FreeFem++

F. Hecht
Laboratoire Jacques-Louis Lions
Université Pierre et Marie Curie
Paris, France

with S. Auliac, P. Jolivet

http://www.freefem.org
mailto:hecht@ann.jussieu.fr

With the support of ANR (French gov.)
ANR-07-CIS7-002-01
http://www.freefem.org/ff2a3/
http://www-anr-ci.cea.fr/

Outline

Last News, FreeFem++ pass 10^9 unknowns

The message from Pierre Jolivet

```
Machine: Curie Thin Node@CEA
Financement: PRACE project HPC-PDE (number 2011050840)
Nombre de coeurs: 6144     Mémoire par coeurs: 4 Go
Eléments finis: P3  Dimension: 2D
Précision: 1e-08
Nombres d'inconnues: 1 224 387 085
Méthode de résolution: GMRES préconditionné par une méthode de
décomposition de domaine à deux niveaux
Nombre d'itérations: 18
Temps de résolution: 2 minutes
Type de problème: équation de diffusion avec coefficients très
hétérogènes (5 ordres de grandeur de variation)
```

Les Journées FreeFem++ les 6 et 7 Décembre 2012, UPMC, Jussieu, Paris

Introduction FreeFem++ is a software to solve numerically partial differential

equations (PDE) in $\mathbb{R}^2)$ and in $\mathbb{R}^3)$ 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 ...

By the way, FreeFem++ is build to play with abstract linear, bilinear form on Finite Element Space and interpolation operator.

FreeFem++ is a freeware and this run on Mac, Unix and Window architecture, in parallel with MPI.

History

1987	MacFem/PCFem les ancêtres (O. Pironneau en Pascal) payant.
1992	FreeFem réécriture de C++ (P1,P0 un maillage) O. Pironneau, D. Bernardi, F. Hecht , C. Prudhomme (adaptation Maillage, bamg).
1996	FreeFem+ réécriture de C++ (P1,P0 plusieurs maillages) O. Pironneau, D. Bernardi, F. Hecht (algèbre de fonction).
1998	FreeFem++ réécriture avec autre noyau élément fini, et un autre langage utilisateur ; F. Hecht, O. Pironneau, K.Ohtsuka.
1999	FreeFem 3d (S. Del Pino) , Une première version de freefem en 3d avec des méthodes de domaine fictif.
2008	FreeFem++ v3 réécriture du noyau élément fini pour prendre en compte les cas multidimensionnels : 1d,2d,3d...

For who, for what !

For what

1. R&D
2. Academic Research ,
3. Teaching of FEM, PDE, Weak form and variational form
4. Algorithmes prototyping
5. Numerical experimentation
6. Scientific computing and Parallel computing

For who : the researcher, engineer, professor, student...

The mailing list `mailto:Freefempp@ljll.math.upmc.fr` with 377 members with a flux of 5-20 messages per day.

More than 1000 true Users (more than 100 download / month)

The main characteristics I/II (2D)(3D)

- Wide range of finite elements : continuous P1,P2 elements, discontinuous P0, P1, RT0,RT1,BDM1, elements ,Edge element, vectorial element, mini-element, ...
- Automatic interpolation of data from a mesh to an other one (with matrix construction if need), so a finite element function is view as a function of (x, y, z) or as an array.
- Definition of the problem (complex or real value) with the variational form with access to the vectors and the matrix.
- Discontinuous Galerkin formulation (only in 2d to day).
- LU, Cholesky, Crout, CG, GMRES, UMFPack, SuperLU, MUMPS, HIPS , SUPERLU_DIST, PASTIX, PARDISO ... sparse linear solver ; eigenvalue and eigenvector computation with ARPACK.
- Online graphics with OpenGL/GLUT/VTK, C++ like syntax.
- An integrated development environment FreeFem++-cs

The main characteristics II/II (2D)(3D)

- Analytic description of boundaries, with specification by the user of the intersection of boundaries in 2d.
- Automatic mesh generator, based on the Delaunay-Voronoï algorithm. (2d,3d (tetgen))
- load and save Mesh, solution
- Mesh adaptation based on metric, possibly anisotropic (only in 2d), with optional automatic computation of the metric from the Hessian of a solution. (2d,3d).
- Link with other soft : parview, gmsh , vtk, medit, gnuplot
- Dynamic linking to add plugin.
- Full MPI interface
- Nonlinear Optimisation tools : CG, Ipopt, NLOpt, stochastic
- Wide range of examples : Navier-Stokes 3d, elasticity 3d, 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. EnterFreeFem++ yours.edp or 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, fraise ... 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.

The Changes form 01/11 to 08/11

- correct problem of comm world on SuperLuDist (complex version)
- correct medit Makefile.am in case of no compilation of medit ..
- Add thresholdings.cpp thresholdings.edp in examples++-load to remove to small coef in a matrix .
- Add lots of DDM Schwarz GMRES preconditioned with a coarse grid solver overlapping :
- Add a true exemple to buld mesh form a image (lg.pgm) , Leman-mesh.edp
- Add New syntaxe in macro generation `NewMacro a(i)` .... `EndMacro` with // comment and macro definition.
- Add interface with special function to GSL `http://www.gnu.org/software/gsl/` the full list is in the example `examples++-loal/gsl.edp`
- correct the precond of gmres algo in A^{-1} : (29 mars 2011) to build a correct Navier-Stokes
- add cast TypeOfSolver on int to parametrize the choise of linear solver.
- put comment in the documentation about negative tgv
- correct `mpiReduce` for complex and real data.
- add `mpiAllReduce(umax,dmaxg,comm,mpiMAX);` where for real umax,dmaxg ;
- update the finite element list in documentation
- add (load "Element-Mixte") NEW FINITE ELEMENT 2D TDNSS1 sym matrix 2x2 conforme in $\{H(divdiv)/div(divs)) \in H^{-1}\}$, RT1 and BDM1 conforme in H(div) Raviart Thomas of Degree 1 and Bezzi, Douglas, Marini RT1 and BDM1ortho conforme in $H(curl)$ Nedelec Finite Elem.
- add matrix multiplication with lapack interface
- add tool to change the region and label number change(Th,fregion= integer function to set the region number)
- nuTriangle now given the tet number in 3D.
- add mpireduce of matrix to build parallel matrix
- correct the Hips interface (not to bad , in test)
- add interface with `MUMPS_4.10.0` version (with automatic download)
- add `P1dc3d` finite element in pluging `"Element_P1dc1"`
- correct bug in gibbs renumbering in 1 case veru sample mesh)
- correct bug in interpolation operator with periodic B.C.

The Changes form 08/11 to 11/12

- add new MUMPS and PADSO interface of linear solver
- correct isoline plugin case of saddle point
- change the compilation tools under windows gcc 4/7 + freeglut / ...)
- change the compile tools on mac pass to clang++
- add formal tools on array [] or matrix [[],[],] for Elasity problem. $trace(A), det(A), Cofactor(A), A : A = \sum_{ij} A_{ij} * A_{ij}, 2 * A, A * 2$
- add integration on levelset line (in test)
- correct pb of C in/output in pluging (in test)
- correct bugs mshmet pluging in case of double eigen value in eigen
- correct typo problem (string size) when a change the default
- add tool to create Quadrature formlar 1d,2d,3d with
- correct download auto compile of mmg3d, mshmet, scotch, ...
- scotch partionner interface see scotch.edp in examples++-load
- add isNaN(x), isInf(x), IsNormal(x) function to check floating point of real value x, see ISO C99.
- add function : NaN() and NaN("") to build NaN real number (double in C) .
- correct error in macro with operator ./= and .*=
- add Ipopt interface (thanks to Sylvain Auliac)
- correct 3d trunc bug in case of internal boundar
- add new type of array , array of array see taboftab :edp in examples++-tuturial `real[int] a; real[int][int] v(10);`
- add plugins with sequentiel mumps without mpi
- add conversion of re and im part of complex sparse matrix A = C.im ; A = C.re ;
- add generation of error in case of periodic boundary condition non scalar problem.
- add tools for adaptation of P2 and P3 finite elements with metrics
- New example in in Chapter 3 Navier Stokes Newton NSNewton.edp
- add code to build matrix and vector form varf in 3d in case of different meshes (in test)
- correct NSprojection.edp chap3 example (PB in out flow BC.)
- correct compile of mmg3d v4 plugins (small change un distrib archive)
- correct bug in cas of resize of array with 2 index (full matrix)
- correct assert in MPI in gather and scatter
- correct bug in case of return in loop for or while.
- correct a=int2d(Th,l)(1) in case simple expression when l is a array.
- build a pkg under macos for distribution .

