Numerical Simulation in Physics and Engineering with FreeFem++

F. Hecht,
LJLL, Sorbonne Université, Paris  projet Alpines, Inria de Paris
with Ionut Danaila, Pierre Jolivet, Frédéric Nataf, Pierre-Henri Tournier, Victorita Dolean

XVIII Spanish-French School Jacques-Louis Lions
Las Palmas de Gran Canaria, 25-29 June 2018

http://www.freefem.org mailto:frederic.hecht@upmc.fr
Outline

1 Introduction
2 Tools
3 Academic Examples
4 Numerics Tools
5 MPI/Parallel
6 No Linear Problem
7 Exercises
8 Technical Remark on freefem++
Outline

1 Introduction
2 Tools
3 Academic Examples
4 Numerics Tools
5 MPI/Parallel
6 No Linear Problem
7 Exercises
8 Technical Remark on freefem++
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.

To try of cell phone https://www.ljll.math.upmc.fr/lehyaric/ffjs/

Info: FreeFem++ solve a problem with $22 \times 10^9$ unknowns in 200 s on 12,000 proc.
Outline

1. Introduction
   - History
     - The main characteristics
     - In progress
     - Basement
     - Weak form
1987  MacFem/PCFem the old ones (O. Pironneau in Pascal) no free.

1992  FreeFem rewrite in C++ (P1,P0 one mesh ) O. Pironneau, D. Bernardi, F. Hecht (mesh adaptation , bamg) , C. Prudhomme .

1996  FreeFem+ rewrite in C++ (P1,P0 more mesh) O. Pironneau, D. Bernardi, F. Hecht (algebra of function).

1998  FreeFem++ rewrite with an other finite element kernel and an new language ; F. Hecht, O. Pironneau, K.Ohtsuka.

1999  FreeFem 3d (S. Del Pino) , a fist 3d version base on fictitious domaine method.

2008  FreeFem++ v3 use a new finite element kernel multidimensionnels: 1d,2d,3d...

2014  FreeFem++ v3.34 parallel version

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 544 members with a flux of 1-10 messages per day.

More than 3000 true Users (more than 1000 download / month)
1 Introduction

- History
- The main characteristics
- In progress
- Basement
- Weak form
The main characteristics I/II

- 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 another one (with matrix construction if needed), so a finite element function is viewed 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, PETSc. ... sparse linear solver; **eigenvalue** and eigenvector computation with ARPACK or SLEPc.

- Online graphics with **OpenGL/GLUT/VTK**, C++ like syntax.

- Javascript version works straight out of an **HTML page**, both online or offline (here).
The main characteristics II/II

- 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. Enter `FreeFem++ yours.edp` or `FreeFem++` must be in one 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, textmate, bbedit, fraise ... and click on the icon of the application `FreeFem++` and load your file via the open file dialog box or drag and drop the icon of your built file on the application `FreeFem++` icon.
Outline

1 Introduction

- History
- The main characteristics
- In progress
- Basement
- Weak form
In progress (future proche ou pas)

- Mesh intersection of get conservative formulation in 2d
- Element on curve and Surface
- BEM method
- DG in 3d
- cmake
- rewrite of sparse matrix kernel
- rewrite of the Finite element kernel of isoparametric FE and to mixte surface FE and 3d FE in a problem.( no template).
- debugger (client-server graphics services)
Outline

1 Introduction
   ● History
   ● The main characteristics
   ● In progress
   ● Basement
   ● Weak form
Element of syntax: the script is like in C/C++

First FreeFem++ is a compiler and after it launch the create code (a kind of byte code).
The language is polymorphe but it is not a objet oriented language.

The key words are reserved and the operator are like in C exempt for: ^ & | + - * / ^ // 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.);
string two=2; int i2 = atoi(two);
for (int i=0;i<n;i++) { ... ;}
if ( <bool exp> ) { ... ;} else { ... ;};
while ( <bool exp> ) { ... ;}
break continue key words

Weakness: all local variables are almost static, so no recursion in function
bug if break before variable declaration in same block.
bug for fespace argument or fespace function argument
x, y, z  // current coord.
label, region  // label of BC (border), (interior)
N.x, N.y, N.z,  // normal
int i = 0;  // an integer
real a = 2.5;  // a real
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)
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[].
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.
a formal array is \[ \text{[ exp1, exp1, \ldots, expn]} \]
the Hermitian transposition is \[ \text{[ exp1, exp1, \ldots, expn]}' \]

```cpp
complex a=1,b=2,c=3i;
func va=[ a,b,c];
// is a formal array in [ ]
a =\text{[ 1,2,3i]}'*va ; cout « a « endl; // Hermitian product
matrix<complex> A=va*\text{[ 1,2,3i]}' ; cout « A « endl;
```

```cpp
matrix<complex> A=va*\text{[ 1,2,3i]}' ; cout « A « endl;
a =\text{[ 1,2,3i]}'*va*2.;
a =(va+[ 1,2,3i])'*va*2.;
va./va;
va.*va; // term to term /
va.*va; // term to term *
```

```cpp
trace(\text{va*\text{[ 1,2,3i]}'}) ; //
(va*\text{[ 1,2,3i]}'\text{)[1][2] ; // get coef
det(\text{[[1,2],[\text{-2},1]]}); // just for matrix 1x1 et 2x2
```

```
usefull macro to def your edp.
macro grad(u) [dx(u),dy(u)] //
macro div(u1,u2) (dx(u1)+dy(u2)) //
```
### List of Plugin

```bash
ls /usr/local/lib/ff++/3.50/lib/
```

<table>
<thead>
<tr>
<th>Plugin</th>
<th>Plugin</th>
<th>Plugin</th>
<th>Plugin</th>
<th>Plugin</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEC.dylib</td>
<td>MUMPS_seq.dylib</td>
<td>ff-cmaes.dylib</td>
<td>mmg3d-v4.0.dylib</td>
<td></td>
</tr>
<tr>
<td>BernardiRaugel.dylib</td>
<td>MetricKuate.dylib</td>
<td>fflapack.dylib</td>
<td>msh3.dylib</td>
<td></td>
</tr>
<tr>
<td>BinaryIO.dylib</td>
<td>MetricPk.dylib</td>
<td>ffnewuoa.dylib</td>
<td>mshmet.dylib</td>
<td></td>
</tr>
<tr>
<td>ClosePoints.dylib</td>
<td>Morley.dylib</td>
<td>ffrandom.dylib</td>
<td>myfunction.dylib</td>
<td></td>
</tr>
<tr>
<td>Curvature.dylib</td>
<td>NewSolver.dylib</td>
<td>freeyams.dylib</td>
<td>myfunction2.dylib</td>
<td></td>
</tr>
<tr>
<td>DxWriter.dylib</td>
<td>PARDISO.dylib</td>
<td>funcTemplate.dylib</td>
<td>pcm2rnm.dylib</td>
<td></td>
</tr>
<tr>
<td>Element_HCT.dylib</td>
<td>SuperLu.dylib</td>
<td>gmsh.dylib</td>
<td>pipe.dylib</td>
<td></td>
</tr>
<tr>
<td>Element_Mixte.dylib</td>
<td>UMFPACK64.dylib</td>
<td>gsl.dylib</td>
<td>ppm2rnm.dylib</td>
<td></td>
</tr>
<tr>
<td>Element_Mixte3d.dylib</td>
<td>VTK_writer.dylib</td>
<td>ilut.dylib</td>
<td>qf11to25.dylib</td>
<td></td>
</tr>
<tr>
<td>Element_P1bl.dylib</td>
<td>VTK_writer_3d.dylib</td>
<td>iohdf5.dylib</td>
<td>scotch.dylib</td>
<td></td>
</tr>
<tr>
<td>Element_P1dc1.dylib</td>
<td>addNewType.dylib</td>
<td>iovtk.dylib</td>
<td>shell.dylib</td>
<td></td>
</tr>
<tr>
<td>Element_P3.dylib</td>
<td>aniso.dylib</td>
<td>isoline.dylib</td>
<td>splitedges.dylib</td>
<td></td>
</tr>
<tr>
<td>Element_P3dc.dylib</td>
<td>bfstream.dylib</td>
<td>isolineP1.dylib</td>
<td>splitmesh3.dylib</td>
<td></td>
</tr>
<tr>
<td>Element_P4.dylib</td>
<td>dfft.dylib</td>
<td>lapack.dylib</td>
<td>splitmesh6.dylib</td>
<td></td>
</tr>
<tr>
<td>Element_P4dc.dylib</td>
<td>distance.dylib</td>
<td>lgbmo.dylib</td>
<td>symmetrizeCSR.dylib</td>
<td></td>
</tr>
<tr>
<td>Element_PkEdge.dylib</td>
<td>exactpartition.dylib</td>
<td>mat_dervieux.dylib</td>
<td>tetgen.dylib</td>
<td></td>
</tr>
<tr>
<td>Element_QF.dylib</td>
<td>ff-AiryBiry.dylib</td>
<td>mat_psi.dylib</td>
<td>thresholdings.dylib</td>
<td></td>
</tr>
<tr>
<td>FreeFemQA.dylib</td>
<td>ff-Ipopt.dylib</td>
<td>medit.dylib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUMPS.dylib</td>
<td>ff-NLopt.dylib</td>
<td>metis.dylib</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Las Palmas de Gran Canaria, 25-29 June 2018

F. Hecht et al.

Cours FreeFem++, XVIII Spanish-French school 20 / 224
Important Plugin

- qf11to25 add more quadrature formulae in 1d, 2d, 3d and tools to build own quadrature
- Element_*, Morlay, BernardiRaugel add new kind of 2d finite element
- UMFPACK64, SuperLu, MUMPS_seq add sequential sparse solver
-metis, scotch mesh Partitioning
-ffrandom true random number generator: srandomdev, srandom, random
-gsl the gsl lib interface (lot of special function)
-shell, pipe directory and file interface, pipe interface
-dfft interface with fftw3 library for FFT.
-msh3, tetgen 3d mesh tools and tetgen interface
-lapack a small interface with lapack library of full linear solver, and full eigen value problem.
-ff-Ipopt interface with Ipopt optimisation software
-ppm2rnm interface with ppm library to read ppm bitmap.
-isoline build a border from isoline.
-distance build a the signed distance approximation to an isoline in 2d and 3d.
Important Plugin with MPI

- **HPDDM** a new parallel linear solver see diffusion-2d.edp example in examples++-hpddm
- **PETSc** a new version of PETSc real interface
- **SLEPc** a new version of SLEPc real interface (include PETSc)
- **PETSc-complex** a new version of complex PETSc interface
- **SLEPc-complex** a new version of complex SLEPc interface (include PETSc-complex)
- **MUMPS** a new version of MUMPS interface
- **MPICG** parallel version of CG, and GMRES
- **mpi-cmaes** parallel version of stochastic optimization algorithm.
- **hips_FreeFem, parms_FreeFem, MUMPS_FreeFem** old parallel linear solver interface.
1 Introduction

- History
- The main characteristics
- In progress
- Basement
- Weak form
Laplace equation, weak form

Let a domain $\Omega$ with a partition of $\partial \Omega$ in $\Gamma_2, \Gamma_e$. Find $u$ a solution in such that:

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

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

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

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

Vh u,v; // set test and unknown 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 $\gamma_2$
plot(u,fill=1,wait=1,value=0,wait=1);

Run:fish.edp    Run:fish3d.edp
```
Tools

- Important Remark
  - Remarks on weak form and boundary conditions
  - Mesh generation
  - Build mesh from image
  - 3d mesh
  - Mesh tools
  - Anisotropic Mesh adaptation
  - 3d adaptation process
Important remark: on geometrical item label and region

- All boundary (internal or nor) was define through a label number and this number is define in the mesh data structure. The support of this label number is a edge in 2d and a face in 3d and **so FreeFem++ never use label on vertices**.
- To defined integration you can use the region (resp. label) number if you compute integrate in domain (resp. boundary). They are no way to compute 1d integral in 3d domain.
- Today they are no Finite Element defined on surface.
- You can put list of label or region in integer array (int[int]).
Outline

2 Tools

- Important Remark
- Remarks on weak form and boundary conditions
- Mesh generation
- Build mesh from image
- 3d mesh
- Mesh tools
- Anisotropic Mesh adaptation
- 3d adaptation process
Remark on varf

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

\begin{verbatim}
varf vb1([u1,u2],[q]) = int2d(Th)( (dy(u1)+dy(u2)) *q)
    +int2d(Th)(1*q) + on(1,u1=2);
varf vb2([v1,v2],[p]) = int2d(Th)( (dy(v1)+dy(v2)) *p)
    +int2d(Th)(1*p) ;
\end{verbatim}

To build matrix \texttt{A} from the bilinear part the the variational form \texttt{a} of type \texttt{varf} do simply

\begin{verbatim}
matrix B1 = vb1(Vh,Wh [, ...] );
matrix<complex> C1 = vb1(Vh,Wh [, ...] );
\end{verbatim}

\begin{verbatim}
// 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.
\end{verbatim}

To build a vector, put \( u_1 = u_2 = 0 \) by setting 0 of on unknown part.

\begin{verbatim}
real[int] b = vb2(0,Wh);
complex[int] c = vb2(0,Wh);
\end{verbatim}

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 B.C.): \( \text{on}(1, \ u = g) \)
  
  The meaning is for all degree of freedom \( i \) (DoF) 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 \( (\Pi_h g)[i] \) is the boundary DoF value given by the interpolation of \( g \).

- An "on" vectorial form: \( \text{on}(1, u_1=g_1, u_2=g_2) \)
  
  If you have vectorial finite element like RT0, the 2 components are coupled, and so you have: \( b[i] = (\Pi_h (g_1, g_2))[i] \times tgv \), where \( \Pi_h \) is the vectorial finite element interpolant.

- A linear form on \( \Gamma \) or \( \Gamma_3 \) (for Neumann B.C. in 2d)
  
  \(-\int 1d(Th) ( f \star w) \) or \(-\int 1d(Th, 3))( f \star w)\)

- A bilinear form on \( \Gamma \) or \( \Gamma_2 \) (for Robin B.C. in 2d)
  
  \(\int 1d(Th) ( K \star v \star w) \) or \(\int 1d(Th, 2))( K \star v \star w)\).

- A linear form on \( \Gamma \) or \( \Gamma_3 \) (for Neumann B.C. in 3d)
  
  \(-\int 2d(Th) ( f \star w) \) or \(-\int 2d(Th, 3))( f \star w)\).
Tools

- Important Remark
- Remarks on weak form and boundary conditions
- Mesh generation
  - Build mesh from image
  - 3d mesh
  - Mesh tools
  - Anisotropic Mesh adaptation
  - 3d adaptation process
First a $10 \times 10$ grid mesh of unit square $]0, 1[^2$

```cpp
int[int] labs=[10, 20, 30, 40]; // bot., right, top, left
mesh Th1 = square(10, 10, label=labs, region=0, [x, y]); //
plot(Th1, wait=1);
int[int] old2newlabs=[10, 11, 30, 31]; // 10 -> 11, 30 -> 31
Th1=change(Th1, label=old2newlabs) ; //
// do Change in 2d or in 3d. region=a, fregion=f ,
// flabel=f

