FreeFem++, a PDEs solver : Basic Tutorial

Yannick Deleuze

Laboratoire Jacques-Louis Lions, Sorbonne Universités, UPMC Univ Paris 06, France
Scientific Computing and Cardiovascular Lab, Department of ESOE, National Taiwan University
Center of Advanced Study in Theoretical Sciences, National Taiwan University

CASTS-LJLL Workshop
on Applied Mathematics and Mathematical Sciences
May 28, 2013

Outline

Outline

FreeFem++

FreeFem++ is an open source partial differential equation solver. It is developped in the Laboratory Jacques-Louis Lions (LJLL) of University Pierre et Marie Curie (Paris, France) to solve PDEs.

- finite elements method, including discontinuous FE spaces;
- 2D/3D
- built on C++, syntax from C++ adapted to mathematics
- automatic mesh generator (2D/3D);
- load, save mesh;
- unstructured meshes;
- mesh adaptation (2D);
- problem definition using variational formulation
- fast linear solvers;
- mpi tools for parallel computing.

Goal

Present a basic introduction to FreeFem++ for beginners with FreeFem++.

Outline

FreeFem++

- Download the current FreeFem++ from Frédéric Hecht website : `http://www.freefem.org/ff++/`
- Full documentation and examples at `http://www.freefem.org/ff++/ftp/freefem++doc.pdf`
- FreeFem++-cs, an integrated environment with graphical interface , from Antoine Le Hyaric website : `http://www.ann.jussieu.fr/~lehyaric/ffcs/index.htm`
- Taiwan Society for Industrial and Applied Mathematics (TWSIAM) FreeFem++ activity group : `http://homepage.ntu.edu.tw/~twhsheu/twsiamff++/freefem.html`
- Tutorial website : `http://www.ljll.math.upmc.fr/deleuze/freefem.html`

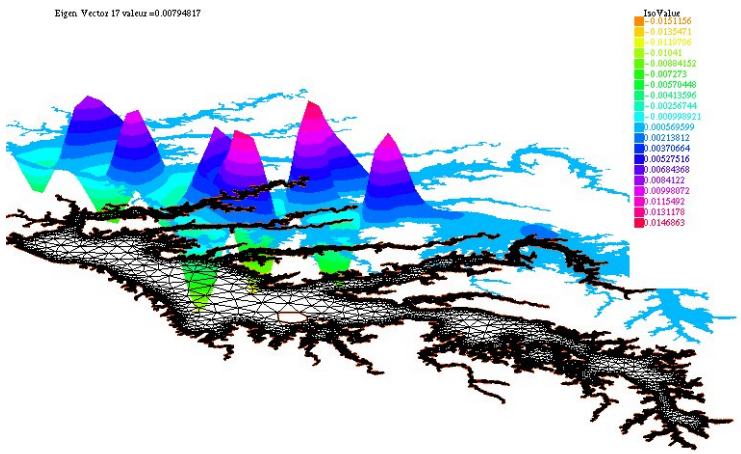
FreeFem++ Documentation

Freefem++

Third Edition, Version 3.20

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

F. Hecht



Keep FreeFem++ documentation close by.

Start using FreeFem++

Installation of FreeFem++ is really easy on Windows and MacOS platforms. On Linux, the software needs to be compiled from source code.

FreeFem++ scripts can be written in any text editor and saved to a ".edp" file. Lauching an '.edp' file depends on your operating system.

I recommend to start using FreeFem++ with FreeFem++-cs, an integrated environment for FreeFem++ (Windows, OS X, Linux). It provides a text editor with FreeFem++ syntax and highlighting of FreeFem++ compilation errors, a text output and visualisation. Developed by Antoine Lehyaric (LJLL).
`http://www.ann.jussieu.fr/~lehyaric/ffcs/index.htm`

Main idea of a FE code using FreeFem++

Main steps to a PDE using the FEM

- define the domain and generate the mesh.
- define the finite element space (basis functions) for the unknowns and test functions.
- set the problem : in FreeFem++, a problem must be given in it's weak form.
- solve the problem.
- visualise the solution.

Workflow of a FreeFem++ script

Generate the mesh Th

```
mesh Th = ... ;
```

Define the finite element space Vh

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

Define the variational problem P

```
problem P(u,v) = a(u,v) - l(f,v) + (Dirichlet boundary condition);
```

solve the problem P

```
P;
```

visualise the solution u

```
plot(u);
```

FreeFem++ syntax

```
x,y,z  // Cartesian coordinates
N.x, N.y, N.z //Normal vector components
int k = 10; // integer
real a=2.5; // real
bool b=(a<3.); /boolean
real [int] array(k); // array of k elements
array[][5]; //6th value of the array
mesh Th; //2d mesh
mesh3 Th3 //3d mesh
fespace Vh(Th,P1); //finite element space
Vh u=x; //finite element function
Vh3<complex> uc = x+ 1.i *y; //complex finite element function
fespace Xh(Th,[P2,P2,P1]);
Xh [u1,u2,p]; // a vectorial finite element function or array
u[]; //the array associated to FE function u
u(1.,0.1,3.); //value of u at point (1.,0.1,3.)
u[].max; // max of the array u
u[].min; // max of the array u
u[].sum; //sum of all elements of  u
u[].l1; // l1 norm of u
u[].l2; // l2 norm of u
u[].linfty; // linfinity norm of u
macro div(u,v) (dx(u)+dy(v))//  EOM
macro Grad(u) [dx(u),dy(u)]//  EOM
func f=x+y; //function
func real f(int i, real a) { .....; return i+a;}
varf a([u1,u2,p],[v1,v2,q])= int2d(...) + on(..)
matrix A = a(Vh,Vh,solver=UMFPACK); //rigid matric of the problem
real[int] b=a(0,V3h); // right hand side
u[] =A^-1*b; // solving u = A^-1*b
```

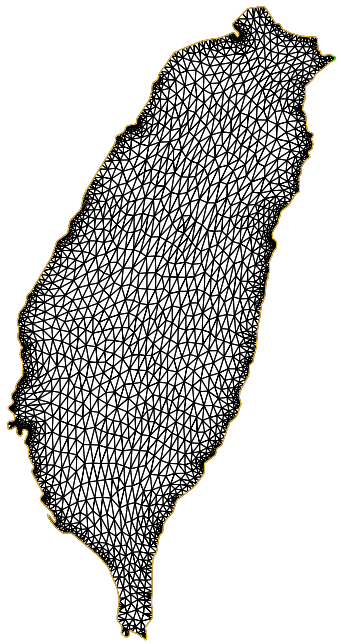
Outline

Outline

Laplace equation

An elastic membrane Ω is attached to a rigid support $\partial\Omega$, and a force $f(x)$ is exerted on the surface. The vertical membrane displacement $u(x)$ is obtained by solving Poisson's equation:

$$\begin{aligned} -\nabla^2 u &= f, \text{ in } \Omega \\ u_{|\partial\Omega} &= g. \end{aligned}$$



Outline

Integration by parts

Let Ω be an open compact domain of $\mathbb{R}^d$ with a smooth boundary $\partial\Omega$. Let $u, v \in \mathcal{C}^2(\bar{\Omega})$. Then

1. Gauss-Ostrogradsky theorem (divergence theorem)

$$\int_\Omega \operatorname{div} U \, \mathrm{d}x = \int_{\partial\Omega} U.n \, \mathrm{d}\Gamma$$

2. Integration by parts

$$\int_\Omega u \frac{\partial v}{\partial x_i} \, \mathrm{d}x = -\int_\Omega \frac{\partial u}{\partial x_i} v \, \mathrm{d}x + \int_{\partial\Omega} u \, v \, \mathrm{d}\Gamma$$

