FreeFem++ a software to solve PDE

F. Hecht

Laboratoire Jacques-Louis Lions

Université Pierre et Marie Curie

Paris, France

with O. Pironneau

http://www.freefem.org

mailto:hecht@ann.jussieu.fr

PLAN

- Introduction `Freefem++`
- syntaxe
- mesh generation
- three formulations of Poisson problem
- Poisson equation in 3D
- variationnal form (Matrix and vector)
- mesh adaptation with metrics
- error indicator computation
- an academic nolineair problem
- a free boundary problem
- exercise : minimal function problem
- exercise : Optimal Hoven, problem
- CFD 2d
- Navier Stokes Incompressible in 3D
- Domain decomposition example
- Mortar Method in FreeFem++
- variationnal Inequation
- Conclusion / Future

http://www.freefem.org/

Introduction

`FreeFem++` is a software to solve numerically partial differential equations (PDE) in $\mathbb{R}^2$) with finite elements methods. We used a user language to set and control the problem. The `FreeFem++` 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 no linear problem and time depend problem. You can solve coupled problem or problem with moving domain or eigenvalue problem, do mesh adaptation , compute error indicator, etc ...

FreeFem++ is a freeware and this run on Mac, Unix and Window architecture.

The main characteristics of FreeFem++ I/II (2D)

- Wide range of finite elements : linear and quadratic Lagrangian elements, discontinuous P1 and Raviart-Thomas elements, vectorial element , mini-element, ...
- 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.
- Definition of the problem (complex or real value) with the variational form with access to the vectors and the matrix if needed
- Discontinuous Galerkin formulation.

The main characteristics of FreeFem++ II/II (2D)

- Analytic description of boundaries, with specification by the user of the intersection of boundaries.
- Automatic mesh generator, based on the Delaunay-Voronoi algorithm.
- load and save Mesh, solution
- Mesh adaptation based on metric, possibly anisotropic, with optional automatic computation of the metric from the Hessian of a solution.
- LU, Cholesky, Crout, CG, GMRES, UMFPack sparse linear solver ; eigenvalue and eigenvector computation with ARPACK.
- Online graphics with OpenGL/GLUT, C++ like syntax.
- Link with other soft : modulef, emc2, medit, gnuplot, ...
- Dynamic linking to add functonality.
- Wide range of of examples : Navier-Stokes, elasticity, fluid structure, eigenvalue problem, Schwarz' domain decomposition algorithm, residual error indicator, ...

How to use

on Unix build a "yours.edp" file with your favorite editor : emacs, vi, nedit, etc. Enter `FreeFem++ yours.edp` or `FreeFem++-nw yours.edp` to execute your script. Remark, this application `FreeFem++` must be in a directory of your `PATH` shell variable.

on Window, MacOs X build a "yours.edp" file with your favorite text editor (raw text, not word text) : emacs, winedit, wordpad, bbedit, ... and click on the icon of the application `FreeFem++` and load you file via de open file dialog box or drag and drop the icon of your built file on the application `FreeFem++` icon.

Element of syntaxe I/II

```
x,y,z , label, N.x, N.y, N.z ,   //    current coordinate, label, normal
int i = 0;                                                    //    an integer
real a=2.5;                                                   //    a reel
bool b=(a<3.);
real[int]  array(10);                                  //    a real array of 10 value
mesh Th; mesh3 Th3;                                    //    a 2d mesh and a 3d mesh
fespace Vh(Th,P1);                                     //    a 2d finite element space;
fespace Vh3(Th3,P13d);                                 //    a 3d finite element space;

Vh u=x;                                          //    a finite element function or array
Vh<complex> uc = x+ 1.i *y;    //    complex valued FE function or array
u(.5,0.6);                          //    value of FE function u at point (.5,.6)
u[];                                //    the array associated to FE function u
u[][5];   //   6th value of the array ( numbering begin at 0 like in C)
func f=x+y;                                                   //    a formal line function
func real g(int i, real a) { .....; return i+a;}
```

Element of syntaxe : Like in C

The key words are reserved

The operator like in C exempt: ^ & |
+ - * / ^ // where a^b= a^b
== != < > <= >= & | // where a|b= a or b, a&b= a and b
= += -= /= *=

BOOLEAN: 0 <=> false , $\neq$ 0 <=> true

```
//   Automatic cast for numerical value : bool, int, reel, complex , so
func heavyside = real(x>0.) ;
```

```
for (int i=0 ;i<n ;i++) { ... ;}
if ( <bool exp> )  { ... ;} else {  ... ;} ;
while ( <bool exp> ) { ... ;}
break  continue  key words
```

Build Mesh 2D

First a 10×10 grid mesh of unit square $]0,1[^2$

```
mesh Th1 =  square(10,10) ;                                  //    boundary label:
plot(Th1,wait=1) ;                          //    1 bottom, 2 right, 3 top, 4, left
```

second a L shape domain $]0,1[^2 \backslash [\frac{1}{2},1[^2$

```
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 ;} ;
border d(t=0.5,1){x=0.5 ; y=t ;  label=4 ;} ;
border e(t=0.5,1){x=1-t ; y=1 ;  label=5 ;} ;
border f(t=0.0,1){x=0 ;   y=1-t ;label=6 ;} ;
plot(a(6) + b(4) + c(4) +d(4) + e(4) + f(6),wait=1) ;          //    to see the 6 borders
mesh Th2 =  buildmesh (a(6) + b(4) + c(4) +d(4) + e(4) + f(6)) ;
```

Get a extern mesh

```
mesh Th2("april-fish.msh") ;
```

build with `emc2`, `bamg`, modulef, etc...

Laplace equation

Let a domain Ω with a partition of $\partial\Omega$ in Γ_2, Γ_e.

Find u a solution in such that :

$$-\Delta u = 1 \text{ in } \Omega, \quad u = 2 \text{ on } \Gamma_2, \quad \frac{\partial u}{\partial \vec{n}} = 0 \text{ on } \Gamma_e \tag{1}$$

Denote $V_g = \{v \in H^1(\Omega)/v_{|\Gamma_2} = g\}$.

The Basic variationnal formulation with is : find $u \in V_2(\Omega)$, such that

$$\int_\Omega \nabla u.\nabla v = \int_\Omega 1v + \int_\Gamma \frac{\partial u}{\partial n} v, \quad \forall v \in V_0(\Omega) \tag{2}$$

Laplace equation I/III

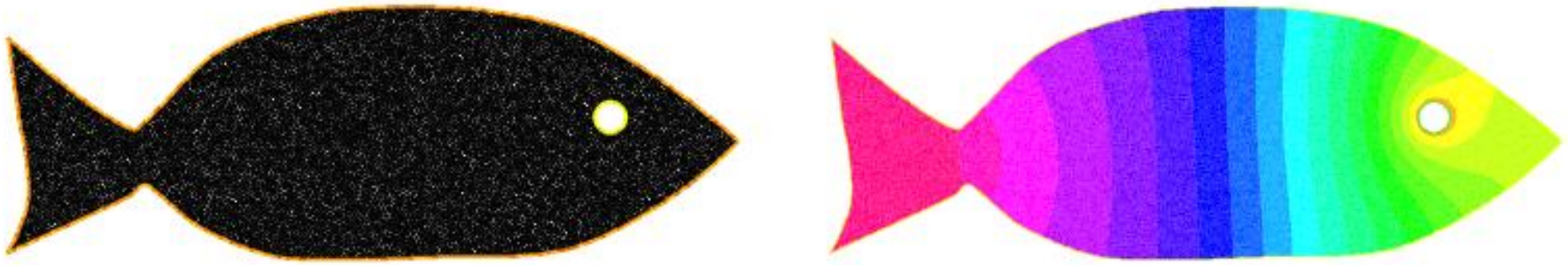
The finite element method is just : replace V_g with a finite element space, and the FreeFem++ code :

```
mesh Th("april-fish.msh") ;
fespace Vh(Th,P1) ;                                  //    define the P1 EF space

 Vh u,v ;

solve laplace(u,v,solver=CG) =
   int2d(Th)(  dx(u)*dx(v)+ dy(u)*dy(v)      )
   -  int2d(Th) ( 1*v)
  + on(2,u=2) ;                                                  //    int on γ2
plot(u,fill=1,wait=1,value=0,ps="april-fish.eps") ;
```

Laplace equation / figure



Execute fish.edp

Laplace equation (mixted formulation) II/III

Now we solve $-\Delta p = f$ on Ω and $p = g$ on $\partial\Omega$, with $\vec{u} = \nabla p$

so the problem becomes :

Find $\vec{u}, p$ a solution in a domain Ω such that :

$$-\nabla.\vec{u} = f,\ \vec{u} - \nabla p = 0 \quad \text{in } \Omega, \qquad p = g \text{ on } \Gamma = \partial\Omega \tag{3}$$

Mixted variationnal formulation

find $\vec{u} \in H_{div}(\Omega)$, $p \in L^2(\Omega)$, such that

$$\int_\Omega q\nabla.\vec{u} + \int_\Omega p\nabla.\vec{v} + \vec{u}.\vec{v} = \int_\Omega -fq + \int_\Gamma g\vec{v}.\vec{n}, \qquad \forall(\vec{v}, q) \in H_{div} \times L^2$$

Laplace equation (mixted formulation) II/III

```
mesh Th=square(10,10) ;
fespace Vh(Th,RT0) ;                    fespace Ph(Th,P0) ;
Vh [u1,u2],[v1,v2] ;                      Ph p,q ;
func f=1. ;
func g=1 ;
problem laplaceMixte([u1,u2,p],[v1,v2,q],solver=LU) =                          //
   int2d(Th)(  p*q*1e-10 + u1*v1 + u2*v2
               + p*(dx(v1)+dy(v2)) + (dx(u1)+dy(u2))*q )
 - int2d(Th) ( -f*q)
 - int1d(Th)( (v1*N.x +v2*N.y)*g) ;                          //    int on gamma
 laplaceMixte ;                                  //    the problem is now solved
 plot([u1,u2],coef=0.1,wait=1,ps="lapRTuv.eps",value=true) ;
 plot(p,fill=1,wait=1,ps="laRTp.eps",value=true) ;
```

