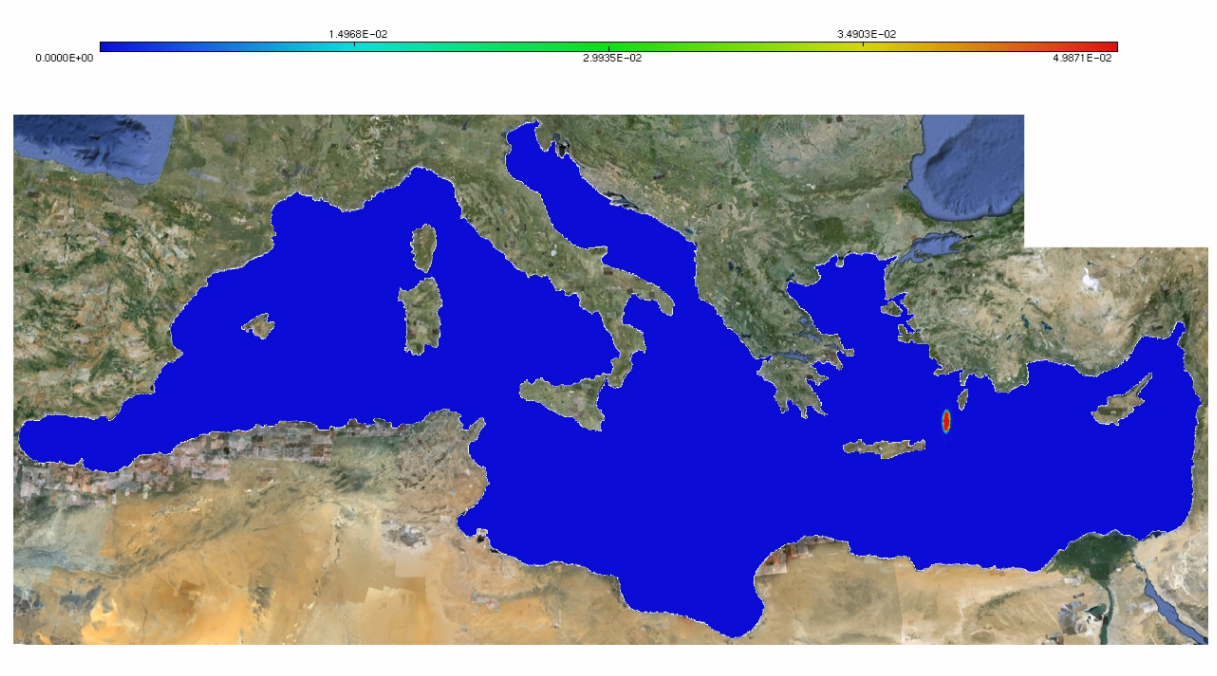

Figure 2: Visualising of the solution using Medit

For **Matlab**, we can save the data with extension **.bb** such as :

```
{ ofstream file("solution.bb");
   file << "2 1 1 "<< Vh.ndof << " 2 \n";
   for (int j=0;j<Vh.ndof ; j++)
      file << uh[][j] << endl;
}
```

And in order to visualize with **Matlab**, you can see the script made by Julien Dambrine at `http://www.downloadplex.com/Publishers/Julien-Dambrine/Page-1-0-0-0-0.html`.

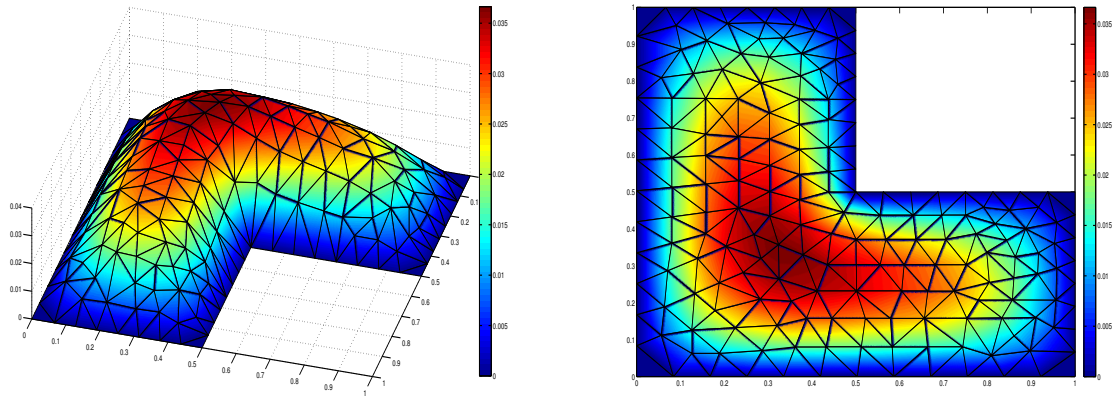

Figure 3: Visualising of the solution and the mesh using matlab

For **Mathematica**, we can save the data with extension **.txt** such as :

```
int k=0;
{ ofstream ff("uhsol."+(1000+k)+".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<<" "<<uh[][Vh(i,j)]<<endl;
   ff<<Th[i][0].x<<" "<< Th[i][0].y<<" "<<uh[][Vh(i,0)]<<"\n";
}
}
k+=1;
```

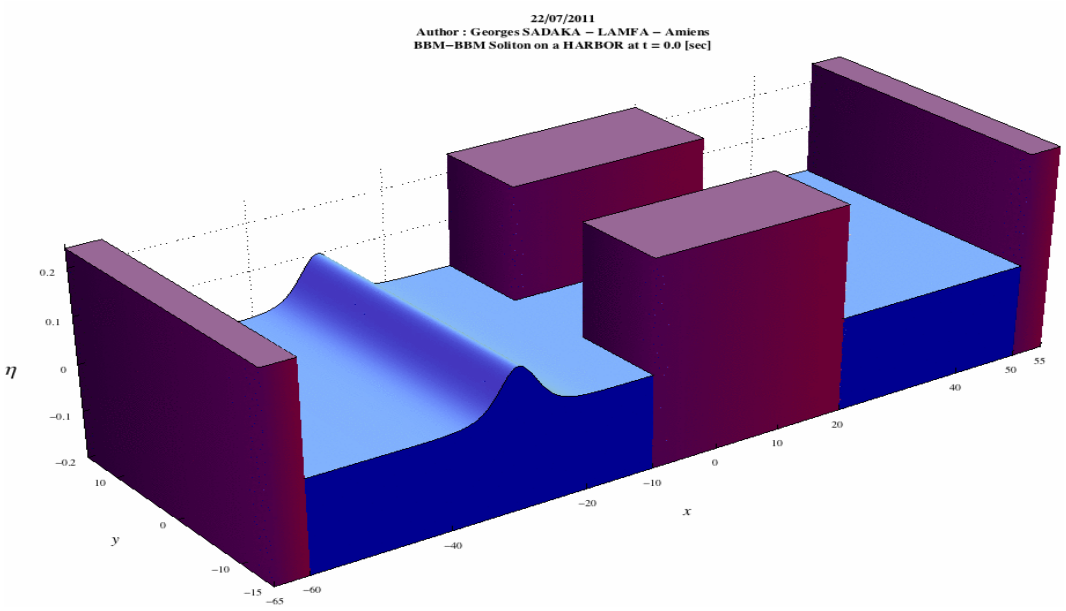

Figure 4: Visualising of the solution using Mathematica

For **Visit**, we can save the data with extension **.vtu** such as :

```
load "iovtk"
int k=0;
int[int] fforder2=[1,1,1];
savevtk("solution."+(1000+k)+".vtu",Th,uh1,uh2,order=fforder2,
   ➥dataname="UH1 UH2", bin=true);
k+=1;
```