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

```cpp
mesh Th = trunc(Th1, (x<0.5) | (y < 0.5), label=1); //
plot(Th, cmm="Th");
mesh Thh = movemesh(Th, [-x, y]);
mesh Th3 = Th+movemesh(Th, [-x, y]); // glumesh ...
plot(Th3, cmm="Th3");
```

Run:mesh1.edp
a Circle with or without an hole:

**Remark:** by default the domain is a left of the border if the number of segments $> 0$.

```latex
border Co(t=0,2*pi) { x=cos(t); y=sin(t); label=1;}
border Ci(t=0,2*pi) { x=cos(t)/2; y=sin(t)/2; label=2;}
plot(Co(30)+Ci(15), wait=1);
mesh Thf=buildmesh(Co(30)+Ci(15)); // without hole
// two region:
cout «" The two Region of Thf : " « Thf(0,0).region« " " « Thf(0,0.9).region « endl;
plot(Thf, wait=1, cmm="Thf");
mesh Thh=buildmesh(Co(30)+Ci(-15)); // without hole
plot(Thh, wait=1, cmm="Thh");

Get a extern mesh
mesh Th2("april-fish.msh");
build with emc2, bamg, modulef, etc...
Run:mesh-circles.edp
```
a L shape domain $]0,1[^2 \setminus [\frac{1}{2},1[^2$ with 6 multi-borders.

```c
int nn=30; real dd=0.5;
real[int,int] XX=[[0,0],[1,0],[1,dd],[dd,dd],[dd,1],[0,1]];
int[int] NN=[nn,nn*dd,nn*(1-dd),nn*(1-dd),nn*dd,nn];
border bb(t=0,1;i)
{
   // i is the index of the multi border loop
   int ii = (i+1)%XX.n; real t1 = 1-t;
   x = XX(i,0)*t1 + XX(ii,0)*t;
   y = XX(i,1)*t1 + XX(ii,1)*t;
   label = 1; ;
}
plot(bb(NN),wait=1);
mesh Th=buildmesh(bb(NN));
plot(Th,wait=1);
Run:mesh-multi.edp```

Outline

2 Tools

- Important Remark
- Remarks on weak form and boundary conditions
- Mesh generation
- Build mesh from image
- 3d mesh
- Mesh tools
- Anisotropic Mesh adaptation
- 3d adaptation process
Build mesh from image 1/2

load "ppm2rnm" load "isoline" load "shell"

string lac="Gran-canaria", lacjpg =lac+".jpg", lacpgm =lac+".pgm";
if(stat(lacpgm)<0) exec("convert -resample 50x50"+lacjpg+"+lacpgm);
real[int,int] Curves(3,1); int[int] be(1); int nc;
{
real[int,int] ff1(lacpgm); // read image
int nx = ff1.n, ny=ff1.m; // grey value in 0 to 1 (dark)
mesh Th=square(nx-1,ny-1,[(nx-1)*(x),(ny-1)*(1-y)]);
fespace Vh(Th,P1); Vh f1; f1[]=ff1; // array to fe function.
real iso =0.08; // try some value to get correct iso
real[int] viso=[iso];
nc=isoline(Th,f1,iso=iso,close=0,Curves,beginend=be,
    smoothing=.1,ratio=0.5);
for(int i=0; i<min(3,nc);++i)
{ int i1=be(2*i),i2=be(2*i+1)-1;
  plot(f1,viso=viso,[Curves(0,i1:i2),Curves(1,i1:i2)],
      wait=1,cmm=i); }}
int[int] iii=[0];  // chose to component ...
int[int] NC=[-500];  // 1 component

border G(t=0,1;i) {
P=Curve(Curves,be(2*iii[i]),be(2*iii[i]+1)-1,t);
    label= iii[i];}

plot(G(NC),wait=1);

mesh Th=buildmesh(G(NC));
plot(Th,wait=1);

real scale = sqrt(AreaIsland/Th.area);
Th=movemesh(Th,[x*scale,y*scale]);

Run:Gran-Canaria.edp  (the Gran Canaria island)
2 Tools

- Important Remark
- Remarks on weak form and boundary conditions
- Mesh generation
- Build mesh from image
- 3d mesh
  - Mesh tools
  - Anisotropic Mesh adaptation
  - 3d adaptation process
A cube with cube

```cpp
load "msh3"                     // buildlayer
mesh3 Th;
int nn=10;
if(1) { int[int] ll=[1,1,1,1,1,2];
    // y=0:front, x=1:right, y=1:back, x=0:left, z=0:down, z=1:up
    Th=cube(nn,nn,nn,label=ll); }
    // Warning bug if no parameter label=ll before version 3.56-1

Th= trunc(Th, ((x<0.5) | (y< 0.5) | (z<0.5)), label=3);
    // remove 1/2 cube
plot("cube",Th);
Run:Cube.edp
```
load "msh3" // buildlayer
load "medit" // medit
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],
    labelmid=rmid, labelup = rup,
    labeldown = rdown); // label def
medit("lac",Th);
Run:Lac.edp Run:3d-leman.edp
func f=2*((.1+(((x/3))*(x-1)*(x-1)/1+x/100))^(1/3.)-(.1)^(1/3.));
real yf=f(1.2,0);
border up(t=1.2,0.) { x=t; y=f; label=0; }
border axe2(t=0.2,1.15) { x=t; y=0; label=0; }
border hole(t=pi,0) { x= 0.15 + 0.05*cos(t); y= 0.05*sin(t);
label=1; }
border axel(t=0,0.1) { x=t; y=0; label=0; }
border queue(t=0,1) { x= 1.15 + 0.05*t; y = yf*t; label =0; }
int np= 100;
func bord= up(np)+axel(np/10)+hole(np/10)+axe2(8*np/10)
+ queue(np/10);
plot( bord); // plot the border ...

mesh Th2=buildmesh(bord);
plot(Th2,wait=1); // the 2d mesh axi mesh

int[int] 123=[0,0,1,1];
Th=buildlayers(Th2,coef= max(.15,y/max(f,0.05)), 50
 ,zbound=[0,2*pi],transfo=[x,y*cos(z),y*sin(z)]
 ,facemerge=1,labelmid=123);

Run:3daximesh.edp
boundary mesh of a Sphere

load "tetgen"
mesh Th=square(10,20,[x*pi-pi/2,2*y*pi]);  //  ]-\pi/2, -\pi/2[ \times ]0, 2\pi[
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 = D F^t D F
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/R; real vv= 1/square(hh);
Th=adaptmesh(Th,m11*vv,m21*vv,m22*vv,IsMetric=1,periodic=perio);
int[int] ref=[0,L];  //  the label of the Sphere to L ( 0 -> L)
mesh3  ThS= movemesh23(Th,transfo=[f1*R,f2*R,f3*R],orientation=1,
label=ref);
Run:Sphere.edp Run:sphere6.edp
include "MeshSurface.idp" // tool for 3d surfaces meshes

// tool for 3d surfaces meshes

mesh3 Th;
try { Th=readmesh3("Th-hex-sph.mesh"); }  // try to read

catch(...) {                               // catch a reading error so build the mesh...
    real hs = 0.2;                              // mesh size on sphere
    int[int] NN=[11,9,10];
    real [int,int] BB=[[-1.1,1.1],[-.9,.9],[-1,1]]; // Mesh Box
    int [int,int] LL=[[1,2],[3,4],[5,6]];          // Label Box

    mesh3 ThHS = SurfaceHex(NN,BB,LL,1)+Sphere(0.5,hs,7,1);  // surface meshes
    real voltet=(hs^3)/6.;                           // volume mesh control.
    real[int] domaine = [0,0,0,1,voltet,0,0,0.7,2,voltet];

    Th = tetg(ThHS,switch="pqaAAYYQ",
               nbofregions=2,regionlist=domaine);

    savemesh(Th,"Th-hex-sph.mesh"); }  // save for next run
load "msh3" load "medit" load "gsl" load "tetgen"
....

mesh3 Ths;  // the surface
{ mesh3 Thc, Thbottom, ThcTop;
    { real[int,int] srneck =
        [ [Htube-0.001, Htube, Htube+Hneck*0.1, Htube+Hneck*0.3 , Htube+Hneck*0.7 ,
            Htube+Hneck*0.9, Htube+Hneck+0.1],
        [Rext , Rext, Rext , Rext*.7+Rneck*.3 , Rext*.1 + Rneck*.9,
            Rneck , Rneck ]];
    gslspline rneck(gslinterpcspline,srneck);  // Curve of neck of the bottle
}

mesh Thc = square(Hbot/hh, 2*pi*Rext/hh, [x*Hbot, y*2*pi]);
fespace V2x(Thc,P1);
func E1 = rneck(x)*cos(y); func E2 = rneck(x)*sin(y); func E3 = x;
...

Thc=adaptmesh(Thc,em11,em21,em22,IsMetric=1,periodic=perio,nbv=100000); }
Thc=change(Thc,fregion=labcyl);
Thc = movemesh23(Thc,transfo=[E1 , E2 , E3]);  // maillage exterior
// extraction of the border of the bottle

int[int] databoder(1); int ncb= getborder(Th3c,databoder);
int ktop= Th3c(databoder[databoder[1]]).z < Th3c(databoder[databoder[0]]).z;
int kbot=1-ktop; // other border

macro DefBorder(bname,kk,Th3,bb,ll)
    int n#bname= bb[kk+1]-bb[kk]; border bname(t=bb[kk], bb[kk+1])
    { real iv = int(t); if( iv == bb[kk+1]) iv = bb[kk];
     iv = bb[iv];x= Th3(iv).x ; y= Th3(iv).y ;label = ll; } // EOM

DefBorder(btop,ktop,Th3c,databoder,1) DefBorder(bbot,kbot,Th3c,databoder,1)

Th3bottom=movemesh23(change(buildmesh(bbot(nbbot),fixeborder=1),fregion=labbottom)
    ,transfo=[x,y,Zbot],orientation=-1);
Th3top=movemesh23(change(buildmesh(btop(-nbtop),fixeborder=1),fregion=labtop)
    ,transfo=[x,y,Ztop],orientation=1);
Ths = Th3c + Th3bottom + Th3top; }

real[int] domaine = [0,0,Htube,1,hh^3/6.];
mesh3 Th=tetg(Ths,switch="pqaAYY",regionlist=domaine); medit("Th",Th);
Run:bottle.edp
2 Tools

- Important Remark
- Remarks on weak form and boundary conditions
- Mesh generation
- Build mesh from image
- 3d mesh

Mesh tools
- Anisotropic Mesh adaptation
- 3d adaptation process
Mesh tools

- **change to change label and region numbering in 2d and 3d.**
- **movemesh checkmovemesh movemesh23 movemesh3**
- **triangulate (2d), tetgconvexhull (3d) build mesh mesh for a set of point**
- **emptymesh (2d) built a empty mesh for Lagrange multiplier**
- **freeyams to optimize surface mesh**
- **mmg3d to optimize volume mesh with constant surface mesh in version 4**
- **mshmet to compute metric**
- **isoline to extract isoline (2d)**
- **trunc to remove peace of mesh and split all element (2d,3d)**
- **splitmesh to split 2d mesh in no regular way.**
Outline

2 Tools
- Important Remark
- Remarks on weak form and boundary conditions
- Mesh generation
- Build mesh from image
- 3d mesh
- Mesh tools
  - Anisotropic Mesh adaptation
  - 3d adaptation process
In Euclidean geometry the length $|\gamma|$ of a curve $\gamma$ 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 $M(x)$ as a field of $d \times d$ symmetric positive definite matrices, and the length $\ell$ of $\Gamma$ w.r.t $M$ is:

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

The key-idea is to construct a mesh where the lengths of the edges are close to 1 accordingly to $M$. 
Metric intersection

The unit ball $\mathcal{B}M$ in a constant 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

\[ u = (10x^3 + y^3) + \tanh(500(sin(5y) - 2x)) \];

\[ v = (10y^3 + x^3) + \tanh(5000(sin(5y) - 2*)) \];

Run: Adapt-uv.edp
The domain is an L-shaped polygon \( \Omega = ]0, 1[^2 \setminus [\frac{1}{2}, 1]^2 \) and the PDE is

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

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

example of Mesh adaptation
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.));
}

Run:CornerLap.edp
2 Tools

- Important Remark
- Remarks on weak form and boundary conditions
- Mesh generation
- Build mesh from image
- 3d mesh
- Mesh tools
- Anisotropic Mesh adaptation
- 3d adaptation process
Example of adaptation process in 3d with mmg3

Let a domain $\Omega = ]0, 1[^3 \setminus [\frac{1}{2}, 1[^3$ The border of $\partial \Omega$ is split in 2 part

- $\Gamma_2$, if $x = 1$, $y = 1$, or $z = 1$
- $\Gamma_1$, else.

Find $u$ a solution in such that:

\[ -\Delta u = 1 \quad \text{in } \Omega, \]
\[ \frac{\partial u}{\partial \vec{n}} = 0 \quad \text{on } \Gamma_2, \]
\[ u = 0 \quad \text{on } \Gamma_1. \]

Thank to mmg v5 tools to do 3d mesh adaptation see http://www.mmgtools.org.

without mmg (isotrope): Run:Laplace-Adapt-3d.edp

with mmg (anisotrope): Run:Laplace-Adapt-aniso-3d.edp
Build of the metric form the solution $u$

Optimal metric norm for interpolation error (function `adaptmesh` in freefem++) for $P_1$ continuous Lagrange finite element

- $L^\infty : \mathcal{M} = \frac{1}{\varepsilon} |\nabla \nabla u| = \frac{1}{\varepsilon} |\mathcal{H}|$ where $\mathcal{H} = \nabla \nabla u$

- $L^p : \mathcal{M} = \frac{1}{\varepsilon} |det(\mathcal{H})|^{\frac{1}{2p+2}} |\mathcal{H}|$ (result of F. Alauzet, A. Dervieux)

In Norm $W^{1,p}$, the optimal metric $\mathcal{M}_\ell$ for the $P_\ell$ Lagrange finite element, Optimal is given by (with only acute triangle) (thanks to J-M. Mirebeau)

$$\mathcal{M}_{\ell,p} = \frac{1}{\varepsilon} (det\mathcal{M}_{\ell})^{\frac{1}{2p+2}} \mathcal{M}_{\ell}$$

and (see `MetricPk` plugin and function )

- for $P_1$: $\mathcal{M}_1 = \mathcal{H}^2$ (for sub optimal case with acute triangle take $\mathcal{H}$)

- for $P_2$: $\mathcal{M}_2 = 3 \sqrt{\left(\begin{array}{cc} a & b \\ b & c \end{array}\right)^2 + \left(\begin{array}{cc} b & c \\ c & a \end{array}\right)^2}$ with

$$D^{(3)}u(x,y) = (ax^3 + 3bx^2y + 3cxy^2 + dy^3)/3!,$$

Run: `adapt.edp` Run: `AdaptP3.edp`
3 Academic Examples

- Laplace/Poisson
  - Linear elasticity equation
  - Stokes equation
  - Optimize Time depend schema
  - Bose Einstein Condensate, result analyse
We find $p$ in $H^1$, such that:

$$-\Delta p = f \text{ in } \Omega, \quad \partial_n p = g_n \text{ on } \Gamma.$$ 

This problem is defined through a constant and mathematically the problem is well pose iff $\int_{\Omega} f + \int_{\Gamma} g_n = 0$ and $p$ is in $H^1/\mathbb{R}$. So we can make a small regularization to remove the problem of constant by find $p_\varepsilon \in H^1$ such that

$$\varepsilon p_\varepsilon - \Delta p_\varepsilon = f \text{ in } \Omega, \quad \partial_n p_\varepsilon = g_n \text{ on } \Gamma$$

and the last problem is trivial to be approximate in FreeFem++:

