Numerical modeling of Geophysical Flows by Finite Element techniques with FreeFem++

F. Hecht

Laboratoire Jacques-Louis Lions

Université Pierre et Marie Curie

Paris, France

with O. Pironneau

http://www.freefem.org

mailto:hecht@ann.jussieu.fr

With the support of ANR (French gov.) ANR-07-CIS7-002-01

http://www.freefem.org/ff2a3/ http://www-anr-ci.cea.fr/

PLAN

- Some Fluids equation
- The equation potentiel flow.
- The Variational formulation.
- The FEM in few minutes.
- How to solve this equation in FreeFem++
- Introduction `Freefem++`
- some syntaxe
- The potentiel flow equation in 2d,3d
- Mesh generation
- Poisson equation with matrix
- Variational/Weak form (Matrix and vector)
- Time dependent problem
- Stokes variational Problem
- Navier-Stokes (characteristic method)
- Navier-Stokes (Newton method)

http://www.freefem.org/

Basic fluids equations

First we suppose the fluids are in a domain $\Omega \subset \mathbb{R}^d$ where $d = 2$ or 3 is the dimension of the physical space.

NOTATION :

$$\boldsymbol{x} = (x_i)_{i=1}^d \in \mathbb{R}^d, \qquad \partial_i u = \frac{\partial u}{\partial x_i}, \qquad \boldsymbol{a}.\boldsymbol{b} = \sum_i a_i b_i, \qquad \nabla u = (\partial_i u)_{i=1}^d.$$

In lot of the case the fluid are incompressible so if $\boldsymbol{u} = (u_i)_{i=1}^d$ is the velocity field of the fluid then $\nabla.\boldsymbol{u} = 0 = \sum_{i=1}^d \partial_i u_i$.

If we suppose the fluid are irrotational, we get the existence of a potential field ϕ such that $\boldsymbol{u} = \nabla\phi = (\partial_i \phi)_{i=1}^d$.

By mixed the both equations we get the potentiel flow equation (also call Poisson equation) :

$$-\Delta\phi = -\nabla.\nabla\phi = 0 \tag{1}$$

where $\Delta\phi = \sum_{i=1}^d \partial_{ii}^2 \phi$.

The boundary condition on the border Γ of Ω of Poisson equation can be

ϕ given (call Dirichlet boundary condition)

$\alpha\phi + \partial_{\boldsymbol{n}}\phi$ given (call Robin or Fourier boundary condition if $\alpha > 0$ and Neumann if $\alpha = 0$), where $\boldsymbol{n}$ is the outer unit normal of Γ and $\partial_{\boldsymbol{n}} = \boldsymbol{n}.\nabla$.

Around a wing Execute Domain-Naca012.edp

In a lac Execute Domain-Leman.edp

Poisson equation, weak form

Let a domain Ω with a partition of $\partial\Omega$ in Γ_d, Γ_r.

Find ϕ a solution in such that :

$$-\Delta\phi = f \text{ in } \Omega, \quad \phi = g_d \text{ on } \Gamma_d, \quad \alpha\phi + \frac{\partial\phi}{\partial\vec{n}} = g_n \text{ on } \Gamma_r \tag{2}$$

Remaind Integration by part also call Stokes formula :

$$-\int_\Omega (\nabla.\boldsymbol{u})v \, \mathrm{d}\Omega = \int_\Omega \boldsymbol{u}.\nabla v \, \mathrm{d}\Omega - \int_\Gamma \boldsymbol{u}.\boldsymbol{n}v \, \mathrm{d}\Gamma$$

Denote $V_g = \{\psi \in H^1(\Omega)/\psi_{|\Gamma_d} = g\}$.

The Basic variationnal formulation after multiply by ψ integrated and integrated by part is : find $\phi \in V_{g_d}(\Omega)$, such that

$$\int_\Omega \nabla\phi.\nabla\psi = \int_\Omega f\psi + \int_\Gamma \frac{\partial\phi}{\partial n}\psi, \quad \forall\psi \in V_0(\Omega) \tag{3}$$