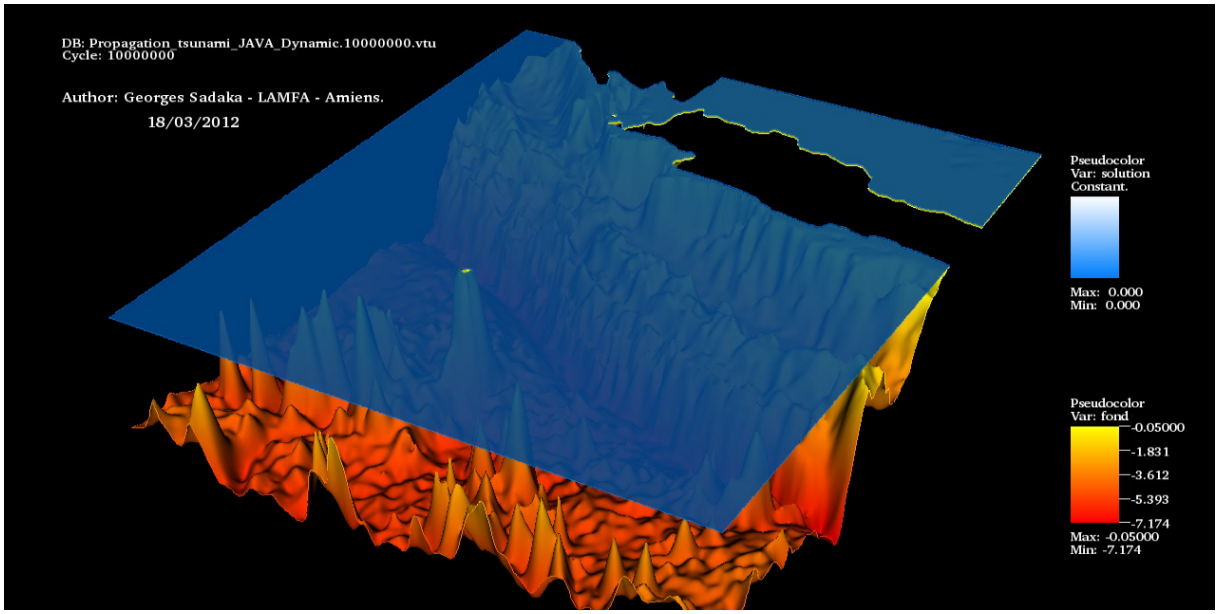

Figure 5: Visualising of the solution using visit

5 Construction of the domain Ω

We note that in FreeFem++ the domain is assumed to described by its boundary that is on the left side of the boundary which is implicitly oriented by the parametrization.
Let Ω be the rectangle defined by its frontier $\partial\Omega = [-5,5] \times [-1,1]$ where his vertices are $A(-5,-1), B(5,-1), C(5,1)$ and $D(-5,1)$, so we must define the border AB, BC, CD and DA of $\partial\Omega$ by using the keyword `border` then the triangulation $\mathcal{T}_h$ of Ω is automatically generated by using the keyword `buildmesh`.

```
real Dx=.2; // discretization space parameter
int aa=-5,bb=5,cc=-1,dd=1;
border AB (t = aa, bb){x = t ;y = cc;label = 1;};
border BC (t = cc, dd){x = bb;y = t ;label = 2;};
border CD (t = bb, aa){x = t ;y = dd;label = 3;};
border DA (t = dd, cc){x = aa;y = t ;label = 4;};
mesh Th = buildmesh( AB(floor(abs(bb-aa)/Dx)) + BC(floor(abs(dd-
   ➥cc)/Dx)) + CD(floor(abs(bb-aa)/Dx)) + DA(floor(abs(dd-cc)/Dx)
   ➥) );
plot( AB(floor(abs(bb-aa)/Dx)) + BC(floor(abs(dd-cc)/Dx)) + CD(
   ➥floor(abs(bb-aa)/Dx)) + DA(floor(abs(dd-cc)/Dx)) ); // to see
   ➥ the border
plot ( Th, ps="mesh.eps"); // to see and save the mesh
```

The keyword `label` can be added to define a group of boundaries for later use (Boundary Conditions for instance). Boundaries can be referred to either by name (AB for example) or by label (1 here).

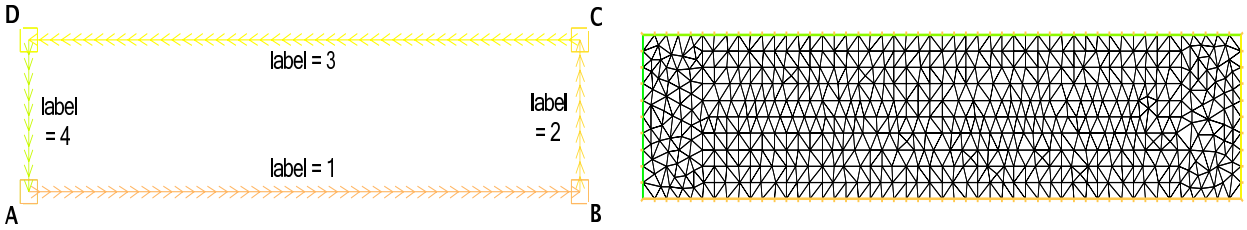

Figure 6: Plot of the border (left) and the mesh (right)

Another way to construct a rectangle domain with isotropic triangle is to use :

```
mesh Th=square(m,n,[x,y]); // build a square with m point on x
   ➥direction and n point on y direction
mesh Th1=movemesh(Th,[x+1,y*2]); // translate the square
   ➥]0,1[*]0,1[ to a rectangle ]1,2[*]0,2[
savemesh(Th1,"Name.msh"); // to save the mesh
mesh Th2("mesh.msh"); // to load the mesh
```

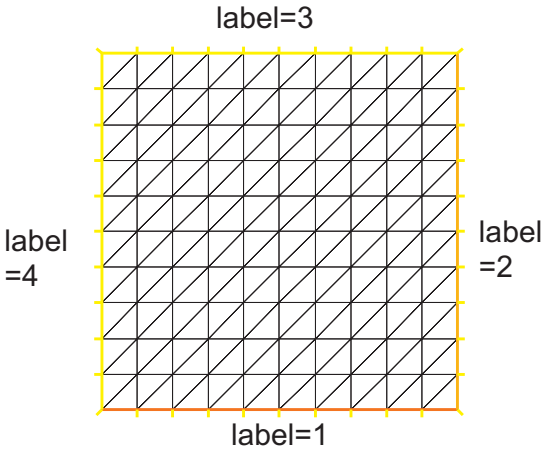

Figure 7: Boundary labels of the mesh by `square(10,10)`

We can also construct our domain defined by a parametric coordinate as:

```
border C(t=0,2*pi){ x=cos(t);y=sin(t);label=1}
mesh Mesh_Name=buildmesh(C(50));
```

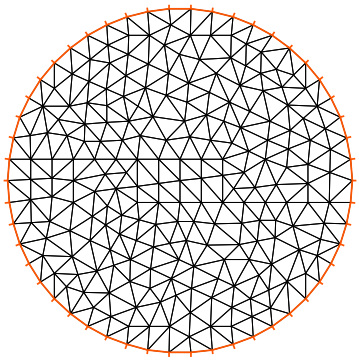

Figure 8: mesh `Th` by `build(C(50))`

To create a domain with a hole we can proceed as:

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

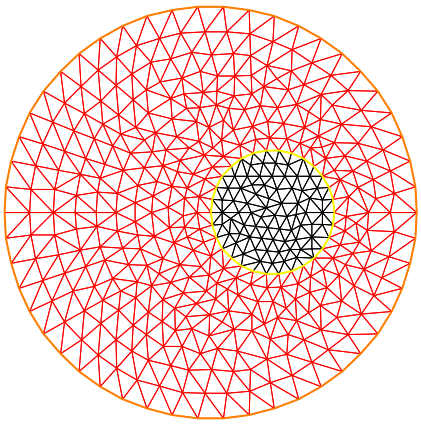

Figure 9: mesh without hole

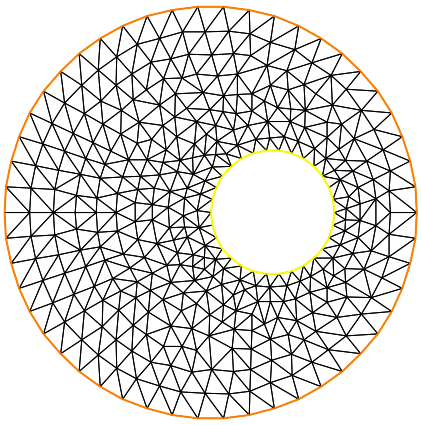

Figure 10: mesh with hole

6 Finite Element Space

A finite element space (F.E.S) is, usually, a space of polynomial functions on elements of $\mathcal{T}_h$, triangles here, with certain matching properties at edges, vertices, ... ; it's defined as :

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

As of today, the known types of F.E.S. are: **P0, P03d, P1, P13d, P1dc, P1b, P1b3d, P2, P23d, P2b, P2dc, P3, P3dc, P4, P4dc, Morley, P2BR, RT0, RT03d, RT0Ortho, Edge03d, P1nc, RT1, RT1Ortho, BDM1, BDM1Ortho, TDNNS1**; where for example:

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

$$P0_h = \left\{ v \in L^2(\Omega) \,\middle|\, \text{for all } K \in \mathcal{T}_h \text{ there is } \alpha_K \in \mathbb{R} : \quad v_{|K} = \alpha_K \right\} \tag{1}$$

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

$$P1_h = \left\{ v \in H^1(\Omega) \,\middle|\, \forall K \in \mathcal{T}_h; \quad v_{|K} \in P_1 \right\} \tag{2}$$

We can see the description of the rest of the F.E.S. in the full documentation of FreeFem++.

7 Boundary Condition

We will see in this section how it's easy to define the boundary condition (B.C.) with FreeFem++, for more information about these B.C., we refer to the full documentation.

7.1 Dirichlet B.C.

To define Dirichlet B.C. on a border $\Gamma_d \subset \mathbb{R}$ like $u|_{\Gamma_d} = f$, we can proceed as `on(gammad,u=f)`, where `u` is the unknown function in the problem.
The meaning is for all degree of freedom i of the boundary referred by the label "`gammad`", the diagonal term of the matrix $a_{ii} = tgv$ with the terrible giant value tgv (=10^{30} by default) and the right hand side $b[i] =$ "$(\Pi_h g)[i]$" $\times\, tgv$, where the "$(\Pi_h g)g[i]$" is the boundary node value given by the interpolation of g. (We are solving here the linear system $AX = B$, where $A = (a_{ij})_{i=1..n;j=1..m}$ and $B = (b_i)_{i=1..n}$).

If u is a vector like $u = (u1, u2)^T$ and we have $u1|_{\Gamma_d} = f1$ and $u2|_{\Gamma_d} = f2$, we can proceed as `on(gammad,u1=f1,u2=f2)`.

7.2 Neumann B.C.

The Neumann B.C. on a border $\Gamma_n \subset \mathbb{R}$, like $\dfrac{\partial u}{\partial n}|_{\Gamma_n} = g$, appear in the Weak formulation of the problem after integrating by parts, for example $\left\langle \dfrac{\partial u}{\partial n}; \Phi \right\rangle_{\Gamma_n} = \langle g; \Phi \rangle_{\Gamma_n} = \displaystyle\int_{\Gamma_n} g \cdot \Phi \, dx =$ `int1d(Th,gamman)(g*phi)`.

7.3 Robin B.C.

The Robin B.C. on a border $\Gamma_r \subset \mathbb{R}$; like $au + \kappa \dfrac{\partial u}{\partial n} = b$ on Γ_r where $a = a(x, y) \geq 0$, $\kappa = \kappa(x, y) \geq 0$ and $b = b(x, y)$; also appear in the Weak formulation of the problem after integrating by parts, for example $-\left\langle \kappa \dfrac{\partial u}{\partial n}; \Phi \right\rangle_{\Gamma_r} = \langle au - b; \Phi \rangle_{\Gamma_r} = \displaystyle\int_{\Gamma_r} au \cdot \Phi \, dx - \int_{\Gamma_r} b \cdot \Phi \, dx =$

`int1d(Th,gammar)(a*u*phi)-int1d(Th,gammar)(b*phi)`.

Important: it is not possible to write in the same integral the linear part and the bilinear part such as in `int1d(Th,gammar)(a*u*phi-b*phi)`.

7.4 Periodic B.C.

In the case of Bi-Periodic B.C., they are achieved in the definition of the periodic F.E.S. such as :

```
fespace Vh( Th, P1,periodic=[[1,x],[3,x],[2,y],[4,y]] );
```

8 Solve the problem

We present here different way to solve the Poisson equation :
Find $u : \Omega =]0,1[\times]0,1[\longrightarrow \mathbb{R}$ such that, for a given $f \in L^2(\Omega)$:

$$\left\{ \begin{array}{rcl} -\Delta u & = & f \text{ in } \Omega \\ u & = & 0 \text{ on } \partial\Omega \end{array} \right. \tag{3}$$

Then the basic variational formulation of (3) is :
Find $u \in H_0^1(\Omega)$, such that for all $v \in H_0^1(\Omega)$,

$$a(u,v) = l(v) \tag{4}$$

where

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \, dxdy \text{ and } l(v) = \int_\Omega f \cdot v \, dxdy$$

To discretize (4), let $\mathcal{T}_h$ denote a regular, quasi uniform triangulation of Ω with triangles of maximum size $h < 1$, let $V_h = \{v_h \in C^0(\bar{\Omega}); v_h|_T \in \mathbb{P}_1(T), \forall T \in \mathcal{T}_h; v_h = 0 \text{ on } \partial\Omega\}$ denote a finite-dimensional subspace of $H_0^1(\Omega)$ where $\mathbb{P}_1$ is the set of polynomials of $\mathbb{R}$ of degrees ≤ 1.
Thus the discretize weak formulation of (4) is :

$$\text{Find } u_h \in V_h : \int_\Omega \nabla u_h \cdot \nabla v_h \, dxdy - \int_\Omega f \cdot v_h \, dxdy = 0 \qquad \forall v_h \in V_h. \tag{5}$$

8.1 solve

The first method to solve (5) is to declare and solve the problem at the same time by using the keyword `solve` such as :

```
solve poisson(uh,vh,init=i,solver=LU) = // Solve Poisson Equation
    int2d(Th)( Grad(uh)'*Grad(vh) )     // bilinear form
    -int2d(Th)(f*vh)                    // linear form
    +on(1,2,3,4,uh=0);                  // Dirichlet B.C.
```

The solver used here is Gauss' LU factorization and when `init` $\neq 0$ the LU decomposition is reused so it is much faster after the first iteration. Note that if the mesh changes the matrix is reconstructed too.
The default solver is **sparsesolver** (it is equal to **UMFPACK** if not other sparce solver is defined) or is set to **LU** if no direct sparse solver is available. The storage mode of the matrix of the underlying linear system depends on the type of solver chosen; for **LU** the matrix is sky-line non symmetric, for **Crout** the matrix is sky-line symmetric, for **Cholesky** the matrix is sky-line symmetric positive definite, for **CG** the matrix is sparse symmetric positive, and for **GMRES**, **sparsesolver** or **UMFPACK** the matrix is just sparse.