```
Vh p,q; real eps=1e-8; // warning eps
   //     must be small but no too small.
solve Laplace(p,q)
   = int2d(Th)( p*q*eps + grad(p)*grad(q))
   - int2d(Th) ( f*q) - int1d(Th) (gn*q) ;
```

Remark: it is hard to put Dirichlet boundary condition on only one point so set the constant due to label definition.
Now we solve \(-\Delta p = f\) in \(\Omega\), \(p = g_d\) on \(\Gamma_d\), \(\partial_n p = g_n\) on \(\Gamma_n\).

\(\Gamma_d, \Gamma_n\) is a partition of \(\partial \Omega\).

with \(\vec{u} = \nabla p\) the problem becomes:

Find \(\vec{u}, p\) such that:

\[-\nabla \cdot \vec{u} = f, \quad \vec{u} - \nabla p = 0 \quad \text{in} \quad \Omega, \quad p = g_d \quad \text{on} \quad \Gamma_d, \quad \partial_n p = g_n \quad \text{on} \quad \Gamma_n \quad (3)\]

Mixte variational formulation is: find \(\vec{u} \in H_{\text{div}}(\Omega), \ p \in L^2(\Omega), \ \vec{u}.n = g_n \) on \(\Gamma_n\) such that

\[
\int_{\Omega} q \nabla \cdot \vec{u} + \int_{\Omega} p \nabla \cdot \vec{v} + \vec{u} \cdot \vec{v} = \int_{\Omega} -fq + \int_{\Gamma_d} g_d \vec{v} \cdot \vec{n}, \quad \forall (\vec{v}, q) \in H_{\text{div}} \times L^2, \text{and} \quad \vec{v}.n = 0 \quad \text{on} \quad \Gamma_n
\]
Laplace equation (mixte formulation) II/III

```
mesh Th=square(10,10); fespace Vh(Th,RT0), Ph(Th,P0);
func gd = 1.; func gn = 1.; func f = 1.;
Vh [u1,u2],[v1,v2];
Ph p,q;
solve laplaceMixte([u1,u2,p],[v1,v2,q],solver=UMFPACK)
    = int2d(Th)( p*q*0e-10 + u1*v1 + u2*v2
                 + p*(dx(v1)+dy(v2)) + (dx(u1)+dy(u2))*q )
    + int2d(Th) ( f*q)
    - int1d(Th,1,2,3)( gd*(v1*N.x +v2*N.y)) // int on \Gamma_d
    + on(4,u1=gn*N.x,u2=gn*N.y); // mean u.n = (gn)n.n
```

Run:LaplaceRT.edp
solve \(-\Delta u = f\) on \(\Omega\) and \(u = g\) on \(\Gamma\)

macro \(\text{dn}(u)\) (N.x*dx(u)+N.y*dy(u)) // def the normal derivative

mesh Th = square(10,10); // unite square
fespace Vh(Th,P2dc); // discontinuous P2 finite element
    // if pena = 0 => Vh must be P2 otherwise penalization
real pena=0; // to add penalization
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

Run:LapDG2.edp
A mathematical Poisson Problem with full Neumann BC. with 1D lagrange multiplier

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

\[
\forall (v, \mu) \in V_h \times \mathbb{R} \quad a(u, v) + b(u, \mu) + b(v, \lambda) = l(v), \quad \text{where } b(u, \mu) = \mu \int_{\Omega} u
\]

```
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];
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
```

Run:Laplace-lagrange-mult.edp
Outline

3 Academic Examples
- Laplace/Poisson
- Linear elasticity equation
- Stokes equation
- Optimize Time depend schema
- Bose Einstein Condensate, result analyse
Linear Lame equation, weak form

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

$$-\nabla \cdot \sigma(u) = f \text{ in } \Omega, \quad u = 0 \text{ on } \Gamma_d, \quad \sigma(u) \cdot n = 0 \text{ on } \Gamma_n$$  \hspace{1cm} (4)

Where $\varepsilon(u) = \frac{1}{2} (\nabla u + ^t \nabla u)$ and $\sigma(u) = A \varepsilon(u)$ with $A$ the linear positive operator on symmetric $d \times d$ matrix corresponding to the material property. Denote $V_g = \{v \in H^1(\Omega)^d / v |_{\Gamma_d} = g\}$.

The Basic displacement variational formulation is: find $u \in V_0(\Omega)$, such that:

$$\int_{\Omega} \varepsilon(v) : A \varepsilon(u) = \int_{\Omega} v \cdot f + \int_{\Gamma} ((A \varepsilon(u)) \cdot n) \cdot v, \quad \forall v \in V_0(\Omega)$$  \hspace{1cm} (5)
The finite element method is just: replace $V_g$ with a finite element space, and the FreeFem++ code:

```cpp
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;
```
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);
Run:beam-3d.edp  Run:beam-EV-3d.edp  Run:free-cyl-3d.edp  Run:beam-3d-Adapt.edp
3 Academic Examples

- Laplace/Poisson
- Linear elasticity equation
- Stokes equation
- Optimize Time depend schema
- Bose Einstein Condensate, result analyse
The Stokes equation is find a velocity field \( u = (u_1, \ldots, u_d) \) and the pressure \( p \) on domain \( \Omega \) of \( \mathbb{R}^d \), such that

\[
-\Delta u + \nabla p = 0 \quad \text{in} \quad \Omega \\
\nabla \cdot u = 0 \quad \text{in} \quad \Omega \\
u = u_\Gamma \quad \text{on} \quad \Gamma
\]

where \( u_\Gamma \) is a given velocity on boundary \( \Gamma \).

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

\[
\forall v \in H^1_0(\Omega)^d, \forall q \in L^2(\Omega)/\mathbb{R}, \quad \int_\Omega \nabla u : \nabla v - p \nabla \cdot v - q \nabla \cdot u = 0
\]

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

\[
\forall v \in H^1_0(\Omega)^d, \forall q \in L^2(\Omega), \int_\Omega \nabla u : \nabla v - p \nabla \cdot v - q \nabla \cdot u - \varepsilon pq = 0
\]
Stokes equation in FreeFem++

... build mesh .... Th (3d) T2d (2d)

```cpp
fespace VVh(Th, [P2,P2,P2,P1]);  // Taylor Hood FE.
macro Grad(u) [dx(u), dy(u), dz(u)]  // EOM
macro div(u1,u2,u3) (dx(u1)+dy(u2)+dz(u3))  // EOM
VVh [u1,u2,u3,p], [v1,v2,v3,q];
solve 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, u1=0, u2=0, u3=0)  + on(2, u1=1, u2=0, u3=0);
```

Run:Stokes-2d.edp   Run:Stokes-bug.edp   Run:Stokes-UzawaCahouetChabart-bug.edp
Run:Stokes-Pipe.edp   Run:Stokes3d.edp
3 Academic Examples
- Laplace/Poisson
- Linear elasticity equation
- Stokes equation
- Optimize Time depend schema
- Bose Einstein Condensate, result analyse
First, it is possible to define variational forms, and use this forms to build matrix and vector to make very fast script (4 times faster here).

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

$$\begin{align*}
\partial_t u - \nabla \cdot (\kappa \nabla u) &= 0 \text{ in } \Omega \times (0, T), \\
u(x, y, 0) &= u_0 + xu_1 \\
u = 30 &\text{ on } \Gamma_24 \times (0, T), \quad \kappa \frac{\partial u}{\partial n} + \alpha (u - u_e) = 0 \text{ on } \Gamma \times (0, T). \tag{6}
\end{align*}$$

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

$$\forall w \in V_0; \quad \int_{\Omega} \frac{u^n - u^{n-1}}{\delta t} w + \kappa \nabla u^n \nabla w + \int_{\Gamma} \alpha (u^n - u_{ue}) w = 0$$

where $V_0 = \{ w \in H^1(\Omega)/w|_{\Gamma_24} = 0 \}$. 
Fast method for Time depend Problem algorithm

So the to code the method with the matrices $A = (A_{ij})$, $M = (M_{ij})$, and the vectors $u^n, b^n, b', b'', b_{cl}$ (notation if $w$ is a vector then $w_i$ is a component of the vector).

$$u^n = A^{-1}b^n, \quad b' = b_0 + Mu^{n-1}, \quad b'' = \frac{1}{\varepsilon} b_{cl}, \quad b_i^n = \begin{cases} b''_i & \text{if } i \in \Gamma_{24} \\ b'_i & \text{else} \end{cases}$$

Where with $\frac{1}{\varepsilon} = \text{tgv} = 10^{30}$:

$$A_{ij} = \begin{cases} \frac{1}{\varepsilon} & \text{if } i \in \Gamma_{24}, \text{ and } j = i \\ \int_{\Omega} w_j w_i / dt + k(\nabla w_j . \nabla w_i) + \int_{\Gamma_{13}} \alpha w_j w_i & \text{else} \end{cases}$$

$$M_{ij} = \begin{cases} \frac{1}{\varepsilon} & \text{if } i \in \Gamma_{24}, \text{ and } j = i \\ \int_{\Omega} w_j w_i / dt & \text{else} \end{cases}$$

$$b_{0,i} = \int_{\Gamma_{13}} \alpha u_{ue} w_i$$

$$b_{cl} = u^0 \text{ the initial data}$$
... 

\( Vh \quad u_0 = f u_0, u = u_0; \)

Create three variational formulation, and build the matrices \( A, M \).

\[
\text{varf } v\text{thermic}(u,v) = \int_{Th} \left( \frac{u \times v}{dt} + k \times (dx(u) \times dx(v) + dy(u) \times dy(v)) \right) + \int_{1d(Th,1,3)} (\alpha \times u \times v) + \text{on}(2,4, u=1);
\]

\[
\text{varf } v\text{thermic0}(u,v) = \int_{1d(Th,1,3)} (\alpha \times u_e \times v);
\]

\[
\text{varf } v\text{Mass}(u,v) = \int_{Th} \left( \frac{u \times v}{dt} \right) + \text{on}(2,4, u=1);
\]

\[
\text{real } tgv = 1e30;
\]

\[
\text{matrix } A = v\text{thermic}(Vh,Vh,tgv=tgv,\text{solver}=\text{CG});
\]

\[
\text{matrix } M = v\text{Mass}(Vh,Vh);
\]
Now, to build the right hand size we need 4 vectors.

```plaintext
real[int] b0 = vthermic0(0,Vh);  // constant part of RHS
real[int] bcn = vthermic(0,Vh);  // tgv on Dirichlet part
// we have for the node i : i ∈ Γ_{24} ⇔ bcn[i] ≠ 0
real[int] bcl = tgv * u0[];     // the Dirichlet B.C. part
```

The Fast algorithm:

```plaintext
for(real t=0; t<T; t+=dt) {
    real[int] b = b0;          // for the RHS
    b += M * u[];             // add the the time dependent part
    u[] = A^{-1} * b;         // Solve linear problem
    plot(u);
}
```

Run: Heat.edp
3 Academic Examples
- Laplace/Poisson
- Linear elasticity equation
- Stokes equation
- Optimize Time depend schema
- Bose Einstein Condensate, result analyse
  - Search all local min
  - Best Fit
  - Delaunay mesh
  - Analyse of a Condensate
Bose Einstein Condensate

With. I. Danaila (Univ. Rouen), G. Vergez (Phd Becasim), P-E Emeriau (Stage ENS)

Just a direct use of Ipopt interface (2 day of works to start script see, + n month )

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

$$
\begin{align*}
    u &= \text{argmin}_{\|u\|=1} \int_{\mathcal{D}} \frac{1}{2} |\nabla u|^2 + V_{\text{trap}} |u|^2 + \frac{g}{2} |u|^4 - \Omega \bar{u}((\frac{-y}{x}), \nabla) u
\end{align*}
$$

to code that in FreeFem++

use

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

The idea to mixte Ipopt and adapt mesh is play this stop criterion, and finally use freefem++ to analyse the result.

Run:BEC.edp
Search all local min

The function `findalllocalmin` find all the local min and use a greedy algorithm to to the local attraction zone, by adding the triangle through the minimal vertices.

```plaintext
mesh Th=square(50,50,[x*2-1,y*2-1]);
load "isoline"
fespace Vh(Th,P1), Ph(Th,P0);
int k =2;
Vh u= sin(k*pi*x)*sin(k*pi*y);
plot(u, wait=1);
Ph r;
int[int] lm=findalllocalmin(Th,u[],r[]);
// lm array gives the vertex number of all the local min
// r is function P0 defined the attraction zone of the local min
// (local min number)
plot(r,u,fill=1,wait=1);
// to see where is the minimuns
Ph mx= Th(lm[real(r)]).x -x, my= Th(lm[real(r)]).y -y;
plot([mx,my],u,wait=1,fill=0);
```

Run:findalllocalmin.edp  Run:findalllocalminbec.edp
(With P-E Emeriau) Just use ipopt to find the arg min of $J(\alpha) = \int_\Omega (u - \phi_\alpha)^2$ where $\alpha$ is the set of parameters.

