An (incomplete) Introduction to FreeFem++

Nathaniel Mays

Wheeling Jesuit University

November 12, 2011
Model Problem - Stokes

\[-\Delta \mathbf{u} + \nabla p = \mathbf{f} \quad \text{in } \Omega \]
\[\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega\]

- Models viscous fluid flow.
- \(\mathbf{u}\) is the velocity vector \(<\,u_1,\,u_2\,>\).
- \(p\) is the pressure.
- \(\nabla \cdot \mathbf{u}\) is the incompressibility condition
Stokes variationally

Alternatively, if we multiply by a test function \((v \in X \text{ and } q \in Q)\) and integrate over the entire space, we end up with: find \((u, p)\) in \((X, Q)\) satisfying

\[
\int_{\Omega} \nabla u : \nabla v \, dx + \int_{\Omega} p \nabla \cdot v \, dx = \int_{\Omega} f \cdot v \, dx \quad \forall v \in X
\]

\[
\int_{\Omega} q \nabla \cdot u \, dx = 0 \quad \forall q \in Q
\]
Now the choice of discretization.

- Triangulate $\Omega$ into $T^h$ by some suitable method.
- Choose $X^h \subset X$ that’s finite dimensional.
- Choose $Q^h \subset Q$ that’s finite dimensional.
- $X^h$ and $Q^h$ need to satisfy the inf-sup condition (bad stuff happens otherwise).
- Typical elements are $P2$-$P1$ (Taylor-Hood) or $P1$bubble-$P1$ (Mini).
Stokes discretely

If we look at discrete function spaces for our test functions ($v^h \in X^h$ and $q^h \in Q^h$) and integrate over the entire space, we end up with: find $(u^h, p^h)$ in $(X^h, Q^h)$ satisfying

$$
\int_{\Omega} \nabla u^h : \nabla v^h \, dx + \int_{\Omega} p^h \nabla \cdot v^h \, dx = \int_{\Omega} f \cdot v^h \, dx \quad \forall v^h \in X^h
$$

$$
\int_{\Omega} q^h \nabla \cdot u^h = 0 \quad \forall q^h \in Q^h
$$
FreeFem++

- Free Finite Element Method software
- Platforms: Unix, Windows, OS X
- Built on C++, so many commands have the same calls
- Wonderful for first calculations

Outline:

- Declarations and structure
- Mesh generation and FE spaces
- Problem statements and programs
- Reports and output
- Future problems
- Parameter loops and post-processing


```c
int i, j, maxIt, one=1;

real error, reynolds=1.0e4;

string errorflag;
string solution=”testvariable”;

func f1=−2.0*y;
func real u1e (real t) {
return −cos(n*pi*x)*sin(n*pi*y)*exp(−2*n*n*pi*pi*t/tau);
}

real[int] isoval(10);
```
// Directories for saving data
string plotdir = "plots/";
string datadir = "data/";
string reportdir = "reports/";

// Identifying prefix that will appear in filenames
string problemPrefix = "channel";

// Create directories if they’re not already here (FreeFem will NOT automatically do this)
exec("mkdir -p " + plotdir);
exec("mkdir -p " + datadir);
exec("mkdir -p " + reportdir);
You can take advantage of FreeFem++'s meshing commands

```cpp
int meshpts=10;
mesh Th = square(meshpts,meshpts);
plot(Th);
```

Or stretch the square

```cpp
real xmin=-2.0,xmax = 5.0,ymin=-1.0,ymax=3.0;
int meshpts=30;
mesh Th = square(meshpts,meshpts,[xmin + (xmax-xmin)*x, ymin + (ymax - ymin)*y]);
plot(Th);
```
// Label 1 will denote no slip boundary conditions. Label 2 will denote parabolic BC.

border bottom(t=0,1) {x = ChLength*t; y = 0.0; label = 1;}
border right(t=0,1) {x = ChLength; y = t; label = 2;}
border top(t=0,1) {x = ChLength*(1.0-t); y = 1; label = 1;}
border left(t=0,1) {x = 0.0; y = (1.0-t); label = 2;}
border obstacle(t=0,2*pi) {x = ObsLength+ObsRadius*cos(t); y = 0.5+ObsRadius*sin(t); label = 1;}

mesh Th = buildmesh( bottom(ChLength*meshperunit) + right( meshperunit) + top(ChLength*meshperunit) + left(meshperunit) + obstacle(-2*meshperunit) );
savemesh(Th,datadir + problemPrefix + ".msh");
Our wonderful mesh
Exercise 1