8.2 problem

The second method to solve (5) is to declare the problem by using the keyword `problem`, and then solve it later by just call his name, such as :

```
problem poisson(uh,vh,init=i,solver=LU)// Definition of the
   ➥problem
    int2d(Th)( Grad(uh)'*Grad(vh) )      // bilinear form
    -int2d(Th)(f*vh)                     // linear form
    +on(1,2,3,4,uh=0);                   // Dirichlet B.C.
Poisson;                                 // Solve Poisson Equation
```

Note that, this technique is used when we have a time depend problem.

8.3 varf

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.

The system (5) is equivalent to :

$$\text{Find } u_h \in V_h : a(u_h, v_h) = l(v_h) \qquad \forall v_h \in V_h. \tag{6}$$

Here,

$$u_h(x,y) = \sum_{i=0}^{M-1} u_{hi}\phi_i(x,y) \tag{7}$$

where $\phi_i = v_{hi}, i = 0, ..., M-1$ are the basis functions of V_h, $M =$ `Vh.ndof` is the number of degree of freedom (i.e. the dimension of the space V_h) and u_{hi} is the value of u_h on each degree of freedom (i.e. $u_{hi} =$`uh[][i]`$=U$).

Thus, using (7), we can rewrite (6) such as :

$$\sum_{j=0}^{M-1} A_{ij}u_{hj} - F_i = 0, \quad i = 0, \cdots, M-1; \tag{8}$$

where

$$A_{ij} = \int_\Omega \nabla\phi_j \nabla\phi_i \, dxdy \qquad \text{and} \qquad F_i = \int_\Omega f\phi_i \, dxdy$$

The matrix $A = (A_{ij})$ is called *stiffness matrix*.
We deduce from the above notation that (8) is equivalent to

$$A \cdot U = F \Longleftrightarrow U = A^{-1} \cdot F \tag{9}$$

which can be solve in FreeFem++ as :

```
int m=10,n=10;
mesh Th= square (m,n ,[x,y]);
fespace Vh( Th , P1 );
Vh uh,vh;
macro Grad (u)[dx(u),dy(u)]// in 2D

func f=1;
varf a(uh ,vh) = int2d (Th)( Grad (uh) '* Grad (vh) ) // bilinear
   ➥ form
+on (1 ,2 ,3 ,4 , uh =0); // Dirichlet B.C.
matrix A=a(Vh ,Vh); // build the matrix
varf l( unused ,vh) = int2d (Th)(f*vh); // linear form
Vh F; F[] = l(0, Vh); // build the right hand side vector
set (A, solver = sparsesolver );
uh [] = A^ -1*F [];
plot(uh);
```

And in 3D :

```
load "msh3"
load "medit"
int m=10,n=10;
mesh Th2= square (m,n ,[x,y]);
mesh3 Th=buildlayers(Th2,10,zbound=[0,1]);
fespace Vh( Th , P13d );
Vh uh,vh;
macro Grad (u)[dx(u),dy(u),dz(u)]// in 2D

func f=1;
varf a(uh ,vh) = int3d (Th)( Grad (uh) '* Grad (vh) ) // bilinear
   ➥ form
+on (0, 1 ,2 ,3 ,4 ,5 , uh =0); // Dirichlet B.C.
matrix A=a(Vh ,Vh); // build the matrix
varf l( unused ,vh) = int3d (Th)(f*vh); // linear form
Vh F; F[] = l(0, Vh); // build the right hand side vector
set (A, solver = sparsesolver );
uh [] = A^ -1*F [];
medit("sol",Th,uh);
```

9 Learning by examples

9.1 Rate of convergence for the Poisson equation

At the beginning, we prove that the rate of convergence in space for the Poisson equation code with P_1 finite element is of order 2.
In this example, we took zero Dirichlet homogenous B.C. on the whole boundary and we have considered the following exact solution :

$$u_{ex} = \sin(\pi x) \cdot \sin(\pi y)$$

Then, we compute the corresponding right hand side $f(x, y)$ in order to obtain the L^2 norm of the error between the exact solution and the numerical one (cf. Table 1)

$$E(u, h_n) = |u_h(h_n) - u_{ex}(h_n)|_{L^2}, \forall i = 1, ..., \texttt{nref}, h = \delta x = 1/N;$$

and then the rate of convergence in space

$$r(u, h_n) = \frac{\log\left(E(u, h_{n-1})/E(u, h_n)\right)}{\log\left(h_{n-1}/h_n\right)}, \forall i = 1, ..., \texttt{nref}$$

We give here a method to compute the right hand side using **Maple**[6] :

```
> u := unapply(sin(pi·x)·sin(pi·y), x, y);
```

$$u := (x, y) \rightarrow \sin(\pi x)\,\sin(\pi y) \quad (1)$$

```
> f := -diff(u(x, y), x, x) - diff(u(x, y), y, y);
```

$$f := 2\sin(\pi x)\,\pi^2\,\sin(\pi y) \quad (2)$$

```
> with(CodeGeneration);
```

[C, Fortran, IntermediateCode, Java, LanguageDefinition, Matlab, Names, Save, Translate, VisualBasic] (3)

```
> Matlab(%%, resultname = "f");
f = 0.2e1 * sin(pi * x) * pi ^ 2 * sin(pi * y);
```

We can copy and paste the result of $f(x, y)$ in the FreeFem++ code.
We present here the script to compute the rate of convergence in space of the code solving the Poisson equation :

```
int nref=4;
real[int] L2error(nref); // initialize the L2 error array
for (int n=0;n<nref;n++) {
    int N=2^(n+4);           // space discretization
    mesh Th= square(N,N);    // mesh generation of a square
    fespace Vh(Th,P1);  // space of P1 Finite Elements
    Vh uh,vh;    // uh and vh belongs to Vh
```

6. http://www.maplesoft.com/

```
    macro Grad(u)[dx(u),dy(u)]//
    Vh uex=sin(pi*x)*sin(pi*y); // exact solution
    Vh f=0.2e1*sin(pi*x)*pi^2*sin(pi*y); // corresponding RHS
    varf a(uh,vh) = int2d(Th)( Grad(uh)'*Grad(vh) ) // bilinear
      ➥form
                    +on(1,2,3,4,uh=0);  // Dirichlet B.C.
    matrix A=a(Vh,Vh);  // build the matrix
    varf l(unused ,vh) = int2d(Th)(f*vh);   //  linear  form
    Vh F; F[] = l(0,Vh);    // build the right hand side vector
    set(A,solver=sparsesolver);
    uh[] = A^-1*F[];
    L2error[n]= sqrt(int2d(Th)((uh-uex)^2));
}
for(int n=0;n<nref;n++)
    cout << "L2error " << n << " = "<<  L2error[n] <<endl;
for(int n=1;n<nref;n++)
    cout <<"convergence rate = "<< log(L2error[n-1]/L2error[n])/
      ➥log(2.)  <<endl;
```

N_n	$E(u,h_n)$	$r(u,h_n)$
16	0.0047854	-
32	0.00120952	1.9842
64	0.000303212	1.99604
128	7.58552e-05	1.99901

Table 1: L^2 norm of the error and the rate of convergence.