```plaintext
real[int] data0 = [ux, x0, y0, s0]; // start point

func real J(real[int] & dat){
  alpha=dat;
  return int2d(Th)(square(u-phialpha));
}

func real[int] dJ(real[int] & dat){
  alpha=dat;
  dat[0]=int2d(Th)(-2*(u-phialpha)*d0phialpha);
  dat[1]=int2d(Th)(-2*(u-phialpha)*d1phialpha);
  dat[2]=int2d(Th)(-2*(u-phialpha)*d2phialpha);
  dat[3]=int2d(Th)(-2*(u-phialpha)*d3phialpha);
  return dat;
}

real[int] data=data0;
verbosity=0;
int r = IPOPT(J,dJ,data,printlevel=0);
```

Run: fit-ipopt.edp
Best Fit axisymmetric

On the domain with no vortex, all just do the L2 projection on axisymmetric space with a laplace regularisation

```plaintext
Ph pok=data7(6,real(r)); // domain with no hole
func r = sqrt(x*x+y*y);
Vh pr = r;
real R = pr[].max;
mesh Th1d=square(200,1,[x*R,y]);
fespace Vh1d(Th1d,P1,periodic=[[1,x],[3,x]]); // dat axi
mesh Th2d = trunc(Thg,pok==1); // mesh
Vh u20 = u1dt(r,0);
// Axi -> 2d
plot(u20,u2,wait=1,dim=3);
```
Delaunay mesh

```plaintext
mesh Thc=triangulate(data7(0,:),data7(1,:));

fespace Eh(Thc,P0edge);// Element P0 / Edge
varf vedge(u,v) = intalledges(Thc,qforder=1)((nTonEdge==2)*v/
    nTonEdge);
real[int] eih=vedge(0,Eh);
int ei=0;
for(int e=0; e < eih.n; ++e) if(eih[e]) eih[ei++]=eih[e];
eih.resize(ei);
real moy = eih.sum/ eih.n ;
// Statistic
real[int] dd = eih;dd-= moy;
real variance = dd.l2 / dd.n;
cout << "\_moy\_eih\_=" << moy<< "\_standart\_deviation\_" << sqrt(
    variance) <<endl;
for(int i=1 ; i<10; ++i)
cout <<"\_quantile\_"<<i/10.<< "\_=" << eih.quantile(i/10.) <<endl;

Run:analyssolbec.edp
```
Outline

4 Numerics Tools
- Connectivity
- Input/Output
- Tricks
- Eigenvalue
- Optimization Tools
mesh Th=square(5,5);

fespace Wh(Th,P2);

cout << " nb of DoF : " << Wh.ndof << endl;
cout << " nb of DoF / K : " << Wh.ndofK << endl;

int k= 2, kdf= Wh.ndofK ;

// element 2

cout << " df of element " << k << ":" ;
for (int i=0;i<kdf;i++) cout << Wh(k,i) << " ";
cout << endl;

Remark on local numbering of Dof by element is
for each sub finite element $P_k$ in $[P_2,P_2,P_1]$ get fist DoF on vertex, second DoF on
edge (opposite to vertex), second on K.

Run:Mesh-info.edp
4 Numerics Tools

- Connectivity
- Input/Output
- Tricks
- Eigenvalue
- Optimization Tools
Save/Restore

uses `cout`, `cin`, `endl`, «, ».
To write to (resp. read from) a file,
declare a new variable `ofstream ofile("filename");`
or
`ofstream ofile("filename", append);` (resp. `ifstream ifile("filename");`)
or
`ofstream ofile("filename", append|binary);` (resp. `ifstream ifile("filename", binary);`)
and use `ofile` (resp. `ifile`) as `cout` (resp. `cin`).

You can use pipe to transfer data to another code here (gnuplot), see `pipe.edp` example:

You can use the plugin `bfstream tp` to make binary io (see `Run: examples++-load/bfstream.edp`)

Run:`pipe.edp` Run:`io.edp`
Outline

4 Numerics Tools
  • Connectivity
  • Input/Output
  • Tricks
  • Eigenvalue
  • Optimization Tools
What is simple to do with freefem++:

- Evaluate variational form with Boundary condition or not.
- Do interpolation
- Do linear algebra
- Solve sparse problem.
? Question, Find the list Degree of Freedom (DoF) of border $k$ for coupling problem.

Idea Take a function increasing negative function $\xi - C_{te}$ on the border, and do a simultaneous sort the to array and DoF numbering, remark we use a PDE on border to build this kind of function

$$\nabla \xi . N^\perp = 1 \text{ on } \Gamma_b$$

or use the macro

`ExtractDofsonBorder(labs, Wh, doflabs, orient)` defined in "ExtractDofsonBorder.idp",

Run: `ListOfDofOnBorder.edp` and see: `ExtractDofsonBorder.idp`.

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

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

\[
J(u) = \int_\Omega F(u); \quad \text{macro } F(u) = \sqrt{1 + \nabla u \cdot \nabla u}
\]

\[
dJ(u)(v) = \int_\Omega dF(u,v); \quad \text{macro } dF(u,v) = \frac{\nabla u \cdot \nabla v}{\sqrt{1 + \nabla u \cdot \nabla u}}
\]

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

\[
\text{macro } ddF(u,v,w) = \frac{\nabla w \cdot \nabla v}{\sqrt{1 + \nabla u \cdot \nabla u}} - \frac{(\nabla u \cdot \nabla v)(\nabla w \cdot \nabla v)}{\sqrt{1 + \nabla u \cdot \nabla u}^3}
\]
Outline

4 Numerics Tools
- Connectivity
- Input/Output
- Tricks
- Eigenvalue
- Optimization Tools
The problem, Find the first $\lambda, u_\lambda$ 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 $1e30 = tg v$ 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

$$\frac{1}{\lambda} v = A^{-1} B v$$

Otherwise we get spurious mode.

Arpack interface:

```cpp
int k=EigenValue(A, B, sym=true, value=ev, vector=eV);
```
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);
Run:Lap3dEigenValue.edp Run:LapEigenValue.edp
Outline

4 Numerics Tools
- Connectivity
- Input/Output
- Tricks
- Eigenvalue
- Optimization Tools
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:

\[
\text{find} \quad x_0 = \text{argmin}_{x \in \mathbb{R}^n} f(x) \\
\text{s.t.} \quad \left\{ \begin{array}{l}
\forall i \leq n, \ x_i^{\text{lb}} \leq x_i \leq x_i^{\text{ub}} \quad \text{(simple bounds)} \\
\forall i \leq m, \ c_i^{\text{lb}} \leq c_i(x) \leq c_i^{\text{ub}} \quad \text{(constraints functions)}
\end{array} \right.
\]

Where $\text{ub}$ and $\text{lb}$ 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 $c_i^{\text{lb}} < c_i^{\text{ub}}$. 

func real J(real[int] &X) {...}  // Fitness Function,
func real[int] gradJ(real[int] &X) {...}  // Gradient

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) {
    ...
    return jacCBuffer;  // fill jacCBuffer
}

The hessian returning function is somewhat different because it has to be the hessian of the lagrangian function:

\[(x, \sigma_f, \lambda) \mapsto \sigma_f \nabla^2 f(x) + \sum_{i=1}^m \lambda_i \nabla^2 c_i(x) \text{ where } \lambda \in \mathbb{R}^m \text{ and } \sigma \in \mathbb{R}. \]

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) {...}
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
...
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);

real[int] b = vP(0,Vh);

int[int] II1=[0],II2=[1];

matrix C1 = interpolate (Wh,Vh, U2Vc=II1);

matrix C2 = interpolate (Wh,Vh, U2Vc=II2);

matrix CC = -1*C1 + C2;

// Constraints matrix

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 ;

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=uzzi[],lz=lzi[],lm=lmi

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);

Run: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: \( \text{cmaes}(J,u[],\ldots)() \)

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);

Run: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++. 
```c
int[3] proc1 = [1, 2, 3], proc2 = [0, 4];
mpiGroup grp(procs); // set MPI_Group to proc 1, 2, 3 in MPI_COMM_WORLD
mpiGroup grp1(comm, proc1); // set MPI_Group to proc 1, 2, 3 in comm
mpiGroup grp2(grp, proc2); // set MPI_Group to grp union proc1

mpiComm comm = mpiCommWorld; // set a MPI_Comm to MPI_COMM_WORLD
mpiComm ncomm(mpiCommWorld, grp); // set the MPI_Comm form grp // MPI_COMM_WORLD
mpiComm ncomm(comm, color, key); // MPI_Comm_split(MPI_Comm comm, // int color, int key, MPI_Comm *ncomm)
mpiComm ncomm(processor(local_comm, local_leader),
              processor(peer_comm, peer_leader), tag);
// build MPI_INTERCOMM_CREATE(local_comm, local_leader, peer_comm,
                           // remote_leader, tag, &ncomm)
mpiComm ncomm(intercomm, high); // build using // MPI_Intercomm_merge(intercomm, high, &ncomm)
mpiRequest rq; // defined an MPI_Request
mpiRequest[int] arq(10); // defined an array of 10 MPI_Request

mpiSize(comm); // return the size of comm (int)
mpiRank(comm); // return the rank in comm (int)
```
processor(i)    // return processor i with no Request in MPI_COMM_WORLD
processor(mpiAnySource) // return processor any source with no Request in MPI_COMM_WORLD
processor(i,comm)  // return processor i with no Request in comm
processor(comm,i)  // return processor i with no Request in comm
processor(i,rq,comm) // return processor i with Request rq in comm
processor(i,rq) // return processor i with Request rq in MPI_COMM_WORLD

processorblock(i) // return processor i in MPI_COMM_WORLD in block mode for synchronously communication
processorblock(mpiAnySource) // return processor any source in MPI_COMM_WORLD in block mode for synchronously communication
processorblock(i,comm) // return processor i in comm in block mode

int status;    // to get the MPI status of send / recv
processor(10) « a « b;    // send a,b asynchronously to the process 1,
processor(10) » a » b;    // receive a,b synchronously from the process 10,
broadcast(processor(10,comm),a);    // broadcast from processor of comm to other comm processor

status=Send( processor(10,comm) , a);    // send synchronously to the process 10 the data a
status=Recv( processor(10,comm) , a);    // receive synchronously from the process 10 the data a;
status=Isend( processor(10,comm) , a);          // send asynchronously to
          // the process 10, the data a without request
status=Isend( processor(10,rq,comm) , a) ;    // send asynchronously to
          // the process 10, the data a with request
status=Irecv( processor(10,rq) , a) ;          // receive synchronously from
broadcast(processor(comm,a));                 // Broadcast to all process of comm

mpiBarrier(comm) ;                           // do a MPI_Barrier on communicator comm,
mpiWait(rq);                                 // wait on of Request,
mpiWaitAll(arq);                             // wait add of Request array,
mpiWtime() ;                                 // return MPIWtime in second (real),
mpiWtick() ;                                 // return MPIWTick in second (real),
mpiAlltoall(a,b[,] ,comm) ;
mpiAllgather(a,b[,] ,comm) ;
mpiGather(a,b,processor(..) ) ;
mpiScatter(a,b,processor(..)) ;
mpiReduce(a,b,processor(..),mpiMAX) ;
mpiAllReduce(a,b,comm, mpiMAX) ;
mpiReduceScatter(a,b,comm, mpiMAX) ;
....
1. Build matrix in parallel by assembling par region remark with the \texttt{change} function you change the region numbering to build region.

```cpp
real c = mpisize/real(Th.nt);
Th=change(Th,fregion= min(mpisize-1,int(nuTriangle*c)));
```

2. Assemble the full matrix in //

```cpp
varf vlaplace(uh,vh) = // definition de problem
  int3d(Th,mpirank)( uh*vh+ dt*Grad(uh)'*grad(vh) )
  + int3d(Th,mpirank)( dt*vh*f ) + on(1,u=g);
matrix A,Ai = vlaplace(Vh,Vh,tgv=ttgv);
mpiReduce(Ai,A,processor(0),mpiSUM); // assemble in //
```

3. Solve the linear using a good parallel solver (MUMPS)

```cpp
load "MUMPS"
  uh[] = A^-1*b ;
  // resolution

Run:Heat3d.edp                   Run:NSCaraCyl-100-mpi2.edp
```
Outline

5 MPI/Parallel

- Schwarz method with overlap
  - Poisson equation with Schwarz method
  - Transfer Part
  - parallel GMRES
  - A simple Coarse grid solver
  - Numerical experiment
  - An other way to build a 2-level Schwarz with oscilation
  - HPDDM example for HPC computation
  - Stroke brain imagery/HPDDM
Outline

5 MPI/Parallel
- Schwarz method with overlap
- Poisson equation with Schwarz method
- Transfer Part
- parallel GMRES
- A simple Coarse grid solver
- Numerical experiment
- An other way to build a 2-level Schwarz with oscilation
- HPDDM example for HPC computation
- Stroke brain imagery/HPDDM
To solve the following Poisson problem on domain $\Omega$ with boundary $\Gamma$ in $L^2(\Omega)$:

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

where $f \in L^2(\Omega)$ and $g \in H^{\frac{1}{2}}(\Gamma)$ are two given functions.

Let introduce $(\pi_i)_{i=1,..,N_p}$ a positive regular partition of the unity of $\Omega$, q-e-d:

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

Denote $\Omega_i$ the sub domain which is the support of $\pi_i$ function and also denote $\Gamma_i$ the boundary of $\Omega_i$.

The parallel Schwarz method is Let $\ell = 0$ the iterator and a initial guest $u^0$ respecting the boundary condition (i.e. $u^0|\Gamma = g$).

$$\forall i = 1..,N_p : -\Delta u_i^\ell = f, \text{ in } \Omega_i, \text{ and } u_i^\ell = u^\ell \text{ on } \Gamma_i$$

$$u^{\ell+1} = \sum_{i=1}^{N_p} \pi_i u_i^\ell$$
Some Remark

We never use finite element space associated to the full domain $\Omega$ because it to expensive. So we use on each domain $i$ we defined $J_i = \{ j \in 1, \ldots, N_p / \Omega_i \cap \Omega_j \neq \emptyset \}$ and we have

$$ (u^{\ell+1})_{|\Omega_i} = \sum_{j \in J_i} (\pi_j u^\ell)_{|\Omega_i} $$

(9)

We denote $u^\ell_{h|i}$ the restriction of $u^\ell_h$ on $V_{h,i}$, so the discrete problem on $\Omega_i$ of problem (7) is find $u^\ell_{h,i} \in V_{h,i}$ such that:

$$ \forall v_{h,i} \in V_{0,i} : \int_{\Omega_i} \nabla v_{h,i} \cdot \nabla u^\ell_{h,i} = \int_{\Omega_i} f v_{h,i}, $$

$$ \forall k \in N_{h,i}^{\Gamma_i} : \sigma^k_i (u^\ell_{h,i}) = \sigma^k_i (u^\ell_{h|i}) $$

where $N_{h,i}^{\Gamma_i}$ is the set of the degree of freedom (Dof) on $\partial \Omega_i$ and $\sigma^k_i$ the Dof of $V_{h,i}$. 
MPI/Parallel

- Schwarz method with overlap
- Poisson equation with Schwarz method

Transfer Part

- parallel GMRES
- A simple Coarse grid solver
- Numerical experiment
- An other way to build a 2-level Schwarz with oscillation
- HPDDM example for HPC computation
- Stroke brain imagery/HPDDM
Transfer Part equation (5)

To compute $v_i = (\pi_i u_i)|_{\Omega_i} + \sum_{j \in J_i} (\pi_j u_j)|_{\Omega_i}$ and can be write the freefem++ function Update with asynchronous send/recv (Otherwise dead lock).

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

5 MPI/Parallel

- Schwarz method with overlap
- Poisson equation with Schwarz method
- Transfer Part
- parallel GMRES
- A simple Coarse grid solver
- Numerical experiment
- An other way to build a 2-level Schwarz with oscillation
- HPDDM example for HPC computation
- Stroke brain imagery/HPDDM
Finally you can easily accelerate the fixe point algorithm by using a parallel GMRES algorithm after the introduction the following affine $S_i$ operator sub domain $\Omega_i$.