3. Green's first identity

$$\int_\Omega \nabla^2 u \, v \, \mathrm{d}x = -\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x + \int_{\partial\Omega} (\nabla u \cdot n) \, v \, \mathrm{d}\Gamma$$

4. Green's second identity

$$\int_\Omega \nabla^2 u \, v \, \mathrm{d}x - \int_\Omega u \, \nabla^2 v \, \mathrm{d}x = \int_{\partial\Omega} ((\nabla u \cdot n) \, v - (\nabla v \cdot n) \, u) \, \mathrm{d}\Gamma$$

Weak form

First, we derive the weak form of the equation.
We multiply the Laplace's equation by a smooth test function v and integrate over the entire domain. We apply Green's identity and it gives

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{dx} - \int_{\partial\Omega} (\nabla u \cdot n)\, v \, \mathrm{d}\Gamma = \int_\Omega f \, v \, \mathrm{dx}.$$

Due to the Dirichlet boundary conditions we choose v such that $v_{|\partial\Omega} = 0$. Let $V_\varphi = \{w \in H^1(\Omega) | w_{|\partial\Omega} = \varphi\}$, then the problem becomes

find $u \in V_g$ such that $\forall v \in V_0$

$$\int_\Omega \nabla u . \nabla v \mathrm{dx} - \int_\Omega f v \mathrm{dx} = 0$$

$a(u, v) = \int_\Omega \nabla u . \nabla v$ is the bilinear form and $l(v) = \int_\Omega f v$ is the linear form.

Finite element method

Freefem++ uses the most common finite elements for the discretisation of the continuous problem.
Consider the continuous piecewise polynomial of degree one (P_1) finite element dizcretization on the triangulation T_h of Ω_h.
The discrete problem becomes

find $u_h \in V_g^h$ such that $\forall v_h \in V_0^h$

$$\int_\Omega \nabla u_h . \nabla v_h - \int_\Omega f v_h = 0$$

Outline

Meshing

It order to solve the problem on a computer, we need to decretize it. The first step of the discretisation is to triangulate the physical domain Ω into the computational domain Th. We can use the keyword square to triangulate the domain and generate the mesh. We need to provide a number N of vertex placed on the borders.

square

```
1  int nn=10;
2  mesh Th = square (nn,nn);
3  plot(Th, wait=1, ps="meshsquare.eps");
```

rectangle

```
1  real x0=0, x1=10; real lx=x1-x0;
2  real y0=0, y1=5; real ly=y1-y0;
3  mesh Th1=square(nn,2*nn,[x0+(x1-x0)*x,y0+(y1-y0)*y]);
4  plot(Th1, wait=1,ps="meshrectangle.eps");
```

Note that we use the keyword plot to plot the mesh and the keyword wait to stop the program at each plot.

Meshing : borders

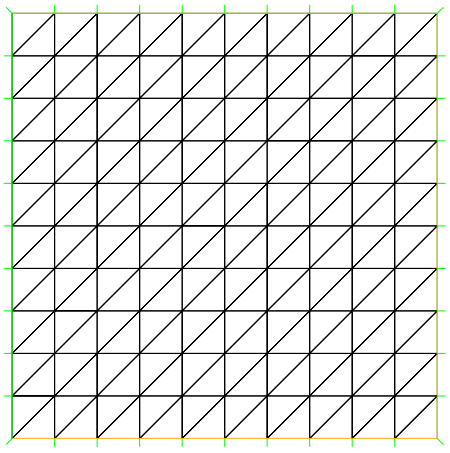


Figure : Square mesh generated with square

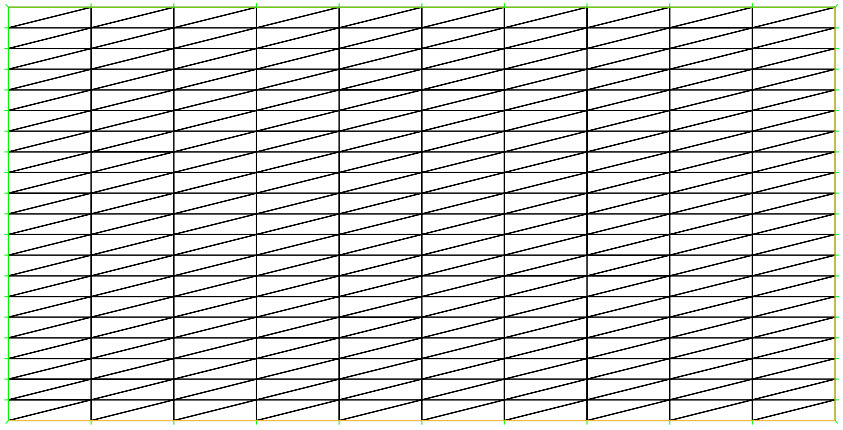


Figure : Rectangle mesh generated with square

Meshing : borders

In FreeFem++, the geometry of the domain can be easily defined by an analytic description of boundaries by pieces and the mesh is automatically generated. Each piece of boundary is described by it's parametric equation. The borders are declared with the type border. The keyword label defines a group of boundaries. The label type can be either an integer or a name. The mesh is then generated by the keyword buildmesh. For is border, we need to provide a number of vertex to be placed on it. The border has to be well defined in the counterclockwise direction. The borders can not cross each others.

```
1  border c1(t=x0,x1){ x=t; y=y0; label=1;};
2  border c2(t=y0,y1){ x=x1; y=t; label=2;};
3  border c3(t=x1,x0){ x=t; y=y1; label=3;};
4  border c4(t=y1,y0){ x=x0; y=t; label=4;};
5  mesh Th2= buildmesh(c1(lx*N/2)+c2(ly*N/2)+c3(lx*N/2)+c4(ly*N/2));
6  plot(Th2, wait=1, ps="mesh.eps");
```

We use the keyword ps to save the plot in an eps file.