Execute LaplaceRT.edp

Laplace equation (Garlerking discutinous formulation) III/III

```
//    solve -Δu = f on Ω and u = g on Γ
macro dn(u) (N.x*dx(u)+N.y*dy(u) )          //    def the normal derivative
mesh Th = square(10,10) ;                                  //    unite square
fespace Vh(Th,P2dc) ;                      //    Discontinous P2 finite element
    //    if pena = 0 => Vh must be P2 otherwise we need some penalisation
real pena=0 ;                              //    a parameter to add penalisation
func f=1 ;    func g=0 ;
Vh u,v ;

problem A(u,v,solver=UMFPACK) =                                          //
     int2d(Th)(dx(u)*dx(v)+dy(u)*dy(v)  )
   + intalledges(Th)(                      //    loop on all edge of all triangle
        ( jump(v)*average(dn(u)) - jump(u)*average(dn(v))
          + pena*jump(u)*jump(v) ) / nTonEdge  )
   - int2d(Th)(f*v)
   - int1d(Th)(g*dn(v)  + pena*g*v) ;
 A ; //    solve DG
```

Execute LapDG2.edp

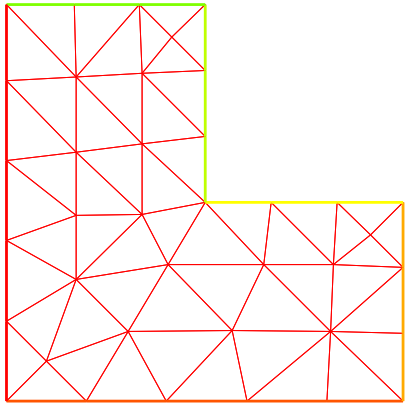
a corner singularity

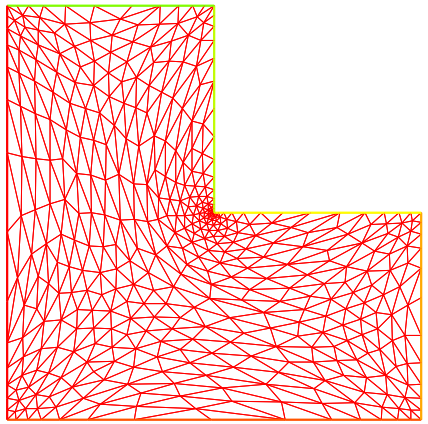
adaptation with metric

The domain is an L-shaped polygon $\Omega =]0,1[^2 \setminus [\frac{1}{2},1]^2$ and the PDE is

$$\text{Find } u \in H_0^1(\Omega) \text{ such that } \quad -\Delta u = 1 \text{ in } \Omega,$$

The solution has a singularity at the reentrant angle and we wish to capture it numerically.





example of Mesh adaptation

FreeFem++ corner singularity program

```
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;};
border d(t=0.5,1){x=0.5; y=t;  label=4;};
border e(t=0.5,1){x=1-t; y=1;  label=5;};
border f(t=0.0,1){x=0;   y=1-t;label=6;};


mesh Th =  buildmesh (a(6) + b(4) + c(4) +d(4) + e(4) + f(6));
fespace Vh(Th,P1);                  Vh u,v;          real error=0.01;
problem Probem1(u,v,solver=CG,eps=1.0e-6) =
     int2d(Th)(  dx(u)*dx(v) + dy(u)*dy(v))  -   int2d(Th)( v)
     + on(1,2,3,4,5,6,u=0);
int i;
for (i=0;i< 7;i++)
{  Probem1;                                                  //    solving the pde problem
   Th=adaptmesh(Th,u,err=error);                    //    the adaptation with Hessian of u
   plot(Th,wait=1);           u=u;      error = error/ (1000^(1./7.));   };
```

The plot of the Finite Basis Function

```
load "Element_P3"
mesh Th=square(1,1) ;
mesh th=square(150,150) ;
fespace Vh(Th,P2) ;

Vh vi=0 ;
for (int i=0 ;i<vi[].n ;++i)
 {
    vi[][i]=1 ;

    plot(vi,wait=0,cmm=" v"+i) ;
    savemesh(th,"mm",[x,y,vi*0.5]) ;                    //   save files for medit
    exec("medit mm ;rm mm.faces mm.points") ;
    vi[]=0 ;

 }
```

Execute plot-fb.edp

Laplace equation I 3d /III

The 3d FreeFem++ code (to day) :

```
mesh3 Th("poisson.mesh") ;
fespace Vh(Th,P13d) ;                              //    define the P1 EF space

Vh u,v ;

macro Grad(u) [dx(u),dy(u),dz(u)] //    EOM

solve laplace(u,v,solver=CG) =
   int3d(Th)( Grad(u)'*Grad(v) )  -  int3d(Th) ( 1*v)
  + on(2,u=2) ;                                                  //    on γ2
```

Matrix and vector

The first 3d FreeFem++ code :

```
mesh3 Th("poisson.mesh") ;
fespace Vh(Th,P13d) ;                          //    define the P1 EF space

Vh u,v ;

macro Grad(u) [dx(u),dy(u),dz(u)] //    EOM

varf vlaplace(u,v,solver=CG) =
   int3d(Th)( Grad(u)'*Grad(v) )  +  int3d(Th) ( 1*v)
  + on(2,u=2) ;                                          //    on γ2

matrix A= vlaplace(Vh,Vh,solver=CG) ;                  //    bilinear part
real[int] b=vlaplace(0,Vh) ;                          //    // linear part
u[] = A^-1*b ;
```

Execute Poisson3d.edp

A mathematical Poisson Problem with full Neumann BC. with 1D lagrange multiplier

The variationnal form is find $(u, \lambda) \in V_h \times \mathbb{R}$ such that

$$\forall (v, \mu) \in V_h \times \mathbb{R} \qquad a(u, v) + b(u, \mu) + b(v, \lambda) = l(v), \text{ where } b(u, \mu) = \int \mu u dx$$

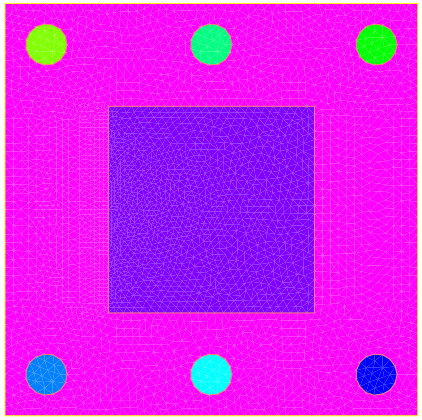
```
mesh Th=square(10,10);          fespace Vh(Th,P1);       //    P1 FE space
int n = Vh.ndof,  n1 = n+1;
func f=1+x-y;                 macro Grad(u) [dx(u),dy(u)]              //   EOM
varf va(uh,vh) = int2d(Th)( Grad(uh)'*Grad(vh)  );
varf vL(uh,vh) = int2d(Th)( f*vh )  ;  varf vb(uh,vh)=  int2d(Th)(1.*vh);
matrix A=va(Vh,Vh);
real[int] b=vL(0,Vh), B = vb(0,Vh);
real[int]  bb(n1),x(n1),b1(1),l(1); b1=0;
matrix AA = [ [ A ,  B ] , [ B', 0 ] ];    bb = [ b, b1];    //    blocks
set(AA,solver=UMFPACK);                  //    set the type of linear solver.
x = AA^-1*bb;     [uh[],l] = x;          //    solve the linear systeme
plot(uh,wait=1);                                  //    set the value
```

Execute Laplace-lagrange-mult.edp

An exercice : Oven problem

Find the power on the 6 resistors of an oven such that the temperature is close as possible to a given temperature in the region 6.

The equation are the stationary Head equation in 2d with classical Fourier boundary condition. the mesh of the domain :



"oven.msh"

let call the u_p the solution of

$$\begin{aligned} \nabla.K\nabla u_p &= \sum_{i=0}^{5} p_i * \chi_i \quad \text{in } \Omega \\ u + K\nabla u_p.n &= 0 \text{ on } \Gamma = \partial\Omega \end{aligned}$$

where χ_i is the characteristics function of the resistance i, $K = 10$ in region 6, $K = 1$ over where.

The problem is find the array p such that

$$p = \text{argmin} \int_{\Omega_6} (u_p - 100)^2 dx$$

Some remark

```
Xh[int]  ur(6) ;              //   to store the 6 finite element functions Xh
```

To day, FreeFem++ as only linear solver on sparse matrix. so a way to solve a full matrix problem is for example :

```
real[int,int] AP(6,6) ;                                  //    a full matrix
real[int] B(6),PR(6) ;                          //    to array (vector of size 6)

... bla bla to compute AP and B

matrix A=AP ;  //    build sparse data structure to store the full matrix
set(A,solver=CG) ;     //    set linear solver to the Conjuguate Gradient
PR=A^-1*B ;                                      //    solve the linear system.
```

The file name of the mesh is `oven.msh`, and the region numbers are 0 to 5 for the resitor, 6 for Ω_6 and 7 for the rest of Ω and the label of Γ is 1.

My solution, build the 6 basics function u_{e_i}

```
int  nbresitor=6 ;              mesh Th("oven.msh") ;
real[int] pr(nbresitor+2), K(nbresitor+2) ;
 K=1 ;      K[regi]=10 ;                                    //    def K
int regi=nbresitor, rege=nbresitor+1, lext=1 ;

macro Grad(u) [dx(u),dy(u)]                                  //   EOM
fespace Xh(Th,P2) ;                Xh u,v ;
int iter=0 ;
problem Chaleur(u,v,init=iter)
     =  int2d(Th)( Grad(u)'*Grad(v)* K[region]) + int1d(Th,lext)(u*v)
     +  int2d(Th)(pr[region]*v) ;

Xh[int]  ur(nbresitor) ;                                     //    to store the 6 u_{e_i}
for(iter=0 ;iter<nbresitor ;++iter)
{  pr=0 ;pr[iter]=1 ;
   Chaleur ;
   ur[iter][]=u[] ;
   plot(ur[iter],fill=1,wait=1) ;  }
```

Computation of the optimal value

```
real[int,int] AP(nbresitor,nbresitor) ;
real[int] B(nbresitor),PR(nbresitor) ;

Xh   ui = 100 ;
for(int i=0 ;i<nbresitor ;++i)
{
    B[i]=int2d(Th,regi)(ur[i]*ui) ;
    for(int j=0 ;j<6 ;++j)
       AP(i,j)= int2d(Th,regi)(ur[i]*ur[j]) ;
}
  matrix A=AP ; set(A,solver=UMFPACK) ;
  PR=A^-1*B ;
  cout << " P R = " << PR << endl ;
  u[]=0 ;
  for (int i=0 ;i<nbresitor ;++i)
    u[] += PR[i]*ur[i][] ;
```

Execute oven-cimpa.edp.edpxxxxx

Error indicator

For the Laplace problem

$$-\Delta u = f \quad \text{in } \Omega, \quad u = g \quad \text{on } \partial\Omega$$

the classical error η_K indicator [C. Bernardi, R. Verfürth] are :

$$\eta_K = \int_K h_K^2 |(f + \Delta u_h)|^2 + \int_{\partial K} h_e |[\frac{\partial u_h}{\partial n}]|^2$$

where h_K is size of the longest edge, h_e is the size of the current edge, n the normal.