```
func real[int] Si(real[int]& U) {
    real[int] V(U.n) ; b = onG .* U;
    b = onG ? b : Bi ;
    V = Ai^-1*b;
    // (7)
    Update(V,U);
    // (??)
    V -= U;   return V; }
```

Where the parallel MPIGMRES or MPICG algorithm is to solve $A_i x_i = b_i, i = 1, .., N_p$ 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; }
```
Outline

5 MPI/Parallel
- Schwarz method with overlap
- Poisson equation with Schwarz method
- Transfer Part
- parallel GMRES
- A simple Coarse grid solver
- Numerical experiment
- An other way to build a 2-level Schwarz with oscilation
- HPDDM example for HPC computation
- Stroke brain imagery/HPDDM
Coarse grid solver

A simple coarse grid is we solve the problem on the coarse grid:

```c
func bool CoarseSolve(real[int]& V, real[int]& U,
                         mpiComm& comm)
{
    if(AC.n==0 && mpiRank(comm)==0)  // first time build
        AC = vPbC(VhC,VhC,solver=sparsesolver);
    real[int] Uc(Rci.n), Bc(Uc.n);
    Uc = Rci*U;  // Fine to Coarse
    mpiReduce(Uc,Bc,processor(0,comm),mpiSUM);
    if(mpiRank(comm)==0)
        Uc = AC^-1*Bc;  // solve of proc 0
    broadcast(processor(0,comm),Uc);
    V = Pci*Uc;  // Coarse to Fine
}
Limitation: if the initial problem, data have oscillation, you must use homogenization technic on coarse problem, or use the F. Nataf and co, preconditionner.
```
So we finally we get 4 algorithms

1. The basic schwarz algorithm \( u^{\ell+1} = S(u^{\ell}) \), where \( S \) is one iteration of schwarz process.

2. Use the GMRES to find \( u \) solution of the linear system \( Su - u = 0 \).

3. Use the GMRES to solve parallel problem \( A_iu_i = b_i \), \( i = 1, \ldots, N_p \), with RAS precondicionneur

4. Use the method with two level precondicionneur RAS and Coarse.

On the SGI UV 100 of the lab:

- Load balance
  - 10
  - 100
  - 1000
  - 10000
  - 100000
  - 1e+06
  - 1e+07
  - 1e+08
  - 1e+09

- Elapse time/s

- Nb of DoF

- Computation
  - \( 2 \times 10^{-6} n^{1.2} \)

- Las Palmas de Gran Canaria, 25-29 June 2018 F. Hecht et al. Cours FreeFem++, XVIII Spanish-French school 120 / 224
Outline

5 MPI/Parallel

- Schwarz method with overlap
- Poisson equation with Schwarz method
- Transfer Part
- parallel GMRES
- A simple Coarse grid solver

Numerical experiment

- An other way to build a 2-level Schwarz with oscilation
- HPDDM example for HPC computation
- Stroke brain imagery/HPDDM
We consider first example in an academic situation to solve Poisson Problem on the cube $\Omega = ]0,1[^3$

$$-\Delta u = 1, \text{ in } \Omega; \quad u = 0, \text{ on } \partial\Omega. \quad (10)$$

With a cartesian meshes $T_{hn}$ of $\Omega$ with $6n^3$ tetrahedron, the coarse mesh is $T_{hm}$, and $m$ is a divisor of $n$.

We do the validation of the algorithm on a Laptop Intel Core i7 with 4 core at 1.8 Ghz with 4 Go of RAM DDR3 at 1067 Mhz,
MPI/Parallel

- Schwarz method with overlap
- Poisson equation with Schwarz method
- Transfer Part
- parallel GMRES
- A simple Coarse grid solver
- Numerical experiment
  - An other way to build a 2-level Schwarz with oscillation
    - Choice of the coarse space
    - First Numerical results
    - Parallel implementation
    - Scalability tests
  - HPDDM example for HPC computation
  - Stroke brain imagery/HPDDM
Motivation

Large discretized system of PDEs
strongly heterogeneous coefficients
(high contrast, nonlinear, multiscale)

E.g. Darcy pressure equation, $P_1$-finite elements:

\[ AU = F \]

\[ \text{cond}(A) \sim \frac{\alpha_{\text{max}}}{\alpha_{\text{min}}} h^{-2} \]

Goal:
iterative solvers
robust in size and heterogeneities

Applications:
flow in heterogeneous / stochastic / layered media
structural mechanics
electromagnetics
etc.
Adding a coarse space

We add a coarse space correction (aka second level)
Let $V_H$ be the coarse space and $Z$ be a basis, $V_H = \text{span } Z$, writing $R_0 = Z^T$ we define the two level preconditioner as:

$$M_{ASM,2}^{-1} := R_0^T (R_0 A R_0^T)^{-1} R_0 + \sum_{i=1}^{N} R_i^T A_i^{-1} R_i.$$ 

The Nicolaides approach is to use the kernel of the operator as a coarse space, this is the constant vectors, in local form this writes:

$$Z := (R_i^T D_i R_i 1)_{1 \leq i \leq N}$$

where $D_i$ are chosen so that we have a partition of unity:

$$\sum_{i=1}^{N} R_i^T D_i R_i = Id.$$

Theoretical convergence result

Theorem (Widlund, Dryija)

Let $M_{ASM,2}^{-1}$ be the two-level additive Schwarz method:

$$\kappa(M_{ASM,2}^{-1} A) \leq C \left( 1 + \frac{H}{\delta} \right)$$

where $\delta$ is the size of the overlap between the subdomains and $H$ the subdomain size.

This does indeed work very well

<table>
<thead>
<tr>
<th>Number of subdomains</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASM</td>
<td>18</td>
<td>35</td>
<td>66</td>
<td>128</td>
</tr>
<tr>
<td>ASM + Nicolaides</td>
<td>20</td>
<td>27</td>
<td>28</td>
<td>27</td>
</tr>
</tbody>
</table>
Failure for Darcy equation with heterogeneities

\[- \nabla \cdot (\alpha(x, y) \nabla u) = 0 \quad \text{in} \quad \Omega \subset \mathbb{R}^2, \]

\[
u = 0 \quad \text{on} \quad \partial \Omega_D, \]

\[
\frac{\partial u}{\partial n} = 0 \quad \text{on} \quad \partial \Omega_N.
\]

Decomposition

<table>
<thead>
<tr>
<th>Jump</th>
<th>1</th>
<th>10</th>
<th>$10^2$</th>
<th>$10^3$</th>
<th>$10^4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASM</td>
<td>39</td>
<td>45</td>
<td>60</td>
<td>72</td>
<td>73</td>
</tr>
<tr>
<td>ASM + Nicolaides</td>
<td>30</td>
<td>36</td>
<td>50</td>
<td>61</td>
<td>65</td>
</tr>
</tbody>
</table>

Our approach

Fix the problem by an optimal and proven choice of a coarse space $Z$. 
Objectives

Strategy

Define an appropriate coarse space $V_{H^2} = \text{span}(Z_2)$ and use the framework previously introduced, writing $R_0 = Z_2^T$ the two level preconditioner is:

$$P_{ASM2}^{-1} := R_0^T (R_0 A R_0^T)^{-1} R_0 + \sum_{i=1}^{N} R_i^T A_i^{-1} R_i.$$  

The coarse space must be

- Local (calculated on each subdomain) $\rightarrow$ parallel
- Adaptive (calculated automatically)
- Easy and cheap to compute
- Robust (must lead to an algorithm whose convergence is proven not to depend on the partition nor the jumps in coefficients)
Abstract eigenvalue problem

**Gen.EVP** per subdomain:

Find \( p_{j,k} \in V_h|\Omega_j \) and \( \lambda_{j,k} \geq 0 \):

\[
\begin{align*}
  a_{\Omega_j}(p_{j,k}, v) &= \lambda_{j,k} a_{\Omega_j}(\Xi_j p_{j,k}, \Xi_j v) \quad \forall v \in V_h|\Omega_j \\
  A_j p_{j,k} &= \lambda_{j,k} X_j A_j^0 X_j p_{j,k} \quad (X_j \ldots \text{diagonal})
\end{align*}
\]

\( \Xi_j \) is the partition unity

\( a_D \ldots \text{restriction of } a \text{ to } D \)

---

**In the two-level ASM:**

Choose first \( m_j \) eigenvectors per subdomain:

\[
V_0 = \text{span}\{\Xi_j p_{j,k}\}_{k=1,\ldots,m_j}^{j=1,\ldots,N}
\]

This automatically includes Zero Energy Modes.
Comparison with existing works

Galvis & Efendiev (SIAM 2010):

\[
\int_{\Omega_j} \kappa \nabla p_{j,k} \cdot \nabla v \, dx = \lambda_{j,k} \int_{\Omega_j} \kappa p_{j,k} v \, dx \quad \forall v \in V_{h|\Omega_j}
\]

Efendiev, Galvis, Lazarov & Willems (submitted):

\[
a_{\Omega_j}(p_{j,k}, v) = \lambda_{j,k} \sum_{i \in \text{neighb}(j)} a_{\Omega_j}(\xi_j \xi_i p_{j,k}, \xi_j \xi_i v) \quad \forall v \in V_{|\Omega_j}
\]

\(\xi_j \ldots\) partition of unity, calculated adaptively (MS)

Our gen.EVP:

\[
a_{\Omega_j}(p_{j,k}, v) = \lambda_{j,k} a_{\Omega_j}(\Xi_j p_{j,k}, \Xi_j v) \quad \forall v \in V_{h|\Omega_j}
\]

both matrices typically singular \(\Rightarrow\) \(\lambda_{j,k} \in [0, \infty]\)
Theory

Two technical assumptions.

Theorem (Spillane, Dolean, Hauret, N., Pechstein, Scheichl)

If for all $j$: $0 < \lambda_{j,m_{j+1}} < \infty$:

$$\kappa(M_{ASM,2}^{-1}A) \leq (1 + k_0) \left[ 2 + k_0 (2k_0 + 1) \max_{j=1}^N \left( 1 + \frac{1}{\lambda_{j,m_{j+1}}} \right) \right]$$

Possible criterion for picking $m_j$: (used in our Numerics)

$$\lambda_{j,m_{j+1}} < \frac{\delta_j}{H_j}$$

$H_j$ . . . subdomain diameter, $\delta_j$ . . . overlap
Numerical results via a Domain Specific Language

FreeFem++ (http://www.freefem.org/ff++), with:

- Metis Karypis and Kumar 1998
- SCOTCH Chevalier and Pellegrini 2008
- UMFPACK Davis 2004
- ARPACK Lehoucq et al. 1998
- MPI Snir et al. 1995

- Intel MKL
- PARDISO Schenk et al. 2004
- MUMPS Amestoy et al. 1998
- PaStiX Hénon et al. 2005
\[ E_1 = 2 \cdot 10^{11} \]
\[ \nu_1 = 0.3 \]
\[ E_2 = 2 \cdot 10^7 \]
\[ \nu_2 = 0.45 \]

**METIS partitions with 2 layers added**

<table>
<thead>
<tr>
<th>subd.</th>
<th>dofs</th>
<th>AS-1</th>
<th>AS-ZEM</th>
<th>((V_H))</th>
<th>GENEO</th>
<th>((V_H))</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>13122</td>
<td>93</td>
<td>134</td>
<td>(12)</td>
<td>42</td>
<td>(42)</td>
</tr>
<tr>
<td>16</td>
<td>13122</td>
<td>164</td>
<td>165</td>
<td>(48)</td>
<td>45</td>
<td>(159)</td>
</tr>
<tr>
<td>25</td>
<td>13122</td>
<td>211</td>
<td>229</td>
<td>(75)</td>
<td>47</td>
<td>(238)</td>
</tr>
<tr>
<td>64</td>
<td>13122</td>
<td>279</td>
<td>167</td>
<td>(192)</td>
<td>45</td>
<td>(519)</td>
</tr>
</tbody>
</table>
Numerics – 3D Elasticity

Iterations (CG) vs. number of subdomains

\[ E_1 = 2 \cdot 10^{11} \]
\[ \nu_1 = 0.3 \]
\[ E_2 = 2 \cdot 10^7 \]
\[ \nu_2 = 0.45 \]

Relative error vs. iterations

16 regular subdomains

<table>
<thead>
<tr>
<th>subd.</th>
<th>dofs</th>
<th>AS-1</th>
<th>AS-ZEM (( V_H ))</th>
<th>GENEO (( V_H ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1452</td>
<td>79</td>
<td>54 (24)</td>
<td>16 (46)</td>
</tr>
<tr>
<td>8</td>
<td>29040</td>
<td>177</td>
<td>87 (48)</td>
<td>16 (102)</td>
</tr>
<tr>
<td>16</td>
<td>58080</td>
<td>378</td>
<td>145 (96)</td>
<td>16 (214)</td>
</tr>
</tbody>
</table>

AS-ZEM (Rigid body motions): \( m_j = 6 \)
Numerical results – Optimality

Layers of hard and soft elastic materials

\[ m_i \] is given automatically by the strategy.