Element of syntax 1/4

```
x,y,z , label,                                  //    current coord., label of BC.
N.x, N.y, N.z,                                                          //    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,P2);                               //   Def. of 2d finite element space;
fespace Vh3(Th3,P1);                             //   Def. of 3d finite element space;
Vh u=x;                                      //    a finite element function or array
Vh3<complex> uc = x+  1i *y;                                 //    complex valued FE
u(.5,.6,.7);                           //    value of FE function u at point (.5,.6,.7)
u[];                              //   the array of DoF value assoc. to FE function u
u[][5];                                 //   6th value of the array (numbering begin
                                                                //   at 0 like in C)
```

Element of syntax 2/4

```
fespace V3h(Th,[P2,P2,P1]);
V3h [u1,u2,p]=[x,y,z];                          //   a vectorial finite element
                                                        //     function or array
                 //   remark u1[] <==> u2[] <==> p[] same array of unknown.
macro div(u,v) (dx(u)+dy(v))//   EOM a macro
                                                     //     (like #define in C )
macro Grad(u) [dx(u),dy(u)]                         //     the macro end with //
varf a([u1,u2,p],[v1,v2,q])=
int2d(Th)(  Grad(u1)'*Grad(v1) +Grad(u2)'*Grad(v2)
                 -div(u1,u2)*q -div(v1,v2)*p)
            +on(1,2)(u1=g1,u2=g2);

matrix A=a(V3h,V3h,solver=UMFPACK);
real[int] b=a(0,V3h);
u2[] =A^-1*b;                             //     or you can put also u1[]= or p[].
```

Element of syntax 3/4

```
 func Heaveside=(x>0);                               //    a formal line function
 func real g(int i, real a) { .....; return i+a;}
 A = A + A'; A = A'*A                           //    matrix operation (only 1/1)
 A = [ [ A,0],[0,A'] ];                                        //    Block matrix.
int[int] I(15),J(15);                            //    two array for renumbering
                       //   the aim is to transform a matrix into a sparse matrix
matrix B;
B = A;                                                        //    copie matrix A
B=A(I,J);                                           //    B(i,j) = A(I(i),J(j))
B=A(I^-1,J^-1);                                      //    B(I(i),J(j))= A(i,j)
B.resize(10,20);                                 //    resize the sparse matrix
                                               //    and remove out of bound terms
int[int] I(1),J(1); real[int] C(1);
[I,J,C]=A;                            //    get of the sparse term of the matrix A
                                                   //    (the array are resized)
A=[I,J,C];                                                //    set a new matrix
matrix D=[diagofA] ;                               //    set a diagonal matrix D
                                                   //    from the array diagofA.
real[int] a=2:12;                                 //   set a[i]=i+2; i=0 to 10.
```

Element of syntax 4/4 (formal computation on array)

```
 a formal array is  [ exp1, exp1, ..., expn]
 the transposition  is  [ exp1, exp1, ..., expn]'

complex a=1,b=2,c=3i;
func va=[ a,b,c];                          //    is a formal array in [ ]
a =[ 1,2,3i]'*va ; cout « a « endl;                //    hermien product
matrix<complex>  A=va*[ 1,2,3i]'; cout « A « endl;
a =[ 1,2,3i]'*va*2.;
a =(va+[ 1,2,3i])'*va*2.;
va./va;                                             //    term to term /
va*/va;                                             //    term to term *
trace(va*[ 1,2,3i]') ;                                              //
(va*[ 1,2,3i]')[1][2] ;                                     //    get coef
det([[1,2],[-2,1]]);                     //    just for matrix 1x1 et 2x2
 usefull to def your edp.
macro grad(u) [dx(u),dy(u)] //
macro div(u1,u2) (dx(u1)+dy(u2)) //
```

Element of syntax : Like in C

```
The key words are reserved

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

BOOLEAN:   0  <=> false , ≠ 0 <=> true = 1

                                          //   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

  weakless:  all local variables are almost static (????)
    bug if break before variable declaration in same block.
    bug for  fespace argument or fespace function argument
```

The C++ kernel / Dehli, (1992)

My early step in

C++

```
typedef double R;
class Cvirt { public: virtual R operator()(R ) const =0;};
class Cfonc : public Cvirt { public:
  R (*f)(R);                                                          //     a function C
  R operator()(R x) const { return (*f)(x);}
  Cfonc( R (*ff)(R)) : f(ff) {} };
class Coper : public Cvirt { public:
  const  Cvirt *g, *d;                                              //     the 2 functions
  R (*op)(R,R);                                                        //     l'opération
  R operator()(R x) const { return (*op)((*g)(x),(*d)(x));}
  Coper( R (*opp)(R,R), const Cvirt *gg, const Cvirt *dd)
    : op(opp),g(gg),d(dd) {}
  ~Coper(){delete g,delete d;} };

static  R Add(R a,R b) {return a+b;}
static  R Sub(R a,R b) {return a-b;}
static  R Mul(R a,R b) {return a*b;}
static  R Div(R a,R b) {return a/b;}
static  R Pow(R a,R b) {return pow(a,b);}
```

How to code differential operator

A differential expression on in a PDE problem is like

$$f * [u_i|\partial_x u_i|\partial_y u_i|\ldots] * [v_j, \partial_x v_j|\partial_y v_i|\ldots]$$

where $[f, |g, \ldots]$ mean f or g, or $\ldots$, and where the unknown part is $[u_i|\partial_x u_i|\partial_x u_i|\ldots] \equiv [(0,i)|(1,i)|(2,i)|\ldots]$ is a pair of $i' \times i$, if we do the same of the test part, the differential expression is a formally sum of :

$$\sum_k f_k \times (i'_k, i_k, j'_k, j_k)$$

So we can easily code this syntaxe :

```
varf a(u,v) = int2d(Th)(Grad(u)'*Grad(v)) - int2d(Th)(f*v) +
on(1,u=0);
matrix A=a(Vh,Vh,solver=UMFPACK);
real[int] b=a(0,Vh);
u[]=A^-1*b;
```

Laplace equation, weak form

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

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

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

Laplace equation in FreeFem++

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

```
mesh3 Th("Th-hex-sph.msh");                              //     read a mesh 3d
fespace Vh(Th,P1);                                 //     define the P1 EF space

 Vh u,v;                          //     set test and unknow FE function in Vh.
macro Grad(u) [dx(u),dy(u),dz(u)]                           //    EOM Grad def
solve laplace(u,v,solver=CG) =
   int3d(Th)(  Grad(u)'*Grad(v)   )
   - int3d(Th) ( 1*v)
   + on(2,u=2);                                                //     int on γ2
plot(u,fill=1,wait=1,value=0,wait=1);
```

Execute fish.edp   Execute fish3d.edp

Mesh tools

- `border` to defined the parametrization of 2d border
- `buildmesh` to build a 2d mesh from 2d borders
- `square` to build a cartesian mesh of distorted square
- `trunc` to remove a part of a mesh and/or split it
- `+` concatened meshes
- `movemesh`, `movemesh23`, `movemesh3` move meshes (with plugin "msh3" for 3d mesh)
- `change` to change label, region , numbering , ...
- `buildlayer` build a 3d from a 2d mesh with layer with can be degenerated. (with plugin "msh3")
- `tetg` fill with tetraedra a closed surface mesh (with "tetgen" plugin)
- `readmesh`, `readmesh3` to read 2d or 3d mesh.

Execute mesh2d.edp Execute mesh3d.edp Execute Leman-3d.edp

Anisotropic Mesh : 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 $\mathcal{M}$.