Theorem : This indicator is optimal with Lagrange Finite element

$$c_0 \sqrt{\sum_K \eta_K^2} \leq ||u - u_h||_{H_1} \leq c_1 \sqrt{\sum_K \eta_K^2}$$

where c_0 and c_1 are two constant independent of h , if $\mathcal{T}_h$ is a regular family of triangulation.

Error indicator in FreeFem++

Test on an other problem :

$$10^{-10}u - \Delta u = x - y \quad \text{in } \Omega, \qquad \frac{\partial u}{\partial n} = 0 \quad \text{on} \Gamma$$

remark, the $10^{-10}u$ term is just to fix the constant.

We plot the density of error indicator :

$$\rho_K = \frac{\eta_K}{|K|}$$

```
fespace Nh(Th,P0) ;

varf indicator2(uu,eta) =
     intalledges(Th)( eta/lenEdge*square( jump( N.x*dx(u)+N.y*dy(u) ) ) )
    + int2d(Th)( eta*square( f+dxx(u)+dyy(u) ) ) ;
eta[] = indicator2(0,Nh) ;
```

Execute adaptindicatorP1.edp

Metric / unit Mesh

In Euclidean geometry the length $|\gamma|$ of a curve γ of $\mathbb{R}^d$ parametrized by $\gamma(t)_{t=0..1}$ is

$$|\gamma| = \int_0^1 \sqrt{< \gamma'(t), \gamma'(t) >} dt$$

We introduce the metric $\mathcal{M}(x)$ as a field of $d \times d$ symmetric positive definite matrices, and the length ℓ of Γ w.r.t $\mathcal{M}$ is :

$$\ell = \int_0^1 \sqrt{< \gamma'(t), \mathcal{M}(\gamma(t))\gamma'(t) >} dt$$

The key-idea is to construct a mesh where the lengths of the edges are close to 1 accordingly to $\mathcal{M}$.

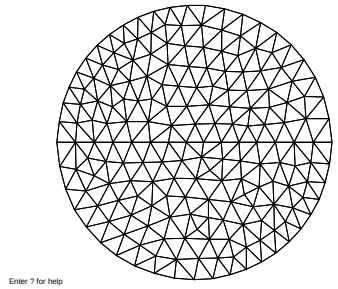
The main IDEA for mesh generation

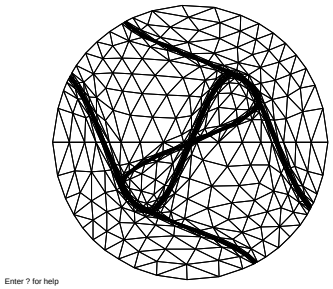
- The difficulty is to find a tradeoff beetween the error estimate and the mesh generation, because this two work are strongty different.
- To do that, we propose way based on a metric $\mathcal{M}$ and unit mesh w.r.t $\mathcal{M}$
- The metric is a way to control the mesh size.
- remark : The class of the mesh which can be created by the metric, is very large.

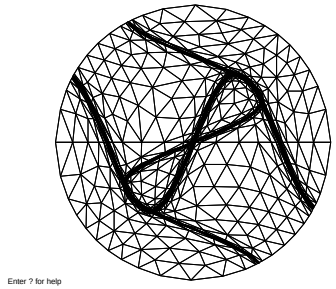
Example of mesh

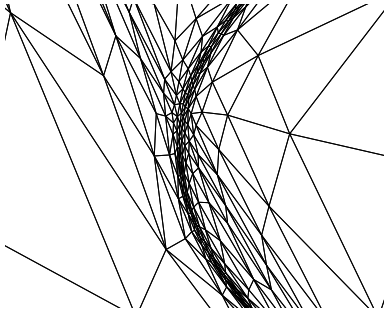
$$f_1 = (10 * x^3 + y^3) + atan2(0.001, (sin(5 * y) - 2 * x))$$

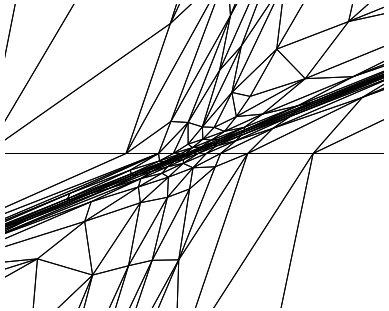
$$f_2 = (10 * y^3 + x^3) + atan2(0.01, (sin(5 * x) - 2 * y)).$$

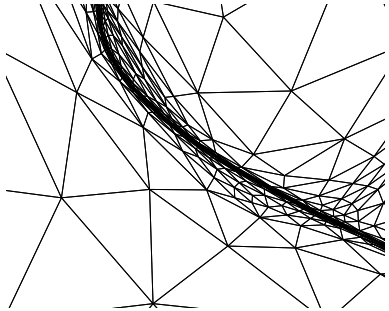












In dimension 2

With the $P1$ finite element the error interpolation is :

$$||u - \Pi_h u||_\infty^T \leq \frac{1}{2} \sup_{x,y,z \in T} ({}^t\vec{xy}\, |\mathcal{H}(z)|\, \vec{xy})$$

where $|\mathcal{H}|$ have the same eigenvectors and the eigenvalue of $|\mathcal{H}|$ is the abs of the eigenvalue of $\mathcal{H}$,

We take

$$\mathcal{M} = \frac{1}{\varepsilon_0} \qquad |\mathcal{H}|$$

and where ε_0 is the expected error.

Comparison : Metric and error indicator :

example of metric mesh : $u = yx^2 + y^3 + \tanh(10\ (\sin(5y) - 2x))$ and $\mathcal{M} = 50 \quad |\mathcal{H}(u)|$

DEMO I

Execute aaa-adp.edp

An academic problem

We propose the solve the following non-linear academic problem of minimization of a functional

$$J(u) = \frac{1}{2}\int_\Omega f(|\nabla u|^2) - u * b$$

where u is function of $H_0^1(\Omega)$. and where f is defined by

$$f(x) = a * x + x - ln(1+x), \quad f'(x) = a + \frac{x}{1+x}, \quad f''(x) = \frac{1}{(1+x)^2}$$

FreeFem++ definition

```
mesh Th=square(10,10) ;                          //    mesh definition of Ω
fespace Vh(Th,P1) ;                               //    finite element space
fespace Ph(Th,P0) ;                                  //    make optimization
```

the definition of f, f', f'' and b

```
real a=0.001 ;

func real f(real u) { return u*a+u-log(1+u) ; }
func real df(real u) { return a+u/(1+u) ;}
func real ddf(real u) { return 1/((1+u)*(1+u)) ;}
Vh b=1 ;                                                    //    to defined b
```

Newton Ralphson algorithm

Now, we solve the problem with Newton Ralphson algorithm, to solve the Euler problem $\nabla J(u) = 0$ the algorithm is

$$u^{n+1} = u^n - \left(\nabla^2 J(u^n)\right)^{-1} \nabla J(u^n)$$

First we introduce the two variational form `vdJ` and `vhJ` to compute respectively ∇J and $\nabla^2 J$

The variational form

```
  Ph ddfu,dfu ;            //   to store f'(|∇u|^2) and 2f''(|∇u|^2) optimization
  //    the variational form of evaluate dJ = ∇J
  //    ----------------------------------------
  //    dJ = f'()*( dx(u)*dx(vh) + dy(u)*dy(vh)
varf vdJ(uh,vh) = int2d(Th)( dfu *( dx(u)*dx(vh) + dy(u)*dy(vh) ) - b*vh)
  + on(1,2,3,4, uh=0) ;


  //    the variational form of evaluate ddJ = ∇^2 J
  //    hJ(uh,vh) = f'()*( dx(uh)*dx(vh) + dy(uh)*dy(vh)
  //                    + f''()( dx(u)*dx(uh) + dy(u)*dy(uh) )
  //                        * (dx(u)*dx(vh) + dy(u)*dy(vh))
varf vhJ(uh,vh) = int2d(Th)( dfu *( dx(uh)*dx(vh) + dy(uh)*dy(vh) )
   + ddfu *(dx(u)*dx(vh) + dy(u)*dy(vh) )*(dx(u)*dx(uh) + dy(u)*dy(uh)))
   + on(1,2,3,4, uh=0) ;
```

Newton Ralphson algorithm, next

```
//    the Newton algorithm
 Vh v,w;
 u=0;
 for (int i=0;i<100;i++)
  {
   dfu = df( dx(u)*dx(u) + dy(u)*dy(u) );                    //    optimization
   ddfu  = 2.*ddf( dx(u)*dx(u) + dy(u)*dy(u) );              //    optimization
   v[]= vdJ(0,Vh);                          //    v = ∇J(u), v[] is the array of v
   real res= v[]'*v[];                                      //    the dot product
   cout << i <<  " residu^2 = " <<  res  << endl;
   if( res< 1e-12) break;
   matrix H= vhJ(Vh,Vh,factorize=1,solver=LU);  //    build and factorize
   w[]=H^-1*v[];                                  //    solve the linear system
   u[] -= w[];
  }
  plot (u,wait=1,cmm="solution with Newton Ralphson");
```

Execute Newton.edp

A Free Boundary problem , (phreatic water)