9.2 Poisson equation over the Fila's face

We present here a method to build a mesh from a photo using Photoshop® and a script in FreeFem++ made by Frédéric Hecht.
We choose here to apply this method on the **Fila**'s face. To this end, we start by the photo in Figure 11, and using Photoshop®, we can remove the region that we wanted out of the domain such as in Figure 12, then fill in one color your domain and use some filter in Photoshop® in order to smooth the boundary as in Figure 13. Then convert your jpg photo to a pgm photo which can be read by FreeFem++ by using in a terminal window :

```
convert fila.jpg fila.pgm
```

Figure 11: Initial photo.

Figure 12: Using Photoshop.

Figure 13: Last photo.

Finally, using the following script :

```
load "ppm2rnm"
load "isoline"
string fila="fila.pgm";
real[int,int] Curves(3,1);
int[int] be(1);
int nc;
{ // build the curve file xy.txt ...
real[int,int] ff1(fila); // read  image and set to an rect. array
//  remark (0,0) is the upper, left corner.
int nx = ff1.n, ny=ff1.m;
// build a cartesain mesh such that the origne is qt 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.
real vmax = f1[].max ;
real vmin = f1[].min ;
real vm = (vmin+vmax)/2;
verbosity=3;
/*
 Usage of isoline
 the named parameter :
 iso=0.25   // value of iso
 close=1, //   to force to have closing curve ...
 beginend=be, // begin  and end of curve
 smoothing=.01, //   nb of smoothing process = size^ratio * 0.01
 where  size is the size of the curve ...
```

```
 ratio=0.5
 file="filename"

 ouptut:
  xx, yy  the array of point of the iso value

a closed curve  number n is

in fortran notation the point of the curve are:
(xx[i],yy[i], i = i0, i1)
with :  i0=be[2*n],  i1=be[2*n+1];

*/
nc=isoline(Th,f1,iso=vm,close=0,Curves,beginend=be,smoothing
   ➥=.005,ratio=0.1);
verbosity=1;
}
int ic0=be(0), ic1=be(1)-1;
  plot([Curves(0,ic0:ic1),Curves(1,ic0:ic1)], wait=1);
// end smoothing the curve ....
macro GG(i)
border G#i(t=0,1)
{
  P=Curve(Curves,be(i*2),be(i*2+1)-1,t);
  label=i+1;
}
real lg#i=Curves(2,be(i*2+1)-1);  //
GG(0)   GG(1)   GG(2)   GG(3)   GG(4) // number of closing curve
real hh= -10;
cout << " .. "<<endl;
func bord = G0(lg0/hh)+G1(lg1/hh)+G2(lg2/hh)+G3(lg3/hh)+G4(lg4/hh
   ➥);
plot(bord,wait  =1);
mesh Th=buildmesh(bord);
cout << " ...  "<<endl;
plot(Th,wait=1);
Th=adaptmesh(Th,5.,IsMetric=1,nbvx=1e6);
plot(Th,wait=1);
savemesh(Th,"fila.msh");
```

we can create the mesh of our domain (cf. Figure 14), and then read this mesh in order to solve the Poisson equation on this domain (cf. Figure 15)

```
mesh Th("fila.msh");
plot(Th);
fespace Vh(Th,P1);
Vh uh,vh;
```

```
func f = 1.;
macro Grad(u)[dx(u),dy(u)]//
solve Poisson(uh,vh) = int2d(Th)(Grad(uh)'*Grad(vh)) - int2d(Th)(
   ➥ f*vh) + on(1,2,3,4,5,uh=0) ;
plot(uh,dim=2,fill=true,value=true);
```

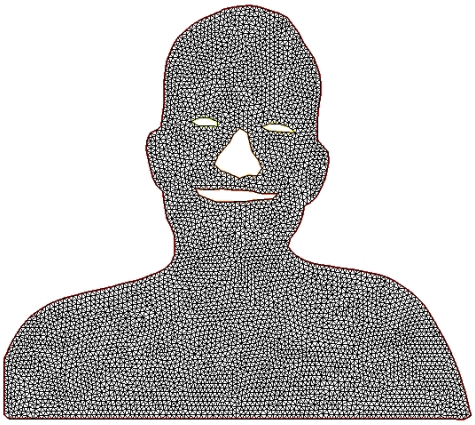

Figure 14: Mesh of the **Fila**'s face.

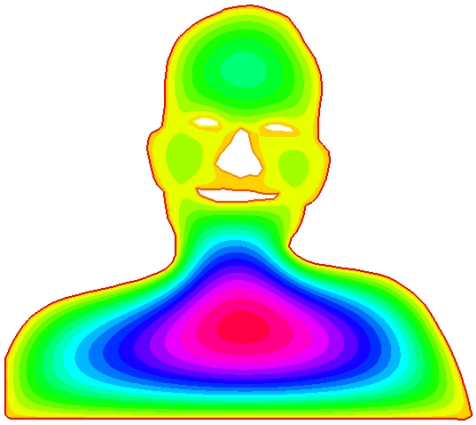

Figure 15: Solution on the **Fila**'s face.

9.3 Rate of convergence for an Elliptic non linear equation

Let $\Omega = B(O,R) \subset \mathbb{R}^2$, it is proposed to solve numerically the problem which consist to find $u(x,y)$ such that

$$\left\{ \begin{array}{rcll} -\Delta u(x,y) + u^3 & = & f & \text{for all } (x,y) \in \Omega \subset \mathbb{R}^2, \\ u(x,y) & = & 0 & \text{for all } (x,y) \text{ on } \partial\Omega. \end{array} \right. \tag{10}$$

9.3.1 Space discretization

Let $\mathcal{T}_h$ be the triangulation of Ω and

$$V_h = \{v_h \in C^0(\bar{\Omega}); v_h|_T \in \mathbb{P}_1(T), \forall T \in \mathcal{T}_h, v_h = 0 \text{ on } \partial\Omega\}.$$

For simplicity, we denote by $\mathcal{V}(u) = u^2$, then the approximation of the variational formulation will be :
Find $u_h \in V_h$ such that $\forall v_h \in V_h$ we have :

$$-\langle \Delta u_h; v_h \rangle + \langle \mathcal{V}(u_h) \cdot u_h; v_h \rangle = \langle f; v_h \rangle$$

thus

$$\langle \nabla u_h; \nabla v_h \rangle + \langle \mathcal{V}(u_h) \cdot u_h; v_h \rangle = \langle f; v_h \rangle \tag{11}$$

In order to solve numerically the non linear term in (13), we will use a semi-implicit scheme such as :

$$\left\langle \nabla u_h^{n+1}; \nabla v_h \right\rangle + \left\langle \mathcal{V}(u_h^n) \cdot u_h^{n+1}; v_h \right\rangle = \langle f; v_h \rangle ,$$

and then we solve our problem by the fixed point method in this way :

```
Set                          u_h^0 = u_0 = 0
Set                          V(u_h^n) = V(u_0)
Set                          err = 1.
while (err > 1e-10)
                             Solve <∇u_h^{p+1}; ∇v_h> = -<V(u_h^n)·u_h^p; v_h> + <f; v_h>
                             Compute err = ||u_h^n - u_h^{p+1}||_{L^2}
                             set V(u_h^n) = V(u_h^{p+1})
                             set u_h^n = u_h^{p+1}
                             p = p + 1;
End while
```