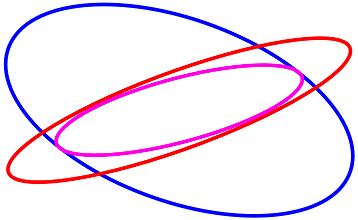
The main IDEA for mesh generation

- The difficulty is to find a tradeoff between the error estimate and the mesh generation, because this two work are strongly 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.f

Metrix intersection

The unit ball $\mathcal{BM}$ in a metric $\mathcal{M}$ plot the maximum mesh size on all the direction, is a ellipse.
If you we have two unknowns u and v, we just compute the metric $\mathcal{M}_u$ and $\mathcal{M}_v$, find a metric $\mathcal{M}_{uv}$ call intersection with the biggest ellipse such that :

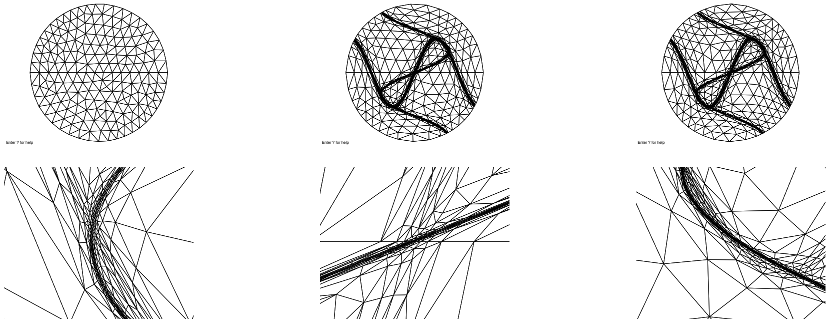
$$\mathcal{B}(\mathcal{M}_v) \subset \mathcal{B}(\mathcal{M}_u) \cap \mathcal{B}(\mathcal{M}_v)$$



Example of mesh

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

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



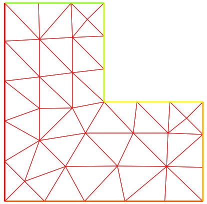
Execute Adapt-uv.edp

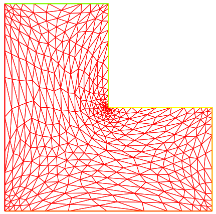
A corner singularity   adaptation with metric

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

$$\text{Find } u \in H^1_0(\Omega) \text{ such that } \quad -\Delta u = 1 \text{ in } \Omega,$$

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





example of Mesh adaptation

FreeFem++ corner singularity program

```
int[int] lab=[1,1,1,1];
mesh Th = square(6,6,label=lab);
Th=trunc(Th,x<0.5 | y<0.5, label=1);

fespace Vh(Th,P1);          Vh u,v;          real error=0.1;
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,u=0);
for (int i=0;i< 7;i++)
{  Probem1;                                                  //    solving the pde
   Th=adaptmesh(Th,u,err=error,nbvx=100000);
                                             //    the adaptation with Hessian of u
   plot(Th,u,wait=1,fill=1);                   u=u;
   error = error/ (1000.^(1./7.)) ;   };
```

Execute CornerLap.edp

Poisson equation with 3d mesh adaptation

```
load "msh3" load "tetgen" load "mshmet" load "medit"
int nn  = 6;
int[int] l1111=[1,1,1,1],l01=[0,1],l11=[1,1];                    //   label numbering
mesh3 Th3=buildlayers(square(nn,nn,region=0,label=l1111),
      nn,  zbound=[0,1], labelmid=l11,labelup = l01,labeldown = l01);
Th3=trunc(Th3,(x<0.5)|(y < 0.5)|(z < 0.5) ,label=1);             //   remove ]0.5, 1[^3

fespace Vh(Th3,P1);               Vh u,v;           //   FE. space definition
macro Grad(u) [dx(u),dy(u),dz(u)]                                //    EOM
problem Poisson(u,v,solver=CG) =
  int3d(Th3)( Grad(u)'*Grad(v) )  -int3d(Th3)( 1*v ) + on(1,u=0);

real errm=1e-2;                                             //   level of error
for(int ii=0; ii<5; ii++)
{
  Poisson;        Vh h;
  h[]=mshmet(Th3,u,normalization=1,aniso=0,nbregul=1,hmin=1e-3,
             hmax=0.3,err=errm);
  errm*= 0.8;                                     //   change the level of error
  Th3=tetgreconstruction(Th3,switch="raAQ",sizeofvolume=h*h*h/6.);
  medit("U-adap-iso-"+ii,Th3,u,wait=1);
}
```

Execute Laplace-Adapt-3d.edp

Move structure in meshes

```
load "msh3"  load "tetgen"  load "medit"  load "mmg3d"
include "MeshSurface.idp" real hs = 0.8;
int[int]  N=[4/hs,8/hs,11.5/hs];   real [int,int]  B=[[-2,2],[-2,6],[-10,1.5]];
int [int,int]  L=[[311,311],[311,311],[311,311]];
mesh3 ThH = SurfaceHex(N,B,L,1); mesh3 ThSg =Sphere(1,hs,300,-1);              //   "gluing" surface meshes
mesh3 ThSd =Sphere(1,hs,310,-1);
ThSd=movemesh3(ThSd,transfo=[x,4+y,z]); mesh3 ThHS=ThH+ThSg+ThSd;
real voltet=(hs^3)/6.;   real[int] domaine = [0,0,-4,1,voltet];
real [int] holes=[0,0,0,0,4,0];
mesh3 Th = tetg(ThHS,switch="pqaAAYYQ",regionlist=domaine,holelist=holes);
medit("Box-With-two-Ball",Th);
                                                                                 //   End build mesh

int[int] opt=[9,0,64,0,0,3];                    //   options of mmg3d see freeem++ doc
real[int] vit=[0,0,-0.3];                                                       //

fespace Vh(Th,P1);   macro Grad(u) [dx(u),dy(u),dz(u)]                           //
Vh uh,vh;                                        //   to compute the displacemnt field
problem Lap(uh,vh,solver=CG) = int3d(Th)(Grad(uh)'*Grad(vh))
  + on(310,300,uh=vit[2] ) + on(311,uh=0.);

for(int it=0; it<29; it++){
  Lap;     plot(Th,uh);
  Th=mmg3d(Th,options=opt,displacement=[0.,0.,uh],memory=1000);   }
```

Execute fallingspheres.edp

Build Matrix and vector of problem

The 3d FreeFem++ code :

```
mesh3 Th("dodecaedre.mesh");                            //     read a mesh from file
fespace Vh(Th,P1);                                     //     define the P1 FE space

macro Grad(u) [dx(u),dy(u),dz(u)] //     End of Macro

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
Vh u;
u[] = A^-1*b;                                         //     solve the linear system
```

Remark on varf

The functions appearing in the variational form are formal and local to the `varf` definition, the only important think is the order in the parameter list, like in

```
 varf vb1([u1,u2],[q]) = int2d(Th)( (dy(u1)+dy(u2)) *q) + int2d(Th)(1*q);
 varf vb2([v1,v2],[p]) = int2d(Th)( (dy(v1)+dy(v2)) *p) + int2d(Th)(1*p);
```

To build matrix A from the bilinear part the the variational form a of type `varf` do simply

```
  matrix B1 = vb1(Vh,Wh [, ...] );
  matrix<complex> C1 = vb1(Vh,Wh [, ...] );
//    where the fespace have the correct number of comp.
 //    Vh is "fespace" for the unknown fields with 2 comp.
 //    ex fespace Vh(Th,[P2,P2]); or fespace Vh(Th,RT);
 //    Wh is "fespace" for the test fields with 1 comp.
```

To build a vector, put $u1 = u2 = 0$ by setting 0 of on unknown part. `real[int]`

```
b = vb2(0,Wh);
complex[int]  c = vb2(0,Wh);
```

Remark : In this case the mesh use to defined , $\int, u, v$ can be different.

The boundary condition terms

First FreeFem use only the label number of edge (2d) or faces (3d).

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

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

- An "on" vectorial form (for Dirichlet) : `on(1,u1=g1,u2=g2)` If you have vectorial finite element like `RT0`, the 2 components are coupled, and so you have : $b[i] = "(\Pi_h(g1, g2))[i]" \times tgv$, where Π_h is the vectorial finite element interpolant.

- a linear form on Γ (for Neumann in 2d)

  `-int1d(Th)( f*w)` or `-int1d(Th,3))( f*w)`

- a bilinear form on Γ or Γ_2 (for Robin in 2d)

  `int1d(Th)( K*v*w)` or `int1d(Th,2)( K*v*w)`.

- a linear form on Γ (for Neumann in 3d)

  `-int2d(Th)( f*w)` or `-int2d(Th,3)( f*w)`

Freefem++ Tricks

What is simple to do with freefem++ :

- Evaluate variational form with Boundary condition or not.
- Do interpolation
- Do linear algebra
- Solve sparse problem.

Freefem++ Trick : extract Dof list of border

? Question how the list Degree of Freedom (DoF) of border.

Idea Take a function negative function, and increasing on the border, and sort do a simultaneous sort this value and DoF numbering.