Poisson equation, weak form

So the border term in green become

$$+\int_{\Gamma}\frac{\partial\phi}{\partial n}\psi = +\int_{\Gamma_r} g_n\psi - \int_{\Gamma_r}\alpha\phi\psi$$

and the equivalent problem is

Find $\phi \in V_{g_d}(\Omega)$, such that

$$\int_{\Omega}\nabla\phi.\nabla\psi + \int_{\Gamma_r}\alpha\phi\psi = \int_{\Omega} f\psi + \int_{\Gamma_r} g_n\psi, \quad \forall\psi \in V_0(\Omega) \tag{4}$$

Poisson equation, weak full Newann BC

Remark, if $\Gamma_r = \emptyset$ and $\alpha = 0$ then ϕ is defined only through derivative, so ϕ is defined to a constant,to fixe the constant we add the constraint $\int_\Omega \phi = 0$.
Now the system is not square, and if we put $\psi = 1$ in (4) we get
$\int_\Omega f + \int_\Gamma g_n = 0$

We can make solve a small perturbated problem with a small positive ε :

Find $\phi_\varepsilon \in H^1(\Omega)$

$$\int_\Omega \nabla\phi_\varepsilon.\nabla\psi + \varepsilon\phi_\varepsilon\psi \, d\Omega = \int_\Omega f\psi \, d\Omega + \int_\Gamma g_n\psi \, d\Gamma, \quad \forall\psi \in H^1(\Omega) \qquad (5)$$

If $\int_\Omega f + \int_\Gamma g_n = 0$, we can prove :

$$\lim_{\varepsilon\to 0} \phi_\varepsilon = \phi, \qquad \text{and} \qquad \int_\Omega \phi_\varepsilon \, d\Omega = 0$$

Finite Element method

Construct a finite dimensional space V_h to approach the functional space $H^1(\Omega)$.

Where h is a parameter which theoretically go to zero (the mesh size).

To build V_h first, we build a triangular mesh $\mathcal{T}_h$ of Ω if $d = 2$ (example Around a wing Execute Domain-Naca012.edp)

or if $d = 3$ we build a tetrahedral mesh (exemple a Execute Cube.edp)

So a mesh $\mathcal{T}_h$ is a "partition" of Ω_h in set of element K (triangle in 2d, or resp. tetraedra in 3d), and and set of border element (segment in 2d or resp. triangle in 3d). We have

$$\overline{\Omega_h} = \bigcup_{K \in \mathcal{T}_h} \overline{K}, \qquad \text{and } K' \cap K = \emptyset \text{ if } K' \neq K \in \mathcal{T}_h$$

Finite Element method/ II

Let call $P_k(K)$ is the space of polynomials function of degree $\leq k$ on K.

So now the simple finite element space are : for each $k > 0$ positive integer we have,

$$V_h^k = \{v \in \mathsf{C}^0(\Omega_h)/\forall K \in \mathcal{T}_h, v_{|K} \in P_k(K)\} \tag{6}$$

And for the $k = 0$ the function are constant on K, the function can not be continuous, so the definition is

$$V_h^0 = \{v \in L^2(\Omega_h)/\forall K \in \mathcal{T}_h, v_{|K} \in P_0(K)\} \tag{7}$$

Finite Element method/ III

It easy to see that it is finite dimensional vectorial space.

We denote $w^i, i \in \mathcal{N}_h^k$ the basis function of this space V_h^k , where $\mathcal{N}_h^k$ is the set of degrees of freedom associated to space V_h^k.

We have the following unique decomposition :

$$u_h \in V_h^k, \qquad u_h = \sum_{i \in \mathcal{N}_h^k} u_i w^i$$

and the interpolation operator is defined by

$$\forall u \in \mathcal{C}^0(\Omega_h), \qquad \mathcal{I}_j^k u = \sum_{i \in \mathcal{N}_h^k} \sigma_i(u) w^i$$

where σ_i are a linear form associated to the degree of freedom which given the value of the degree of freedom.

For V_h^1, you can easily prove that : $\mathcal{N}_h^1$ is the set of vertices of $\mathcal{T}_h$ and $\sigma_i(u) = u(q_i)$ (remaind u is a function), where q_i is the coordinate of vertex i.