In order to test the convergence of this method we will study the rate of convergence in space (cf. Table 3) of the system (12) with $R = 1$ and the exact solution :

$$u_{ex} = \sin(x^2 + y^2 - 1).$$

Then, we compute the corresponding right hand side $f(x, y)$ using **Maple** such as :

```
> u := unapply(sin(x^2 + y^2 - 1), x, y);
```

$$u := (x, y) \rightarrow \sin(x^2 + y^2 - 1) \quad (1)$$

```
> f := -diff(u(x, y), x, x) - diff(u(x, y), y, y) + u(x, y)^3;
```

$$f := 4\sin(x^2 + y^2 - 1)x^2 - 4\cos(x^2 + y^2 - 1) + 4\sin(x^2 + y^2 - 1)y^2 + \sin(x^2 + y^2 - 1)^3 \quad (2)$$

```
> simplify(%);
```

$$4\sin(x^2 + y^2 - 1)x^2 - 4\cos(x^2 + y^2 - 1) + 4\sin(x^2 + y^2 - 1)y^2 + \sin(x^2 + y^2 - 1) - \sin(x^2 + y^2 - 1)\cos(x^2 + y^2 - 1)^2 \quad (3)$$

```
> with(CodeGeneration);
```

[*C*, *Fortran*, *IntermediateCode*, *Java*, *LanguageDefinition*, *Matlab*, *Names*, *Save*, *Translate*, *VisualBasic*] (4)

```
> Matlab(%%, resultname = "f");
f = 0.4e1 * sin((x ^ 2 + y ^ 2 - 1)) * (x ^ 2) - 0.4e1 * cos((x ^
2 + y ^ 2 - 1)) + 0.4e1 * sin((x ^ 2 + y ^ 2 - 1)) * (y ^ 2) + sin
((x ^ 2 + y ^ 2 - 1)) - sin((x ^ 2 + y ^ 2 - 1)) * cos((x ^ 2 + y
^ 2 - 1)) ^ 2;
```

We present here the corresponding script to compute the rate of convergence in space of the code solving the Elliptic non linear equation (12):

```
verbosity=0.;
int nraff=7;
real[int] L2error(nraff); // initialize the L2 error array
```

```
for (int n=0;n<nraff;n++) {
    int N=2^(n+4);          // space discretization
    real R=1.; // radius
    border C(t=0.,2.*pi){x=R*cos(t);y=R*sin(t);label=1;};
    mesh Th=buildmesh(C(N));
    fespace Vh(Th,P1);
    Vh uh, uh0=0, V=uh0^2, vh;
    Vh uex=sin((x ^ 2 + y ^ 2 - 1));
    Vh f=0.4e1 * sin((x ^ 2 + y ^ 2 - 1)) * (x ^ 2) - 0.4e1 * cos
      ➥((x ^ 2 + y ^ 2 - 1)) + 0.4e1 * sin((x ^ 2 + y ^ 2 - 1))
      ➥* (y ^ 2) + sin ((x ^ 2 + y ^ 2 - 1)) - sin((x ^ 2 + y ^
      ➥2 - 1)) * cos((x ^ 2 + y ^ 2 - 1)) ^ 2;
    macro Grad(u)[dx(u),dy(u)]//
    problem ELLNL(uh,vh) =
            int2d(Th)(Grad(uh)'*Grad(vh))  // bilinear term
            + int2d(Th) ( uh*V*vh )          // non linear term
            - int2d(Th)( f*vh )              // right hand side
            + on(1,uh=0);                    // Dirichlet B.C.
    real err=1.; // for the convergence
    while (err >= 1e-10){
        ELLNL;
        err=sqrt(int2d(Th)((uh-uh0)^2));
        V=uh^2;          // actualization
        uh0=uh;
    }
    L2error[n]= sqrt(int2d(Th)((uh-uex)^2));
}
for(int n=0;n<nraff;n++)
    cout << "L2error " << n << " = "<<  L2error[n] <<endl;
for(int n=1;n<nraff;n++)
    cout <<"convergence rate = "<< log(L2error[n-1]/L2error[n])/
      ➥log(2.)  <<endl;
```

N_n	$E(u,h_n)$	$r(u,h_n)$
16	0.015689	-
32	0.0042401	1.88758
64	0.00117866	1.84695
128	0.00032964	1.83819
256	8.48012e-05	1.95873
512	1.9631e-05	2.11095
1024	4.88914e-06	2.00548

Table 2: L^2 norm of the error and the rate of convergence.

9.4 Rate of convergence for an Elliptic non linear equation with big Dirichlet B.C.

Let $\Omega = B(O,R) \subset \mathbb{R}^2$, it is proposed to solve numerically the problem which consist to find $u(x,y)$ such that

$$\left\{ \begin{array}{rcl} \Delta u(x,y) & = & \mathcal{V}(u) \cdot u \quad \text{for all } (x,y) \in \Omega \subset \mathbb{R}^2, \\ u(x,y) & = & DBC \to +\infty \quad \text{for all } (x,y) \text{ on } \partial\Omega. \end{array} \right. \tag{12}$$

9.4.1 Space discretization

Let $\mathcal{T}_h$ be the triangulation of Ω and

$$V_h = \{v_h \in C^0(\bar{\Omega}); v_h|_T \in \mathbb{P}_1(T), \forall T \in \mathcal{T}_h, v_h = p \text{ on } \partial\Omega\}, p \longrightarrow +\infty.$$

Then the approximation of the variational formulation will be :
Find $u_h \in V_h$ such that $\forall v_h \in V_h$ we have :

$$\langle \Delta u_h; v_h \rangle = \langle \mathcal{V}(u_h) \cdot u_h; v_h \rangle$$

thus

$$-\langle \nabla u_h; \nabla v_h \rangle = \langle \mathcal{V}(u_h) \cdot u_h; v_h \rangle \tag{13}$$

In order to solve numerically the non linear term in (13), we will use a semi-implicit scheme such as :

$$-\left\langle \nabla u_h^{n+1}; \nabla v_h \right\rangle = \left\langle \mathcal{V}(u_h^n) \cdot u_h^{n+1}; v_h \right\rangle,$$

and then we solve our problem by the fixed point method in this way :

`Set`	$u_h^0 = u_0 = DBC, p = 0$
`Set`	$\mathcal{V}(u_h^n) = \mathcal{V}(u_0)$
`Set`	$err = 1.$
`while` $(err > 1e-10)$	
	`Solve` $-\left\langle \nabla u_h^{p+1}; \nabla v_h \right\rangle = \langle \mathcal{V}(u_h^n) \cdot u_h^p; v_h \rangle$
	`Compute` $err = \|u_h^n - u_h^{p+1}\|_{L^2}$
	`set` $\mathcal{V}(u_h^n) = \mathcal{V}(u_h^{p+1})$
	$p = p + 1;$
`End while`	

In order to test the convergence of this method we will study the rate of convergence in space (cf. Table 3) for an application of (12), where $R = 1, \mathcal{V}(u) = u, DBC = 0$ or $DBC = 50$.
The system to be solved is then

$$\Delta u(x,y) - u^2 = f \quad \text{for all } (x,y) \in \Omega = B(O,1) \subset \mathbb{R}^2, \tag{14}$$

$$u(x,y) = DBC \quad \text{for all } (x,y) \text{ on } \partial\Omega. \tag{15}$$

In this case, we will the following exact solution :

$$u_{ex} = DBC + \sin(x^2 + y^2 - 1).$$

Then, we compute the corresponding right hand side $f(x,y)$ using **Maple** such as :

