Variational Methods for Computational Fluid Dynamics
Année 2013 - 2014, X 2011.

Memento FreeFem++

1 Memento

- Comments
```
// A comment
```

- Variables: *type variables;* or *type variable = value;*
```
real mu = 0.0025; //viscosity
int Niter = 25; //Number of iterations
real[int] X = [1.,2.,3.]; // an array
```

- Functions *func type name_func(type1 variable1,··· ,typen variablen)*
```
func real bezier(real p0,real p1,real q1,real q2,real t)
{ return p0*(1-t)^3 + q1*3*(1-t)^2*t+q2*3*(1-t)*t^2+p1*t^3; }
```

- Meshes
  - Squares/rectangles
    ```
    mesh Th=square(L*Np,Np,[L*x,y]);
    ```
  - Borders
    ```
    border border1(t=0,1) {x=cos(t);y=sin(t); label = 1;};
    ```
  - Buildmesh
    ```
    mesh Th = buildmesh(±border1(15)···±bordern(20));
    ```
    Pay attention that the sign gives the direction in which the border is oriented, while the parameter in the parentheses gives the number of points used to discretize the border.
  - Saving the mesh
    ```
    savemesh(Th,"Th.msh");
    ```
  - Plotting the mesh
    ```
    plot(Th,ps="Th.eps",bw=1,wait=1);
    ```

- Finite element spaces
```
fespace Vh(Th,P1b), Xh(Th,P1);
```

- Finite element variables
```
Vh u = 0, v=x;
```

- Variational formulations
```
problem Laplace(u,tu)=
     int2d(Th)(dx(u)*dx(tu)+dy(u)*dy(tu))
```

- Dirichlet boundary conditions
```
     +on(1,3,5,u=0);
```

- Neumann boundary conditions
```
     -int1d(Th,3)(g*tu);
```

- Loops
```
for (int iter=0 ; iter<=Niter ; iter++)
     {
     bloc of instructions
     }
```

- Printing
```
cout << "Mean veloc= " << int1d(Th,6)(-uy)/(2*r0) << endl;
```

- Plotting the solution
```
plot([ux,uy],bw=1,coef=20,value=1,wait=0) ; //vectorfields
plot(uh,bw=1,wait=0) ; //scalars
plot(Tx,fill=1,grey=1,wait=0,value=0,nbiso=50,ps="fig.ps");

om = dy(ux)-dx(uy); //Vorticity
plot(om,fill=1,value=1,wait=1);
```

- Saving files
```
string fn="./RES/", fname ;
```
and then, in a time loop:
```
for (int iter=1000 ; iter<Niter+1000 ; iter++) {
 ...
fname = fn + "vv"+ iter + ".ps";
plot(Tx,fill=1,ps=fname);
...
}
```

- Convection of a variable
```
Vh vh = convect([pux,puy],-dt,uh));
```

2 Variational formulations

2.1 Stokes

```
problem Stokes([ux,uy,p],[tux,tuy,tp])=
```

- $\mu \int \nabla u : \nabla v$
```
     int2d(Th)(mu*(dx(ux)*dx(tux)+dy(uy)*dy(tuy)+
     dx(uy)*dx(tuy)+ dy(ux)*dy(tux)))
```

- $\mu \int (\nabla u + {}^t\nabla u) : \nabla v$

```
int2d(Th)(mu*(2*dx(ux)*dx(tux)+2*dy(uy)*dy(tuy)+
dx(uy)*dx(tuy)+ dy(ux)*dy(tux)+
dx(uy)*dy(tux)+ dy(ux)*dx(tuy)))
```

- $-\int p \text{ div } v - \int q \text{ div } u$

```
+int2d(Th)( - p*dx(tux) - p*dy(tuy) +
- tp*dx( ux) - tp*dy(uy))
```

- $-\int f \cdot v = 0$

```
+int2d(Th)( - f1*tux+f2*tuy
```

2.2 Navier-Stokes

```
problem Stokes([ux,uy,p],[tux,tuy,tp])=
```

- Eulerian (characterisctics)

```
int2d(Th)((ux*tux+uy*tuy)/dt)
+int2d(Th)(-convect([pux,puy],-dt,pux)*tux/dt
-convect([pux,puy],-dt,puy)*tuy/dt)
```

- Lagrangian
  First move the mesh

```
Th = movemesh(Th,[x+dt*ux,y+dt*uy]);
tmp=ux[]; pux=0; pux[]=tmp ;
tmp=uy[]; puy=0; puy[]=tmp ;
```

  and then solve with the following terms in the variational formulation:

```
int2d(Th)((ux*tux+uy*tuy)/dt)
+int2d(Th)(-(pux*tux+puy*tuy)/dt)
```

- ALE First move the mesh

```
MeshVeloc ; // Computes the ALE velocity [cx,cy]
Th = movemesh(Th,[x+dt*cx,y+dt*cy]); //Moves the mesh
tmp=ux[]; pux=0; pux[]=tmp ;//Update the variables
tmp=uy[]; puy=0; puy[]=tmp ;
tmp=cx[]; cx=0; cx[]=tmp ;
tmp=cy[]; cy=0; cy[]=tmp ;
```

  and then solve with the following terms in the variational formulation:

```
int2d(Th)((ux*tux+uy*tuy)/dt)
+int2d(Th)(-convect([pux-cx,puy-cy],-dt,pux)*tux/dt
-convect([pux-cx,puy-cy],-dt,puy)*tuy/dt)
```

  Notice that `tmp` is an array defined by

```
real[int] tmp(ux[].n);
```

3 Fluid-solid

3.1 Penalization

3.2 Constraints