Execute ListOfDofOnBorder.edp

Computation of error estimate $\eta_K = \sqrt{\int_K \mathrm{blabla}} = \sqrt{\int_\Omega w_k \mathrm{blabla}}$ where w_k is the basic function of `fespace Ph(Th,P0)`.

```
varf vetaK(unused,wK) = int2d(Th)( blabla * wK);
Ph etaK; etaK[] = vetaK(0,Ph);  etaK=sqrt(etaK);
```

Freefem++ Tricks

to Change Default sparse solver add following line :

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

if `MUMPS-seq` is available in file `$(HOME)/.freefem++.pref`

Diff How to compute, differential : use of `macro`

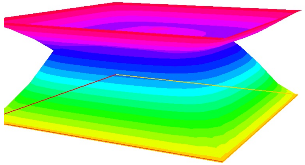
$$J(u) = \int_\Omega F(u); \qquad \texttt{macro F(u) =} \sqrt{1+\nabla u.\nabla u}$$

$$dJ(u)(v) = \int_\Omega dF(u,v); \qquad \texttt{macro dF(u,v)} = \frac{\nabla u.\nabla v}{\sqrt{1+\nabla u.\nabla u}}$$

$$ddJ(u)(v,w) = \int_\Omega ddF(u,v,w);$$

$$\texttt{macro ddF(u,v,v)} = \frac{\nabla w.\nabla v}{\sqrt{1+\nabla u.\nabla u}} - \frac{(\nabla u.\nabla v)(\nabla w.\nabla v)}{\sqrt{1+\nabla u.\nabla u}^3}$$

Optimization examples



The 2 membrane V.I. is find $(u^-, u^+) \in H^1(\Omega)^2$ $u^-_{|\Gamma} = g^-, u^+_{|\Gamma} = g^+$

$$\underset{u^- < u^+}{\text{argmin}} \ \frac{1}{2} \int_\Omega |\nabla u^-|^2 + |\nabla u^+|^2 - \int_\Omega u^- f^- + u^+ f^-$$

with $f^\pm, g^\pm$ are given function.

Variational Inequality

To solve just make a change of variable $u = u^+ - u^-, u > 0$ and $v = u^+ + u^-$, and we get a classical VI problem on u and and the Poisson on v.

So we can use the algorithm of Primal-Dual Active set strategy as a semi smooth Newton Method HinterMuller , K. Ito, K. Kunisch SIAM J. Optim. V 13, I 3, 2002.

In this case , we just do all implementation by hand in FreeFem++ language

Execute VI-2-membrane-adap.edp

Ipopt Data The IPOPT optimizer in a FreeFem++ script is done with the `IPOPT`

function included in the `ff-Ipopt` dynamic library. IPOPT is designed to solve constrained minimization problem in the form :

$$\begin{array}{ll} \text{find} & x_0 = \operatorname*{argmin}_{x \in \mathbb{R}^n} f(x) \\ \text{s.t.} & \left\{ \begin{array}{lr} \forall i \leq n, \ x_i^{\text{lb}} \leq x_i \leq x_i^{\text{ub}} & \text{(simple bounds)} \\ \forall i \leq m, \ c_i^{\text{lb}} \leq c_i(x) \leq c_i^{\text{ub}} & \text{(constraints functions)} \end{array} \right. \end{array} \qquad (3)$$

Where ub and lb stand for "upper bound" and "lower bound". If for some $i, 1 \leq i \leq m$ we have $c_i^{\text{lb}} = c_i^{\text{ub}}$, it means that c_i is an equality constraint, and an inequality one if $c_i^{\text{lb}} < c_i^{\text{ub}}$.

Ipopt Data, next

```
func real J(real[int] &X) {...}                              //   Fitness Function, return scalar
func real[int] gradJ(real[int] &X) {...}                              //   Gradient is a vector

func real[int] C(real[int] &X) {...}                                          //   Constraints
func matrix jacC(real[int] &X) {...}                                  //   Constraints jacobian

matrix jacCBuffer;                                         //   just declare, no need to define yet
func matrix jacC(real[int] &X)
{
  ...                                                                     //   fill jacCBuffer
  return jacCBuffer;
}
```

The hessian returning function is somewhat different because it has to be the hessian of the lagrangian function : $(x, \sigma_f, \lambda) \mapsto \sigma_f \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 c_i(x)$ where $\lambda \in \mathbb{R}^m$ and $\sigma \in \mathbb{R}$. Your hessian function should then have the following prototype :

```
matrix hessianLBuffer;                                                 //   just to keep it in mind
func matrix hessianL(real[int] &X,real sigma,real[int] &lambda) {...}
```

Ipopt Call

```
real[int] Xi = ... ;                                        //   starting point
IPOPT(J,gradJ,hessianL,C,jacC,Xi, ... );

IPOPT(J,gradJ,C,jacC,Xi,...);                              //   IPOPT with BFGS
IPOPT(J,gradJ,hessianJ,Xi,...);                               //   Newton IPOPT
                                                        //    without constraints
IPOPT(J,gradJ,Xi, ... );                                //   BFGS, no constraints
IPOPT(J,gradJ,Xi, ... );                                //   BFGS, no constraints
IPOPT([b,A],CC,ui1[],lb=lb1[],clb=cl[]..);                     //   affine case
  ...
```

Ipopt interface

```
load  "ff-Ipopt"
varf vP([u1,u2],[v1,v2]) = int2d(Th)(Grad(u1)'*Grad(v1)+ Grad(u2)'*Grad(v2))
- int2d(Th)(f1*v1+f2*v2);

matrix A = vP(Vh,Vh);                                         //   Fitness function matrix...
real[int] b = vP(0,Vh);                                                //   and linear form
int[int] II1=[0],II2=[1];                                           //   Constraints matrix
matrix C1 =  interpolate (Wh,Vh, U2Vc=II1);
matrix C2 =  interpolate (Wh,Vh, U2Vc=II2);
matrix CC = -1*C1 + C2;                                                     //    u2 - u1 >0
Wh cl=0;                               //   constraints lower bounds (no upper bounds)
varf vGamma([u1,u2],[v1,v2]) = on(1,2,3,4,u1=1,u2=1);
real[int] onGamma=vGamma(0,Vh);
Vh [ub1,ub2]=[g1,g2];
Vh [lb1,lb2]=[g1,g2];
ub1[] = onGamma ? ub1[] : 1e19  ;                            //   Unbounded in interior
lb1[] = onGamma ? lb1[] : -1e19  ;
Vh [uzi,uzi2]=[uz,uz2],[lzi,lzi2]=[lz,lz2],[ui1,ui2]=[u1,u2];;
Wh lmi=lm;
IPOPT([b,A],CC,ui1[],lb=lb1[],clb=cl[],ub=ub1[],warmstart=iter>1,uz=uzi[],lz=lzi[],lm=lmi[]);
```

Execute IpoptLap.edp   Execute IpoptVI2.edp   Execute IpoptMinSurfVol.edp

NLopt interface

```
load  "ff-NLopt"
...
if(kas==1)
   mini = nloptAUGLAG(J,start,grad=dJ,lb=lo,ub=up,IConst=IneqC,
           gradIConst=dIneqC,subOpt="LBFGS",stopMaxFEval=10000,
           stopAbsFTol=starttol);
else if(kas==2)
  mini = nloptMMA(J,start,grad=dJ,lb=lo,ub=up,stopMaxFEval=10000,
         stopAbsFTol=starttol);
else if(kas==3)
  mini = nloptAUGLAG(J,start,grad=dJ,IConst=IneqC,gradIConst=dIneqC,
         EConst=BC,gradEConst=dBC,
         subOpt="LBFGS",stopMaxFEval=200,stopRelXTol=1e-2);
else if(kas==4)
  mini = nloptSLSQP(J,start,grad=dJ,IConst=IneqC,gradIConst=dIneqC,
        EConst=BC,gradEConst=dBC,
        stopMaxFEval=10000,stopAbsFTol=starttol);
```

Execute VarIneq2.edp

Stochastic interface

This algorithm works with a normal multivariate distribution in the parameters space and try to adapt its covariance matrix using the information provides by the successive function evaluations. Syntaxe : `cmaes(J,u[],..)` ( )
From `http://www.lri.fr/~hansen/javadoc/fr/inria/optimization/cmaes/package-summary.html`

Stochastic Exemple

```
load  "ff-cmaes"

real mini = cmaes(J,start,stopMaxFunEval=10000*(al+1),
                stopTolX=1.e-4/(10*(al+1)),
                initialStdDev=(0.025/(pow(100.,al))));
SSPToFEF(best1[],best2[],start);
```

Execute cmaes-VarIneq.edp

```
load  "mpi-cmaes"

real mini = cmaesMPI(J,start,stopMaxFunEval=10000*(al+1),
                stopTolX=1.e-4/(10*(al+1)),
                initialStdDev=(0.025/(pow(100.,al))));
SSPToFEF(best1[],best2[],start);
```

remark, the FreeFem `mpicommworld` is used by default. The user can specify his own MPI communicator with the named parameter "`comm=`", see the MPI section of this manual for more informations about communicators in `FreeFem++`.