Meshing : borders

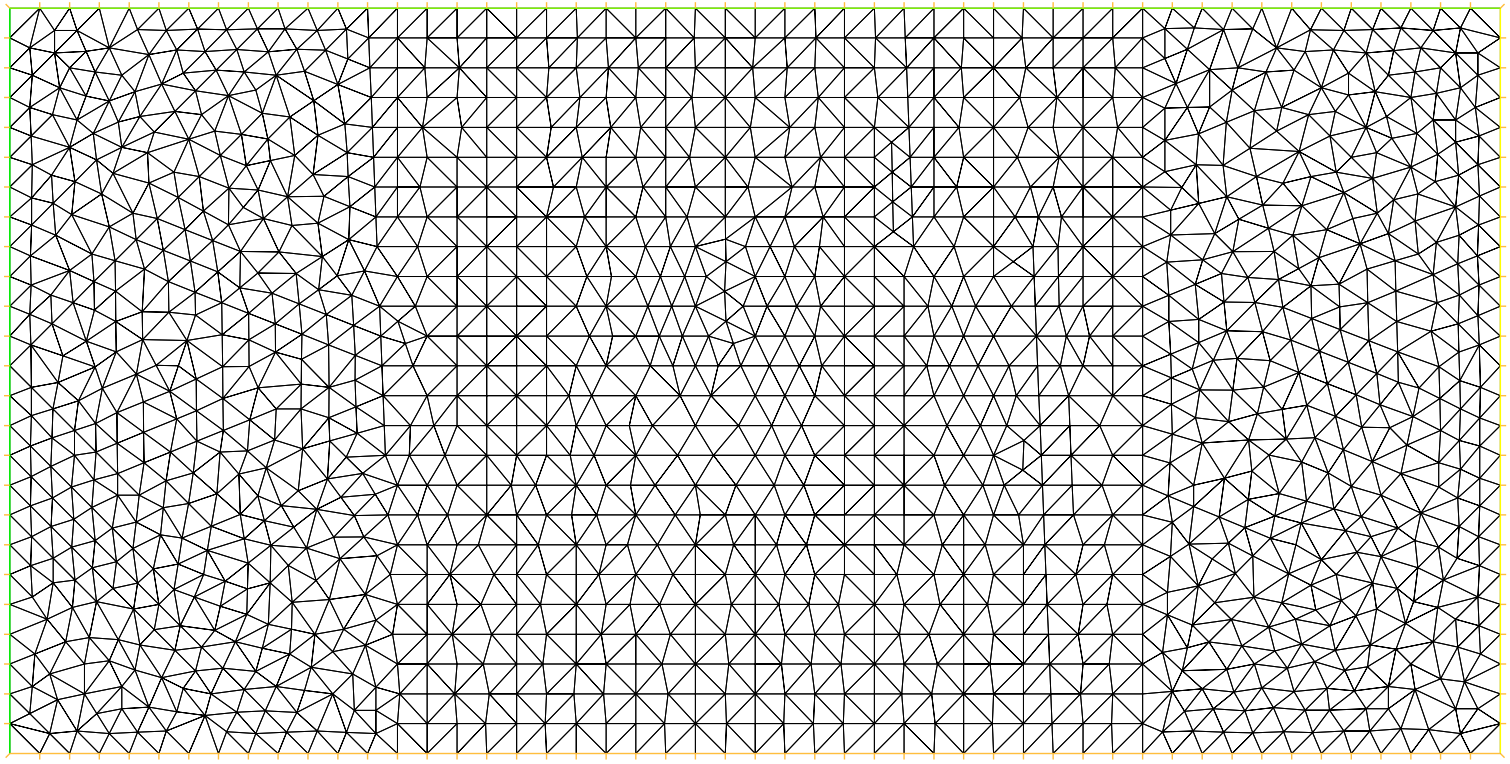


Figure : Mesh generated with border and build mesh

Meshing : borders

```
1  int circle=0;
2  border c(t=0,2*pi){x=1+3*cos(t);y=-1+3*sin(t);label=circle;}
3  mesh Th3=buildmesh(c(100));
4  plot(Th3,wait=1, ps="circle.eps", cmm="Circle mesh");
```

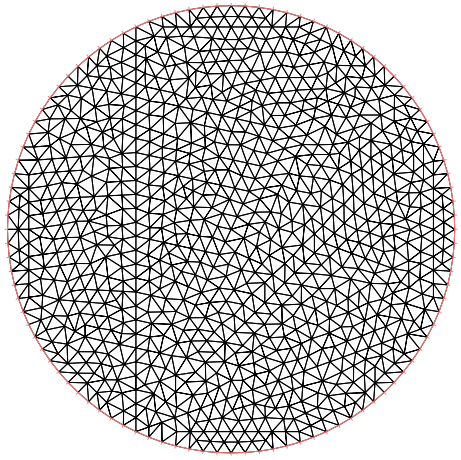
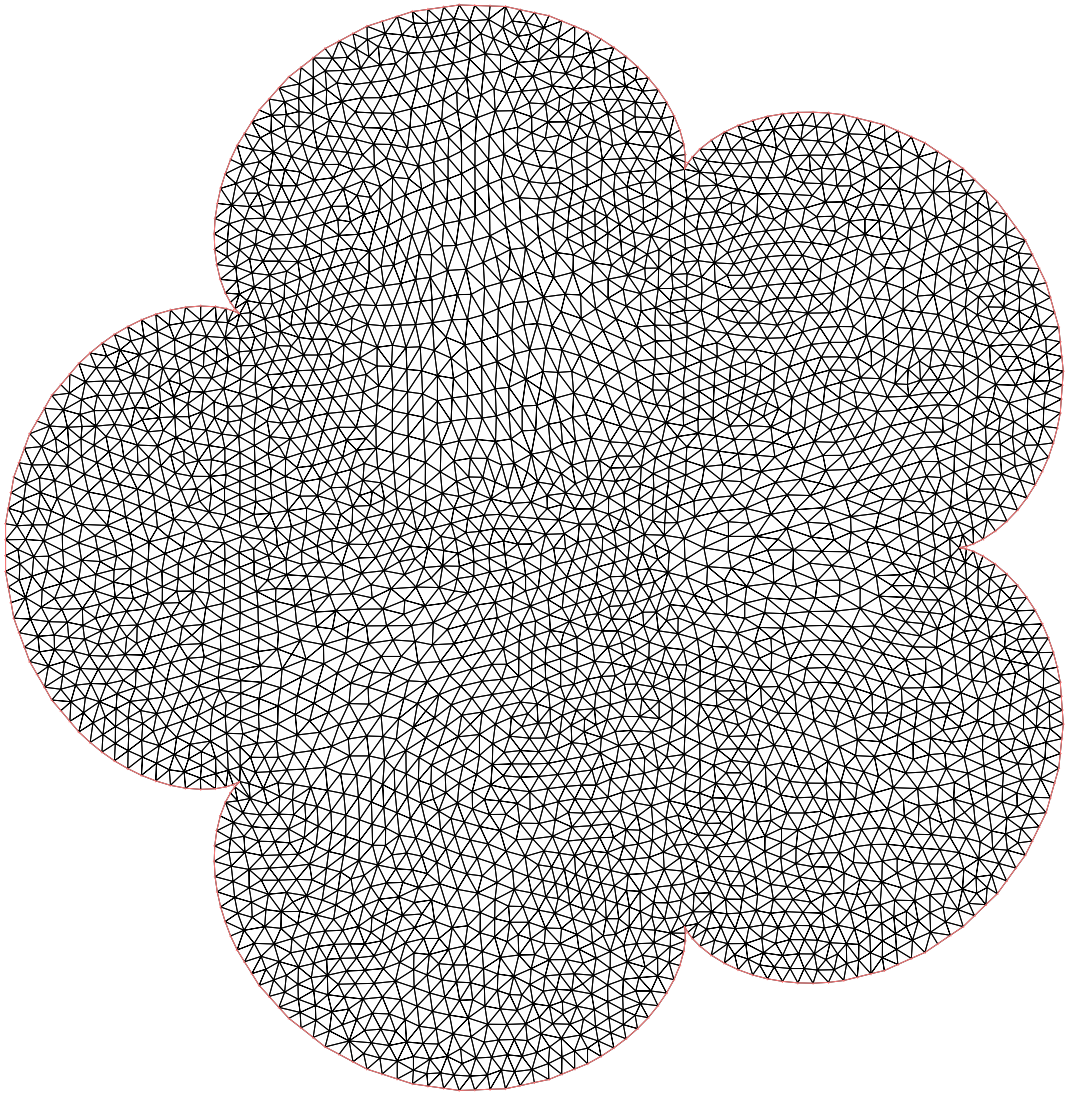