We present here the corresponding script to compute the rate of convergence in space of the code solving the Elliptic non linear equation (14):

```
> u := unapply(DBC + sin(x^2 + y^2 - 1), x, y);
```

$$u := (x, y) \rightarrow DBC + \sin\left(x^2 + y^2 - 1\right) \tag{1}$$

```
> f := diff(u(x, y), x, x) + diff(u(x, y), y, y) - u(x, y)^2;
```

$$f := -4\sin\left(x^2 + y^2 - 1\right)x^2 + 4\cos\left(x^2 + y^2 - 1\right) - 4\sin\left(x^2 + y^2 - 1\right)y^2 - \left(DBC + \sin\left(x^2 + y^2 - 1\right)\right)^2 \tag{2}$$

```
> with(CodeGeneration);
```

$$[C, Fortran, IntermediateCode, Java, LanguageDefinition, Matlab, Names, Save, Translate, VisualBasic] \tag{3}$$

```
> Matlab(%%, resultname = "f");
f = -0.4e1 * sin((x ^ 2 + y ^ 2 - 1)) * (x ^ 2) + 0.4e1 * cos((x ^
2 + y ^ 2 - 1)) - 0.4e1 * sin((x ^ 2 + y ^ 2 - 1)) * (y ^ 2) -
(DBC + sin((x ^ 2 + y ^ 2 - 1))) ^ 2;
```

```
int nref=7;
real[int] L2error(nref); // initialize the L2 error array
for (int n=0;n<nref;n++) {
    int N=2^(n+4);          // space discretization
    real R=1.; // radius
    border C(t=0.,2.*pi){x=R*cos(t);y=R*sin(t);label=1;};
    mesh Th=buildmesh(C(N));
    fespace Vh(Th,P1);
    real DBC=0.;
    Vh uh, uh0=DBC, V=uh0, vh;
    Vh uex=DBC+sin((x ^ 2 + y ^ 2 - 1));
    Vh f=-0.4e1 * sin((x ^ 2 + y ^ 2 - 1)) * (x ^ 2) + 0.4e1 *
       ➥cos((x ^ 2 + y ^ 2 - 1)) - 0.4e1 * sin((x ^ 2 + y ^ 2 -
       ➥1)) * (y ^ 2) - (DBC + sin((x ^ 2 + y ^ 2 - 1))) ^ 2;
    macro Grad(u)[dx(u),dy(u)]//
    problem ELLNL(uh,vh) =
            - int2d(Th)(Grad(uh)'*Grad(vh))   // bilinear term
            - int2d(Th) ( uh*V*vh )           // non linear term
            - int2d(Th)( f*vh )               // right hand side
            + on(1,uh=DBC);                   // Dirichlet B.C.
    real err=1.; // for the convergence
    while (err >= 1e-10){
        ELLNL;
        err=sqrt(int2d(Th)((uh-V)^2));
        V=uh;        // actualization
    }
    L2error[n]= sqrt(int2d(Th)((uh-uex)^2));
}
for(int n=0;n<nref;n++)
```

```
    cout << "L2error " << n << " = "<<  L2error[n] <<endl;
for(int n=1;n<nref;n++)
    cout <<"convergence rate = "<< log(L2error[n-1]/L2error[n])/
      ➥log(2.)  <<endl;
```

N_n	$E(u,h_n)$, DBC=0	$r(u,h_n)$, DBC=0	$E(u,h_n)$, DBC=50	$r(u,h_n)$, DBC=50
16	0.0159388	-	0.00610357	-
32	0.00455562	1.80683	0.00244016	1.32268
64	0.00118025	1.94855	0.000767999	1.6678
128	0.000335335	1.81542	0.000210938	1.86429
256	8.6533e-05	1.95428	5.67798e-05	1.89337
512	1.9715e-05	2.13395	1.40771e-05	2.01203
1024	4.90847e-06	2.00595	3.56437e-06	1.98163

Table 3: L^2 norm of the error and the rate of convergence.

9.5 Rate of convergence for the Heat equation

Let $\Omega =]0,1[^2$, we want to solve the Heat equation :

$$\begin{cases} \dfrac{\partial u}{\partial t} - \mu \cdot \Delta u &= f(x,y,t) \quad \text{for all } (x,y) \in \Omega \subset \mathbb{R}^2, t, \mu \in \mathbb{R}^+, \\ u(x,y,0) &= u_0(x,y), \\ u &= 0 \quad \text{on } \partial\Omega. \end{cases} \tag{16}$$

9.5.1 Space discretization

Let $\mathcal{T}_h$ be the triangulation of Ω and

$$V_h = \{v_h \in C^0(\bar{\Omega}); v_h|_T \in \mathbb{P}_1(T), \forall T \in \mathcal{T}_h, v_h = 0 \text{ on } \partial\Omega\}.$$

Then the approximation of the variational formulation will be :
Find $u_h \in V_h$ such that $\forall v_h \in V_h$ we have :

$$\left\langle \frac{\partial u_h}{\partial t}; v_h \right\rangle - \langle \mu \cdot \Delta u_h; v_h \rangle = \langle f; v_h \rangle$$

Thus

$$\left\langle \frac{\partial u_h}{\partial t}; v_h \right\rangle + \mu \cdot \langle \nabla u_h; \nabla v_h \rangle = \langle f; v_h \rangle \tag{17}$$

9.5.2 Time discretization

We will use here a θ-scheme to discretize the Heat equation (17) as :

$$\left\langle \frac{u_h^{n+1} - u_h^n}{\Delta t}; v_h \right\rangle + \mu \cdot \theta \left\langle \nabla u_h^{n+1}; \nabla v_h \right\rangle + \mu \cdot (1-\theta) \left\langle \nabla u_h^n; \nabla v_h \right\rangle = \theta \cdot \left\langle f^{n+1}; v_h \right\rangle + (1-\theta) \cdot \left\langle f^n; v_h \right\rangle .$$

Therefore :

$$\left\langle \frac{u_h^{n+1}}{\Delta t}; v_h \right\rangle + \mu\cdot\theta \left\langle \nabla u_h^{n+1}; \nabla v_h \right\rangle = \left\langle \frac{u_h^{n}}{\Delta t}; v_h \right\rangle - \mu\cdot(1-\theta) \left\langle \nabla u_h^{n}; \nabla v_h \right\rangle + \theta\cdot\left\langle f^{n+1}; v_h \right\rangle + (1-\theta)\cdot\left\langle f^{n}; v_h \right\rangle . \tag{18}$$

To resolve (18) with FreeFem++, we will write it as a linear system of the form :

$$\mathcal{A}\mathbf{X} = \mathcal{B} \Longleftrightarrow \sum_{j=0}^{M-1} \mathcal{A}_{i,j} \cdot \mathbf{X}_j = \mathcal{B}_i \text{ for } i = 0; \ldots; M-1;$$

where, M is the degree of freedom, the matrix $\mathcal{A}_{i,j}$ and the arrays $\mathbf{X}_j$ and $\mathcal{B}_i$ are defining as :

$$\mathbf{X}_j = u_{j,h}^{n+1}, \qquad \mathcal{A}_{i,j} = \begin{cases} tgv = 10^{30} & \text{if } i \in \partial\Omega \text{ and } j = i \\ \displaystyle\int_\Omega \frac{\varphi_i\varphi_j}{\Delta t} + \mu\cdot\theta\cdot\nabla\varphi_i\nabla\varphi_j dxdy & \text{if } j \neq i \end{cases}$$

$$\mathcal{B}_i = \begin{cases} tgv = 10^{30} & \text{if } i \in \partial\Omega \\ \displaystyle\int_\Omega \frac{u_h^n\varphi_i}{\Delta t} - \mu\cdot(1-\theta)\cdot\nabla u_h^n\nabla\varphi_i + \left(\theta\cdot f^{n+1} + (1-\theta)\cdot f^n\right)\varphi_i dxdy & \text{otherwise} \end{cases}$$