<table>
<thead>
<tr>
<th>#Z per subd.</th>
<th>one level</th>
<th>ZEM</th>
<th>GenEO</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \max(m_i - 1, 3) ]</td>
<td></td>
<td></td>
<td>2600 (93)</td>
</tr>
<tr>
<td>[ m_i ]</td>
<td>5.1 \times 10^5 (184)</td>
<td>1.4 \times 10^4 (208)</td>
<td>53 (35)</td>
</tr>
<tr>
<td>[ m_i + 1 ]</td>
<td></td>
<td></td>
<td>45 (25)</td>
</tr>
</tbody>
</table>

Condition number (iteration count) for one and two level ASMs

- Taking one fewer eigenvalue has a huge influence on the iteration count
- Taking one more has only a small influence
Eigenvalues and eigenvectors

Eigenvector number 1 is $-3.07387 \times 10^{-15}$; exaggeration coefficient is: $1000000000$

Eigenvector number 2 is $8.45471 \times 10^{-16}$; exaggeration coefficient is: $1000000000$

Eigenvector number 3 is $5.3098 \times 10^{-15}$; exaggeration coefficient is: $1000000000$

Eigenvector number 4 is $1.15244 \times 10^{-05}$; exaggeration coefficient is: $100000$

Eigenvector number 5 is $1.87668 \times 10^{-05}$; exaggeration coefficient is: $100000$

Eigenvector number 6 is $4.99451 \times 10^{-05}$; exaggeration coefficient is: $100000$

Eigenvector number 7 is $0.000132778$; exaggeration coefficient is: $100000$

Eigenvector number 8 is $0.000141253$; exaggeration coefficient is: $100000$

Eigenvector number 9 is $0.000396054$; exaggeration coefficient is: $100000$

Eigenvector number 10 is $0.169032$; exaggeration coefficient is: $100000$

Eigenvector number 11 is $0.169212$; exaggeration coefficient is: $100000$

Eigenvector number 12 is $0.169217$; exaggeration coefficient is: $100000$

Eigenvector number 13 is $0.16922$; exaggeration coefficient is: $1000000$

Eigenvector number 14 is $0.169515$; exaggeration coefficient is: $100000$

Eigenvector number 15 is $0.170536$; exaggeration coefficient is: $10000$

Logarithmic scale
Figure: Two dimensional diffusivity $\kappa$
Channels and inclusions: $1 \leq \alpha \leq 1.5 \times 10^6$, the solution and partitionings (Metis or not)
Parallel implementation

PhD of Pierre Jolivet.
Since version 1.16, bundled with the Message Passing Interface.
FreeFem++ is working on the following parallel architectures (among others):

<table>
<thead>
<tr>
<th>System</th>
<th>N° of cores</th>
<th>Memory</th>
<th>Peak perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpc1@LJLL</td>
<td>160@2.00 Ghz</td>
<td>640 Go</td>
<td>∼ 10 TFLOP/s</td>
</tr>
<tr>
<td>titane@CEA</td>
<td>12192@2.93 Ghz</td>
<td>37 To</td>
<td>140 TFLOP/s</td>
</tr>
<tr>
<td>babel@IDRIS</td>
<td>40960@850 Mhz</td>
<td>20 To</td>
<td>139 TFLOP/s</td>
</tr>
<tr>
<td>curie@CEA</td>
<td>92000@2.93 Ghz</td>
<td>315 To</td>
<td>1.6 PFLOP/s</td>
</tr>
</tbody>
</table>

http://www.idris.fr, Orsay, France.
http://www.prace-project.eu.
Strong scalability in two dimensions heterogeneous elasticity

Elasticity problem with heterogeneous coefficients

Speed-up for a 1.2 billion unknowns 2D problem. Direct solvers in the subdomains.
Strong scalability in three dimensions heterogeneous elasticity

Elasticity problem with heterogeneous coefficients

Speed-up for a 160 million unknowns 3D problem. Direct solvers in subdomains. Peak performance wall-clock time: 36s.
Weak scalability in two dimensions

Darcy problems with heterogeneous coefficients

Efficiency for a 2D problem. Direct solvers in the subdomains. Final size: 22 billion unknowns. Wall-clock time: \( \sim 200s \).
Weak scalability in three dimensions

Darcy problems with heterogeneous coefficients

Efficiency for a 3D problem. Direct solvers in the subdomains. Final size: 2 billion unknowns. Wall-clock time: $\sim 200s$. 

Las Palmas de Gran Canaria, 25-29 June 2018
F. Hecht et al.
Cours FreeFem++, XVIII Spanish-French school
Preprints available on HAL: [http://hal.archives-ouvertes.fr/](http://hal.archives-ouvertes.fr/)


Outline

5 MPI/Parallel

- Schwarz method with overlap
- Poisson equation with Schwarz method
- Transfer Part
- parallel GMRES
- A simple Coarse grid solver
- Numerical experiment
- An other way to build a 2-level Schwarz with oscillation

- HPDDM example for HPC computation
- Stroke brain imagery/HPDDM
Reader the book of V. Dolean, P. Jolivet and F. Nataf, An Introduction to Domain Decomposition Methods: algorithms, theory and parallel implementation SIAM bookstore, 2015. (see pdf), (see Erratum)

Run:diffusion-3d.edp
Run:diffusion-3d-PETSc.edp
Run:elasticity-3d.edp
Run:elasticity-3d-PETSc.edp
Run:Stokes-3d.edp
Run:Stokes-3d-PETSc.edp
Run:helmholtz-2d.edp
Run:helmholtz-2d-PETSc.edp
Outline

5 MPI/Parallel

- Schwarz method with overlap
- Poisson equation with Schwarz method
- Transfer Part
- parallel GMRES
- A simple Coarse grid solver
- Numerical experiment
- An other way to build a 2-level Schwarz with oscillation
- HPDDM example for HPC computation
- Stroke brain imagery/HPDDM
Motivation: types of brain stroke

2 types of stroke:

- **Ischemic**
- **Hemorrhagic**

The correct treatment depends on the type of stroke:

- $\Rightarrow$ restore blood flow
- $\Rightarrow$ lower blood pressure
Motivation:

In order to differentiate between ischemic and hemorrhagic stroke, CT scan or MRI is typically used. **Microwave tomography** is a novel and promising imaging technique, especially for medical and brain imaging.

<table>
<thead>
<tr>
<th></th>
<th>CT scan</th>
<th>MRI</th>
<th>microwave tomography</th>
</tr>
</thead>
<tbody>
<tr>
<td>resolution</td>
<td>excellent</td>
<td>excellent</td>
<td>good</td>
</tr>
<tr>
<td>fast</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>safe</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>mobile</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>cost</td>
<td>~1 000 000 €</td>
<td>~1 000 000 €</td>
<td>~300 000 €</td>
</tr>
</tbody>
</table>

Diagnosing a stroke at the earliest possible stage is crucial for all following therapeutic decisions.
EMTensor GmbH, Vienna, Austria.

First-generation prototype: cylindrical chamber composed of 5 rings of 32 antennas (ceramic-loaded waveguides).
Motivation

MEDIMAX ANR project: development of a **new robust inversion tool associated with** the electromagnetic forward problem **in highly heterogeneous media based on the high-level integrated development environment FreeFem++.**

**Partners:**

- **LEAT**: Christian Pichot (project coordinator), Iannis Aliferis, Claire Migliaccio and Ibtissam El Kanfoud;
- **JAD**: Victorita Dolean, Francesca Rapetti, Richard Pasquetti and Marcella Bonazzoli;
- **MAP5**: Maya de Buhan and Marion Darbas;
- **LJLL**: Frédéric Nataf, Frédéric Hecht, Antoine Le Hyaric and Pierre-Henri Tournier.
We consider in $\Omega$ a linear, isotropic, non-magnetic, dispersive, dissipative dielectric material. The direct problem consists in finding the electromagnetic field distribution in the whole chamber, given a known material and transmitted signal.
The direct problem

For each of the $5 \times 32$ antennas, the associated electric field $E_i$ is the solution of Maxwell’s equations:

\[
\begin{align*}
\nabla \times (\nabla \times E_i) - \mu_0 \left( \omega^2 \varepsilon + i \omega \sigma \right) E_i &= 0 \quad \text{in } \Omega, \\
E_i \times n &= 0 \quad \text{on } \Gamma_{\text{metal}}, \\
(\nabla \times E_i) \times n + i \beta E_i \times n &= g \quad \text{on } \Gamma_i, \\
(\nabla \times E_i) \times n + i \beta E_i \times n &= 0 \quad \text{on } \Gamma_j, \ j \neq i,
\end{align*}
\]

where $\mu_0$ is the permittivity of free space, $\omega$ is the incident angular frequency, $\beta$ is the wavenumber of the waveguide, $\varepsilon > 0$ is the dielectric permittivity and $\sigma > 0$ is the conductivity.
The inverse problem

Solving the inverse problem corresponds to minimizing the following cost functional:

\[
J(\kappa) = \frac{1}{2} \sum_{i=1}^{160} \sum_{j=1}^{160} \left| S_{ij}(\kappa) - S_{ij}^{obs} \right|^2
\]

\[
= \frac{1}{2} \sum_{i=1}^{160} \sum_{j=1}^{160} \left| \frac{\int_{\Gamma_j} \overrightarrow{E_i(\kappa)} \cdot \overrightarrow{E_0} d\gamma}{\int_{\Gamma_j} |\overrightarrow{E_0}|^2 d\gamma} - S_{ij}^{obs} \right|^2,
\]

where \(\kappa(x) \in \mathbb{C}\) and \(S_{ij}(\kappa)\) depends on the solution \(\overrightarrow{E_i(\kappa)}\) to

\[
\begin{aligned}
\nabla \times (\nabla \times \overrightarrow{E_i}) - \kappa \overrightarrow{E_i} &= 0 & \text{in } \Omega, \\
\overrightarrow{E_i} \times \overrightarrow{n} &= 0 & \text{on } \Gamma_{\text{metal}}, \\
(\nabla \times \overrightarrow{E_i}) \times \overrightarrow{n} + i\beta \overrightarrow{E_i} \times \overrightarrow{n} &= \overrightarrow{g} & \text{on } \Gamma_i, \\
(\nabla \times \overrightarrow{E_i}) \times \overrightarrow{n} + i\beta \overrightarrow{E_i} \times \overrightarrow{n} &= 0 & \text{on } \Gamma_j, \ j \neq i.
\end{aligned}
\]
Numerical experiment - hemorrhagic stroke

Idea: reconstruct the permittivity slice by slice, by taking into account the transmitting antennas corresponding to only one ring and truncating the computational domain.

⇒ Reconstructed images corresponding to one ring obtained in less than 5 minutes.
6 No Linear Problem
- Newton Method
- Navier-Stokes
- Variational Inequality
- Ground water
- Hyper elasticity equation
- Periodic Surface Acoustic Waves Transducer Analysis
- Phase change with Natural Convection
Newton methods

To solve $F(u) = 0$ the Newton’s algorithm is

1. $u^0$ a initial guess
2. do
   1. find $w^n$ solution of $DF(u^n)w^n = F(u^n)$
   2. $u^{n+1} = u^n - w^n$
   3. if($||w^n|| < \varepsilon$) break;

The Optimize Newton Method if $F = C + L + N$, where $C$ is the constant part, $L$ is Linear part and $N$ is Non linear part of $F$. we have $DF = L + DN$ and $DF(u^n)u^{n+1} = DF(u^n)u^n - F(u^n) = DN(u^n)u^n - N(u^n) - C$.

So the change in algorithm are:

2. find $u^{n+1}$ solution of $DF(u^n)u^{n+1} = DN(u^n)u^n - N(u^n) - C$
3. if($||u^{n+1} - u^n|| < \varepsilon$) break;
Outline

6 No Linear Problem
- Newton Method
- Navier-Stokes
- Variational Inequality
- Ground water
- Hyper elasticity equation
- Periodic Surface Acoustic Waves Transducer Analysis
- Phase change with Natural Convection
For Navier Stokes problem the Newton algorithm is: \( \forall v, q, \)

\[
F(u, p) = \int_\Omega (u \nabla)u.v + u.v + \nu \nabla u : \nabla v - q \nabla . u - p \nabla . v + BC
\]

\[
DF(u, p)(w, w_p) = \int_\Omega (w \nabla)u.v + (u \nabla)w.v
\]

\[
+ \int_\Omega \nu \nabla w : \nabla v - q \nabla . w - p_w \nabla . v + BC0
\]

Run:cavityNewton.edp  Run:NSNewtonCyl-100-mpi.edp
incompressible Navier-Stokes equation with characteristics 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

\[
\frac{1}{\tau} (u^{n+1} - u^n \circ X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} = 0, \\
\nabla \cdot u^{n+1} = 0
\]  

(11)

The term \( X^n(x) \approx x - \tau u^n(x) \) will be computed by the interpolation operator or
convect operator.
Or better we use an order 2 schema, BDF1

\[
\frac{\partial u}{\partial t} + (u \cdot \nabla)u \approx \frac{(3u^{n+1} - 4u^n \circ X_1^n + u^{n-1} \circ X_2^n)}{2\tau}
\]

with \( u^* = 2u^n - u^{n-1} \), and \( X_1^n(x) \approx x - \tau u^*(x) \), \( X_2^n(x) \approx x - 2\tau u^*(x) \)
The generalise Stokes problem is find \( u, p \) solution of

\[
Au + Bp = f, \quad ^tBu = 0
\]

with \( A \equiv (\alpha \text{Id} + \nu \Delta) \) and \( B \equiv \nabla \). remark, if \( A \) est symmetric positive the you can use a conjugate gradient to solve the the following problem

\[
^tBA^{-1}Bp = ^tBA^{-1}f
\]

Now in a periodic domain, all differential operators commute and the Uzawa algorithm comes to solving the linear operator \(-\nabla.((\alpha \text{Id} - \nu \Delta)^{-1}\nabla)\), where \( \text{Id} \) is the identity operator. So the preconditioner suggested is \(-\alpha \Delta^{-1} + \nu \text{Id}\).

the term \( \frac{\partial u}{\partial t} + (u \cdot \nabla)u \) is the total derivative and discretization in time

\[
\frac{1}{\tau}(u^{n+1} - u^n \circ X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} = 0, \\
\nabla \cdot u^{n+1} = 0
\]

The term \( X^n(x) \approx x - \tau u^n(x) \) will be computed with convect operator.
The **NSI** 3d code

```plaintext
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;
  b=vNS(0, VVh);
  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, time ="+t, wait=0); }
Run:NSI3d.edp
```

---

Las Palmas de Gran Canaria, 25-29 June 2018  F. Hecht et al. Cours FreeFem++, XVIII Spanish-French school 163 / 224
No Linear Problem
- Newton Method
- Navier-Stokes
- Variational Inequality
- Ground water
- Hyper elasticity equation
- Periodic Surface Acoustic Waves Transducer Analysis
- Phase change with Natural Convection
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 \( v \).


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

Run:VI-2-membrane-adap.edp
6 No Linear Problem
- Newton Method
- Navier-Stokes
- Variational Inequality
- Ground water
  - Hyper elasticity equation
  - Periodic Surface Acoustic Waves Transducer Analysis
  - Phase change with Natural Convection
A Free Boundary problem, (phreatic water)

Let a trapezoidal domain $\Omega$ defined in FreeFem++:

```plaintext
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 $\Gamma_a$
border b(t=0,h1){x=L;y=t;label=2;}; // the source $\Gamma_b$
border f(t=L,0){x=t;y=t*(h1-h)/L+h;label=3;}; // $\Gamma_f$
border d(t=h,0){x=0;y=t;label=4;}; // Left impermeable $\Gamma_d$

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 problem is: find $p$ and $\Omega$ such that:

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

(\text{Neumann})

(\text{Dirichlet})

where the input water flux is $q = 0.02$, and $K = 0.5$. The velocity $u$ of the water is given by $u = -\nabla p$. 
We use the following fix point method: (with bad main B.C. Run:freeboundaryPB.edp) 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 \( p = y \) on \( \Gamma^k_b \) et on \( \Gamma^k_f \)
\[
\int_{\Omega^k} \nabla p \nabla p' = 0, \quad \forall p' \in V \text{ with } p' = 0 \text{ on } \Gamma^k_b \cup \Gamma^k_f
\]

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 \( \Gamma^k_a \) (bottom)
\[
\int_{\Omega^k} \nabla v \nabla v' = \int_{\Gamma^k_f} \left( \frac{\partial p}{\partial n} - \frac{q}{K} n_x \right) v', \quad \forall v' \in V \text{ with } v' = 0 \text{ sur } \Gamma^k_a
\]
remark: we can use the previous equation to evaluate
\[
\int_{\Gamma^k} \frac{\partial p}{\partial n} v' = - \int_{\Omega^k} \nabla p \nabla v'
\]
Implementation

The new domain is: \( \Omega^{k+1} = \mathcal{F}(\Omega^k) \) Warning if is the movement is too large we can have triangle overlapping.