Finite Element method/ IIII

To get a approximation of (4) Find $\phi \in V_{g_d}(\Omega)$, such that

$$\int_\Omega \nabla\phi.\nabla\psi + \int_{\Gamma_r} \alpha\phi\psi = \int_\Omega f\psi + \int_{\Gamma_r} g_n\psi, \quad \forall\psi \in V_0(\Omega) \tag{8}$$

We just replace the space V_g by the corresponding finite element space $V_{h\,g}$ where $V_{h\,g}$

First defined $\mathcal{N}_{0h}$ the subset $\mathcal{N}_h$ with are not the border Γ_d, (c-a-d : If the support of σ_i is include in Γ_d then i is not in $\mathcal{N}_{0h}$) and then we can compute $\sigma_i(g_d)$ so the definition $V_{g\,h}$ is :

$$V_{h\,g} = \{\psi_h \in V_h \quad : \quad \forall i \in \mathcal{N}_h \setminus \mathcal{N}_{0\,h} \quad \sigma_i(\psi_h) = \sigma_i(g)\}$$

The discrete problem is :
Find $\phi_h \in V_{h\,g_d}(\Omega)$, such that

$$\int_{\Omega_h} \nabla\phi_h.\nabla\psi_h + \int_{\Gamma_{he}} \alpha\phi_h\psi_h = \int_{\Omega_h} f\psi_h + \int_{\Gamma_{hr}} g_n\psi_h, \quad \forall\psi_h \in V_{h\,0}(\Omega) \tag{9}$$

Introduction to install, 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 ...

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

Fist install freefem++ from
`http://www.freefem.org/ff++/ftp//FreeFem++-3.8.exe`
and notepad++ from
`http://sourceforge.net/project/showfiles.php?group_id=95717&package_id=102072`‘

Configuration de Notepad++

Open Notepad++ and Enter F5
In the new window enter the command
FreeFem++ "$(FULL_CURRENT_PATH)"
Click on Save, and enter FreeFem++ in the box "Name", now choose
the short cut keyboard to launch directly FreeFem++ for example alt+shitt+R

To add the Color Syntax Compatible to FreeFem++ In Notepad++,
In Menu "Parameters"->"Configuration of the Color Syntax".
In the list "Language" select C++
Add "edp" in the field "add ext"
Select "INSTRUCTION WORD" in the list "Description" and in the field
"supplementary key word", cut and past the following list

```
P0 P1 P2 P3 P4 P5 P1dc P2dc P3dc P4dc P5dc RT0 RT1 RT2 RT3 RT4 RT5 macro plot int1d int2d
solve movemesh adaptmesh trunc checkmovemesh on func buildmesh square Eigenvalue min max
imag exec LinearCG NLCG Newton BFGS LinearGMRES
catch try intalledges jump average mean load savemesh convect abs
sin cos tan atan asin acos cotan sinh cosh tanh cotanh atanh asinh acosh pow
exp log log10 sqrt dx dy endl cout
```

Select "TYPE WORD" in the list "Description" and ... " "supplementary key word",
cut and past the following list

```
mesh real fespace varf matrix problem string border complex ifstream ofstream
```

Click on Save& Close. Now nodepad++ is configure.

Element of syntaxe 1/2

```
x,y,z , label, region    //    current coordinate, label for BC , region
N.x, N.y, N.z                                          //    normal vector
int i = 0;                                               //    an integer
real a=2.5;                                                //    a reel
bool b=(a<3.);
real[int]  array(10) ;                          //    a real array of 10 value
mesh Th; mesh3 Th3;                              //    a 2d mesh and a 3d mesh
int reg = Th(0,0).region;    //    get the region number of (0,0) in Th.
fespace Vh(Th,P2);                              //    a 2d finite element space;
fespace Vh3(Th3,P1);                            //    a 3d finite element space;
Vh u=x;                              //    a finite element function or array
real xmax= u[].max ;                           //    get x max bound of the domain
Vh3<complex> uc = x+ 1.i *y;   //    complex valued FE function or array
u(.5,.6,.7);                  //    value of FE function u at point (.5,.6,.7)
u[];                             //    the array associated to FE function u
u[][5];    //   6th value of the array ( numbering begin at 0 like in C)
```

Element of syntaxe 1/2

```
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
macro Grad(u) [dx(u),dy(u)]//   EOM
varf a([u1,u2,p],[v1,v2,q])=
 int2d(Th,reg)(  Grad(u1)'*Grad(v1) +Grad(u2)'*Grad(v2)   //    reg is a
                    -div(u1,u2)*q -div(v1,v2)*p)     //    region number
         +on(1,2)(u1=g1,u2=g2);                  //    1,2 are label number

matrix A=a(V3h,V3h,solver=UMFPACK);
real[int] b=a(0,V3h);
u[] =A^-1*b;                                                    //
func f=x+y;                                    //    a formal line function
func real g(int i, real a) { .....; return i+a;}
A = A + A'; A = A'*A  //    matrix operation (only one by one operation)
A = [ A,0],[0,A']];                                  //    Block matrix.
```