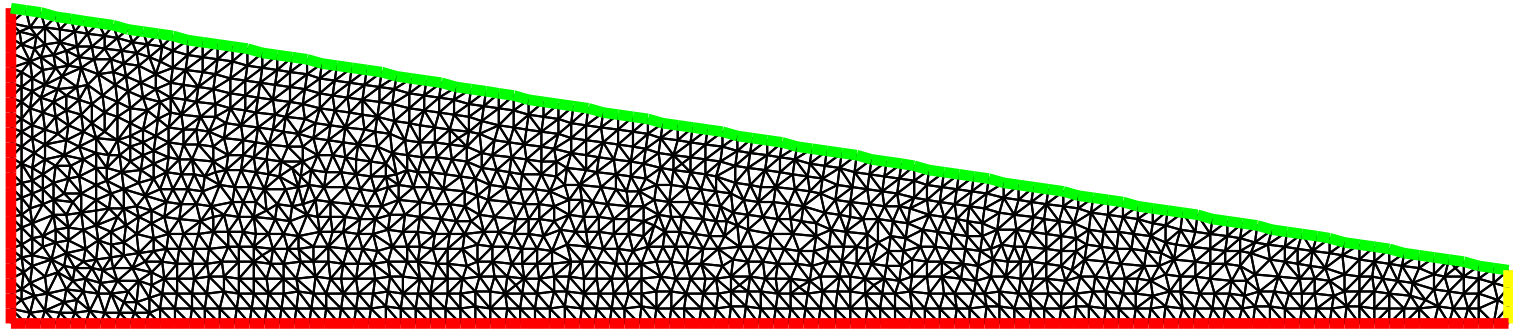
Let a trapezoidal domain Ω defined in FreeFem++ :

```
real L=10 ;                                                    //    Width
real h=2.1 ;                                                   //    Left height
real h1=0.35 ;                                                 //    Right height

border a(t=0,L){x=t ;y=0 ;label=1 ;} ;    //    bottom impermeable Γ_a
border b(t=0,h1){x=L ;y=t ;label=2 ;} ;    //    right, the source Γ_b
border f(t=L,0){x=t ;y=t*(h1-h)/L+h ;label=3 ;} ; //    the free surface Γ_f
border d(t=h,0){x=0 ;y=t ;label=4 ;} ;    //    Left impermeable Γ_d

int n=10 ;
mesh Th=buildmesh (a(L*n)+b(h1*n)+f(sqrt(L^2+(h-h1)^2)*n)+d(h*n)) ;
plot(Th,ps="dTh.eps") ;
```

The initial mesh



The problem is, find p and Ω such that :

$$\begin{cases} -\Delta p = 0 & \text{in } \Omega \\ p = y & \text{on } \Gamma_b \\ \dfrac{\partial p}{\partial n} = 0 & \text{on } \Gamma_d \cup \Gamma_a \\ \dfrac{\partial p}{\partial n} = \frac{q}{K} n_x & \text{on } \Gamma_f \quad (Neumann) \\ p = y & \text{on } \Gamma_f \quad (Dirichlet) \end{cases}$$

where the input water flux is $q = 0.02$, and $K = 0.5$. The velocity u of the water is given by $u = -\nabla p$.

algorithm

We use the following fix point method : let be, $k = 0$, $\Omega^k = \Omega$.

First step, we forgot the Neumann BC and we solve the problem : Find p in $V = H^1(\Omega^k)$, such $p = y$ onon Γ_b^k et on Γ_f^k

$$\int_{\Omega^k} \nabla p \nabla p' = 0, \quad \forall p' \in V \text{ with } p' = 0 \text{ on } \Gamma_b^k \cup \Gamma_f^k$$

With the residual of the Neumann boundary condition we build a domain transformation $\mathcal{F}(x, y) = [x, y - v(x)]$ where v is solution of : $v \in V$, such than $v = 0$ on Γ_a^k (bottom)

$$\int_{\Omega^k} \nabla v \nabla v' = \int_{\Gamma_f^k} (\frac{\partial p}{\partial n} - \frac{q}{K} n_x) v', \quad \forall v' \in V \text{ with } v' = 0 \text{ sur } \Gamma_a^k$$

remark : we can use the previous equation to evaluate

$$\int_{\Gamma^k} \frac{\partial p}{\partial n} v' = - \int_{\Omega^k} \nabla p \nabla v'$$

The new domain is : $\Omega^{k+1} = \mathcal{F}(\Omega^k)$

Warning if is the movement is too large we can have triangle overlapping.

```
problem Pp(p,pp,solver=CG) = int2d(Th)( dx(p)*dx(pp)+dy(p)*dy(pp))
  + on(b,f,p=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(p)*N.x+dy(p)*N.y))) ;
while(errv>1e-6)
{   j++ ;  Pp ;  Pv ;    errv=int1d(Th,f)(v*v) ;
     coef = 1 ;
    //    Here french cooking if overlapping see the example
    Th=movemesh(Th,[x,y-coef*v]) ;                          //    deformation
}
```

Execute freeboundary.edp

3D mesh of a Sphere/ I

```
load "tetgen"
mesh Th=square(10,20,[x*pi-pi/2,2*y*pi]);                    //    ]-pi/2, -pi/2[x]0,2π[
                                                  //    a parametrization of a sphere
func f1 =cos(x)*cos(y); func f2 =cos(x)*sin(y); func f3 = sin(x);
                          //    the 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^t DF
func m11=f1x^2+f2x^2+f3x^2;
func m21=f1x*f1y+f2x*f2y+f3x*f3y;
func m22=f1y^2+f2y^2+f3y^2;
func perio=[[4,y],[2,y],[1,x],[3,x]];
real hh=0.1;
real vv= 1/square(hh);
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
mesh3 Th3=tetgtransfo(Th,transfo=[f1,f2,f3]);
```

3D layer mesh of a Lac / II

```
load "glumesh"//    buildlayer
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,uz,p2 ;
int[int] rup=[0,2],  rdown=[0,1], rmid=[1,1] ;
func zmin= 2-sqrt(4-(x*x+y*y)) ;   func zmax= 2-sqrt(3.) ;
//    we get nn*coef layers
mesh3 Th=buildlayers(Th2,nn,
                     coef= max((zmax-zmin)/zmax,1./nn),
                     zbound=[zmin,zmax],
                     reffacemid=rmid,  reffaceup = rup,
                     reffacedown = rdown) ;                    //    label def
savemesh(Th,"Th.meshb") ;
exec("medit Th ;rm Th.meshb") ;
```

Execute Lac.edp

Stokes equation

The Stokes equation is find a velocity field $\boldsymbol{u} = (u_1, .., u_d)$ and the pressure p on domain Ω of $\mathbb{R}^d$, such that

$$\begin{aligned} -\Delta \boldsymbol{u} + \nabla p &= 0 \quad \text{in} \quad \Omega \\ \nabla \cdot \boldsymbol{u} &= 0 \quad \text{in} \quad \Omega \\ \boldsymbol{u} &= \boldsymbol{u}_\Gamma \quad \text{on} \quad \Gamma \end{aligned}$$

where $\boldsymbol{u}_\Gamma$ is a given velocity on boundary Γ.

The classical variationnal formulation is : Find $\boldsymbol{u} \in H^1(\Omega)^d$ with $\boldsymbol{u}_{|\Gamma} = u_\Gamma$, and $p \in L^2(\Omega)/\mathbb{R}$ such that

$$\forall \boldsymbol{u} \in H_0^1(\Omega)^d, \ \forall q \in L^2(\Omega)/\mathbb{R}, \qquad \int_\Omega \nabla \boldsymbol{u} : \nabla \boldsymbol{v} - p \nabla . v - q \nabla . u = 0$$

or now find $p \in L^2(\Omega)$ such than (with $\varepsilon = 10^{-10}$)

$$\forall \boldsymbol{u} \in H_0^1(\Omega)^d, \ \forall q \in L^2(\Omega), \int_\Omega \nabla \boldsymbol{u} : \nabla \boldsymbol{v} - p \nabla . v - q \nabla . u + \varepsilon p q = 0$$

An exercice in FreeFem++

The geometrical problem : Find a function $u : C^1(\Omega) \mapsto \mathbb{R}$ where u is given on $\Gamma = \partial\Omega$, (e.i. $u_{|\Gamma} = g$) such that the area of the surface S parametrize by $(x, y) \in \Omega \mapsto (x, y, u(x, y))$ is minimal.

So the problem is arg min$J(u)$ where

$$J(u) = \iint_\Omega \left\| \begin{pmatrix} 1 \\ 0 \\ \partial_x u \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ \partial_y u \end{pmatrix} \right\|_2 dxdy = \iint_\Omega \sqrt{1 + (\partial_x u)^2 + (\partial_y u)^2} dxdy$$

So the Euler equation associated to the minimisaton is :

$$\forall v / v_{|\Gamma} = 0 \quad : \quad DJ(u)v = -\int \frac{(\partial_x v \partial_x u + \partial_y v_y \partial_y u)}{\sqrt{1 + (\partial_x u)^2 + (\partial_y u)^2}} = 0$$

So find the solution for $\Omega =]0, \pi[^2[$ and $g(x, y) = cos(2 * x) * cos(2 * y)$. by using the Non Linear Conjugate gradient `NLCG` like in the example : `algo.edp` in `examples++-tutorial`.