```cpp
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
}
Run:freeboundary.edp
```
6 No Linear Problem
   - Newton Method
   - Navier-Stokes
   - Variational Inequality
   - Ground water
   - Hyper elasticity equation
   - Periodic Surface Acoustic Waves Transducer Analysis
   - Phase change with Natural Convection
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

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

where $J_1 = I_1^{\frac{1}{3}}$, $J_2 = I_2^{\frac{2}{3}}$, $J = I_3^{\frac{1}{3}}$.

let $u$ the displacement, when

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

The problem is find

$$u = \arg\min_u W(I_1, I_2, I_3)$$
Hyper elasticity equation

```cpp
fespace Wh(Th,[P2,P2]);

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;
}
```

Run:Hyper-Elasticity-2d.edp see:ElasticLaw2d.idp see:CiarletGemona.idp
6 No Linear Problem
- Newton Method
- Navier-Stokes
- Variational Inequality
- Ground water
- Hyper elasticity equation
- Periodic Surface Acoustic Waves Transducer Analysis
- Phase change with Natural Convection
A true industrial numerical problem (2d):

1. 3 EDP: Dielectric, Elasticity, Piezoelectric, Linear, harmonic approximation
2. $\alpha$-periodic B.C. (New simple idea)
3. semi-infinite Dielectric domain (classical Fourier/Floquet transforme)
4. semi-infinite Piezoelectric domain (Hard)

In 9 month, we build with P. Ventura (100%, me 10%) a numerical simulator from scratch (8 months for the validation), The only thing to add to freefem++ is a interface with lapack to compute eigenvector of full $8 \times 8$ matrix.

A good message: Les calculs des paramètres physiques des transducteurs dans la bande d’arrêt et les évaluations de capacité statiques sont très satisfaits par rapport aux résultats expérimentaux!
In the **dielectric medium** $\Omega_d$,

$$D = \varepsilon_d E$$  \hspace{1cm} (13)

In the **elastic medium** $\Omega_e$

$$T = C_e : S$$  \hspace{1cm} (14)

With $C_e$ is the elastic tensor for the elastic metallic domain. In the **piezoelectric domain**

$$\begin{cases} 
T = C_{\text{p}}^E : S - eE \\
D = e^T S + \varepsilon S E 
\end{cases}$$  \hspace{1cm} (15)
The material domain $\Omega_m$ obeys Newton’s second law:

$$\nabla \cdot T = \rho \frac{\partial^2 u}{\partial t^2}$$  \hspace{1cm} (16)

The quasi static Maxwell’s equation is assumed for the whole domain $\Omega$:

$$\nabla \cdot D = 0$$  \hspace{1cm} (17)

By using the divergence relationship and the Green’s integral formula, it results the general weak formulation of the periodic $\gamma$-harmonic problem:
The variational form

Find \((u, \phi)\) in \(V_\gamma^3(\Omega_m) \times V_\gamma(\Omega)\) (verifying the equipotential boundary condition in the electrode), such that for all \((v, \psi)\) in \(V_\gamma^3(\Omega_m) \times V_\gamma^3(\Omega)\), satisfying the zero equipotential boundary condition, we have:

\[
\int_{\Omega_m} \overline{S(v)} : T(u) \, d\Omega - \omega^2 \int_{\Omega_m} \rho \, \overline{v} \cdot u \, d\Omega \\
- \int_{\Omega} \overline{E(\psi)} \cdot (\varepsilon \overline{S(u)} + \varepsilon E(\phi)) \, d\Omega \\
- \int_{\Gamma_d} \overline{v} \cdot (T(u) \cdot n) \, d\Gamma - \int_{\Gamma_u \cup \Gamma_d} \overline{\psi} (D(\phi) \cdot n) \, d\Gamma = 0 \quad (18)
\]

With, \(V_\gamma(\Omega)\) is the mathematical space of \(L^2(\Omega)\) with the derivative in \(L^2(\Omega)\) satisfying \(\gamma\)-harmonic periodic boundary conditions.
The $\gamma$-harmonic periodic boundary trick

Let us first define $\varphi_\gamma (x) = e^{-j2\pi \gamma \frac{x}{p}}$, $\varphi_\gamma (x)$ is a $\gamma$-harmonic periodic function satisfying:

$$\varphi_\gamma (x + p) = e^{-j2\pi \gamma} \varphi_\gamma (x) \tag{19}$$

We just do the change of variable:

$$\begin{align*}
  u(x, y) &= \varphi_\gamma (x) u^\diamond (x, y) \\
  \phi(x, y) &= \varphi_\gamma (x) \phi^\diamond (x, y)
\end{align*} \tag{20}$$

Where $u^\diamond (x)$ and $\phi^\diamond (x)$ are $p$-periodic functions.

The main idea is to define a new differential operator $\nabla_\gamma$ by:

$$\nabla_\gamma u^\diamond = \nabla (\varphi_\gamma u^\diamond) = \varphi_\gamma \nabla u^\diamond + \varphi_\gamma' u^\diamond \tag{21}$$

Because the physical fields $E, D, T$, and $S$ are expressed using partial derivative of $u$, and $\phi$, it is possible to define the operators $E_\gamma (\phi^\diamond) = E (\varphi_\gamma \phi^\diamond)$, $D_\gamma (\phi^\diamond) = D (\varphi_\gamma \phi^\diamond)$, $T_\gamma (u^\diamond) = T (\varphi_\gamma u^\diamond)$, ....
The new variational form with period BC.

Find \((u^\diamond, \phi^\diamond)\) in \(V_1^3(\Omega_m) \times V_1(\Omega)\) (verifying the equipotential boundary condition), such that for all \((v^\diamond, \psi^\diamond)\) in \(V_1^3(\Omega_m) \times V_1^3(\Omega)\), satisfying the zero equipotential boundary condition), we have:

\[
\int_{\Omega_m} S_\gamma(v^\diamond) : T_\gamma(u^\diamond) \, d\Omega - \omega^2 \int_{\Omega_m} \rho \overline{v^\diamond} \cdot \overline{u^\diamond} \, d\Omega
\]

\[
- \int_{\Omega} E_\gamma(\psi^\diamond) \cdot (eS_\gamma(u^\diamond) + \varepsilon E_\gamma(\phi^\diamond)) \, d\Omega
\]

\[
- \int_{\Gamma_d} \overline{\varphi_\gamma v^\diamond} \cdot (T_\gamma(u^\diamond) \, n) \, d\Gamma - \int_{\Gamma_u \cup \Gamma_d} \overline{\varphi_\gamma \psi^\diamond}(D_\gamma(\phi^\diamond) \cdot n) \, d\Gamma = 0 \quad (22)
\]

Where, \(V_1(\Omega)\) is the mathematical space of \(L^2(\Omega)\) with derivative in \(L^2(\Omega)\) satisfying \(p\)-periodic boundary conditions.
We have to modelize the following term:

\[- \int_{\Gamma_d} \overline{\varphi \gamma v^\diamond} \cdot (T_{\gamma} (u^\diamond) \mathbf{n}) \, d\Gamma - \int_{\Gamma_u \cup \Gamma_d} \overline{\varphi \gamma \psi^\diamond} (D_{\gamma} (\phi^\diamond) \cdot \mathbf{n}) \, d\Gamma, \quad (23)\]

also called border terms.
First from (23), let us look at the boundary integral $A_{\Gamma_u}$, at the interface $\Gamma_u$ of the semi-infinite dielectric semi-space.

\[A_{\Gamma_u} = \int_{\Gamma_u} \overline{\varphi \gamma \psi^\diamond} D_{\gamma} (\phi^\diamond) \cdot \mathbf{n} \, d\Gamma \quad (24)\]
The elementary coefficients to compute are for all finite element basic functions \( w^\diamond_i \) introduce in (??), only for node \( i \in \mathcal{N}_u \) the set of node on \( \Gamma_u \).

\[
\forall (i, j) \in \mathcal{N}_u^2, \quad (A_{\Gamma_u})_{ij} = -\varepsilon d \int_{\Gamma_u} \overline{\varphi_\gamma w^\diamond_i} \partial_n (\varphi_\gamma w^\diamond_j) \, d\Gamma \tag{25}
\]

According [2], it is possible to expand, at the interface \( \Gamma_u \) the \( \gamma \)-harmonic periodic \( \varphi_\gamma w^\diamond_j \) into the Floquet’s basis function

\[
f_m (x, y) = e^{-2\pi(j(m+\gamma)x-|m+\gamma|(y-y_u))/p} = \varphi_\gamma (x) f^\diamond_m (x, y). \tag{26}
\]

where \( y_u \) is the \( y \) coordinate of \( \Gamma_u \).

\[
\varphi_\gamma (x) w^\diamond_j (x, y) = \sum_{m=-\infty}^{+\infty} c^j_m f_m (x, y) \tag{27}
\]

With the \( L^2(\Gamma_u) \) orthogonality of Fourier’s basis \( f^\diamond_m \), we have:

\[
c^j_m = \frac{1}{p} \int_{\Gamma_u} w^\diamond_j \overline{f^\diamond_m} \, d\Gamma, \tag{28}
\]
and on $\Gamma_u$ the normal derivative $\partial_n f_m(x, y) = \partial_y f_m(x, y)$ satisfies:

$$\partial_n f_m = -g_m f_m, \quad \text{with} \quad g_m = \frac{2\pi}{p} |\gamma + m|$$

(29)

Leading to the relationship:

$$\partial_n (\varphi_\gamma \omega^\diamond j) = - \sum_{m=-\infty}^{+\infty} c_m^j g_m f_m$$

(30)

Finally the term $(A_{\Gamma_u})_{ij}$ is

$$(A_{\Gamma_u})_{ij} = \frac{\varepsilon d}{p} \sum_{m=\infty}^{+\infty} g_m \int_{\Gamma_u} \overline{\omega^\diamond i} \phi^\diamond m d\Gamma \int_{\Gamma_u} \omega^\diamond j \phi^\diamond m d\Gamma$$

(31)

Run: BEM.edp
Outline

- No Linear Problem
  - Newton Method
  - Navier-Stokes
  - Variational Inequality
  - Ground water
  - Hyper elasticity equation
  - Periodic Surface Acoustic Waves Transducer Analysis
  - Phase change with Natural Convection
Phase change with Natural Convection

The starting point of the problem is Brainstorming session (part I) of the third FreeFem++ days in December 2011, 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. 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 $\theta_r$ (resp $\theta_l$) at the right and left boundaries.

The model is: find the field: the velocity $u = (u_1, u_2)$, the pressure $p$ and temperature $\theta$:

$$
\begin{align*}
\partial_t u + (u \nabla) u + \nabla \mu \nabla u + \nabla p &= -c_T e_2 & \text{in } \Omega_f \\
\nabla \cdot u &= 0 & \text{in } \Omega_f \\
\partial_t \theta + (u \nabla) \theta + \nabla k_T \nabla \theta &= \partial_t S(T) & \text{in } \Omega
\end{align*}
$$

(32)

Where $\Omega_f$ is the fluid domain and the solid domain is $\Omega_s = \Omega \setminus \Omega_f$. 

Las Palmas de Gran Canaria, 25-29 June 2018 F. Hecht et al. Cours FreeFem++, XVIII Spanish-French school 185 / 224
Phase change with Natural Convection

The enthalpy of the change of phase is given by the function \( S \); \( \mu \) is the relative viscosity, \( k_T \) the thermal diffusivity.

In \( \Omega_f = \{ x \in \Omega; \theta > \theta_f \} \), with \( \theta_m \) the melting temperature the solid has melt. We modeled, the solid phase as a fluid with huge viscosity, so:

\[
\mu = \begin{cases} 
\theta < \theta_f & \sim 10^6 \\
\theta \geq \theta_m & \sim \frac{1}{Re}
\end{cases}
\]

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

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

Las Palmas de Gran Canaria, , 25-29 June 2018 F. Hecht et al. Cours FreeFem++, XVIII Spanish-French school 186 / 224
The true device
We apply a fixed point algorithm for the phase change part (the domain $\Omega_f$ is fixed at each iteration) and a full no-linear Euler implicit scheme with a fixed domain for the rest. We use a Newton method to solve the non-linearity.

- if we don’t make mesh adaptation, the Newton method do not converge
- if we use explicit method diverge too,
- if we implicit the dependance in $\Omega_s$ the method also diverge.

This is a really difficult problem.
The finite element space to approximate $u_1, u_2, p, \theta$ is defined by

```plaintext
fespace Wh(Th, [P_2, P_2, P_1, P_1]);
```

We do mesh adaptation at each time step, with the following code:

```plaintext
Ph ph = S(T), pph=S(Tp);
Th= adaptmesh(Th, T, Tp, ph, pph, [u_1, u_2], err=errh,
hmax=hmax, hmin=hmax/100, ratio = 1.2);
```

This means, we adapt with all variables plus the 2 melting phases at time $n+1$ and $n$ and we smooth the metric with a ratio of 1.2 to account for the movement of the melting front.
the fixed point are implemented as follows