- Create a domain with a rectangle with base 2 and height 3 and a semicircle (of radius 1) on the top.
- For the mesh, use approximately 10 mesh points per unit length (so 20 on the side of length 2, 31 for the semicircle).
- Plot the mesh.
// Defining velocity and pressure spaces.
fespace Xh(Th,P2);
fespace Qh(Th,P1);
// Initialize velocity and pressure functions
Xh u1,u2;
Qh p;
// Initialize velocity and pressure test functions
Xh v1,v2;
Qh q;
problem stokes([u1,u2,p],[v1,v2,q],solver=UMFPACK) =
  int2d(Th)(
    ( dx(u1) * dx(v1) + dx(u2) * dx(v2) + dy(u1) * dy(v1) + dy(u2) *
      dy(v2) )
    - p * ( dx(v1) + dy(v2) )
    - q * ( dx(u1) + dy(u2) )
    + PRESSURESTABILIZE * p * q
  )
  -int2d(Th)(
    f1*v1 + f2*v2
  )
  + on( 1, u1=0.0, u2=0.0)
  + on( 2, u1=u1Boundary, u2=0.0);
Macros make your life a little less chaotic

- They only need to be debugged once.
- They clean up your code.
- Variables are just placeholders. They don’t need to be initialized.
- They’re slightly finicky (parenthesis)
- They don’t end with ;, but // instead.

```plaintext
macro div(u1,u2) ( dx(u1) + dy(u2) ) //
macro H1norm(T,u1,u2) ( sqrt(int2d(T)( dx(u1)^2 + dy(u1)^2 + dx(u2)^2 + dy(u2)^2 ) )) //
```
macro gradgrad(u1,u2,v1,v2) ( dx(u1) * dx(v1) + dx(u2) * dx(v2) + dy(u1) * dy(v1) + dy(u2) * dy(v2) ) //
macro div(u1,u2) ( dx(u1) + dy(u2) ) //

problem stokes([u1,u2,p],[v1,v2,q],solver=UMFPACK) =
  int2d(Th)(
    gradgrad(u1, u2, v1, v2)
    - p * div(v1, v2)
    - q * div(u1, u2)
    + PRESSURESTABILIZE * p * q
  )
  -int2d(Th)(
    f1*v1 + f2*v2
  )
  + on( 1, u1=0.0, u2=0.0)
  + on( 2, u1=u1Boundary, u2=0.0);
Exercise 2

The Poisson equation is

$$\Delta u = f.$$ 

Create a problem statement in FreeFEM to solve this problem.

- Define the solution space (use P1 functions)
- Define the variables (u)
- Write out the variational form.
- Use Dirichlet boundary conditions ($u = g$ on $\partial \Omega$)

Using true solution $u_t = 1 + x^2 + y^2$, we know $f = 4$ and $g = 1 + x^2 + y^2$. Calculate the error between your solution $u$ and the true solution with:

```cpp
cout << "L^2-Error=" << sqrt( int2d(Th)((u - (1+x^2+y^2))^2) ) << endl;
```
startTime = clock();
stokes;
compTime = clock();

plot(Th, wait=1, cmm="Our wonderful mesh", ps=plotdir+
    problemPrefix + "mesh.eps"); // Plots the mesh and then waits to
    continue.
plot(p, value=1, fill=1, cmm="Contour of pressures", ps=plotdir+
    problemPrefix + "pressure.eps", wait=1);
plot([u1, u2], value=1, coef=0.1, cmm="Velocity vectors", ps=plotdir+
    problemPrefix + "velocity.eps", wait=1);
plot(p, [u1, u2], value=1, fill=1, coef=0.1, cmm="Pressure and velocity",
    wait=1); // Note the order of pressure and then velocity

cout << " Time = " << compTime - startTime << " Solution norm = " << int2d(Th)(u1^2 + u2^2) << endl;
<table>
<thead>
<tr>
<th>Vec Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0809877</td>
</tr>
<tr>
<td>0.161975</td>
</tr>
<tr>
<td>0.242963</td>
</tr>
<tr>
<td>0.323951</td>
</tr>
<tr>
<td>0.404939</td>
</tr>
<tr>
<td>0.485926</td>
</tr>
<tr>
<td>0.566914</td>
</tr>
<tr>
<td>0.647902</td>
</tr>
<tr>
<td>0.728889</td>
</tr>
<tr>
<td>0.809877</td>
</tr>
<tr>
<td>0.890865</td>
</tr>
<tr>
<td>0.971853</td>
</tr>
<tr>
<td>1.05284</td>
</tr>
<tr>
<td>1.13383</td>
</tr>
<tr>
<td>1.21482</td>
</tr>
<tr>
<td>1.2958</td>
</tr>
<tr>
<td>1.37679</td>
</tr>
<tr>
<td>1.45778</td>
</tr>
<tr>
<td>1.53877</td>
</tr>
</tbody>
</table>

Velocity vectors
Pressure and velocity

IsoValue
-37.3391
-28.9141
-23.2973
-17.6806
-12.0639
-6.44722
-0.830515
4.78619
10.4029
16.0196
21.6363
27.253
32.8697
38.4864
44.1031
49.7199
55.3366
60.9533
66.57
80.6117

Vec Value
0
0.0809877
0.161975
0.242963
0.323951
0.404939
0.485926
0.566914
0.647902
0.728889
0.809877
0.890865
0.971853
1.05284
1.13383
1.21482
1.2958
1.37679
1.45778
1.53877
Hyperbolic problems, like the heat equation

\[ u_t - \mu \Delta u = f, \]

need another approach. One approach is Backward-Euler time stepping.