Tools

Example of use of NLCG function :

```
   func real J(real[int] & xx)                          //    the functionnal to mininized
   {  real s=0;
     ...                          //    add code to copy xx array of finite element function
     return s; }
  func real[int] DJ(real[int] &xx)                              //    the grad of functionnal
    { ....                        //    add code to copy xx array of finite element function
     return xx; };                                    //    return of an existing variable ok
  ...
  NLCG(DJ,x,eps=1.e-6,nbiter=20,precon=matId);
```

Two useful operators on array :real[int]

```
   real[int]  a(10),b(10);
   ...
   int n= a.n                                          //    give the size of array a.
   a =  b? 1. : 0;                              //    a[i] = 1 if b[i] else a[i]=0. ∀i
```

To see the 3D plot of the surface with medit

```
 savemesh(Th,"mm",[x,y,u]);//   save mm.points and mm.faces file for medit
 exec("medit mm;rm mm.bb mm.faces mm.points");                //    call medit
```

My solution First the functionnal

```
func g=cos(2*x)*cos(2*y) ;                                  //    valeur au bord
mesh Th=square(20,20,[x*pi,y*pi]) ;                         //    mesh definition of Ω
fespace Vh(Th,P1) ;

func real J(real[int] & xx)                                 //    the functionnal to minimise
  { Vh u ;u[]=xx ;      //    to set finite element function u from xx array
    return int2d(Th)( sqrt(1 +dx(u)*dx(u) + dy(u)*dy(u) ) ) ; }

func real[int] dJ(real[int] & x)                            //    the grad of the J
  { Vh u ;u[]=xx ;      //    to set finite element function u from xx array
    varf au(uh,vh) = int2d(Th)( ( dx(u)*dx(vh) + dy(u)*dy(vh) )
             / sqrt(1. +dx(u)*dx(u) + dy(u)*dy(u) )     )
             + on(1,2,3,4,uh=0) ;
    return xx= au(0,Vh) ;  }          //    warning no return of local array
```

My solution

Second the call

```
Vh u=G ;
verbosity=5 ;                                          //    to see the residual
int conv=NLCG(dJ,u[],nbiter=500,eps=1e-5) ;
cout << " the  surface =" << J(u[])  << endl ;
                                                       //    so see the surface un 3D
savemesh(Th,"mm",[x,y,u]) ;                            //    save surface mesh for medit
exec("medit mm ;rm mm.bb mm.faces mm.points") ;
```

Execute minimal-surf.edp

Stokes equation in FreeFem++

```
int nn=10;                          mesh Th2=square(nn,nn);
fespace Vh2(Th2,P2);                Vh2 ux,uz,p2;
mesh3 Th=buildlayers(Th2,nn);   savemesh(Th,"c10x10x10.mesh");
fespace VVh(Th,[P23d,P23d,P23d,P13d]);   //    Taylor Hood Finite element.
macro Grad(u) [dx(u),dy(u),dz(u)]                                    //    EOM
macro div(u1,u2,u3) (dx(u1)+dy(u2)+dz(u3))                            //   EOM
varf vStokes([u1,u2,u3,p],[v1,v2,v3,q]) =  int3d(Th)(
    Grad(u1)'*Grad(v1) +  Grad(u2)'*Grad(v2) +  Grad(u3)'*Grad(v3)
     - div(u1,u2,u3)*q - div(v1,v2,v3)*p + 1e-10*q*p )
 + on(1,2,3,4,6,u1=0,u2=0,u3=0)   +  on(5,u1=1,u2=0,u3=0);
matrix A=vStokes(VVh,VVh);
set(A,solver=UMFPACK);
real[int] b= vStokes(0,VVh);
VVh [u1,u2,u3,p];      u1[]= A^-1 * b;
ux= u1(x,0.5,y);  uz= u3(x,0.5,y);  p2= p(x,0.5,y);
plot([ux,uz],p2,cmm=" cut y = 0.5");
```

Execute Stokes3d.edp

incompressible Navier-Stokes equation with caracteristics methods

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \nu \Delta u + \nabla p = 0, \quad \nabla \cdot u = 0$$

with the same boundary conditions and with initial conditions $u = 0$.

This is implemented by using the interpolation operator for the term $\frac{\partial u}{\partial t} + u \cdot \nabla u$, giving a discretization in time

$$\begin{array}{ll} \frac{1}{\tau}(u^{n+1} - u^n \circ X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} & = 0, \\ \nabla \cdot u^{n+1} & = 0 \end{array} \tag{4}$$

The term $X^n(x) \approx x - u^n(x)\tau$ will be computed by the interpolation operator.

The ff++ NSI 3d code

```
 real alpha =1./dt ;
 varf vNS([uu1,uu2,uu3,p],[v1,v2,v3,q]) =
  int3d(Th)( alpha*(uu1*v1+uu2*v2+uu3*v3)
  + nu*(Grad(uu1)'*Grad(v1) +  Grad(uu2)'*Grad(v2) +  Grad(uu3)'*Grad(v3))
  - div(uu1,uu2,uu3)*q - div(v1,v2,v3)*p + 1e-10*q*p )
  + on(1,2,3,4,5,uu1=0,uu2=0,uu3=0)
  + on(6,uu1=4*(1-x)*(x)*(y)*(1-y),uu2=0,uu3=0)
  + int3d(Th)( alpha*(
      u1(X1,X2,X3)*v1  +   u2(X1,X2,X3)*v2 +   u3(X1,X2,X3)*v3 )) ;
A = vNS(VVh,VVh) ;   set(A,solver=UMFPACK) ; //   build and factorize matrix
real t=0 ;
for(int i=0 ;i<50 ;++i)
  { t += dt ;  X1[]=XYZ[]-u1[]*dt ;            //    set χ=[X1,X2,X3] vector
    b=vNS(0,VVh) ;                                    //    build NS rhs
    u1[]= A^-1 * b ;                          //    solve the linear systeme
    ux= u1(x,0.5,y) ;  uz= u3(x,0.5,y) ;  p2= p(x,0.5,y) ;
    plot([ux,uz],p2,cmm=" cut y = 0.5, time ="+t,wait=0) ;  }
```

Execute NSI3d.edp

Domain decomposition Problem

We present, three classique exemples, 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.

Schwarz-overlap.edp

To solve

$$-\Delta u = f, \quad \text{in } \Omega = \Omega_1 \cup \Omega_2 \quad u|_\Gamma = 0$$

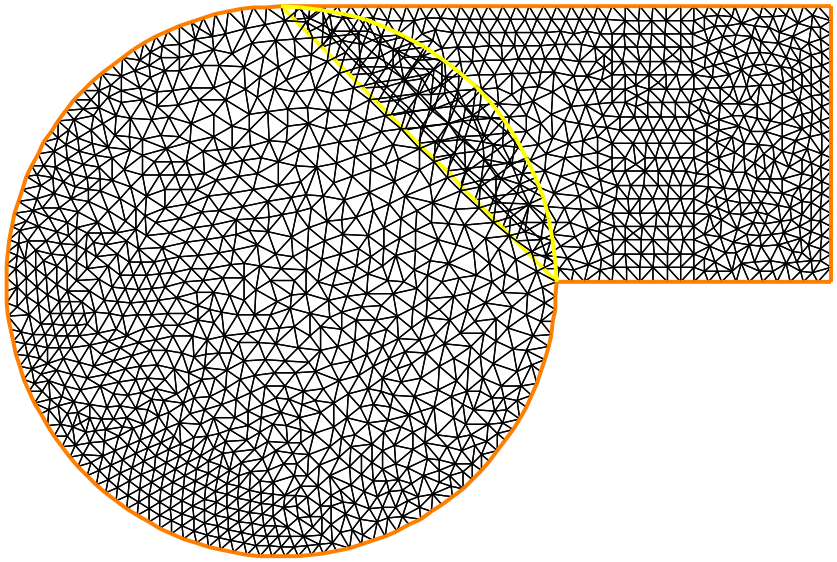
the Schwarz algorithm runs like this

$$-\Delta u_1^{m+1} = f \text{ in } \Omega_1 \quad u_1^{m+1}|_{\Gamma_1} = u_2^m$$

$$-\Delta u_2^{m+1} = f \text{ in } \Omega_2 \quad u_2^{m+1}|_{\Gamma_2} = u_1^m$$

where Γ_i is the boundary of Ω_i and on the condition that $\Omega_1 \cap \Omega_2 \neq \emptyset$ and that u_i are zero at iteration 1.

Here we take Ω_1 to be a quadrangle, Ω_2 a disk and we apply the algorithm starting from zero.



Schwarz-overlap.edp / Mesh generation

```
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
```

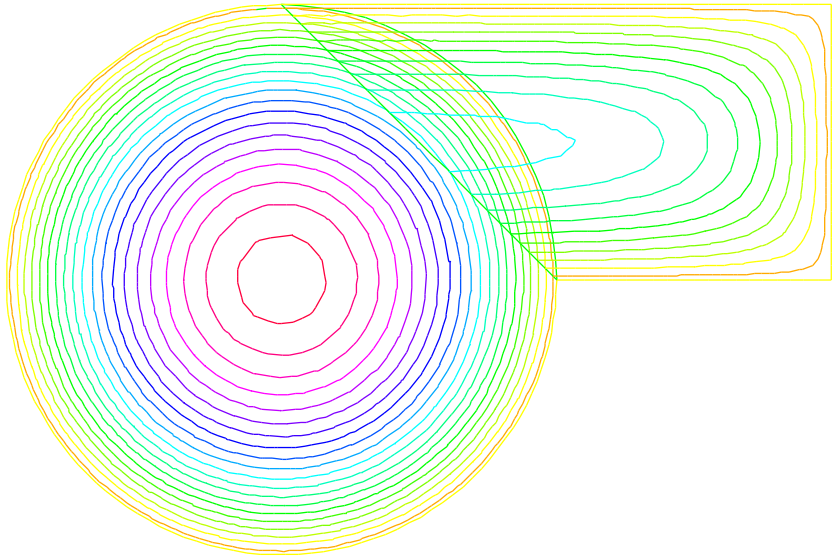
Schwarz-overlap.edp

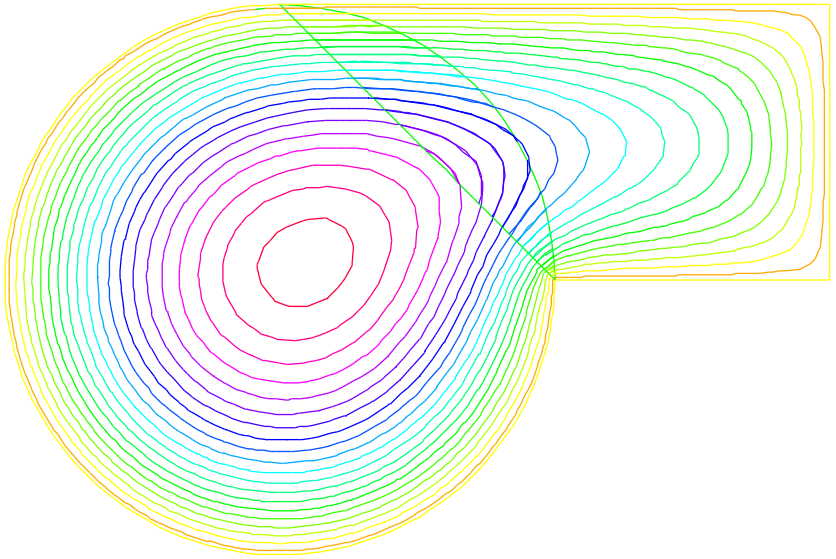
The space and problem definition is :

```
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 ) ;
for ( i=0 ;i< 10 ; i++)
{
   PB ;   pb ;       plot(U,u,wait=true) ;
} ;
```