```c
real err=1e100, errp;
for(int kk=0; kk<2; ++kk) // 2 step of fixed point on \( \Omega_s \)
{ nu = nuT; // recompute the viscosity in \( \Omega_s, \Omega_f \)
  for(int niter=0; niter<20; ++niter) // newton loop
  { BoussinesqNL;
    err = u1w[].linfty;
    cout << niter << " err\_NL\_" << err << endl;
    u1[] -= u1w[];
    if(err < tolNewton) break; // convergence ..
  }
}
```
The linearized problem

\[
\text{problem} \quad \text{BoussinesqNL([u1w,u2w,pw,Tw],[v1,v2,q,TT])} \\
= \int2d(Th) \left( \\
[u1w,u2w,Tw]'*[v1,v2,TT]*cdt \\
+ UgradV(u1,u2,u1w,u2w,Tw)' *[v1,v2,TT] \\
+ UgradV(u1w,u2w,u1,u2,T)' *[v1,v2,TT] \\
+ (Grad(u1w,u2w)'*Grad(v1,v2)) * nu \\
+ (Grad(u1,u2)'*Grad(v1,v2)) * dnu* Tw \\
+ cmT*T*w2 + grad(Tw)'*grad(TT)*kT \\
- div(u1w,u2w)*q -div(v1,v2)*pw - eps*pw*q \\
+ dS(T)*Tw*TT*cdt \right) \\
- \int2d(Th) \left( \\
[u1,u2,T]'*[v1,v2,TT]*cdt \\
+ UgradV(u1,u2,u1,u2,T)' *[v1,v2,TT] \\
+ (Grad(u1,u2)'*Grad(v1,v2)) * nu \\
+ cmT*T*v2 - eps*p*q + grad(T)'*grad(TT)*kT \\
- div(u1,u2)*q -div(v1,v2)*p \\
+ S(T)*TT*cdt - [u1p,u2p,Tp]'*[v1,v2,TT]*cdt \\
- S(Tp)*cdt*TT) \\
+ \text{on}(1,2,3,4, \ u1w=0,u2w=0)+\text{on}(2,Tw=0)+\text{on}(4,Tw=0) ;
\]
The parameters of the computation


θ_m = 0, Re = 1, S_{te} = 0.045, P_r = 56.2, R_a = 3.27 \times 10^5 , \theta_l = 1, \theta_r = -0.1 \text{ so in this case } cmT = c_T = -R_a / P_r , kT = k_T = 1 / P_r, \epsilon_s = 10^{-6}, \text{ time step } \delta t = 10^{-1}, cdT = 1 / \delta t, \text{ at time } t = 80 \text{ and we get a good agreement with the article.}
Phase change with Natural Convection

So now, a real problem, get the physical parameter of the real experiment.
Run:Orange-Newton.edp Run:Buoyancy-driven-cavite-eau-valide.edp
Outline

1 Introduction
2 Tools
3 Academic Examples
4 Numerics Tools
5 MPI/Parallel
6 No Linear Problem
7 Exercises
8 Technical Remark on freefem++
Exercices

- An exercice: Oven problem
- An exercice: Min surface problem
- Heat equation with thermic resistance
- Benchmark: Navier-Stokes
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:

let call the $u_p$ the solution of

$$-\nabla \cdot K \nabla u_p = \sum_{i=0}^{5} p_i \cdot \chi_i \text{ in } \Omega$$

$$u + K \nabla u_p \cdot n = 0 \text{ on } \Gamma = \partial \Omega$$

where $\chi_i$ 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

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

[Image of oven mesh and characteristics function]
Some remark

build the mesh with multi border trick.

Xh[int] ur(6); // to store the 6 FE. functions Xh
FreeFem++ as only linear solver on sparse matrix by default, but in the lapack plugin you have access to full matrix solver (see examples++-load/lapack.edp) 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; // full matrix to sparse of or use of lapack
set(A,solver=CG); // set linear solver to the C.G.
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 \( \Omega_6 \) and 7 for the rest of \( \Omega \) and the label of \( \Gamma \) is 1.
My solution, build the 6 basics function $u_{e_i}$

```c
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

```cpp
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][];

Run: oven.edp
```
Exercises

- An exercise: Oven problem
- An exercise: Min surface problem
- Heat equation with thermic resistance
- Benchmark: Navier-Stokes
An exercise: Min surface problem

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

$$\arg \min J(u) = \int_{\Omega} \left\| \begin{pmatrix} 1 \\ 0 \\ \partial_x u \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ \partial_y u \end{pmatrix} \right\| d\Omega = \int_{\Omega} \sqrt{1 + (\partial_x u)^2 + (\partial_y u)^2} d\Omega$$

So the Euler-Lagrange equation associated to the minimization is:

$$\forall v/v|\Gamma = 0 : DJ(u)v = -\int_{\Omega} \frac{(\partial_x v \partial_x u + \partial_y v \partial_y u)}{\sqrt{1 + (\partial_x u)^2 + (\partial_y u)^2}} d\Omega = 0$$

So find the solution for $\Omega = [0, \pi]^2$ and $g(x, y) = \cos(2x) \times \cos(2y)$. by doing fixed point method, by using the Non Linear Conjugate gradient NLCG like in the example: algo.edp in examples++-tutorial, IPOPT interface, or Newton method.
Fixed Point algorithm

So simple:

```c
int nn=10;
mesh Th=square(nn,nn);
fespace Vh(Th,P1); Vh u=0,up,v;
func g = cos(pi*x)*cos(2*pi*y);
for(int i=0; i< 100; ++i)
{
    up=u; // set the previous value
    solve Pb(u,v) = int2d(Th)( (dx(u)*dx(v) + dy(u)*dy(v) ) /sqrt( 1+ (dx(up)*dx(up) + dy(up)*dy(up) ) ))
        +on(1,2,3,4,u=g);
    real area = int2d(Th)( sqrt(1+ (dx(u)*dx(u) + dy(u)*dy(u) ) ));
    real err= sqrt(int2d(Th)( (u-up)^2)); // Error L2
    cout << i << " surface = " << area << " err L2 = " << err << "endl;
    plot(u, dim=3,fill=3, cmm=i+" area="+area+" err= "+err);
    if(err<1e-5) break;
}
```

Run: minimal-surf-fixePoint.edp
Example of use of NLCG function:

```cpp
func real J(real[int] & xx) // the functional to minimized
{
    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 functional
{
    // 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);
```

Useful operator on array `real[int]`

```cpp
real[int] a(10),b(10);
...
 a = b ? 1. : 0 ; // a[i] = 1 if b[i] else a[i]=0. \forall i
```

To see the 3D plot of the surface

```cpp
plot(u,dim=3);
```
a solution with NLCG: first functional

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

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

func real[int] dJ(real[int] & xx)  // the grad of the J
{ Vh u; u[] = xx;  // to set FE.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
```
Solution 1:

Vh u=G;
verbosity=5; \qquad // to see the residual
int conv=NLCG(dJ,u[],nbiter=500,eps=1e-5);
cout << " the surface =" << J(u[]) << endl;
\qquad // so see the surface un 3D
plot(u,dim=3);
Run:minimal-surf.edp
Exercises

- An exercise: Oven problem
- An exercise: Min surface problem
- Heat equation with thermic resistance
- Benchmark: Navier-Stokes
Heat equation with thermic resistance

let $\Omega$ be a domain cut with internal boundary $\Gamma_0$ in 2 sub-domain $\Omega_i, (i = 1, 2)$

We have Heat equation (Poisson) on $\Omega$, but on $\Gamma_0$ we have a jump $[u]$ on the temperature $u$ proportional to the temperature flux which is continue

So the equation to solve is:

Find $u$ such that $u|_{\Omega_i} \in H(\Omega_i)$ for $i = 1, 2$ and

$$-\nabla \kappa \nabla u = f_i, \quad \text{in } \Omega_i$$

$$\alpha[u] - \kappa \nabla u \cdot n = 0, \quad [\kappa \nabla u \cdot n] = 0, \quad \text{on } \Gamma_0$$

+ external boundary condition on $\partial\Omega$.

For the test take:

$L = 3$, $\Omega = [-L, L] \times [0, 1]$, $\Gamma_0 = \{ \sin(\pi y)/5, y \in [0, 1] \}$, take $\kappa = i$ in $\Omega_i$.

The external boundary condition on $\partial\Omega$ are: $\kappa \nabla u \cdot n = 0$ on upper and lower boundary, $u = 0$ at the left part, $u = 1$ at the right part.
Heat equation with thermic resistance

Method 1: Solve 2 coupled problems and use the block matrix tools to defined the linear system of the problem.
Exercises

- An exercice: Oven problem
- An exercice: Min surface problem
- Heat equation with thermic resistance
- Benchmark: Navier-Stokes
An exercice: Navier-Stokes

Try to make the 2d benchmark of:
http://www.mathematik.tu-dortmund.de/lsiii/cms/papers/
SchaeferTurek1996.pdf

The mesh can be set:

```cpp
int n=15; // parameter ...
real D=0.1, H=0.41;
real cx0 = 0.2, cy0 = 0.2; // center of cyl.
real xa = 0.15, ya=0.2, xe = 0.25,ye =0.2; // point for pressure..
```

```cpp
border fr1(t=0,2.2){x=t; y=0; label=1;}
border fr2(t=0,H){x=2.2; y=t; label=2;}
border fr3(t=2.2,0){x=t; y=H; label=1;}
border fr4(t=H,0){x=0; y=t; label=1;}
border fr5(t=2*pi,0){x=cx0+D*sin(t)/2; y=cy0+D*cos(t)/2; label=3;}

mesh Th=buildmesh(fr1(5*n)+fr2(n)+fr3(5*n)+fr4(n)+fr5(-n*3));
plot(Th, wait=1);
```
Technical Remark on freefem++

- compilation process
  - Plugin
  - Plugin to read image
  - Plugin of link code through a pipe
  - FreeFem++ et C++ type
For Windows see http://www.freefem.org/ff++/windows.php
For Linux see see http://www.freefem.org/ff++/linux.php
For Mac OS see see http://www.freefem.org/ff++/macosx.php
to add your C++ function in FreeFem++.
First, like in cooking, the first true difficulty is how to use the kitchen.
I suppose you can compile the first example for the examples++-load

numerrar11:FH-Seville hecht# ff-c++ myppm2rnm.cpp
...
add tools to read pgm image
Technical Remark on freefem++

- compilation process
- Plugin
  - Plugin to read image
  - Plugin of link code through a pipe
  - FreeFem++ et C++ type
Dynamics Load facility

Or How to add your C++ function in FreeFem++.
First, like in cooking, the first true difficulty is how to use the kitchen.
I suppose you can compile the first example for the `examples++-load`

`numercal11: FH-Seville hecht# ff-c++ myppm2rnm.cpp`

`...`

add tools to read `pgm` image
8 Technical Remark on freefem++
   - compilation process
   - Plugin
   - Plugin to read image
   - Plugin of link code through a pipe
   - FreeFem++ et C++ type
The interesting code

```cpp
#include "ff++.hpp"
typedef KNM<double> * pRnm; // real[int,int] array variable type
typedef KN<double> * pRn; // real[int] array variable type
typedef string ** string; // the ff++ string variable type

pRnm read_image( pRnm const & a, const pstring & b); // the function to read image

pRn seta( pRn const & a, const pRnm & b)
{ *a=*b;
  KN_<double> aa=*a;
  return a;}

void Init()
{ // the link with FreeFem++
  TheOperators->Add("<-",
    new OneOperator2_<KNM<double> *, KNM<double> *, string*>(&read_image) );
  TheOperators->Add("=",
    new OneOperator2_<KN<double> *, KN<double> *, KNM<double>* >(seta));
}

LOADFUNC(Init); // to call Init Function at load time

Remark, TheOperators is the ff++ variable to store all world operator, Global is to store function.
```
Technical Remark on freefem++

- compilation process
- Plugin
- Plugin to read image
- Plugin of link code through a pipe
- FreeFem++ et C++ type
How to extend

A true simple example How to make dynamic gnuplot
Idea: use a pipe to speak with gnuplot the C code :

```c
FILE * gp = popen("gnuplot");
for( double f=0; f < 3.14; f += 0.01)
    fprintf(gp,"plot sin(x+%f)\n",f);
```
To do this add a new constructor of ofstream in freefem++
A way to pass info between to code

Make a pipe, under unix (with a use of pstream tools)

```cpp
#include "ff++.hpp"
#include "pstream.h"
typedef redb::pstream pstream;
typedef std::string string;
static pstream ** pstream_init(pstream **const & p, string * const & a)
{ *p = new pstream(a->c_str());
  return p;);

void inittt()
{
  // add new pointer type * pstream
  Dcl_TypeandPtr<pstream*>(0,0,::InitializePtr<pstream*>,::DeletePtr<pstream*>);
  // add cast operation to make std iostream read and write
  atype<istream* >() -> AddCast( new E_F1_funcT<istream*,pstream**>(UnRef<istream* >));
  atype<ostream* >() -> AddCast( new E_F1_funcT<ostream*,pstream**>(UnRef<ostream* >));
  // the constructor from a string.
  TheOperators->Add("<-",new OneOperator2_<pstream**,pstream**,string*>(pstream_init) );
  // add new keyword type pstream
  zzzfff->Add("pstream", atype< pstream ** >());
}
LOADFUNC(inittt);
```

a small test: Run:gnuplot.edp
Outline

8 Technical Remark on freefem++
  - compilation process
  - Plugin
  - Plugin to read image
  - Plugin of link code through a pipe
  - FreeFem++ et C++ type
FreeFem++ et C++ type

The tools to add a operator with 2 arguments:

```cpp
OneOperator2_<returntype ,typearg1 ,typearg2>(& thefunction ));
returntype thefunction(typearg1 const &, typearg2 const &)
```

To get the C++ type of all freefem++ type, method, operator, just do in examples++-tutorialdirectory

```
c++filt -t < lestables
    Cmatrix 293 Matrice_Creuse<std::complex<double> >
    R3 293 Fem2D::R3
    bool 293 bool*
    complex 293 std::complex<double>**
    element 293 (anonymous namespace)::lgElement
    func 294 C_F0
    ifstream 293 std::basic_istream<char, std::char_traits<char> >
    int 293 long*
    matrix 293 Matrice_Creuse<double>
    mesh 293 Fem2D::Mesh**
    mesh3 293 Fem2D::Mesh3**
    ofstream 293 std::basic_ostream<char, std::char_traits<char> >
    problem 294 Problem
    real 293 double*
    solve 294 Solve
    string 293 std::basic_string<char, std::char_traits<char>, std::allocator<char> >
    varf 294 C_args
    vertex 293 (anonymous namespace)::lgVertex
```
FreeFem++ Triangle/Tet capability

```cpp
// soit T un Element de sommets A, B, C ∈ R²
// ------------------------
// number of vertices of triangle (here 3)
const Element::Vertex & V = T[i];
// the vertex i of T (i ∈ {0, 1, 2})
// mesure of T
double a = T.mesure();
// edge vector
Rd AB = T.Edge(2);
Rd hC = T.H(2);
// gradient of 2 base function
R l = T.lenEdge(i);
// length of i edge oppose of i
(Selection) T;
// label of T (region number)
R2 G(T(R2(1./3, 1./3)));
// The barycentre of T in 3d
```

Las Palmas de Gran Canaria, 25-29 June 2018 F. Hecht et al.
Cours FreeFem++, XVIII Spanish-French school 223 / 224
FreeFem++ Mesh/Mesh3 capability

Mesh Th("filename"); // read the mesh in "filename"
Th.nt; // number of element (triangle or tet)
Th.nv; // number of vertices
Th.neb or Th.nbe; // number of border element (2d) or (3d)
Th.area; // area of the domain (2d)
Th.peri; // length of the border

typedef Mesh::Rd Rd; // R2 or R3
Mesh2::Element & K = Th[i]; // triangle i, int i ∈ [0, nt]
Rd A=K[0]; // coor of vertex 0 of triangle K
Rd G=K(R2(1./3,1./3)); // the barycentre de K.
Rd DLambda[3]; // compute the 3 ∇λ^K for i = 0, 1, 2
K.Graddlambda(DLambda); // vertex j, int j ∈ [0, nv]
Mesh::Vertex & V = Th(j); // border element l ∈ [0, nbe]
Mesh::BorderElement & BE=th.be(l); // coord of vertex 1 on Seg BE
Rd B=BE[1]; // middle of BE.
Rd M=BE(0.5);
int j = Th(i,k); // global number of vertex k ∈ [0, 3] of tria. i ∈ [0, nt]
Mesh::Vertex & W=Th[i][k]; // vertex k ∈ [0, 3] of triangle i ∈ [0, nt]

int ii = Th(K); // number of triangle K
int jj = Th(V); // number of triangle V
int ll = Th(BE); // number of Seg de bord BE
assert( i == ii && j == jj ); // check.