Figure : Mesh generated with border and buildmesh

Meshing : ranunculoid

Ranunculoid mesh

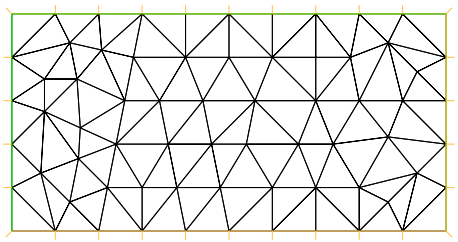


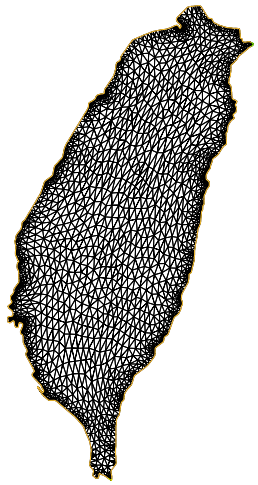
Meshing : save & load

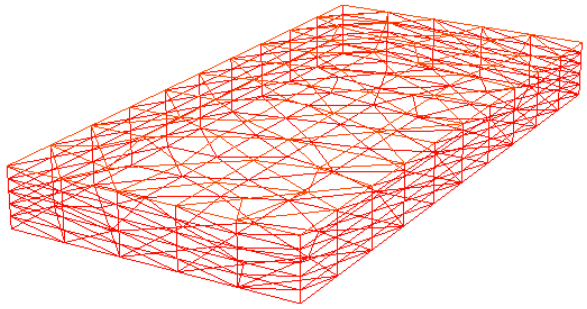
Use savemesh and readmesh to save or load a mesh file ".msh"

```
1   savemesh(Th2,"meshfile.msh");
```

```
1   mesh Th3 = readmesh("meshfile.msh");
```







Outline

Finite Element Spaces

The FEM approximates all function
$u(x,y) \simeq u_0\Phi_0(x,y) + ... + u_{M-1}\Phi_{M-1}(x,y)$ with finite element basis functions $\Phi_k(x,y)$.

FreeFem++ handles Lagrangian finite elements (P0,P1,P2,P3,P4), bubble elements (P1b,P2b), discontinuous P1, Raviart-Thomas, ... (see full documentation).

fespace defines the the discrete FE space for the unknowns and test functions;

$$Vh(Th, P1) = \left\{ w(x,y) | w(x,y) = \Sigma_{k=1}^{M} w_k \Phi_k(x,y), w_k \in \mathbb{R} \right\}$$

Declare the finite element space

```
1   fespace Vh(Th,P1);      // P1 FE space
```

Declare the unknown and the test functions

```
1   Vh u,v;                 // unkown and test function.
```

Finite Element Spaces

- **P0** piecewise constant discontinuous finite element. The degrees of freedom are the barycenter element value.
- **P1** piecewise linear continuous finite element. The degrees of freedom are the vertices values.
- **P2** piecewise continuous quadratic finite element. The degrees of freedom are the vertices values and the middle point of each edge.
- **P3** piecewise continuous cubic finite element. (need : load "Element_P3")
- **P4** piecewise continuous quartic finite element. (need : load "Element_P4")
- **P1b** piecewise linear continuous plus bubble. [1]
- **P2b** piecewise quadratic continuous plus bubble. [1]
- **P1dc** piecewise linear discontinuous finite element.
- **P2dc** piecewise P2 discontinuous finite element.

For full list, refer to FreeFem++ documentation.

[1] Brezzi, F. & Russo, A. Choosing Bubbles for Advection-Diffusion Problems. Mathematical Models and Methods in Applied Sciences 04, 571587 (1994). (The residual-free bubbles technique interprets the stabilization parameter as the mean value of the solution of a differential equation defined at the element level.)

Discrete variational form

In FreeFem++, the problem can be declared with the keyword

problem P(u,v) = a(u,v) - l(f,v) + (Dirichlet BC);

The bilinear form and the linear form should not be written under the same integral.

$$\int_\Omega \nabla u.\nabla v \rightarrow \text{int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v))}$$

$$\int_\Omega fv \rightarrow \text{int2d(Th)(f*v)}$$

where dx(u) $= \frac{\partial u}{\partial x}$, dy(u) $= \frac{\partial u}{\partial y}$. The keyword on defines the Dirichlet boundary condition on the borders 1, 2, 3 and 4.

```
1   func f=1;                  //  right hand side function
2   func g=0;                  //  boundary condition function
3
4   //  definion of  the problem
5   problem laplace(u,v) =
6         int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v) ) //bilinear form
7         - int2d(Th)( f*v )                     //  linear form
8         + on(1,2,3,4,u=g) ; // boundary condition form
```

Solve the problem

FreeFem++ solves the linear problem of the form

$$\Sigma_{j=0}^{M-1} A_{ij} u_j - F_i = 0, \quad i = 1, ..., M-1, \quad F_i = \int_\Omega f \Phi_i$$

FreeFem++ automatically builds the associate matrix and the second member vector.

```
1    laplace;
```

FreeFem++ solves elliptic equation. The user should specify it's own algorithms to solve parabolic and hyperbolic problems.

The user can specify the solver to solve the linear problem associated. FreeFem++ provides a large variety of linear direct and iterative solvers (LU, Cholesky, Crout, CG, GMRES, multi-frontal method UMFPACK, MUMPS (MUltifrontal Massively Parallel sparse direct Solver), SuperLU, ...).

Visualization

Use the keyword plot to plot the solution with useful options :

- ps= string to save the plot in a eps file;
- cmm = string to add comment
- if value= 1 plot the value of isolines
- if fill=1 color fill between isolines
- if wait=1 stop the program at this plot

```
1    plot(u,ps="Laplace.eps",value=1, wait=1, fill=1, cmm="Solution u of
         the Laplace equation in \Omega");
```

Outline

Outline

Heat problem

Freefem++ is able to solve time indecent problems but it is the user's responsibility to provide the algorithms.

$$\begin{cases} \frac{\partial u}{\partial t} - \nabla \cdot (\kappa \nabla u) = 0 \text{ in } (0, T) \times \Omega \\ u_{|_{t=0}} = U_0 \text{ in } \Omega \\ \nabla u \cdot n = U_N \text{ on } (0, T) \times \Gamma_N \\ u = U_D \text{ on } (0, T) \times \Gamma_D \end{cases}$$

Time discretization

First, let us consider a time discretization of the heat equation (35) . Time is discretize with finite difference schemes such as

θ - scheme [Raviart-Thomas 1998]

We discretize time $(0, T)$ in $M-1$ intervals $\Delta t = \frac{T}{M-1}$ and M points $(t_m)_{m=0}^{M-1}$ such that $U(t_m, x) = U^m(x)$.

$$\frac{\partial U}{\partial t} \approx \frac{U^{n+1} - U^n}{\Delta t} = \theta F(U^{n+1}) + (1-\theta) F(U^n)$$

- Euler explicit ($\theta = 0$)
- Euler implicit ($\theta = 1$)
- Crank-Nicolson ($\theta = \frac{1}{2}$)

Time discretization

Let us choose the Euler implicit quadrature. The heat problem in its semi-discrete form becomes

$$\frac{u^{n+1} - u^n}{\Delta t} - \nabla \cdot (\kappa \nabla u^{n+1}) = 0$$

Exercise : Implement the Crank-Nicolson scheme or any higher order scheme of your choice.

Spatial discretization

We multiply the Laplace's equation by a smooth test function v and integrate over the entire domain. We apply Green's identity and it gives

$$\int_\Omega \frac{1}{\Delta t} u^{n+1}\, v + \int_\Omega \kappa \nabla u^{n+1} \nabla v - \int_{\partial\Omega} \kappa (\nabla u^{n+1} \cdot n)\, v - \int_\Omega \frac{1}{\Delta t} u^n\, v = 0$$

Due to the Dirichlet boundary conditions we choose v such that $v_{|\partial\Omega} = 0$ on Γ_D and we use the Neumann boundary condition on Γ_N

$$\int_\Omega \frac{1}{\Delta t} u^{n+1}\, v + \int_\Omega \kappa \nabla u^{n+1} \nabla v - \int_{\partial\Gamma_N} \kappa\, U_N\, v - \int_\Omega \frac{1}{\Delta t} u^n\, v = 0$$

We can rewrite it, find $u^{n+1} \in V$ such that $\forall v \in V_0$

$$a(u, v) = l(v)$$

with

- $a(u, v) = \int_\Omega \frac{1}{\Delta t} u^{n+1}\, v + \int_\Omega \kappa \nabla u^{n+1} \nabla v$ the bilinear form,
- and $l(v) = -\int_{\partial\Gamma_N} \kappa\, U_N\, v - \int_\Omega \frac{1}{\Delta t} u^n\, v$ the linear form.