Element of syntaxe : Like in C

The key words are reserved

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

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

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

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

Element of syntaxe array tools

```
real[int]  a(5),b(5),c(5),d(5);
a = 1; b = 2;  c = 3;
a[2]=0;
d = ( a? b : c );          //    for i = 0, n-1 : d[i] = a[i]? b[i] : c[i] ,
cout << " d = ( a? b : c )  is " << d << endl;
d = ( a? 1 : c );        //    for i = 0, n-1: d[i] = a[i]? 1 : c[i] ,
d = ( a? b : 0 );        //    for i = 0, n-1: d[i] = a[i]? b[i] : 0 ,
d = ( a? 1 : 0 );         //    for i = 0, n-1: d[i] = a[i]? 0 : 1 ,
tab.sort ;                              //    sort the array tab (version 2.18)

int[int] ii(0:d.n-1);           //    set array ii to 0,1, ..., d.n-1
d=-1:-5;                       //    set d to -1,-2, .. -5
sort(d,ii);                             //    sort array d and ii in parallel

real[int] methods=[   //    ---- some methods norme, min max, dot product--
  a.l1 ,  a.l2 ,   a.linfty , a.sum  , a.max ,  a.min ,  (a'*a)
];
```

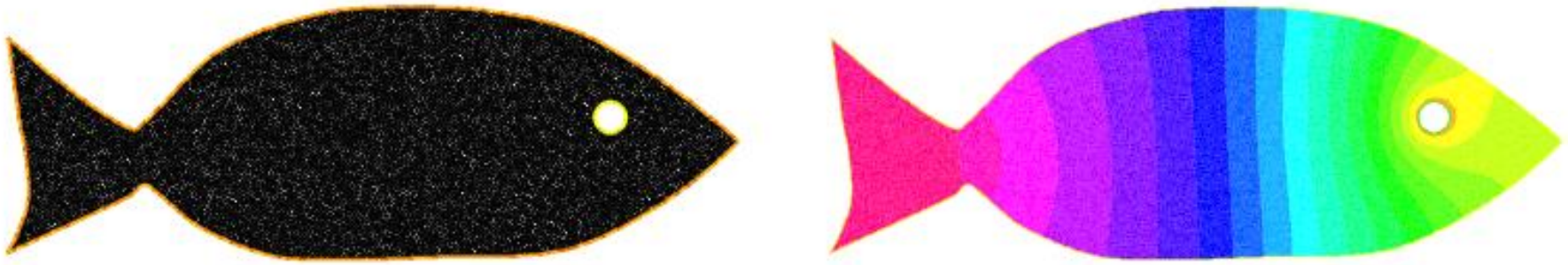
Laplace equation in FreeFem++

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

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

Vh phi,psi;
macro Grad(u) [dx(u),dy(u),dz(u)]                                    //   EOM
solve laplace(phi,psi,solver=CG) =
   int3d(Th)(  Grad(phi)'*Grad(psi)   )   -  int3d(Th) ( 1*psi)
  + on(2,phi=2);                                                     //    int on γ2
plot(phi,fill=1,wait=1,value=0,wait=1);
```

Laplace equation 2d / figure



Execute fish.edp Execute Laplace3d.edp Execute EqPoisson.edp

Build Mesh with tetgen or read mesh

```
include "MeshSurface.idp"                    //    tool for 3d surfaces meshes
mesh3 Th;
try {  Th=readmesh3("Th-hex-sph.mesh"); }                  //    try to read
catch(...) {                          //    catch an error to 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); //    "gluing"
                                                             //    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"); }
```

Build form a extern file mesh

```
mesh3 Th2("Th-hex-sph.mesh");
```

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

A cube with buildlayer (simple)