Bose Einstein Condensate

The problem is find a complex field u on domain $\mathcal{D}$ such that :

$$u = \underset{||u||=1}{\operatorname{argmin}} \int_{\mathcal{D}} \frac{1}{2}|\nabla u|^2 + V_{trap}|u|^2 + \frac{g}{2}|u|4 - \Omega i\overline{u}(\left(\begin{smallmatrix}-y\\x\end{smallmatrix}\right).\nabla)u$$

How to code that in FreeFem++

use

- Ipopt interface (`https://projects.coin-or.org/Ipopt`)
- Adaptation de maillage

Execute BEC.edp

This example illustrates the coupling of natural convection modeled by the Boussinesq approximation and liquid to solid phase change in $\Omega =]0,1[^2$, No slip condition for the fluid are applied at the boundary and adiabatic condition on upper and lower boundary and given temperature θ_r (resp θ_l) at the right and left boundaries. The starting point of the problem is Brainstorming session (part I) of the third FreeFem++ days in december , 2012, this is almost the Orange Problem is describe in web page
`http://www.ljll.math.upmc.fr/~hecht/ftp/ff++days/2011/Orange-problem.pdf`.
So the full model is : find $\boldsymbol{u} = (u_1, u_2)$, the velocity field, p the pressure field and θ the temperature flied in domain Ω such that

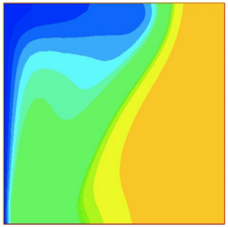
$$\left\{ \begin{array}{rll} \boldsymbol{u} & \text{given} & \text{in } \Omega_s \\ \partial_t \boldsymbol{u} + (\boldsymbol{u}\nabla)\boldsymbol{u} + \nabla.\mu\nabla\boldsymbol{u} + \nabla p & = -c_T \boldsymbol{e}_2 & \text{in } \Omega_f \\ \nabla.\boldsymbol{u} & = 0 & \text{in } \Omega_f \\ \partial_t \theta + (\boldsymbol{u}\nabla)\theta + \nabla.k_T\nabla\theta & = \partial_t S(T) & \text{in } \Omega \end{array} \right. \tag{4}$$

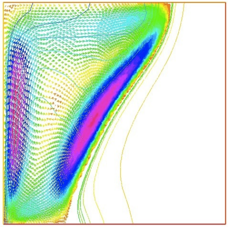
Where Ω_f is the fluid domain and the solid domain is $\Omega_s = \Omega \setminus \Omega_f$. The enthalpy of the change of phase is given by the function S ; μ is the relative viscosity, k_T the thermal diffusivity.
In $\Omega_f = \{x \in \Omega; \theta > \theta_f\}$, with θ_m the melting temperature the solid has melt.
We modeled, the solid phase as a fluid with huge viscosity, so :

$$\mu = \left\{ \begin{array}{lcc} \theta < \theta_f & \sim & 10^6 \\ \theta \geq \theta_m & \sim & \frac{1}{\mathrm{Re}} \end{array} \right. ,$$

The Stefan enthalpy S_c with defined by $S_c(\theta) = H(\theta)/S_{th}$ where S_{the} is the stefan number and H is the Heaviside function with use the following smooth the enthalpy :

$$S(\theta) = \frac{\tanh(50(\theta - \theta_m)))}{2S_{te}}.$$



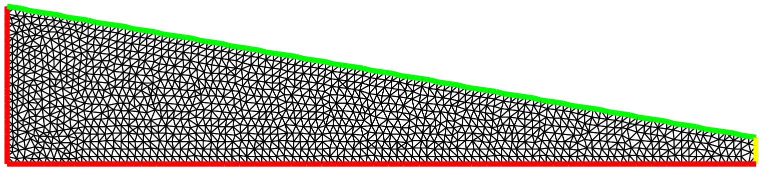


A Free Boundary problem , (phreatic water)

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;}; //    impermeable Γa
border b(t=0,h1){x=L;y=t;label=2;};     //    the source Γb
border f(t=L,0){x=t;y=t*(h1-h)/L+h;label=3;}; //    Γf
border d(t=h,0){x=0;y=t;label=4;};     //    Left impermeable Γd
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)

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

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

Eigenvalue/ Eigenvector example

The problem, Find the first λ, u_λ such that :

$$a(u_\lambda, v) = \int_\Omega \nabla u_\lambda \nabla v = \lambda \int_\Omega u_\lambda v = \lambda b(u_\lambda, v)$$

the boundary condition is make with exact penalization : we put $1e30 = tgv$ on the diagonal term of the lock degree of freedom. So take Dirichlet boundary condition only with a variational form and not on b variational form , because we compute eigenvalue of

$$w = A^{-1} B v$$

Otherwise we get spurios mode.

Eigenvalue/ Eigenvector example code

```
...
fespace Vh(Th,P1);
macro Grad(u) [dx(u),dy(u),dz(u)]                                    //    EOM
varf  a(u1,u2)= int3d(Th)(  Grad(u1)'*Grad(u2) +  on(1,u1=0) ;
varf b([u1],[u2]) = int3d(Th)(  u1*u2 ) ;                            //    no BC
matrix A= a(Vh,Vh,solver=UMFPACK),
        B= b(Vh,Vh,solver=CG,eps=1e-20);

int nev=40;                    //    number of computed eigenvalue close to 0
real[int] ev(nev);                                 //    to store nev eigenvalue
Vh[int] eV(nev);                                  //    to store nev eigenvector
int k=EigenValue(A,B,sym=true,value=ev,vector=eV);
k=min(k,nev);
for (int i=0;i<k;i++)
   plot(eV[i],cmm="ev  "+i+" v =" + ev[i],wait=1,value=1);
```

Execute Lap3dEigenValue.edp Execute LapEigenValue.edp

Linear elasticty equation, weak form

Let a domain $\Omega \subset \mathbb{R}^d$ with a partition of $\partial\Omega$ in Γ_d, Γ_n.
Find the displacement $\boldsymbol{u}$ field such that :

$$-\nabla.\sigma(\boldsymbol{u}) = \boldsymbol{f} \text{ in } \Omega, \quad \mathbf{u} = \mathbf{0} \text{ on } \mathbf{\Gamma_d}, \quad \sigma(\mathbf{u}).\mathbf{n} = \mathbf{0} \text{ on } \mathbf{\Gamma_n} \tag{5}$$

Where $\varepsilon(\boldsymbol{u}) = \frac{1}{2}(\nabla\boldsymbol{u} + {}^t\nabla\boldsymbol{u})$ and $\sigma(\boldsymbol{u}) = \boldsymbol{A}\varepsilon(\boldsymbol{u})$ with $\boldsymbol{A}$ the linear positif operator on symmetric $d \times d$ matrix corresponding to the material propriety. Denote
$V_{\boldsymbol{g}} = \{\boldsymbol{v} \in H^1(\Omega)^d / \boldsymbol{v}_{|\Gamma_d} = \boldsymbol{g}\}$.
The Basic displacement variational formulation with is : find $\boldsymbol{u} \in V_0(\Omega)$, such that

$$\int_\Omega \varepsilon(\boldsymbol{v}) : \boldsymbol{A}\varepsilon(\boldsymbol{u}) = \int_\Omega \boldsymbol{v}.\boldsymbol{f} + \int_\Gamma ((\boldsymbol{A}\varepsilon(\boldsymbol{u})).n).v, \quad \forall \boldsymbol{v} \in V_0(\Omega) \tag{6}$$

Poisson equation with Schwarz method

- Introduction `Freefem++`
- new parallel solver
- Schwarz algorithm to solve Poisson
    - Construction of the local partition of the unity
    - Construction of the overlapping,
    - the Schwarz alternating method,
    - asynchronous Send/Receive (Hard)
    - an GMRES version for Schwarz alternating method,
    - Parallel visualisation for debugging

Poisson equation with Schwarz method

To solve the following Poisson problem on domain Ω with boundary Γ in $L^2(\Omega)$:

$$-\Delta u = f, \text{ in } \Omega, \text{ and } u = g \text{ on } \Gamma,$$

where f and g are two given functions of $L^2(\Omega)$ and of $H^{\frac{1}{2}}(\Gamma)$,
Let introduce $(\pi_i)_{i=1,..,N_p}$ a regular partition of the unity of Ω, q-e-d :

$$\pi_i \in \mathcal{C}^0(\Omega) : \quad \pi_i \geq 0 \text{ and } \sum_{i=1}^{N_p} \pi_i = 1.$$