- Choose \( dt \) (time step size)
- Given \( u^m \)
- Solve

\[
\frac{u^{m+1} - u^m}{dt} - \mu \Delta u^{m+1} = f(t^{m+1})
\]
Saving Data

```c++
{
    ofstream file(datadir + problemPrefix + "_mesh_" + meshperunit + 
                   "_u1.txt");
    file << u1[] << endl;
}
{
    ofstream file(datadir + problemPrefix + "_mesh_" + meshperunit + 
                   "_u2.txt");
    file << u2[] << endl;
}
{
    ofstream file(datadir + problemPrefix + "_mesh_" + meshperunit + 
                   "_p.txt");
    file << p[] << endl;
}
```
Reports

Things that you probably should include in a report:

- Computation time (perhaps broken up into individual tasks/iterations as well as the entire run) `clock()` is the function to call system time.

- Mesh size (any parameters used to create the mesh, as well as number of triangles, min/max size of triangles, etc)

- Degrees of freedom (function of FEspaces, problem, mesh)

- List all relevant parameters. ANYTHING necessary to reproduce the experiment (and some things that aren’t). Convergence criterion for nonlinear solvers, stabilization terms, everything.

- Directories where everything is saved.
ofstream report(reportdir + problemPrefix + ”report.txt”);
report << endl << ”**************************************************************************” << endl << endl;
report << ”Problem Parameters” << endl;
report << ” ChLength = ” << ChLength << endl;
report << ” ObsLength = ” << ObsLength << endl;
report << ” ObsRadius = ” << ObsRadius << endl;
report << ” meshperunit = ” << meshperunit << endl << endl;
report << ”Boundary condition parameters” << endl;
report << ” maxSpeed = ” << maxSpeed << endl << endl;
report << ”Numerical solver parameters” << endl;
report << ” PRESSURESTABILIZE = ” << PRESSURESTABILIZE << endl << endl;
report << ”Results” << endl;
report << ” Time = ” << compTime − startTime << endl;
report << ” Solution norm = ” << int2d(Th)(u1^2 + u2^2) << endl;
Parameter Loops

- Easiest thing to change in code is parameters
- Multiple ways of changing
  - for loop over parameters inside code
  - `cin` command to make a call from commandline
- Types of things to check:
  - Convergence properties with different Reynolds number (fluid flow)
  - Solution properties with different borders
Parameter sweep example

// Initialize meshperunit and all other variables needed
mesh Th=square(2,2); // This is just to initialize the mesh, we’re going
to redefine it later.
real meshperunitINITIAL = 4.0;
int meshIndexMAX = 3;

for(int meshIndex = 0; meshIndex < meshIndexMAX; meshIndex++)
{ // Begin meshing loop
    meshperunit = meshperunitINITIAL * 2.0^meshIndex;

    Th = buildmesh( ... ) // from earlier

    //Rest of the code to have problem statement and the program.
    //Report anything that you want reported from this mesh level.
} // End meshing loop
Post Processing

There are many reasons to set up your code for post processing.

- Running your code on a remote computer and don’t have imaging abilities.
- You don’t know what you need to know at the beginning of the experiment.
- Need new plots of old data.
- Need error reports with different norms.
```c
mesh Th = readmesh(datadir + problemPrefix + ")msh");
fespace Xh(Th, P2);
Xh u1, u2;
{
    ifstream data(datadir + problemPrefix + "_mesh_" + meshperunit + "
    u1.txt");
data >> u1[];
}
{
    ifstream data(datadir + problemPrefix + "_mesh_" + meshperunit + "
    u2.txt");
data >> u2[];
}
```
Another way to do parameter sweeps:

```c
real reynolds, meshperunit, dt;
cout << ”Please enter the reynolds number, mesh points per unit length, and time step” << endl;
cin >> reynolds >> meshperunit
```

In linux:

```bash
for reynolds in 1 1e2 1e4; do
  for meshing in 4 8 16; do
    echo $reynolds $meshing | FreeFem++ stokes.edp > report${reynolds}_${meshing}.txt&;
  done
done
```

Also useful:

```bash
for i in ‘ls *.eps‘; do epstopdf $i; done
```