```
load "msh3"                                                    //    buildlayer
int nn=10;
int[int] rup=[0,2],   //    label of the upper face 0-> 2 (region -> label)
      rdown=[0,1],    //    label of the lower face 0-> 1 (region -> label)
      rmid=[1,1 ,2,1 ,3,1 ,4,1 ], //   4 Vert. faces: 2d label -> 3d label
      rtet[0,0];
real zmin=0,zmax=1;
mesh3 Th=buildlayers(square(nn,nn),nn,
                     zbound=[zmin,zmax],
                     reftet=rtet,
                     reffacemid=rmid,
                     reffaceup = rup,
                     reffacelow = rdown);
```

Execute Cube.edp

3D layer mesh of a Lac with buildlayer

```
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],
                     reffacemid=rmid,  reffaceup = rup,
                     reffacelow = rdown);                    //    label def
medit("lac",Th);
```

Execute Lac.edp Execute 3d-leman-mesh.edp

a 3d fish Mesh with buildlayer

```
func f=2*((0.1+(((x/3))*(x-1)*(x-1)/1+x/100))^(1/3.)-(0.1)^(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 axe1(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)+axe1(np/10)+hole(np/10)+axe2(8*np/10)+ queue(np/10);
plot( bord);                                          //    plot the border ...
mesh Th2=buildmesh(bord);                             //    the 2d mesh axi mesh
plot(Th2,wait=1);
int[int] l23=[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,reffacemid=l23);
```

Execute EqPoisson.edp

The plot of the Finite Basis Function (3d plot)

```
load "Element_P3"                                    //    load P3 finite element
mesh Th=square(3,3);                                 //    a mesh with 2 elements
fespace Vh(Th,P3);

Vh vi=0;
for (int i=0;i<vi[].n;++i)
 {
    vi[][i]=1;                                  //    def the i+1^th basis function

    plot(vi,wait=0,cmm=" v"+i,dim=3);
    vi[]=0;                                       //    undef i+1^th basis function

 }
```

Execute plot-fb.edp

Build Matrix and vector of problem

The 3d FreeFem++ code :

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

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

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

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

Remark on varf

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

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

To build matrix A from the bilinear part and Boundary conditions of the variational form $vb1$ of type `varf` do simply

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

To build the vector form linear part and Boundary conditions, we have $(u1 = u2 = 0)$ and just say

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

The boundary condition terms

- An "on" scalar form (for Dirichlet) : `on(1, u = g )`
  The meaning is for all degree of freedom i of this associated boundary, the diagonal term of the matrix $a_{ii} = tgv$ with the *terrible giant value* `tgv` (=10^{30} by default) and the right hand side $b[i] = "(\Pi_h g)[i]" \times tgv$, where the $"(\Pi_h g)g[i]"$ is the boundary node value given by the interpolation of g.
- An "on" vectorial form (for Dirichlet) : `on(1,u1=g1,u2=g2)` If you have vectorial finite element like `RT0`, the 2 components are coupled, and so you have : $b[i] = "(\Pi_h(g1, g2))[i]" \times tgv$, where Π_h is the vectorial finite element interpolant.
- a linear form on Γ (for Neumann in 2d)
  `-int1d(Th)( f*w)` or `-int1d(Th,3))( f*w)`
- a bilinear form on Γ or Γ_2 (for Robin in 2d)
  `int1d(Th)( K*v*w)` or `int1d(Th,2)( K*v*w)`.
- a linear form on Γ (for Neumann in 3d)
  `-int2d(Th)( f*w)` or `-int2d(Th,3)( f*w)`
- a bilinear form on Γ or Γ_2 (for Robin in 3d)
  `int2d(Th)( K*v*w)` or `int2d(Th,2)( K*v*w)`.

Fast method for Time depend Problem / formulation

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.

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

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

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).

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

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

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

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

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

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

Fast The Time depend Problem/ edp

```
 ...
Vh u0=fu0,u=u0;
```

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

```
varf vthermic (u,v)= int2d(Th)(u*v/dt + k*(dx(u) * dx(v) + dy(u) * dy(v)))
  +  int1d(Th,1,3)(alpha*u*v)  + on(2,4,u=1);
varf vthermic0(u,v) =   int1d(Th,1,3)(alpha*ue*v);
varf vMass (u,v)= int2d(Th)( u*v/dt)  + on(2,4,u=1);

real tgv = 1e30;
 A= vthermic(Vh,Vh,tgv=tgv,solver=CG);
matrix M= vMass(Vh,Vh);
```

Fast The Time depend Problem/ edp

Now, to build the right hand size we need 4 vectors.