We note that the θ-scheme is stable under the CFL condition (when $\theta \in [0, 1/2[$) :

$$\mu\frac{\Delta t}{\Delta x} + \mu\frac{\Delta t}{\Delta y} \leq \frac{1}{2\cdot(1-2\theta)}$$

In our test, we will consider that $\Delta x = \Delta y$ and that $CFL \in]0, 1]$, then when $\theta \in [0, 1/2[$, the θ-scheme is stable under this condition :

$$\Delta t \leq \frac{CFL\cdot(\Delta x)^2}{4\cdot\mu\cdot(1-2\theta)},$$

and for $\theta \in [1/2, 1]$, the θ-scheme is always stable.
We note also that due to the consistency error :

$$\varepsilon_{i,j}^n \leq c\Delta t\,|2\theta - 1| + \mathcal{O}\left(\Delta x^2\right) + \mathcal{O}\left(\Delta y^2\right) + \mathcal{O}\left(\Delta t^2\right),$$

the θ-scheme is consistent of order 1 in time and 2 in space when $\theta = 0$ (with the CFL condition) and when $\theta = 1$ (with $\Delta t = (\Delta x)^2$) and the θ-scheme is consistent of order 2 in time and in space when $\theta = 1/2$ (with $\Delta t = \Delta x$) (cf. Table 4).

Remark. When we use finite element, mass lumping is usual with explicit time-integration schemes (as when $\theta \in [0, 1/2[$). It yields an easy-to-invert mass matrix at each time step, while improving the CFL condition [Hug87]. In FreeFem++, mass lumping are defined as `int2d(Th,qft=qf1pTlump)`.

In order to test numerically the rate of convergence in space and in time of the θ-scheme, we will consider the following exact solution :

$$u_{ex} = \sin(\pi x)\cdot\sin(\pi y)e^{sin(t)}.$$

Then, we compute the corresponding right hand side $f(x, y)$ using **Maple** such as :

```
> u := unapply(sin(pi·x)·sin(pi·y)·exp(sin(t)), x, y, t);
```

$$u := (x, y, t) \to \sin(\pi x)\sin(\pi y)\, e^{\sin(t)} \tag{1}$$

```
> f := diff(u(x, y, t), t) − mu·diff(u(x, y, t), x, x) − mu·diff(u(x, y, t), y, y);
```

$$f := \sin(\pi x)\sin(\pi y)\cos(t)\, e^{\sin(t)} + 2\,\mu \sin(\pi x)\,\pi^2 \sin(\pi y)\, e^{\sin(t)} \tag{2}$$

```
> simplify(%);
```

$$\sin(\pi x)\sin(\pi y)\, e^{\sin(t)}\left(\cos(t) + 2\,\mu\,\pi^2\right) \tag{3}$$

```
> with(CodeGeneration);
```

[C, Fortran, IntermediateCode, Java, LanguageDefinition, Matlab, Names, Save, Translate, VisualBasic] (4)

```
> Matlab(%%, resultname = "f");
f = sin(pi * x) * sin(pi * y) * exp(sin(t)) * (cos(t) + 0.2e1 * mu
* pi ^ 2);
```

we remind that the rate of convergence in time is

$$r(u, dt_n, h_n) = \frac{\log\left(E(u, h_{n-1})/E(u, h_n)\right)}{\log\left(dt_{n-1}/dt_n\right)}, \forall n = 1, ..., \texttt{nref}$$

```
macro Grad(u)[dx(u),dy(u)]//
macro uex(t) (sin(pi*x)*sin(pi*y)*exp(sin(t)))//
macro f(t)  ( sin(pi * x) * sin(pi * y) * exp(sin(t)) * (cos(t) +
   ➥ 0.2e1 * mu * pi ^ 2) ) //

real t, dt, h,T=.1, mu=1., CFL=1., theta=0.;
int nref=4;
real[int] L2error(nref); // initialize the L2 error array
real[int] Dx(nref); // initialize the Space discretization array
real[int] DT(nref); // initialize the Time discretization array

for (int n=0;n<nref;n++) {
    int N=2^(n+4);
    t=0;
    h=1./N;
    Dx[n]=h;
    if (theta<.5)
        dt=CFL*h^2/4./(1.-2.*theta)/mu;
    else if (theta==.5)
        dt=h;
    else
        dt=h^2;
    DT[n]=dt;
    mesh Th=square(N,N);
    fespace Vh(Th,P1);
    Vh u,u0,B;
```

```
    varf a(u,v) = int2d(Th,qft=qf1pTlump)(u*v/dt + Grad(u)'*Grad(
       ➥v)*theta*mu) + on(1,2,3,4,u=0);
    matrix A = a(Vh,Vh);
    varf b(u,v) = int2d(Th,qft=qf1pTlump)(u0*v/dt - Grad(u0)'*
       ➥Grad(v)*(1.-theta)*mu ) + int2d(Th,qft=qf1pTlump)( (f(t+
       ➥dt)*theta+f(t)*(1.-theta))*v ) + on(1,2,3,4,u=0);
    u=uex(t);
    for (t=0;t<=T;t+=dt){
        u0=u;
        B[] = b(0,Vh);
        set(A,solver=sparsesolver);
        u[] = A^-1*B[];
    }
    L2error[n]=sqrt(int2d(Th)(abs(u-uex(t))^2));
}
for(int n=0;n<nref;n++)
    cout << "L2error " << n << " = "<<  L2error[n] <<endl;
for(int n=1;n<nref;n++){
    cout <<"Space convergence rate = "<< log(L2error[n-1]/L2error
       ➥[n])/log(Dx[n-1]/Dx[n])  <<endl;
    cout <<"Time convergence rate = "<< log(L2error[n-1]/L2error[
       ➥n])/log(DT[n-1]/DT[n])  <<endl;
}
```

N_n	16	32	64	128
$E(u,h_n)$, $\theta=0$	0.00325837	0.000815303	0.000203872	5.09709e-05
$r(u,h_n)$, $\theta=0$	-	1.99874	1.99967	1.99992
$r(u,dt_n,h_n)$, $\theta=0$	-	0.99937	0.999834	0.99996
$E(u,h_n)$, $\theta=1/2$	0.00325537	0.000819141	0.000203817	5.08854e-05
$r(u,h_n)$, $\theta=1/2$	-	1.99064	2.00684	2.00195
$r(u,dt_n,h_n)$, $\theta=1/2$	-	1.99064	2.00684	2.00195
$E(u,h_n)$, $\theta=1$	0.00323818	0.000807805	0.000201833	5.04512e-05
$r(u,h_n)$, $\theta=1$	-	2.0031	2.00084	2.0002
$r(u,dt_n,h_n)$, $\theta=1$	-	1.00155	1.00042	1.0001

Table 4: L^2 norm of the error and the rate of convergence in space and in time for different θ.

10 Conclusion

We presented here a basic introduction to FreeFem++ for the beginner with FreeFem++. For more information, go to the following link `http://www.freefem.org/ff++`.

Acknowledgements : This work was done during the CIMPA School - Caracas 16-27 of April 2012. I would like to thank Frédéric Hecht (LJLL, Paris), Antoine Le Hyaric (LJLL, Paris) and Olivier Pantz (CMAP, Paris) for fruitful discussions and remarks.

References

[Hug87] Thomas J. R. Hughes. The finite element method. Prentice Hall Inc., Englewood Cliffs, NJ, 1987. Linear static and dynamic finite element analysis, With the collaboration of Robert M. Ferencz and Arthur M. Raefsky. 29

[LucPir98] Brigitte Lucquin and Olivier Pironneau. *Introduction to Scientific Computing. Wiley*, 1998. PDF. 3