Heat equation on the unit square

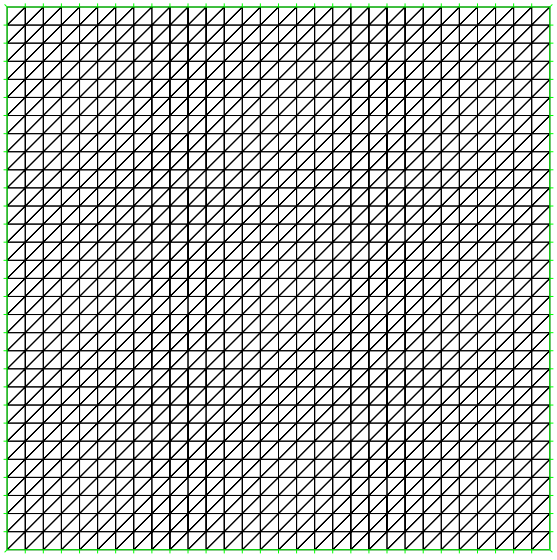
Let's consider the problem

$$\begin{cases} \frac{\partial u}{\partial t} - \nabla \cdot (\kappa \nabla u) = 0 \text{ in } (0, T) \times \Omega, \\ u_{|t=0} = \sin(\pi x) \sin(\pi y) \text{ in } \Omega, \\ u = 0 \text{ on } (0, T) \times \Gamma_D = \partial\Omega. \end{cases}$$

Then, the exact solution is

$$u(x, y, t) = \sin(\pi x) \sin(\pi y) \, \mathrm{e}^{-2\pi^2 t}$$

Generating mesh



```
1  int CD=10; // could be anything
2  real x0=0, x1=1;
3  real y0=0, y1=1;
4  int nn=30;
5  mesh Ths=square(nn,nn,[x0+(x1-x0)*x,y0+(y1-y0)*y]);
6  int[int] labels=[1,CD,2,CD,3,CD,4,CD]; // change the labels from
       1,2,3,4 to CD
7  Ths=change(Ths,label=labels);
8  plot(Ths, wait=1, ps="square.eps", cmm="square mesh");
```

Defining the problem

Parameters

```
int i; //time loop iterator
int M=10; //time iterations
real dt = 0.00001; //time step
```

Finite element space

```
fespace Vhs(Ths,P1);
Vhs u,u0,v; //unknow and test function
```

Initial condition

```
u = sin(pi*x)*sin(pi*y);
```

Variational problem

```
real kappa= 1;
macro F(u,v) kappa*(dx(u)*dx(v)+dy(u)*dy(v)) //eom
problem heat(u,v,init=i)
        = int2d(Ths)(u*v/dt)
        + int2d(Ths)(F(u,v))
        - int2d(Ths)(u0*v/dt)
    + on(CD,u=0)
;
```

Solving the system

Time loop

```
1           for(i=0;i<M; i++) {
2                   u0=u; //update from previous step time
3                   heat; //solve the linear system
4                   plot(u,wait=0,fill=1,value=1,cmm="solution at time "
                        +i*dt, viso=viso); // plot the solution at each
                         step time
5           }
```

Visualize the solution

Visualize the solution

```
1           real time= M*dt; //current time
2           plot(u,value=true,wait=1,fill=true,cmm="solution at time "+
                time, viso=viso);
```

Heat equation with Dirichlet and Neumann boundary conditions

Exercise: Now, let's consider the problem

$$\begin{cases} \frac{\partial u}{\partial t} - \nabla \cdot (\kappa \nabla u) = 0 \text{ in } (0,T) \times \Omega, \\ u_{|_{t=0}} = U_D = 20 \text{ in } \Omega, \\ \nabla u \cdot n = U_N = -1 \text{ on } (0,T) \times \Gamma_N, \\ u = U_D = 20 \text{ on } (0,T) \times \Gamma_D. \end{cases}$$

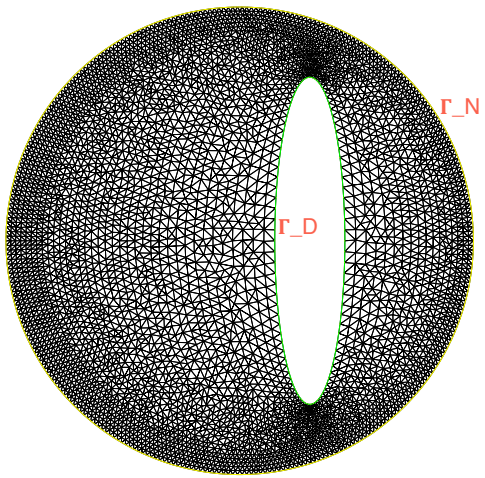


Figure : Ω

1D integration on the border Γ

$$\int_\Gamma \nabla u \cdot n v \rightarrow \text{int1d(Th)((dx(u)*N.x + dy(u)*N.y) v)}$$

Outline

Convection

Consider the convection equation in $\Omega \times (0, T)$

$$\frac{\partial u}{\partial t} + a \cdot \nabla u = f,\ u_{|\partial\Omega} = 0$$

Characteristic-Galerkin method

The convection equation can be discretised as [See FreeFem++ documentation]

$$\frac{u^{n+1}(x) - u^n(X^n)}{\Delta t} = f^n(x).$$

- Euler scheme: $X^n = x - a^n(x)\,\Delta t$
- Second order Runge-Kutta: $X^n = x - a^n(x - a^n(x)\,\frac{\Delta t}{2})\,\Delta t$

The problem becomes: find $u_h^{n+1} \in Vh_0$ such that $\forall v_h \in Vh_0$:

$$\int_\Omega \frac{1}{dt} u_h^{n+1} v_h - \int_\Omega \frac{1}{dt} u_h^n \circ X^n\, v_h - \int_\Omega f v_h = 0$$

Convection

Find $u_h^{n+1} \in Vh_0$ such that $\forall v_h \in Vh_0$:

$$\int_\Omega \frac{1}{dt} u_h^{n+1} v_h - \int_\Omega \frac{1}{dt} u_h^n(X^n)\, v_h - \int_\Omega f v_h = 0$$

FreeFem++ provides an interpolation operator
convect$(a^n, -\Delta t, u^n) \approx u^n(X^n) = u^n(x - a^n(x)\,\Delta t)$ such that the code for the problem is

```
problem convect (u,w)
        = int2d(Th) ( 1/dt*u*w)
        - int2d(Th)(1/dt*convect([a1,a2],-dt,up)*w)
        - int2d(Th)(f*w);
```

Operators

FreeFem++ provides many operators

- dx(u), dy(u), dxx(u), dyy(u), dxy(u)
- convect(a,dt,u)
- sin(u), exp(u), ...
- int1d(u), int2d(u)

but you can make your own

- macro div(u,v) (dx(u)+dy(v)) //EOM

Exercise : Write macros to enhance your codes.

Solvers

The user can provide the solver to solve the linear problem associated. FreeFem++ provides a large variety of linear direct and iterative solvers (LU, Cholesky, Crout, CG, GMRES, UMFPACK, MUMPS, SuperLU, ...).

A useful option is init. If init $\neq 0$, than the decomposition of the stiff matrix associated to the problem is reused.

Exercise :

- Change the solver used.
- Use the init parameter to save computational time.