```
real[int]  b0  = vthermic0(0,Vh);                    //   constant part of the RHS
real[int]  bcn = vthermic(0,Vh);   //   tgv on Dirichlet boundary node(!=0)
                        //    we have for the node i : i ∈ Γ24   ⇔   bcn[i] ≠ 0
real[int]  bcl=tgv*u0[];          //    the Dirichlet boundary condition part
```

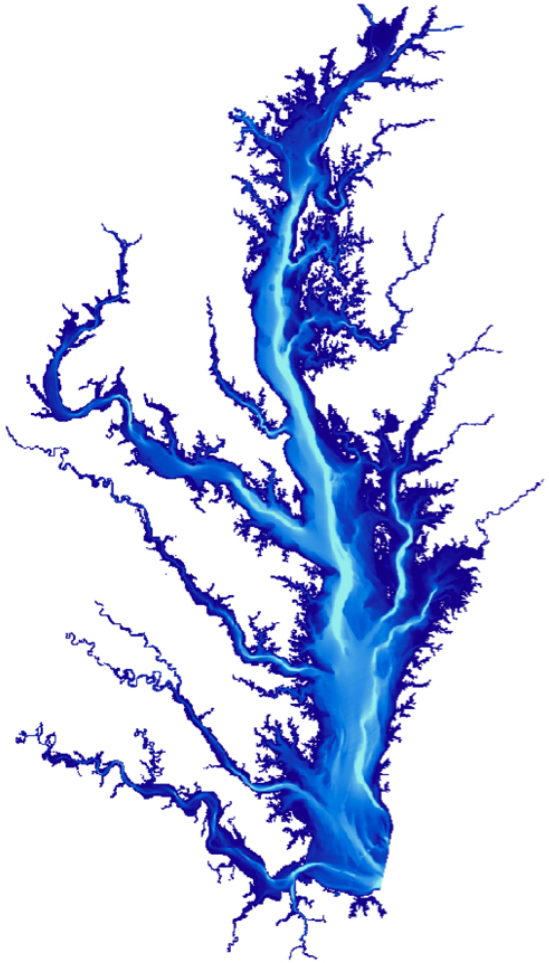
The Fast algorithm :

```
for(real t=0;t<T;t+=dt){
    real[int] b = b0 ;                                          //    for the RHS
    b += M*u[];                                 //    add the the time dependent part
    b = bcn? bcl  : b ;                 //    do ∀i: b[i] = bcn[i]? bcl[i] : b[i] ;
    u[] = A^-1*b;
    plot(u);
}
```

Execute Heat.edp

Some Idea to build meshes

The problem is to compute eigenvalue of a potential flow on the Chesapeake bay (Thank to Mme. Sonia Garcia, smg@usna.edu).



- Read the image in freefem, `adaptmesh` , `trunc` to build a first mesh of the bay and finally remove no connected component. We use : $\xi > 0.9||\xi||_\infty$ where ξ is solution of

$$10^{-5}\xi - \Delta\xi = 0 \quad \text{in } \Omega; \qquad \frac{\partial \xi}{\partial n} = 1 \quad \text{on } \Gamma.$$

  Remark, on each connect component ω of Ω, we have

$$\xi_{|\omega} \simeq 10^5 \frac{\int_{\partial\omega} 1}{\int_\omega 1}.$$

  Execute Chesapeake/Chesapeake-mesh.edp
- Solve the eigen value, on this mesh.
- Execute Chesapeake/Chesapeake-flow.edp

Eigenvalue/ Eigenvector example

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

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

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

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

Stokes equation

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

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

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

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

$$\forall \boldsymbol{v} \in H_0^1(\Omega)^d,\ \forall q \in L^2(\Omega)/\mathbb{R}, \quad \int_\Omega \nu\nabla\boldsymbol{u} : \nabla\boldsymbol{v} - p\nabla.v - q\nabla.u = \int_\Gamma (\nu\frac{\partial \boldsymbol{u}}{\partial \vec{n}} + p\vec{n}).v$$

Remark, The term is fundamental to change the type of boundary condition.

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

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

The compatibility condition for Stokes

Like with full Neumann boundary condition , we have here, compatibility.

Just plug $v = 0, q = 1$ in previous equation then we get :

$$\int_\Omega \nabla . \boldsymbol{u} = \int_\Gamma \boldsymbol{u}.\boldsymbol{n} = 0$$

.

This condition mush be true also a discrete level, otherwise the problem have no solution.

And it is not so easy to respect strongly this mathematical constraint, but generally, in a flow we do not what is a good boundary condition at outlet, a simple idea is put a Neumann like boundary condition (Green term is zero. $\int_{\Gamma_{out}} (\nu \frac{\partial \boldsymbol{u}}{\partial \vec{n}} + p\vec{n}).v = 0$). In this term p appear so the pressure is now not defined to a constant, and this compatibility problem disappear. Execute Stokes-Compatibily-Problem.edp

Stokes equation in FreeFem++