Schwarz-overlap.edp





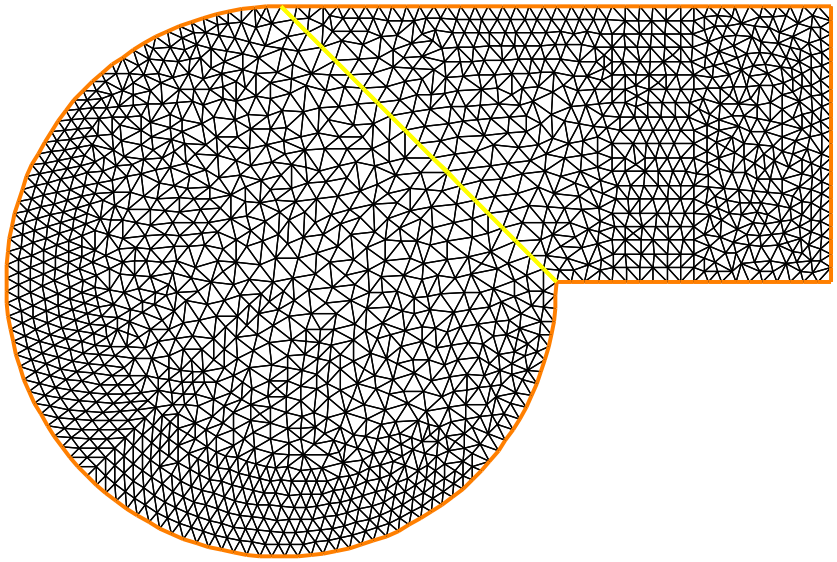
Isovalues of the solution at iteration 0 and iteration 9

Schwarz-no-overlap.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



The two none overlapping mesh `TH` and `th`

Let introduce Γ_i is common the boundary of Ω_1 and Ω_2 and $\Gamma_e^i = \partial\Omega_i \setminus \Gamma_i$.

The probem find λ such that $(u_1|_{\Gamma_i} = u_2|_{\Gamma_i})$ where u_i is solution of the

following Laplace problem :

$$-\Delta u_i = f \text{ in } \Omega_i \quad u_i|_{\Gamma_i} = \lambda \quad u_i|_{\Gamma_e^i} = 0$$

To solve this problem we just make a loop with upgradingλ with

$$\lambda = \lambda \pm \frac{(u_1 - u_2)}{2}$$

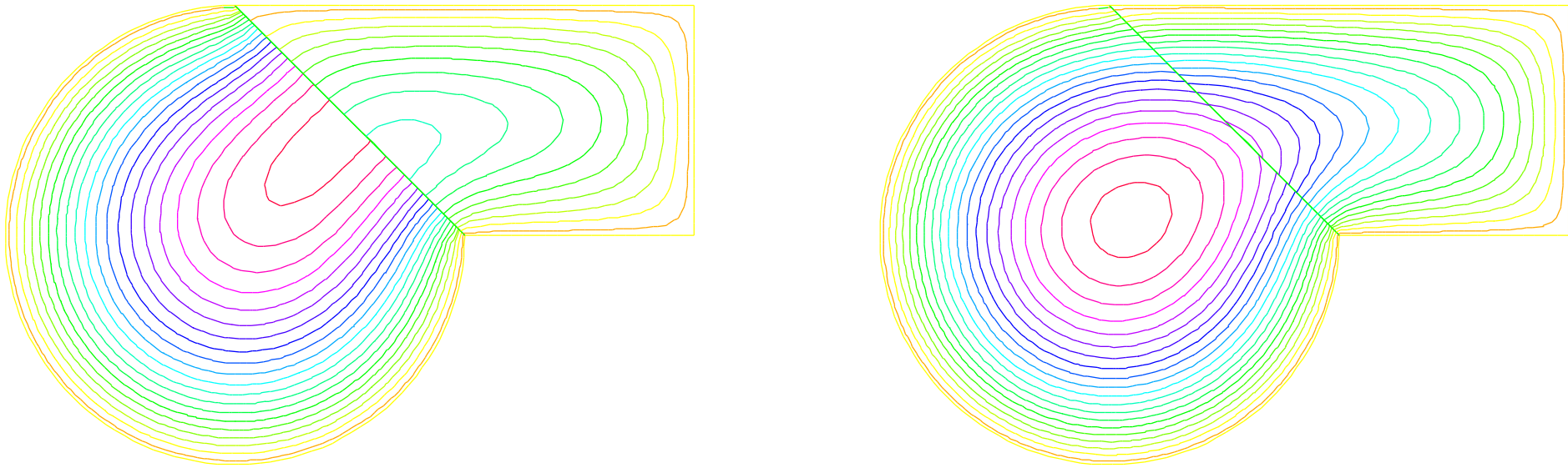
where the sign $+$ or $-$ of $\pm$ is choose to have convergence.

Schwarz-no-overlap.edp

```
                                                //    schwarz1 without overlapping
int inside = 2;          int outside = 1;
...  build mesh th and TH
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;  };
```

Schwarz-no-overlap.edp



Isovalues of the solution at iteration 0 and iteration 9 without overlapping

Schwarz-gc.edp

To solve

$$-\Delta u = f \text{ in } \Omega = \Omega_1 \cup \Omega_2 \quad u|_\Gamma = 0,$$

the Schwarz algorithm for domain decomposition without overlapping runs like this

Let introduce Γ_i is common the boundary of Ω_1 and Ω_2 and $\Gamma_e^i = \partial\Omega_i \setminus \Gamma_i$.

The probem find λ such that $(u_1|_{\Gamma_i} = u_2|_{\Gamma_i})$ where u_i is solution of the following Laplace problem :

$$-\Delta u_i = f \text{ in } \Omega_i \quad u_i|_{\Gamma_i} = \lambda \quad u_i|_{\Gamma_e^i} = 0$$

The version of this example for Shur componant. The border problem is solve with conjugate gradient.

First, we construct the two domain

```
                    //    Schwarz without overlapping (Shur complement Neumann -> Dirichet)
real cpu=clock();
int inside = 2;
int outside = 1;

border Gamma1(t=1,2){x=t;y=0;label=outside;};
border Gamma2(t=0,1){x=2;y=t;label=outside;};
border Gamma3(t=2,0){x=t;y=1;label=outside;};

border GammaInside(t=1,0){x = 1-t; y = t;label=inside;};

border GammaArc(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;
                                                          //    build the mesh of Ω1 and Ω2
mesh Th1 = buildmesh( Gamma1(5*n) + Gamma2(5*n) + GammaInside(5*n) + Gamma3(5*n));
mesh Th2 = buildmesh ( GammaInside(-5*n) + GammaArc(25*n) );
plot(Th1,Th2);

                                                                //    defined the 2 FE space
fespace Vh1(Th1,P1),      Vh2(Th2,P1);
```

Schwarz-gc.edp, next

Remark, to day is not possible to defined a function just on a border, so the λ function is defined on the all domain Ω_1 by :

```
Vh1 lambda=0 ;                                          //    take λ ∈ V_h1
```

The two Laplace problem :

```
Vh1 u1,v1 ;                  Vh2 u2,v2 ;
int i=0 ;                                    //    for factorization optimization
problem Pb2(u2,v2,init=i,solver=Cholesky) =
    int2d(Th2)( dx(u2)*dx(v2)+dy(u2)*dy(v2) )
  + int2d(Th2)( -v2)
  + int1d(Th2,inside)(-lambda*v2) +    on(outside,u2= 0 ) ;
problem Pb1(u1,v1,init=i,solver=Cholesky) =
    int2d(Th1)( dx(u1)*dx(v1)+dy(u1)*dy(v1) )
  + int2d(Th1)( -v1)
  + int1d(Th1,inside)(+lambda*v1) +    on(outside,u1 = 0 ) ;
```

Schwarz-gc.edp, next I

Now, we define a border matrix , because the λ function is none zero inside the domain Ω_1 :

```
varf b(u2,v2,solver=CG) =int1d(Th1,inside)(u2*v2) ;
matrix B= b(Vh1,Vh1,solver=CG) ;
```

The boundary problem function,

$$\lambda \longrightarrow \int_{\Gamma_i} (u_1 - u_2) v_1$$

```
func real[int] BoundaryProblem(real[int] &l)
{
   lambda[]=l ;                                    //    make FE function form l
   Pb1 ;      Pb2 ;
   i++ ;                                           //    no refactorization i !=0
   v1=-(u1-u2) ;
   lambda[]=B*v1[] ;
   return lambda[] ;
} ;
```

Schwarz-gc.edp, next II

Remark, the difference between the two notations `v1` and `v1[]` is : `v1` is the finite element function and `v1[]` is the vector in the canonical basis of the finite element function `v1` .

```
Vh1 p=0,q=0;
                                //    solve the problem with Conjugue Gradient
LinearCG(BoundaryProblem,p[],eps=1.e-6,nbiter=100);
          //    compute the final solution, because CG works with increment
BoundaryProblem(p[]);                    //    solve again to have right u1,u2

cout << " -- CPU time  schwarz-gc:" <<  clock()-cpu << endl;
plot(u1,u2);                                                    //    plot
```

Mortar Method

Let be a partition without overlap $\Omega = \cup_{i=0,..,4}\Omega_i$.