Denote Ω_i the sub domain which is the support of π_i function and also denote Γ_i the boundary of Ω_i.
The parallel Schwarz method is Let $\ell = 0$ the iterator and a initial guest u^0 respecting the boundary condition (i.e. $u^0_{|\Gamma} = g$).

$$\forall i = 1.., N_p : \quad -\Delta u_i^\ell = f, \text{ in } \Omega_i, \quad \text{ and } u_i^\ell = u^\ell \text{ on } \Gamma_i \setminus \Gamma, \; u_i^\ell = g \text{ on } \Gamma_i \cap \Gamma \tag{7}$$

$$u^{\ell+1} = \sum_{i=1}^{N_p} \pi_i u_i^\ell \tag{8}$$

After discretization with the Lagrange finite element method, with a compatible mesh $\mathcal{T}_{h\,i}$ of Ω_i, i. e., the exist a global mesh $\mathcal{T}_h$ such that $\mathcal{T}_{h\,i}$ is include in $\mathcal{T}_h$.
Let us denote :

- $V_{h\,i}$ the finite element space corresponding to domain Ω_i,
- $\mathcal{N}_{h\,i}$ is the set of the degree of freedom σ_i^k,
- $\mathcal{N}_{hi}^{\Gamma_i}$ is the set of the degree of freedom of $V_{h\,i}$ on the boundary Γ_i of Ω_i,
- $\sigma_i^k(v_h)$ is the value the degree of freedom k,
- $V_{0h\,i} = \{v_h \in V_{h\,i} : \forall k \in \mathcal{N}_{hi}^{\Gamma_i}, \quad \sigma_i^k(v_h) = 0\}$,
- the conditional expression $a\ ?\ b : c$ is defined like in `C` of `C++` language by

$$a?b : c \equiv \begin{cases} \text{if } a \text{ is true then return } b \\ \text{else return } c \end{cases} .$$

Remark we never use finite element space associated to the full domain Ω because it to expensive.

We have to defined to operator to build the previous algorithm :
We denote $u_{h\,|i}^\ell$ the restriction of u_h^ℓ on $V_{h\,i}$, so the discrete problem on Ω_i of problem (7) is find $u_{h\,i}^\ell \in V_{h\,i}$ such that :

$$\forall v_{h\,i} \in V_{0i} : \int_{\Omega_i} \nabla v_{h\,i} . \nabla u_{h\,i}^\ell = \int_{\Omega_i} f v_{h\,i}, \quad \forall k \in \mathcal{N}_{hi}^{\Gamma_i} \ : \ \sigma_i^k(u_{h\,i}^\ell) = (k \in \Gamma)\ ?\ g_i^k : \sigma_i^k(u_{h\,|i}^\ell)$$

where g_i^k is the value of g associated to the degree of freedom $k \in \mathcal{N}_{hi}^{\Gamma_i}$.

In FreeFem++, it can be written has with U is the vector corresponding to $u^{\ell}_{h|i}$ and the vector U1 is the vector corresponding to $u^{\ell}_{h\,i}$ is the solution of :

```
  real[int] U1(Ui.n)   ,    b= onG .* U;
  b   = onG ? b  : Bi ;       U1  = Ai^-1*b;
```

where $\mathtt{onG}[i] = (i \in \Gamma_i \setminus \Gamma)?1 : 0$, and Bi the right of side of the problem, are defined by

```
  fespace Whi(Thi,P2);                                                                      //    def of the Finite element space.
  varf vPb(U,V)= int3d(Thi)(grad(U)'*grad(V)) + int3d(Thi)(F*V) +on(1,U=g) + on(10,U=G);
  varf vPbon(U,V)=on(10,U=1);
  matrix Ai = vPb(Whi,Whi,solver=sparsesolver);
  real[int] onG = vPbon(0,Whi);
  real[int] Bi=vPb(0,Whi);
```

where the freefem++ label of Γ is 1 and the label of $\Gamma_i \setminus \Gamma$ is 10.

To build the transfer/update part corresponding to (8) equation on process i, let us call njpart the number the neighborhood of domain of Ω_i (i.e : π_j is none 0 of Ω_i), we store in an array jpart of size njpart all this neighborhood. Let us introduce two array of matrix, Smj[j] to defined the vector to send from i to j a neighborhood process, and the matrix $rMj[j]$ to after to reduce owith neighborhood j domain.
So the tranfert and update part compute $v_i = \pi_i u_i + \sum_{j\in J_i} \pi_j u_j$ and can be write the freefem++ function Update :

```
func bool Update(real[int] &ui, real[int] &vi)
{ int n= jpart.n;
  for(int j=0;j<njpart;++j)  Usend[j][]=sMj[j]*ui;
  mpiRequest[int] rq(n*2);
  for (int j=0;j<n;++j) Irecv(processor(jpart[j],comm,rq[j  ]), Ri[j][]);
  for (int j=0;j<n;++j) Isend(processor(jpart[j],comm,rq[j+n]), Si[j][]);
  for (int j=0;j<n*2;++j) int k= mpiWaitAny(rq);
  vi = Pii*ui;//    set to $\pi_i u_i$ // apply the unity local partition .
   for(int j=0;j<njpart;++j)  vi += rMj[j]*Vrecv[j][];                                                        //    add $\pi_j u_j$
 return true; }
```

where the buffer are defined by :

```
  InitU(njpart,Whij,Thij,aThij,Usend)  //    defined the send buffer
  InitU(njpart,Whij,Thij,aThij,Vrecv)  //    defined the revc buffer
```

with the following macro definition :

```
macro InitU(n,Vh,Th,aTh,U) Vh[int] U(n); for(int j=0;j<n;++j) {Th=aTh[j];  U[j]=0;}                              //   EOM
```

Finally you can easily accelerate the fixe point algorithm by using a parallel GMRES algorithm after the introduction the following affine $\mathcal{A}_i$ operator sub domain Ω_i.

```
func real[int] Ai(real[int]& U) {
 real[int] V(U.n)    ,  b= onG .* U;
  b  = onG ? b : Bi ;
  V = Ai^-1*b;
  Update(V,U);
  V -= U;   return V; }
```

Where the parallel MPIGMRES or MPICG algorithm is just a simple way to solve in parallel the following $A_i x_i = b_i, i = 1,.., N_p$ by just changing the dot product by reduce the local dot product of all process with the following MPI code :

```
template<class R> R ReduceSum1(R s,MPI_Comm * comm)
{   R r=0;
    MPI_Allreduce( &s, &r, 1 ,MPI_TYPE<R>::TYPE(),   MPI_SUM,  *comm );
    return r; }
```

This is done in MPIGC dynamics library tool.

We have all ingredient to solve in parallel if we have et the partitions of the unity. To build this partition we do : the initial step on process 1 to build a coarse mesh, $\mathcal{T}_h{}^*$ of the full domain, and build the partition π function constant equal to i on each sub domain $\mathcal{O}_i, i = 1,.., N_p$, of the grid with the Metis graph partitioner and on each process i in $1.., N_p$ do

1. Broadcast from process 1, the mesh ${\mathcal{T}_h}^*$ (call Thii in freefem++ script), and π function,
2. remark that the characteristic function $\mathbf{1}_{\mathcal{O}_i}$ of domain $\mathcal{O}_i$, is defined by $(\pi = i)?1:0$,
3. let us call Π^2_P (resp. Π^2_V) the L^2 on P^*_h the space P0 on ${\mathcal{T}_h}^*$ (resp. V^*_h the space P1 one ${\mathcal{T}_h}^*$). build in parallel the π_i and Ω_i, such that $\mathcal{O}_i \ \subset \Omega_i$ where $\mathcal{O}_i = supp((\Pi^2_V \Pi^2_C)^m \mathbf{1}_{O_i})$, and m is a the overlaps size on the coarse mesh We choose a function $\pi^*_i = (\Pi^2_1 \Pi^2_0)^m \mathbf{1}_{\mathcal{O}_i}$ so the partition of the unity is simply defined by $\pi_i = \pi^*_i (\sum_{j=1}^{N_p} \pi^*_j)^{-1}$

   The set J_i of neighborhood of the domain Ω_i, $V_{h\,i}$, jpart and njpart can be defined with :

```
Vhi pii=$\pi^*_i$ ;  Vhi[int] pij(npij);                        //    local partition of 1 = pii + $\sum_j$ pij[j]
int[int] jpart(npart);   int njpart=0;     Vhi sumphi =   $\pi^*_i$  ;
for (int i=0;i<npart;++i)
  if(i != ipart ) {
     if(int3d(Thi)( $\pi^*_j$)>0) {  pij[njpart]=$\pi^*_j$;
       sumphi[] += pij[njpart][];   jpart[njpart++]=i;}}}
     pii[]=pii[] ./ sumphi[];
for (int j=0;j<njpart;++j) pij[j][] = pij[j][] ./ sumphi[];
jpart.resize(njpart);
```

4. We call ${\mathcal{T}_h}^*_{ij}$ the sub mesh part of $\mathcal{T}_{h\,i}$ where π_j are none zero. and tank to the function trunc,

```
for(int jp=0;jp<njpart;++jp)  aThij[jp]  = trunc(Thi,pij[jp]>1e-10,label=10);
```

5. At this step we have all on the coarse mesh , so we can build the final fine mesh by splitting all meshes : Thi, Thij[j],Thij[j] with freefem++ trunc mesh function.