```
 ...  build mesh .... Th (3d) T2d ( 2d)
fespace VVh(Th,[P2,P2,P2,P1]);              //    Taylor Hood Finite element.
macro Grad(u) [dx(u),dy(u),dz(u)]                 //    EOM
macro div(u1,u2,u3) (dx(u1)+dy(u2)+dz(u3))              //   EOM
varf vStokes([u1,u2,u3,p],[v1,v2,v3,q]) =  int3d(Th)(
    Grad(u1)'*Grad(v1) +  Grad(u2)'*Grad(v2) +  Grad(u3)'*Grad(v3)
     - div(u1,u2,u3)*q - div(v1,v2,v3)*p + 1e-10*q*p )
 + on(1,u1=0,u2=0,u3=0)   +  on(2,u1=1,u2=0,u3=0);
matrix A=vStokes(VVh,VVh); set(A,solver=UMFPACK);
real[int] b= vStokes(0,VVh);
VVh [u1,u2,u3,p];       u1[]= A^-1 * b;

                                                  //    2d intersection of plot
fespace V2d(T2d,P2);                             //   2d finite element space ..
V2d ux= u1(x,0.5,y);  V2d uz= u3(x,0.5,y);  V2d p2= p(x,0.5,y);
plot([ux,uz],p2,cmm=" cut y = 0.5");
```

Execute Stokes3d.edpExecute Stokes-3d-leman.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

$$\begin{array}{rl} \frac{1}{\tau}(u^{n+1} - u^n \circ X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} & = 0, \\ \nabla \cdot u^{n+1} & = 0 \end{array} \tag{10}$$

The term $X^n(x) \approx x - u^n(x)\tau$ will be computed by the interpolation operator, or with convect operator (work form version 3.3)

The ff++ NSI 3d code

```
real alpha =1./dt;
varf vNS([uu1,uu2,uu3,p],[v1,v2,v3,q]) =
 int3d(Th)( alpha*(uu1*v1+uu2*v2+uu3*v3)
 + nu*(Grad(uu1)'*Grad(v1) +  Grad(uu2)'*Grad(v2) +  Grad(uu3)'*Grad(v3))
 - div(uu1,uu2,uu3)*q - div(v1,v2,v3)*p + 1e-10*q*p )
 + on(1,2,3,4,5,uu1=0,uu2=0,uu3=0)
 + on(6,uu1=4*(1-x)*(x)*(y)*(1-y),uu2=0,uu3=0)
 + int3d(Th)( alpha*(
    u1(X1,X2,X3)*v1  +   u2(X1,X2,X3)*v2 +   u3(X1,X2,X3)*v3 ));
```

or with convect tools change the last line by

```
 + int3d(Th,optimize=1)( alpha*convect([u1,u2,u3],-dt,u1)*v1
                        +alpha*convect([u1,u2,u3],-dt,u2)*v2
                        +alpha*convect([u1,u2,u3],-dt,u2)*v3);
```

The ff++ NSI 3d code/ the loop in times

```
A = vNS(VVh,VVh);   set(A,solver=UMFPACK); //   build and factorize matrix
real t=0;
for(int i=0;i<50;++i)
  { t += dt;  X1[]=XYZ[]-u1[]*dt;              //    set χ=[X1,X2,X3] vector
    b=vNS(0,VVh);                                    //    build NS rhs
    u1[]= A^-1 * b;                             //    solve the linear system
    ux= u1(x,0.5,y);  uz= u3(x,0.5,y);  p2= p(x,0.5,y);
    plot([ux,uz],p2,cmm=" cut y = 0.5, time ="+t,wait=0);  }
```

Execute NSI3d.edp

Newton Algorithm to solve nolinear Problem

The problem is find $u \in \mathbb{R}^n$ such that $F(u) = 0$ where $F : \mathbb{R}^n \mapsto \mathbb{R}^n$.

The algorithm is

1. choose $u_0 \in \mathbb{R}^n$, $i = 0$;
2. do
   (a) compute $w_i = DF(u_i)^{-1} F(u_i)$;
   (b) $u_{i+1} = u_i - w_i$;
   (c) $i = i + 1$;
   until $||w_i|| > \varepsilon$.

Solve incompressible Navier Stokes flow (3 Slides)