Remark Ω is the open set without the skeleton $\mathcal{S}$ and the external boundary is Γ. So the Mortar problem is : Find $u \in H^1(\Omega)$ such that $u_{|\Gamma} = g$ and $\lambda \in H^{-\frac{1}{2}}(\mathcal{S})$ and

$$\forall v \in H^1(\Omega), \quad v_{|\Gamma} = 0, \qquad \int_\Omega \nabla u \nabla v + \int_{\mathcal{S}} [v]\lambda = \int_{\Omega_i} fv$$

$$\forall \mu \in H^{-\frac{1}{2}}(\mathcal{S}), \qquad \int_{\mathcal{S}} [u]\mu = 0$$

For each sub domain Ω_i ,

$$\forall v \in H^1(\Omega_i), \quad v_{|\Gamma} = 0, \quad \int_{\Omega_i} \nabla u \nabla v + \int_{\mathcal{S}\cap\partial\Omega_i} \varepsilon_i \, \lambda v = \int_{\Omega_i} fv$$

$$\forall \mu \in H^{-\frac{1}{2}}(\mathcal{S}), \quad \sum_i \int_{\mathcal{S}\cap\partial\Omega_i} \varepsilon_i \, \mu u = O$$

Where $\varepsilon_i = \mathbf{n}_{\mathcal{S}}.\mathbf{n}_i$, $\varepsilon_i = \pm 1$ and $\sum_i \varepsilon_i = 0$.

Mortar Method Precond

$$J'(\lambda)(\mu) = -\int_{\mathcal{S}} [u_\lambda]\mu = 0 \forall \mu$$

$$\forall v \in H^1(\Omega_i), \quad v_{|\Gamma} = 0, \quad \int_{\Omega_i} \nabla u_l \nabla v + \int_{\mathcal{S}\cap\partial\Omega_i} \varepsilon_i \, \lambda v = \int_{\Omega_i} f v$$

For each sub domain Ω_i ,

$$\forall v \in H^1(\Omega_i), \quad v_{|\Gamma} = 0, \quad \int_{\Omega_i} \nabla u \nabla v + \int_{\mathcal{S}\cap\partial\Omega_i} \varepsilon_i \, \lambda v = \int_{\Omega_i} f v$$

$$\forall \mu \in H^{-\frac{1}{2}}(\mathcal{S}), \quad \sum_i \int_{\mathcal{S}\cap\partial\Omega_i} \varepsilon_i \, \mu u = O$$

Where $\varepsilon_i = \mathbf{n}_{\mathcal{S}}.\mathbf{n}_i$, $\varepsilon_i = \pm 1$ and $\sum_i \varepsilon_i = 0$.

Mortar method, compute ε_i

```
func f=1+x+y ;        real g=1 ;
...                                                    //    def of sub-domain
fespace Lh(Thm,P1) ;          Lh  lh=0,rhsl=0 ;
fespace RTh(Tha,[P0edge,P0edge]) ;//   Finite element constant on each edge
varf  vNN([ux,uy],[nx,ny]) = int1d(Tha,1)(( nx*N.x + ny*N.y)/lenEdge) ;
Nmx[]= vNN(0,RTh) ;                                              //    Def of n_S

                                    //    in a macro where i is the ssd number
macro defeps(i)
fespace Eh#i(Th#i,P0edge) ;
varf veps#i(u,v)= int1d(Th#i,1,qforder=5)( (Nmx*N.x + Nmy*N.y)*v/lenEdge) ;
   Eh#i eps#i = 0 ;
   eps#i[]= veps#i(0,Eh#i) ;
   eps#i = -real(eps#i <-0.01) + real(eps#i >0.01) ;              //    a ±1
```

Mortar method

```
macro defspace(i)
    cout << " Domaine " << i<< " --------" << endl ;
    fespace Vh#i(Th#i,P1) ;
    Vh#i u#i ;                    Vh#i rhs#i ;
    defeps(i)
    varf vLapM#i([u#i],[v#i]) =
        int2d(Th#i)(  Grad(u#i)'*Grad(v#i) )    +  int2d(Th#i) (f*v#i)
     +  on(labext,u#i=g)   ;
    varf cc#i([l],[u]) = int1d( Thmm,1)(l*u*eps#i) ;
    matrix C#i = cc#i(Lh,Vh#i) ;
    matrix A#i = vLapM#i(Vh#i,Vh#i,solver=GMRES) ;
    rhs#i[]=vLapM#i(0,Vh#i) ;                    //    End of macro defspace(i)
defspace(0) defspace(1) defspace(2) defspace(3)
```

Mortar method/ Direct Method

```
varf  vDD(u,v) = int2d(Thm)(u*v*1e-10);
matrix DD=vDD(Lh,Lh);                    //    a trick to make a invertible matrix.
matrix A=[  [ A0 ,0  ,0  ,0  ,C0 ],
            [ 0  ,A1 ,0  ,0  ,C1 ],
            [ 0  ,0  ,A2 ,0 , C2 ],
            [ 0  ,0  ,0  ,A3, C3 ],
            [ C0',C1',C2',C3',DD ] ];
real[int] xx(M.n),   bb=[rhs0[], rhs1[],rhs2[],rhs3[],rhsl[] ];
set(A,solver=UMFPACK);                   //    choose the solver associed to M
xx = A^-1 * bb;
[u0[],u1[],u2[],u3[],lh[]] = xx;                              //    dispatcher
```

Execute DDM18-Mortar.edp

Mortar method/ GC method

```
func real[int] SkPb(real[int] &l)
{  int verb=verbosity;   verbosity=0;   itera++;
   v0[] = rhs0[]; v0[]+= C0* l; u0[] = A0^-1*v0[];
   v1[] = rhs1[]; v1[]+= C1* l; u1[] = A1^-1*v1[];
   v2[] = rhs2[]; v2[]+= C2* l; u2[] = A2^-1*v2[];
   v3[] = rhs3[]; v3[]+= C3* l; u3[] = A3^-1*v3[];
   l  = C1'*u1[];    l += C0'*u0[];   l += C2'*u2[];
   l += C3'*u3[];   l= lbc? lbc0: l;
   verbosity=verb;
   return l; };
verbosity=100; lh[]=0;
LinearCG(SkPb,lh[],eps=1.e-2,nbiter=30);
```

Mortar method/ in parallel : The functionnal

```
func real[int] SkPb(real[int] &l)
{
   int verb=verbosity;  verbosity=0;  itera++;
   broadcast(processor(0),what);
   if(what==2) return l;
   broadcast(processor(0),l);
   v0[] = rhs0[]; v0[]+= C0* l; u0[] = A0^-1*v0[];
   l  = C0'*u0[];
   l= lbc? lbc0: l;                                  //    put zero on boundary
   if(mpirank==0)                                       //    on master process
    for (int i=1;i<4;++i)
     { processor(i) >> lw[];
      l += lw[];    }
   else  processor(0) << l;                             //    on salve process
   verbosity=verb;
   return l; }
```

Mortar method/ CG in parallel

```
what=1 ;
verbosity=100 ;
if(mpirank==0)
 {                                                    //    on master process
  LinearCG(SkPb,lh[],eps=1.e-3,
           nbiter=20) ;                          //   future withprocess=[0,1,2,3]
  what=2 ;  SkPb(lh[]) ;
 }
 else                                                  //    on slave process
   while(what==1)
      SkPb(lh[]) ;
```

Execute DDM18-mortar-mpi.edp

Fluid-Structure interactions

The Stokes system for viscous fluids with velocity $\vec{u}$ and pressure p :

$$-\Delta\vec{u} + \vec{\nabla}p = 0,\ \nabla\cdot\vec{u} = 0, \quad \text{in} \quad \Omega,\ \vec{u} = \vec{u}_\Gamma \quad \text{on} \quad \Gamma = \partial\Omega$$

where u_Γ is the velocity of the boundaries. The force that the fluid applies to the boundaries is the normal stress

$$\vec{h} = (\nabla\vec{u} + \nabla\vec{u}^T)\vec{n} - p\vec{n}$$

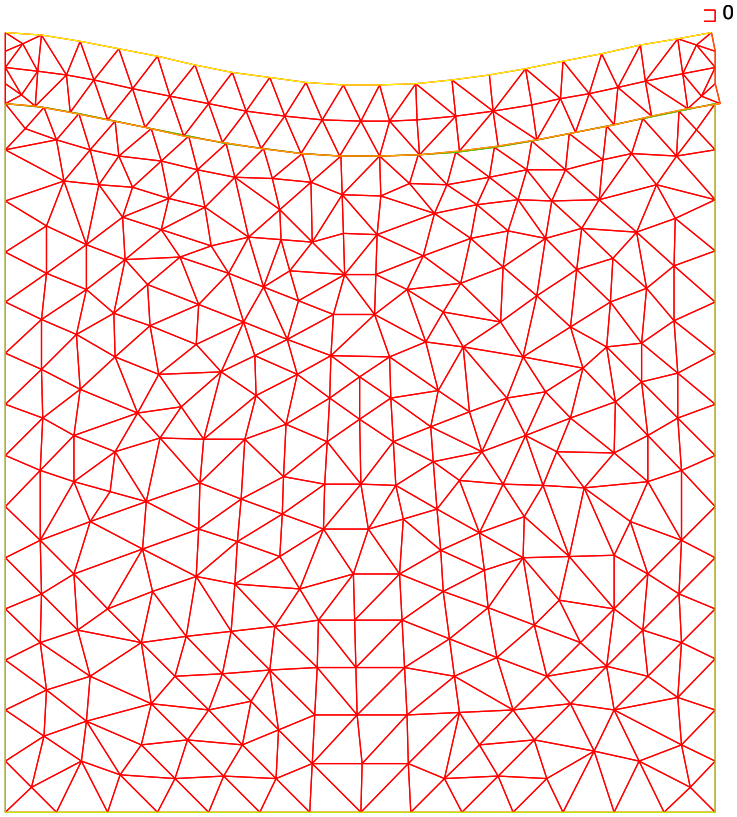
Elastic solids subject to forces deform : a point in the solid, originally at (x,y) goes to (X,Y) after sometimes. When the displacement vector $\vec{v} = (v_1, v_2) = (X - x, Y - y)$ is small, Hooke's law relates the stress tensor σ inside the solid to the deformation tensor ϵ :