Input-Output in FreeFem++

The user can import/export data from FreeFem++ such as meshes files, postscript images, text files. Other format are handle with FreeFem++ to interact with third parties softwares.

Terminal

The syntax write/read in the terminal is similar to C++ with the keywords cin, cout, <<, >>, endl.

```
1  int nn ;
2  cout << "number of nodes on the borders nn= " << endl; //writing in
       the terminal
3  cin >> nn; //read value from the terminal
4  mesh Tht=square(nn,nn);
```

Files

To write/read a file, first the user need to declare a variable with ofstream (output) or ifstream (input). Then, the syntax for input and output in a file remains the same.

```
1  ifstream fin("data.txt"); //declare an input from file
2  real ff, fg;
3  fin >> ff; //reads the 1srt value from the file
4  fin >> fg; //reads the 2nd  value from the file
5  func f=ff;
6  func g=fg;
```

Input-Output in FreeFem++

Mesh

To import/export mesh the function read mesh and save mesh are available. Refer to the documentation for full list of format one can import in FreeFem++.

```
savemesh(Tht,"meshfile.msh"); //export mesh
mesh Th = readmesh("meshfile.msh"); //import mesh
```

Finite element function (array of value)

To import/export the solution of a finite element function we use the following syntax

```
{
ofstream fout("solution.txt");
fout << u[]; //export the solution
} //use bracket to close to file (call the destructor)
```

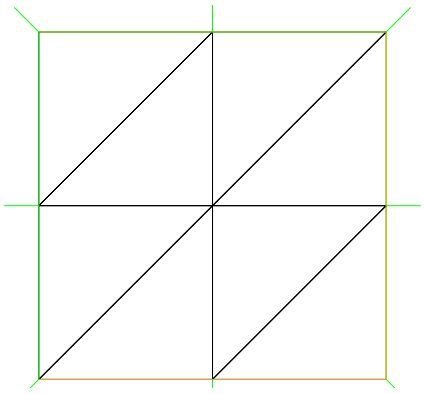
```
Vh u4plot;
{ifstream fu("solution.txt");
fu >> u4plot[];} //import a FE solution
plot(u4plot,ps="Laplace.eps"); //export image
```

Exercise : use input/output functionalities in your codes.

Mesh file

```
9 8 8
0 0 4
0.5 0 1
1 0 2
0 0.5 4
0.5 0.5 0
1 0.5 2
0 1 4
0.5 1 3
1 1 3
1 2 5 0
1 5 4 0
2 3 6 0
2 6 5 0
4 5 8 0
4 8 7 0
5 6 9 0
5 9 8 0
1 2 1
2 3 1
3 6 2
6 9 2
8 7 3
9 8 3
4 1 4
7 4 4
```

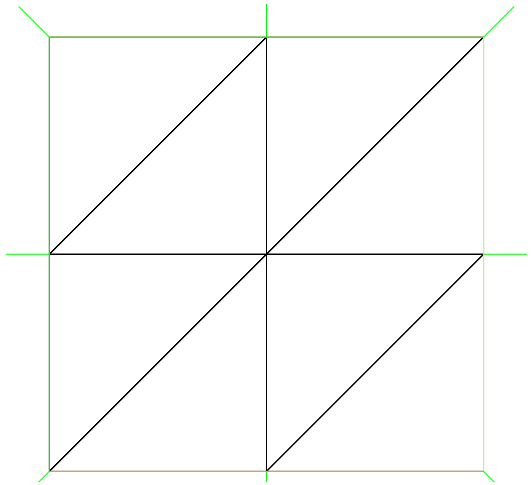
```
NbVertices NbElements NbBorderElem
//coordinates of vertices
x1 y1 label //1srt vertice
x2 y2 label //2nd vertice
...
//element with 3 vertices 1,2,3
v1 v2 v3 region
v1 v2 v4 region
v3 v4 v5 region
...
// boundary element is a segment
     of 2 vertices
v1 v2 label
v2 v3 label
...
```



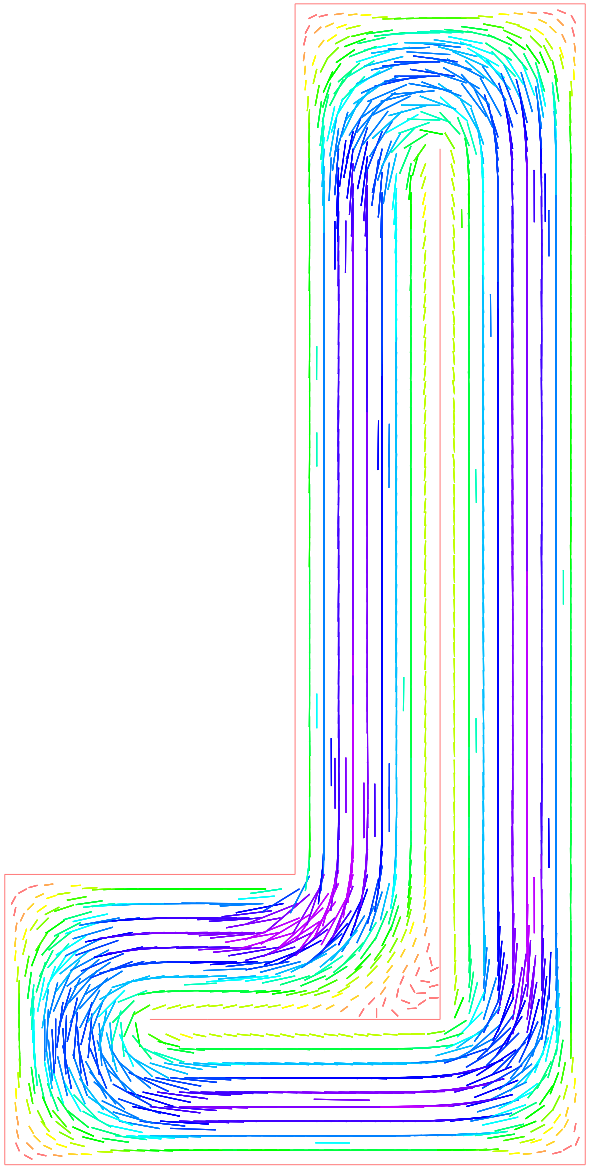
FE solution file

```
9
        6.25e-62        6.25e-32        6.25e-62        6.25e-32
                    0.0625
        6.25e-32        6.25e-62        6.25e-32        6.25e-62
```

```
NbVertices
        u(1)  u(2)  u(3)
        u(4)  u(5)  ...
        ...         u(NbVertices)
```



Pure convection



Pure convection

Generating the mesh

```
real Lx = 10;
real Ly = 20;

real[int] A(2), B(2), C(2), D(2), E(2), F(2), G(2), H(2), I(2);

A=[0,0]; B=[Lx,0]; C=[Lx,Ly]; D=[Lx/2.,Ly];
E=[Lx/2.,Ly/4.]; F= [0,Ly/4.];
G=[Lx/4.,Ly/8.]; H=[3*Lx/4.,Ly/8.]; I=[3*Lx/4.,7*Ly/8.];

border l1(t=0,1){x=(1-t)*A[0]+t*B[0]; y=(1-t)*A[1]+t*B[1]; label=0;}
border l2(t=0,1){x=(1-t)*B[0]+t*C[0]; y=(1-t)*B[1]+t*C[1]; label=0;}
border l3(t=0,1){x=(1-t)*C[0]+t*D[0]; y=(1-t)*C[1]+t*D[1]; label=0;}
border l4(t=0,1){x=(1-t)*D[0]+t*E[0]; y=(1-t)*D[1]+t*E[1]; label=0;}
border l5(t=0,1){x=(1-t)*E[0]+t*F[0]; y=(1-t)*E[1]+t*F[1]; label=0;}
border l6(t=0,1){x=(1-t)*F[0]+t*A[0]; y=(1-t)*F[1]+t*A[1]; label=0;}

border i1(t=0,1){x=(1-t)*G[0]+t*H[0]; y=(1-t)*G[1]+t*H[1]; label=0;}
border i2(t=0,1){x=(1-t)*H[0]+t*I[0]; y=(1-t)*H[1]+t*I[1]; label=0;}

int nn=4;
mesh Th=buildmesh(l1(nn*Lx)+l2(nn*Ly)+l3(Lx/2.*nn)+l4(3.*Ly/4.*nn)+
      l5(Lx/2.*nn)+l6(Ly/4.*nn)+i1(Lx/2.*nn)+i2(Ly*6./8.*nn));

plot(Th, wait=1, ps="pureconvect.eps");
```