```
                                                //    define finite element space Taylor Hood.
fespace XXMh(th,[P2,P2,P1]);
XXMh [u1,u2,p], [v1,v2,q];

                                                                                  //    macro
macro div(u1,u2) (dx(u1)+dy(u2))                                                  //
macro grad(u1,u2) [dx(u1),dy(u2)]                                                 //
macro ugrad(u1,u2,v) (u1*dx(v)+u2*dy(v))                                          //
macro Ugrad(u1,u2,v1,v2) [ugrad(u1,u2,v1),ugrad(u1,u2,v2)]                        //

                                                              //    solve the Stokes equation
solve Stokes ([u1,u2,p],[v1,v2,q],solver=UMFPACK) =
    int2d(th)( ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
             +  dx(u2)*dx(v2) + dy(u2)*dy(v2) )
             + p*q*(0.000001)
             - p*div(v1,v2) - q*div(u1,u2)           )
  + on(1,u1=u1infty,u2=u2infty)
  + on(3,u1=0,u2=0);
real nu=1./100.;
```

the tangent PDE

```
XXMh [up1,up2,pp];                                          //    the previous solution...
varf    vNS ([u1,u2,p],[v1,v2,q]) =                                          //    the RHS
    int2d(th)(
              + nu * ( dx(up1)*dx(v1) + dy(up1)*dy(v1)
              + dx(up2)*dx(v2) + dy(up2)*dy(v2) )
              + pp*q*(0.000001)
              + pp*dx(v1)+ pp*dy(v2)
              + dx(up1)*q+ dy(up2)*q
              + Ugrad(up1,up2,up1,up2)'*[v1,v2]
             )
  + on(1,2,3,u1=0,u2=0)
;

varf    vDNS ([u1,u2,p],[v1,v2,q]) =                        //    Derivative (bilinear part
    int2d(th)(
              + nu * ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
              +  dx(u2)*dx(v2) + dy(u2)*dy(v2) )
              + p*q*1e-6  + p*dx(v1)+ p*dy(v2)
              + dx(u1)*q+ dy(u2)*q
              + Ugrad(u1,u2,up1,up2)'*[v1,v2]
              + Ugrad(up1,up2,u1,u2)'*[v1,v2]          )
  + on(1,2,3,u1=0,u2=0);
```

Nolinear Loop, to solve INS with Newton Method

```
for(int rre = 100; rre <= 100; rre *= 2)                    //    continuation on the reylnods
{   re=min(real(rre),100.);
    th=adaptmesh(th,[u1,u2],p,err=0.1,ratio=1.3,nbvx=100000,
                 hmin=0.03,requirededges=lredges);
    [u1,u2,p]=[u1,u2,p];                    //    after mesh adapt: interpolated old -> new th
    [up1,up2,pp]=[up1,up2,pp];
    real[int] b(XXMh.ndof),w(XXMh.ndof);
    int kkkk=3;
    for (i=0;i<=15;i++)                     //    solve steady-state NS using Newton method
    {  if  (i%kkkk==1)
         {  kkkk*=2;                                                //    do mesh adatpation
            th=adaptmesh(th,[u1,u2],p,err=0.05,ratio=1.3,nbvx=100000,hmin=0.01);
            [u1,u2,p]=[u1,u2,p];                          //    resize of array (FE fonction)
            [up1,up2,pp]=[up1,up2,pp];
            b.resize(XXMh.ndof); w.resize(XXMh.ndof); }
        nu =LL/re;                                                  //    set the viscsity
        up1[]=u1[];
        b = vNS(0,XXMh);
        matrix Ans=vDNS(XXMh,XXMh);                                 //    build sparse matrix
        set(Ans,solver=UMFPACK);
        w = Ans^-1*b;                                               //    solve sparse matrix
        u1[] -= w;
        if(w.l2<1e-4) break; }}
```

Execute cavityNewtow.edp