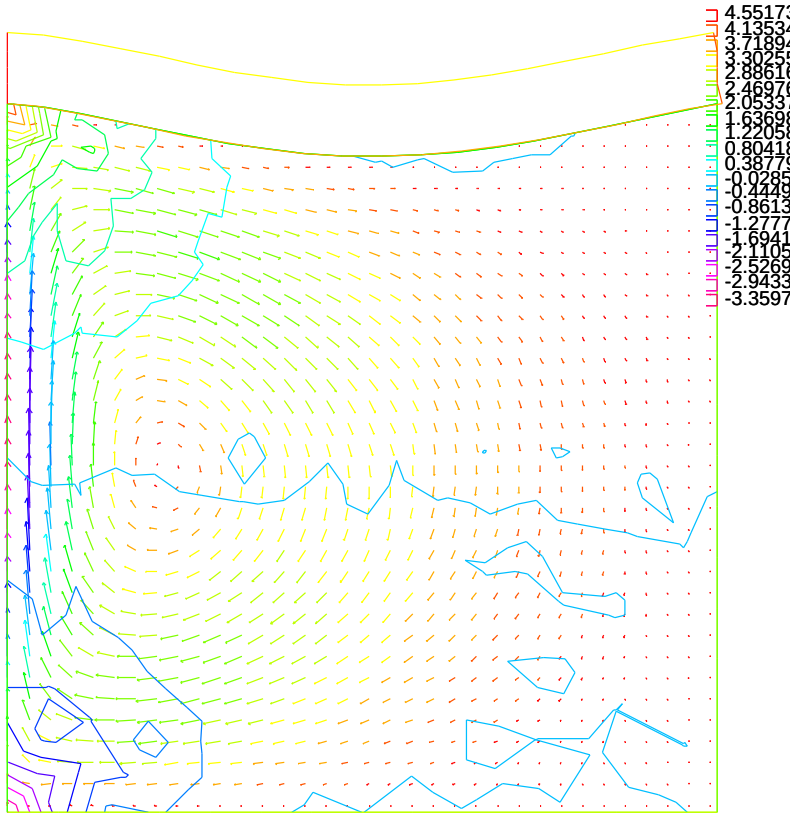
$$\sigma_{ij} = \lambda \delta_{ij} \nabla . \vec{v} + \mu \epsilon_{ij},\ \epsilon_{ij} = \frac{1}{2}(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i})$$

where δ is the Kronecker symbol and where λ, μ are two constants describing the material mechanical properties in terms of the modulus of elasticity, and Young's modulus.

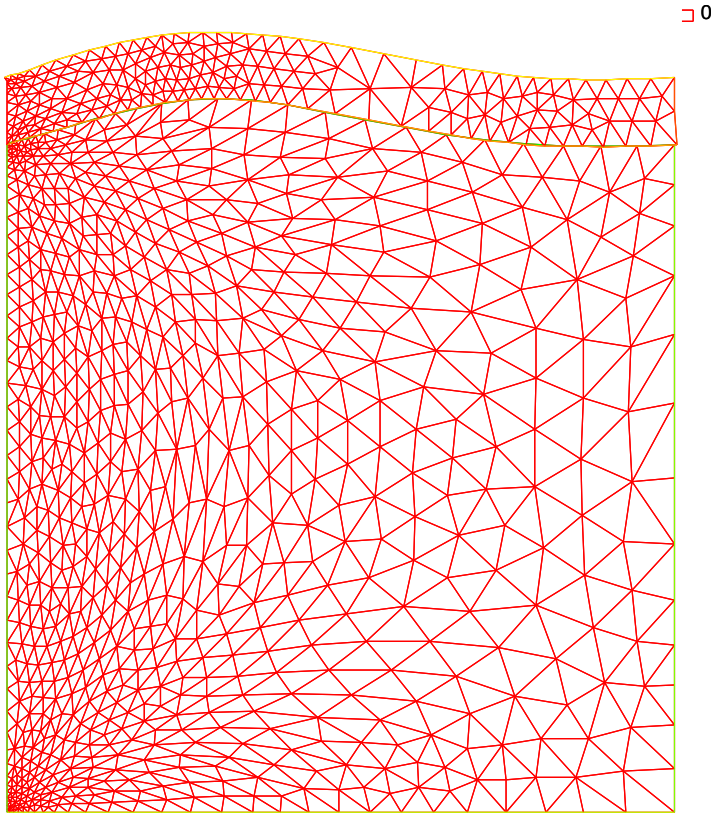
The equations of elasticity are naturally written in variational form for the displacement vector $v(x) \in V$ as

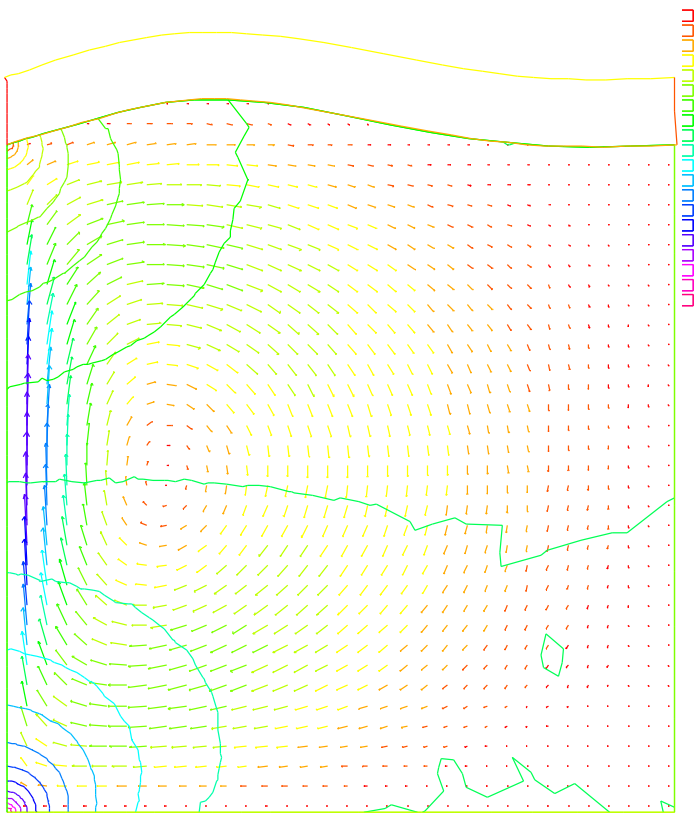
$$\int_\Omega [\mu \epsilon_{ij}(\vec{v}) \epsilon_{ij}(\vec{w}) + \lambda \epsilon_{ii}(v) \epsilon_{jj}(\vec{w})] = \int_\Omega \vec{g} \cdot \vec{w} + \int_\Gamma \vec{h} \cdot \vec{w}, \forall \vec{w} \in V$$


0


4.55173
4.13534
3.71894
3.30255
2.88616
2.46976
2.05337
1.63698
1.22058
0.80418
0.38779
-0.0285
-0.4449
-0.8613
-1.2777
-1.6941
-2.1105
-2.5269
-2.9433
-3.3597

Iteration 1


0


2.32825
2.03496
1.74168
1.44839
1.15511
0.86182
0.56853
0.27524
-0.01804
-0.31133
-0.60462
-0.89791
-1.19119
-1.48448
-1.77777
-2.07106
-2.36434
-2.65763
-2.95092
-3.24421

Iteration 7

Eigenvalue

This tools is based on the arpack++ * the object-oriented version of ARPACK eigenvalue package.

The function EigenValue compute the generalized eigenvalue of $Au = \lambda Bu$ where sigma $= \sigma$ is the shift of the method.

The return value is the number of converged eigenvalue (can be greater than the number of eigen value nev=)

```
int k=EigenValue(OP,B,nev= , sigma= ) ;
```

where the sparce matrix $OP = A - \sigma B$ is defined with a solver and with boundary condition, and the matrix B is just sparce.

*. http://www.caam.rice.edu/software/ARPACK/

Eigenvalue/ parameter

`sym=` the problem is symmetric (all the eigen value are real)

`nev=` the number desired eigenvalues (nev) close to the shift.

`value=` the array to store the real part of the eigenvalues

`ivalue=` the array to store the imag. part of the eigenvalues

`vector=` the array to store the eigenvectors. For real nonsymmetric problems, complex eigenvectors are given as two consecutive vectors, so if eigenvalue k and $k+1$ are complex conjugate eigenvalues, the kth vector will contain the real part and the $k+1$th vector the imaginary part of the corresponding complex conjugate eigenvectors.

`tol=` the relative accuracy to which eigenvalues are to be determined ;

`sigma=` the shift value ;

`maxit=` the maximum number of iterations allowed ;

`ncv=` the number of Arnoldi vectors generated at each iteration of ARPACK.

Eigenvalue/ example

In the first example, we compute the eigenvalue and the eigenvector of the Dirichlet problem on square $\Omega =]0, \pi[^2$.

The problem is find : λ, and u_λ in $\mathbb{R} \times H_0^1(\Omega)$

$$\int_\Omega \nabla u_\lambda \nabla v = \lambda \int_\Omega uv \quad \forall v \in H_0^1(\Omega)$$

The exact eigenvalues are $\lambda_{n,m} = (n^2 + m^2), (n, m) \in {\mathbb{N}_*}^2$ with the associated eigenvectors are $u_{\lambda_{m,n}} = sin(nx) * sin(my)$.

We use the generalized inverse shift mode of the `arpack++` library, to find 20 eigenvalue and eigenvector close to the shift value $\sigma = 20$.

Eigen value/ source

```
verbosity=10;
mesh Th=square(20,20,[pi*x,pi*y]);
fespace Vh(Th,P2);
Vh u1,u2;
real sigma = 20;                                                    //    value of the shift
                                                  //    OP = A - sigma B; // the shifted matrix
varf  op(u1,u2)= int2d(Th)(  dx(u1)*dx(u2) + dy(u1)*dy(u2) - sigma* u1*u2 )
                     +  on(1,2,3,4,u1=0);                           //    Boundary condition
varf b([u1],[u2]) = int2d(Th)(  u1*u2 );                          //    no Boundary condition
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 penalisation:
//    we put 1e30=tgv on the diagonal term to lock the degre of freedom.
//    So take dirichlet boundary condition just on a variationnal form
//    and not on b variationnanl form.
//    because we solve w = OP^{-1} * B * v
```

Eigen value/ source next

```
int nev=20 ;                //    number of computed eigen valeu 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) ;
                                  //    return the number of computed eigenvalue
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) ;
}
```

Load facility examples

A little hard, because you have to compile C++ code

Conclusion and Future

It is a useful tool to teaches Finite Element Method, and to test some nontrivial algorithm.

– Optimization FreeFem*
– All graphic with OpenGL
– Galerkin discontinue (fait)
– complex problem (fait)
– 3D (in constructoin)
– Suite et FIN. (L'avenir ne manque pas de future et lycée de Versailles)

Thank, for your attention ?