Generating the velocity field

```
1   func fy=x-Lx/2;
2   fespace Uh(Th, P1b);
3   fespace Vh(Th, P1);
4   Uh a1, a2, a1h, a2h;
5   Vh p,ph;
6
7   solve Stokes([a1, a2, p],[a1h, a2h,ph])
8           = int2d(Th)(dx(a1)*dx(a1h)+dy(a1)*dy(a1h))
9           + int2d(Th)(dx(a2)*dx(a2h)+dy(a2)*dy(a2h))
10          + int2d(Th)(dx(p)*a1h+dy(p)*a2h)
11          + int2d(Th)(dx(a1)*ph+dy(a2)*ph)
12          - int2d(Th)(fy*a2h)
13          + on(0, a1=0, a2=0);
14
15  plot([a1, a2], ps="velocity.eps", wait=1);
```

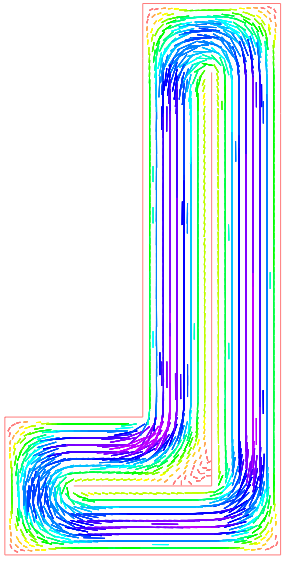


Figure :
Velocity field
$a = [a_2, a_2]$

Pure convection

```
1  Vh u = (sqrt((x-Lx/2.)^2+(y-Ly/16.)^2)<Ly/32.);
2  real masse0 = int2d(Th)(u); //initial mass
3  cout << "initial mass M0=" << masse0 << endl;
4  plot(u, nbiso=40, fill=1, wait=1);
5
6  real time = 0; //current time
7  real dt=1; // time step
8  real itmax=30; // max iterations
9  real[int] itplot(itmax), masse(itmax);
10
11 for (int it=0; it<itmax; it++){ //time iteration
12         itplot[it] = it;
13         u = convect([a1,a2], -dt, u);
14         time += dt;
15         plot(u, nbiso=40, fill=1);
16         cout << "Masse0 = " << masse0
17        << "; Masse(t) = " << masse[it] << endl;
18         plot(u, nbiso=60, fill=1, cmm="u("+time+")",
               ps="u_time="+time+".eps");
19 }
```

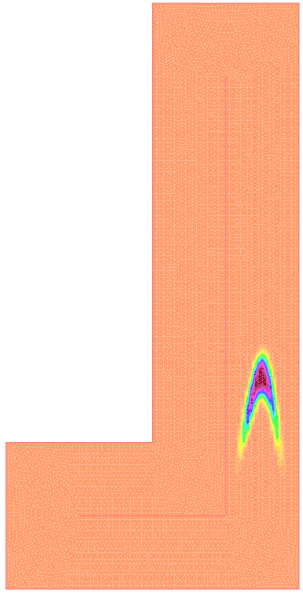


Figure :
Velocity field
$a = [a_2, a_2]$

Outline

Stationary Stokes equations

We consider the pseudo-compressible Stokes equations on the smooth spatial domain Ω :

$$\begin{aligned} -\frac{1}{\mathrm{Re}}\nabla^2 u + \nabla p &= f, \\ \nabla . u + \varepsilon p &= 0, \\ u &= g \text{ on } \Gamma_1, \\ u &= 0 \text{ on } \Gamma_2, \\ p &= 0 \text{ on } \Gamma_3, \\ \nabla u \cdot n &= 0 \text{ on } \Gamma_3. \end{aligned}$$

where $u = (u_1, u_2)$ is the velocity vector and p the pressure. ∇^2, here, stands for the vector Laplacian. In Cartesian coordinantes, $\nabla^2 u = (\nabla^2 u_1, \nabla^2 u_2)$.

Variational problem

To solve a problem in FreeFem++, let us write the variational formulation of the problem. We multiply the first equation by a smooth test function $v = (v_1, v_2)$, we get after applying the Green's formula

$$\frac{1}{\mathrm{Re}} \int_\Omega \nabla u_i \nabla v_i - \frac{1}{\mathrm{Re}} \int_{\partial\Omega} \nabla u_i \cdot n \, v_i - \int_\Omega p \partial_i v_i + \int_{\partial\Omega} p \, n_i v_i - \int_\Omega f_i v_i = 0.$$

And we multiply the second equation by a smooth function q and integrate over Ω. We get

$$\int_\Omega \nabla . u \, q + \int_\Omega \varepsilon p q = 0.$$

We consider the space $V_\phi = \left\{(w, r) \in [H_0^1(\Omega)]^2 \times L^2(\Omega) | w_{|\Gamma_1} = \phi, w_{|\Gamma_2} = 0\right\}$ and choose $(u, p) \in V_g$ and $(v, q) \in V_0$. The variational form reduces to

$$\frac{1}{\mathrm{Re}} \int_\Omega \nabla u_i \nabla v_i - \int_\Omega p \partial_i v_i - \int_\Omega f_i v_i = 0, \;\; i = 1, 2,$$

Variational problem

The problem is to find $(u, p) \in V_g$ such that for all $(v, q) \in V_0$. The variational form reduces to

$$\frac{1}{\mathrm{Re}} \int_\Omega (\nabla u_1 \nabla v_1 + \nabla u_2 \nabla v_2) - \int_\Omega p(\partial_x v_1 + \partial_y v_2)$$
$$+ \int_\Omega (\partial_x u_1 + \partial_y u_2)\, q + \int_\Omega \varepsilon p q - \int_\Omega (f_1 v_1 + f_2 v_2) = 0$$

```
1  //-----------------------------------------------------------
2  //Finite Element Spaces
3  //-----------------------------------------------------------
4  fespace Xh(Th,P2);
5  fespace Mh(Th,P1);
6  Xh u2,v2;
7  Xh u1,v1;
8  Mh p,q;
9
10 //-----------------------------------------------------------
11 //Stokes
12 //-----------------------------------------------------------
13 func g=4*y*(1-y);
14
15 solve Stokes ([u1,u2,p],[v1,v2,q],solver=UMFPACK) =
16  //Implement the stokes problem
17    + on(1,u1=g,u2=0)
18    + on(2,4,u1=0,u2=0)
19    + on(3,p=0);
```