6. The construction of the send/recv matrices sMj and rMj : can done with this code :

```
mesh3 Thij=Thi;  fespace Whij(Thij,Pk);                              //    variable fespace meshes..
matrix Pii; Whi wpii=pii; Pii = wpii[];                        //    Diagonal matrix corresponding $\times\pi_i$
matrix[int] sMj(njpart), rMj(njpart);                                  //    M send/rend case..
 for(int jp=0;jp<njpart;++jp)
   { int j=jpart[jp]; Thij = aThij[jp];                          //   change mesh to change Whij,Whij
     matrix I = interpolate(Whij,Whi);                                      //    Whij <- Whi
     sMj[jp] = I*Pii;       rMj[jp] = interpolate(Whij,Whi,t=1); }}         //    Whij -> Whi
```

To build a not to bad application, I have add code tout change variable from parameter value with the following code

```
include "getARGV.idp"
verbosity=getARGV("-vv",0);
...
bool gmres=getARGV("-gmres",1);
bool dplot=getARGV("-dp",0);
```

And small include to make graphic in parallel of distribute solution of vector u on mesh T_h with the following interface :

```
include "MPIplot.idp"
func bool  plotMPIall(mesh3 &Th,real[int] & u,string  cm)
{ PLOTMPIALL(mesh3,Pk, Th, u,{cmm=cm,nbsio=3});                              //    MPI plot
 return 1;}
```

A Parallel Numerical experiment on laptop

We consider first example in an academic situation to solve Poisson Problem on the cube $\Omega =]0,1[^3$

$$-\Delta u = 1, \text{ in } \Omega; \qquad u = 0, \text{ on } \partial\Omega. \tag{9}$$

With a cartesian meshes $\mathcal{T}_{hn}$ of Ω with $6n^3$ tetrahedron, the coarse mesh is $\mathcal{T}^*_{hm}$, and m is a divisor of n.
We do the validation of the algorithm on a Laptop Intel Core i7 with 4 core at 2.66 Ghz with $8Go$ of RAM DDR3 at 1067 Mhz,

Execute DDM-Schwarz-Lap-2dd.edp Execute DDM-Schwarz-Lame-2d.edp Execute DDM-Schwarz-Lame-3d.edp

Linear Lame equation, weak form

Let a domain $\Omega \subset \mathbb{R}^d$ with a partition of $\partial\Omega$ in Γ_d, Γ_n.
Find the displacement $\boldsymbol{u}$ field such that :

$$-\nabla.\sigma(\boldsymbol{u}) = \boldsymbol{f} \text{ in } \Omega, \quad \mathbf{u} = \mathbf{0} \text{ on } \mathbf{\Gamma_d}, \quad \sigma(\mathbf{u}).\mathbf{n} = \mathbf{0} \text{ on } \mathbf{\Gamma_n} \tag{10}$$

Where $\varepsilon(\boldsymbol{u}) = \frac{1}{2}(\nabla\boldsymbol{u} + {}^t\nabla\boldsymbol{u})$ and $\sigma(\boldsymbol{u}) = \boldsymbol{A}\varepsilon(\boldsymbol{u})$ with $\boldsymbol{A}$ the linear positif operator on symmetric $d \times d$ matrix corresponding to the material propriety. Denote
$V_{\boldsymbol{g}} = \{\boldsymbol{v} \in H^1(\Omega)^d / \boldsymbol{v}_{|\Gamma_d} = \boldsymbol{g}\}$.
The Basic displacement variational formulation is : find $\boldsymbol{u} \in V_0(\Omega)$, such that

$$\int_\Omega \varepsilon(\boldsymbol{v}) : \boldsymbol{A}\varepsilon(\boldsymbol{u}) = \int_\Omega \boldsymbol{v}.\boldsymbol{f} + \int_\Gamma ((\boldsymbol{A}\varepsilon(\boldsymbol{u})).n).v, \quad \forall \boldsymbol{v} \in V_0(\Omega) \tag{11}$$

Linear elasticty equation, in FreeFem++

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

```
load "medit"   include "cube.idp"
int[int]  Nxyz=[20,5,5];
real [int,int]  Bxyz=[[0.,5.],[0.,1.],[0.,1.]];
int [int,int]  Lxyz=[[1,2],[2,2],[2,2]];
mesh3 Th=Cube(Nxyz,Bxyz,Lxyz);
                                                              //    Alu ...
real rhoAlu = 2600, alu11=  1.11e11 , alu12 = 0.61e11 ;
real alu44= (alu11-alu12)*0.5;
func Aalu = [  [alu11, alu12,alu12,   0.   ,0.   ,0.  ],
               [alu12, alu11,alu12,   0.   ,0.   ,0.   ],
               [alu12, alu12,alu11,   0.   ,0.   ,0.   ],
               [0. ,   0. ,   0. ,    alu44,0.   ,0.   ],
               [0. ,   0. ,   0. ,    0.   ,alu44,0.   ],
               [0. ,   0. ,   0. ,    0.   ,0.   ,alu44]   ];
real gravity = -9.81;
```

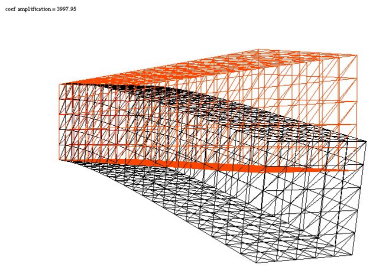
Linear elasticty equation, in FreeFem++

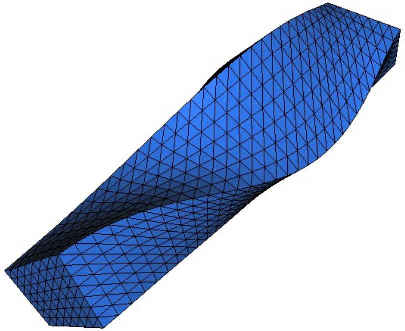
The problem code :

```
fespace Vh(Th,[P1,P1,P1]);
Vh [u1,u2,u3], [v1,v2,v3];
macro Strain(u1,u2,u3) [ dx(u1), dy(u2),dz(u3), (dz(u2)
  +dy(u3)), (dz(u1)+dx(u3)), (dy(u1)+dx(u2)) ]                       //    EOM
solve Lame([u1,u2,u3],[v1,v2,v3])=
  int3d(Th)(  Strain(v1,v2,v3)'*(Aalu*Strain(u1,u2,u3))  )
           - int3d(Th) ( rhoAlu*gravity*v3)
           + on(1,u1=0,u2=0,u3=0) ;

real coef= 0.1/u1[].linfty;  int[int] ref2=[1,0,2,0];
mesh3 Thm=movemesh3(Th,
     transfo=[x+u1*coef,y+u2*coef,z+u3*coef],
     label=ref2);
plot(Th,Thm, wait=1,cmm="coef  amplification = "+coef );
medit("Th-Thm",Th,Thm);
```

Lame equation / figure





Execute beam-3d.edp Execute beam-EV-3d.edp Execute beam-3d-Adapt.edp

Hyper elasticity equation

The Hyper elasticity problem is the minimization of the energy $W(I_1, I_2, I_3)$ where I_1, I_2, I_3 are the 3 invariants. For example The Ciarlet Geymonat energy model is

$$W = \kappa_1(J_1 - 3) + \kappa_2(J_2 - 3) + \kappa(J - 1) - \kappa \ln(J)$$

where $J_1 = I_1 I_3^{-\frac{1}{3}}$, $J_2 = I_2 I_3^{-\frac{2}{3}}$,$J = I_3^{\frac{1}{2}}$,
let u the deplacement, when

- $F = I_d + \nabla u$
- $C = {}^tFF$
- $I_1 = \mathrm{tr}(C)$
- $I_2 = \frac{1}{2}(\mathrm{tr}(C)^2 - \mathrm{tr}(C^2))$
- $I_3 = \det(C)$

The problem is find

$$u = \underset{u}{\mathtt{argmin}}\, W(I_1, I_2, I_3)$$

Hyper elasticity equation

Defined $C2 \equiv C$ from d and the differential at order 1 and 2 call $dC2$, $ddC2$

```
macro C2(d) [
1. + 2.*dx(d[0]) + dx(d[0])*dx(d[0]) + dx(d[1])*dx(d[1]) ,
1. + 2.*dy(d[1]) + dy(d[0])*dy(d[0]) + dy(d[1])*dy(d[1]) ,
 dy(d[0]) + dx(d[1])  + dx(d[0])*dy(d[0]) + dx(d[1])*dy(d[1]) ] //   OEM
```

Definition of the 3 invariants from the vector C and the differential at order 1 and 2 call $dI2C$, $dI2C$

```
macro I2C(C)  [
  C[0] + C[1] + 1.,
  C[0]*C[1] + C[1] + C[0] - C[2]*C[2],
  C[0]*C[1] - C[2]*C[2] ] //   EOM
```

Use the chain rule of compute de differential of the energy W.

```
macro I2d(d) I2C(C2(d))                                                    //
macro dI2d(d,dd)  dI2C( C2(d) , dC2(d,(dd)) )                              //
macro ddI2d(d,dd,ddd) (ddI2C( dC2(d,(dd)),dC2(d,(ddd)))+ dI2C( C2(d),ddC2((dd),(ddd)))) //
macro W2d(d) W(I2d(d))                                                     //
macro dW2d(d,dd)  dW( I2d(d) , dI2d(d,(dd))  )                             //
macro ddW2d(d,dd,ddd)
      (ddW(I2d(d),dI2d(d,(dd)),dI2d(d,(ddd)))  + dW( I2d(d) , ddI2d(d,(dd),(ddd)) ) )   //
```

remark : To slow in 3d, I will polynomn of operator to accelerate, because the application $d \mapsto [I_1, I_2, I_2]$ are polynome-n of ∇d

Hyper elasticity equation

```
fespace Wh(Th,[P2,P2]);
                                                    //    Newton's Method ..
Wh [d1,d2]=[0,0];
Wh [w1,w2],[v1,v2];
for(int i=0;i<Nnewton;++i)
 {
    solve dWW([w1,w2],[v1,v2]) =
         int2d(Th)( ddW2d([d1,d2],[w1,w2],[v1,v2]) )
       - int2d(Th)( dW2d([d1,d2],[v1,v2]) -[v1,v2]'*[f1,f2] )
       + on(1,w1=0,w2=0);

     d1[] -= w1[];
     real err = w1[].linfty;
     if(err< epsNewton) break;
 }
```

Execute Hyper-Elasticity-2d.edp   Execute ElasticLaw2d.idp   Execute CiarletGemona.idp

The formulation Primitive equation, variational form

find $\boldsymbol{u}_H \in \boldsymbol{V}_g$ and $\overline{p}^z \in W$ such that $\forall \boldsymbol{v}_H \in \boldsymbol{V}_0, \overline{q}^z \in W$:

$$\begin{aligned}\int_\Omega \nu_H \nabla_H \boldsymbol{u}_H . \nabla_H \boldsymbol{v}_H + \nu_z \nabla_z \boldsymbol{u}_H . \nabla_z \boldsymbol{v}_H \\ - \int_\omega \overline{\nabla_H . \boldsymbol{u}_H}^z \overline{q}^z \\ - \int_\omega \overline{\nabla_H . \boldsymbol{v}_H}^z \overline{p}^z \quad &= \quad - \int_\Omega f \boldsymbol{v}_H + ...\end{aligned}$$

where Ω is the horizontal trace of Ω. and u_3 is reconstruct from $\boldsymbol{u}_H$ with :

$$\forall v_3 \in V_0, \quad \int_\Omega \partial_z u_3 \partial_z v_3 = - \int_\Omega \nabla_H . \boldsymbol{u}_H \partial_z v_3$$

with $V_0 = \{ v \in H^1(\Omega), v_{|\Gamma_d} = 0, v_{|\Gamma_0} = 0 \}$.

The formulation Primitive equation, FreeFem++

$\boldsymbol{V}$ is `fespace(Th,[P2,P2])`,
W is `fespace(Thp,[P1],periodic[[G0,x,y],[[Gd,x,y]])`, where `Thp` is a mesh with one layer to build a 2d FE element space because No interpolation between 2d and 3d `fespace`.

```
mesh Th2D=square(nn,nn);
int[int] refm=[1,1,2,1,3,10,4,1], refu=[0,3], refd=[0,2];
fespace Vh2(Th2D,P2);
mesh3 Th=buildlayers(Th2D,nn,zbound=[-1.,0.],
          labelmid=refm,labelup=refu,labeldown=refd);
mesh3 Thp=buildlayers(Th2D,1,zbound=[-1.,0.],
          labelmid=refm,labelup=refu,labeldown=refd);

fespace Vh(Th,[P2,P2,P1]) ;                                        //    for u1,u2, p
fespace Uh(Th,[P2]) ;
fespace Wh(Th,[P2,P2]) ;                                              //    for u1,u2
fespace Ph2(Thp,[P1],periodic=[[2,x,y],[3,x,y]]) ;                          //   2d
```

Injection , FreeFem++

The Injection $I_h : (u_1, u_2, \overline{p}^z) \mapsto (u_1, u_2, p)$

```
matrix Ih;                              //    matrix du go from 3d,3d,2d -> 3d,3d,3d
{
                                               //    numbering first u1,u2 second p.
int[int] V2Pc=[-1,-1,0],  V2Wc=[0,1,-1];                           //    u1, u2

matrix IPh =interpolate(Vh,Ph2,U2Vc=V2Pc);
                                                                 //    Π_Vh(v ∈ Ph2)
matrix IWh =interpolate(Vh,Wh,U2Vc=V2Wc);
int n = IPh.n;
int m =  IWh.m+IPh.m;
int[int] nuV=0:n-1;                                                     //    id
int[int] nuW=0:IWh.m-1;                                              //    u1,u2
int[int] nuP=IWh.m:m-1;                                             //    p (2d)
matrix JJh = IPh(nuV^-1,nuP^-1);
matrix IIh = IWh(nuV^-1,nuW^-1);
Ih =  JJh + IIh;
}
```

Matrix , FreeFem++

```
varf vPrimitive([u1,u2,p3],[v1,v2,q3]) =
      int3d(Th)(
          nuH * ( GradH(u1)'*GradH(v1)
                + GradH(u2)'*GradH(v2)  )
        + nuz * (dz(u1)*dz(v1) + dz(u2)*dz(v2) )
        - div2(u1,u2)* q3                                      //    - du3*q3
        - div2(v1,v2)* p3                                      //    - dv3*p3
        - 1e-6*p3*q3                    //     mean value for p3 is zero ..
      )                                                      //     bil. form
    + int2d(Th,3) ( (u1*v1 + u2*v2)*0.00001)
    + int2d(Th,3)  ( v1*0.0001 )                                      //
    + on(10,u1=x*(1-x),u2=0)+ on(1,u1=0,u2=0)
    + on(2,u1=0,u2=0);
matrix AA = vPrimitive(Vh,Vh,tgv=ttgv) ;
matrix A = AA*Ih;  A = Ih'*A;                                //    A = Ih'*A*Ih .
```

Solve , FreeFem++

```
real[int] bb= vPrimitive(0,Vh,tgv=ttgv) ;
real[int] b=Ih'*bb;
real[int]  uu= A^-1*b;
Vh [u1,u2,p3] ;
u1[] = Ih*uu;

varf vU3(u3,v3)=int3d(Th)( dz(u3)*dz(v3) )
        + on(3,2,u3=0);
varf vU3U([u1,u2],[v3])=
     int3d(Th)(-1.*div2(u1,u2)*dz(v3));

matrix A3= vU3(Uh,Uh), B3= vU3U(Vh,Uh);

real[int] b3 = B3*u1[];
Uh u3;  u3[] = A3^-1*b3;
```

Conclusion/Future

Freefem++ v3 is

- very good tool to solve non standard PDE in 2D/3D
- to try new domain decomposition domain algorithm

The the future we try to do :

- Build more graphic with VTK, paraview , ... (in progress)
- Add Finite volume facility for hyperbolic PDE (just begin C.F. FreeVol Projet)
- 3d anisotrope mesh adaptation
- automate the parallel tool

Thank for you attention.