Implementation of the Stokes problem

```
1   //-----------------------------------------------------------------
2   //Finite Element Spaces
3   //-----------------------------------------------------------------
4   fespace Xh(Th,P2);
5   fespace Mh(Th,P1);
6   Xh u2,v2;
7   Xh u1,v1;
8   Mh p,q;
9
10
11  //-----------------------------------------------------------------
12  //Stokes
13  //-----------------------------------------------------------------
14  func g=4*y*(1-y);//*(cos(0.1*i)+2); //pulsative flow
15
16  solve Stokes ([u1,u2,p],[v1,v2,q],solver=Crout) =
17      int2d(Th)( ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
18                 +  dx(u2)*dx(v2) + dy(u2)*dy(v2) )
19                 + p*q*(0.000001)
20                 + p*dx(v1)+ p*dy(v2)
21                 + dx(u1)*q+ dy(u2)*q
22                )
23    + on(1,u1=g,u2=0)
24    + on(2,4,u1=0,u2=0)
25    + on(3,p=0);
26
27  plot(coef=0.2,cmm=" [u1,u2] and p  ",p,[u1,u2], wait=1);
```

Outline

Convection-Diffusion Equation

The convection-diffusion equation describes the physical phenomena where particles or other physical quantities are transferred inside a physical system due to two processes: diffusion and convection.

$$\begin{aligned} \frac{\partial \phi}{\partial t} + \nabla.(u\phi) - \nabla.(\nu \nabla \phi) &= 0, \\ \phi_{|t=0} &= \phi^0, \\ \phi &= \phi_D \text{ on } \Gamma_D \\ \nabla \phi.\vec{n} &= u.\vec{n}\phi \text{ on } \Gamma_N. \end{aligned}$$

where $u = (u_1, u_2)$ is the velocity vector of a fluid vector, such that $\nabla.u = 0$ in Ω and ν is the diffusion coefficient.

In most common situation, the diffusion coefficient is constant and the velocity field describes an incompressible flow i.e. $\nabla.u = 0$. Then the problem simplifies to

$$\begin{aligned} \frac{\partial \phi}{\partial t} + u.\nabla \phi - \nu \nabla^2 \phi &= 0, \\ \phi_{|t=0} &= \phi^0, \\ \phi &= \phi_D \text{ on } \Gamma_D \\ \nabla \phi.\vec{n} &= u.\vec{n}\phi \text{ on } \Gamma_N. \end{aligned}$$

Diffusion-convection of a pollutant in a lake

$$\begin{aligned} \frac{\partial \phi}{\partial t} + u.\nabla\phi - \nu\nabla^2\phi &= 0, \\ \phi_{|t=0} &= \phi^0, \\ \nabla\phi.\vec{n} &= 0 \text{ on } \Gamma. \end{aligned}$$

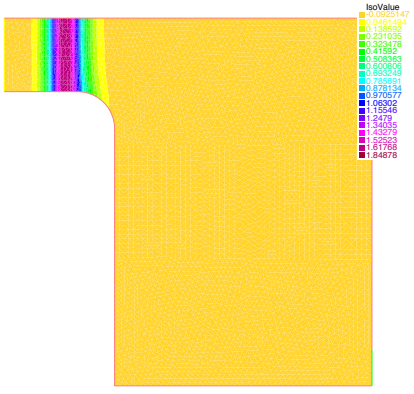


Figure : ϕ^0

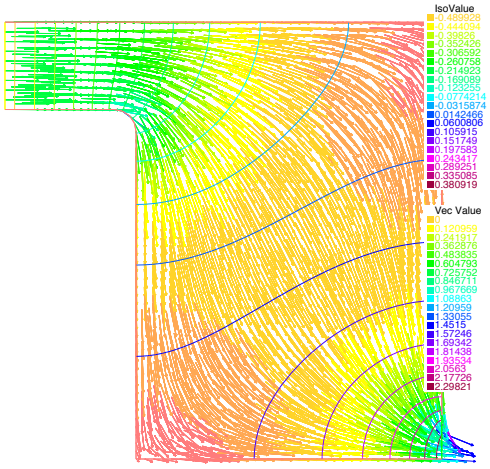


Figure : Flow potential Φ and vector fields $u = \nabla\Phi$ in a model lake with a river inlet and a river outlet. A pollutant will be released on one side of the lake and transported by the water flow.

Diffusion-convection of particles

We model the the evolution of the density of particles in a Stokes flow. The the evolution of the density of pollutant ϕ is described by the following problem : find the numerical solution $\phi_h \in V_h$ such that $\forall v_h \in V_h$

$$\int_\Omega \frac{\partial \phi_h}{\partial t} v_h - u.\nabla v_h \phi_h + \nu \nabla \phi_h . \nabla v_h = 0, \tag{1}$$

where $u = (u_1, u_2)$ is the velocity of the fluid and ν is the diffusivity of the particles.

Diffusion-convection of particles in a Stokes flow

```
1  //////////////////////////////////////////////////////
2  //Defining the time problem
3  //////////////////////////////////////////////////////
4  real nu=0.01; //difusivity
5  real dt=0.01; //time step
6  int M=300; //time iterations
7  int ite=0; //time iteration
8
9  //bilinear form
10 macro at(phih,vh) (1./dt*phih*vh + nu * ( dx(phih)*dx(vh) + dy(phih
       )*dy(vh) ) + (u1 * dx(phih) + u2 * dy(phih) ) * vh) //eom
11 //linear form
12 macro l(vh) (1./dt*phip*vh) //eom
13
14 problem CD (phi,w,init=ite) =
15           int2d(Th)  ( at(phi,w) )
16          -int2d(Th)(l(w));
```

Ex1 : Write the loop in time.

Ex2 : Write a Cranck-Nicolson scheme in time.

Diffusion-convection of particles in a Stokes flow

```
1  ////////////////////////////////////////////////////
2  //Solving the problem
3  ////////////////////////////////////////////////////
4  //initial condition : pollution
5  func initphi = 2*exp(-500*((x-0.5)^2+(y-0.5)^2));
6  phi= initphi;
7  plot(phi, fill=1, value=1, wait=1 , cmm="phi(0)");
8
9  for(ite=0; ite<M; ite++){
10           phip=phi;
11           CD;
12 plot(phi, wait=0, fill=1, cmm=" phi("+ite*dt+")");
```

Ex : Reduce the diffusivity. What do you observe ?

Exercise : convect operator and solver

FreeFem++ provides an interpolation operator convect for the $\partial_t u + ((u.\nabla)u)$.
We can rewrite the problem giving the discretisation in time :

find $\phi_h^{n+1} \in Vh$ such that $\forall v_h \in Vh$:

$$\int_\Omega \frac{1}{dt}(\phi_h^{n+1} - \phi_h^n \circ X^n).v_h + \nabla\phi_h^{n+1}.\nabla v_h - u.\nabla\phi_h^{n+1} v_h = 0$$

The term $\phi_h^n \circ X^n$ will be computed by the operator convect.

```
1  macro grad(u) [dx(u),dy(u)] //eom
2
3  problem CDconvect (phi,w) =
4            int2d(Th)  ( 1/dt*phi*w + nu*grad(phi)'*grad(w) )
5          - int2d(Th)(1/dt*convect([u1,u2],-dt,phip)*w)
6  ;
```

Exercise : different meshes

1. Use different meshes provided in the website to compute the convection-diffusion of particles in a stokes flow in different situation.
2. Change the value of the diffusivity to observe different results. Be sure you computation are stable using more grid points or the Characterisitic method.

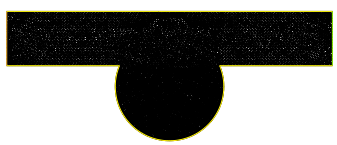


Figure : Aneurysm

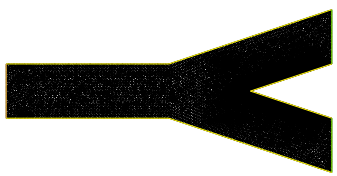


Figure : Bifurcation

Figure : Stenosis