FreeFem++, part III

M. M. Sussman

sussmanm@math.pitt.edu

Office Hours: 11:10AM-12:10PM, Thack 622

May 12 – June 19, 2014

Topics

Exercise 21 was hard to program

- The term

$$\int_\Omega \frac{\partial u_j v_j w_i}{\partial x_i} = \int_\Omega \nabla \cdot \Big((u \cdot v) w \Big)\, dx$$
$$= \int_\Gamma (u \cdot v)(w \cdot n)\, ds$$
$$= 0 \quad \text{for Dirichlet b.c.}$$

- For Neumann conditions, need to include it.
- How to get $w \cdot n$?
- How to integrate only over exit?

The normal

- `n = FacetNormal(mesh)` identifies the norma
- It is a function on facets
- Used in `exampleDG.py`
- Searching the FEniCS book for “normal” finds a discussion of “Flux Functionals” on page 27, explaining how to use it.
- Example code from page 27

```
n = FacetNormal(mesh)
flux = -p*dot(nabla_grad(u), n) * ds
total_flux = assemble( flux )
```

- Suggests using `ds(i)` for integral over part of the boundary.
- Points to Section 1.5.3 (our `example6.py`)

Implementation

- Need to use a mesh function to mark the exit boundary
- Follow **`example6.py`**
- Add extra term to weak form
- Doesn't work.

Debugging

1. Begin with not-working code
2. Modify so choose skew-symmetric or tradtional easily
3. Remove plots
4. Compute error to give working/not working indicator
5. `print "integral u dot n ds(1) = ",assemble(inner(u,n)*ds(1))`
   - Gives 0.0
6. `print "integral u dot n ds = ",assemble(inner(u,n)*ds)`
   - Gives roundoff, not zero
   - Inlet - exit = roundoff
7. Go to **`example6.py`**, add similar **`assemble`** statements at end
   - Gives 0.0 for **`ds(1)`**
   - Gives nonzero for **`ds`**
8. Change **`g`** so solution is no longer the exact solution
9. Solution error becomes large!
10. Why does **`inner(u,n)*ds(1)`** work inside weak form, not inside **`print`**?

Make exercise 21 code look like `example6.py`

- Change things one step at a time!
- Test after each change

11. Check code for marking b.c.: essentially same
12. Exercise 21 code uses `solve(NSE == LNSE, w, bcs, ...`
13. Change to `assemble` and `solve` as in `example6.py`
14. Notice `exterior_facet_domains=boundary_parts` in `assemble`
15. Works now!

Topics

Available elements

- From the book, Chapter 6, p. 148ff
- **P0, P03d** discontinuous constant Lagrange elements
- **P1, P13d** continuous linear Lagrange elements
- **P1dc** discontinuous linear Lagrange elements
- **P1b, P1b3d** continuous linear with bubble
- **P2, P23d** continuous quadratic Lagrange elements
- **P2b** continuous quadratic with bubble
- **P2dc** discontinuous quadratic
- **P3**, **P3dc** continuous and discontinuous cubic
- **P4**, **P4dc** continuous and discontinuous quartic

Available elements, cont'd

- **Morley** Nonconforming element P2 element with $\frac{\partial u}{\partial n}$ continuous across element sides. Good for biharmonic equation.
- **P2BR** "Bernaid Raugel" P1 vector element with edge bubbles aimed at the Stokes equation
- **RT0, RT03d** Raviart-Thomas $H(div)$-conforming element. RT0-P0 (MINI element) is LBB-stable.
- **RT0Ortho** Raviart-Thomas Orthogonal (Nédélec, type I). Good for computational electromagnetics
- **Edge03d** Nédélec, type II, degree 0. Good for computational electromagnetics
- **RT1** Raviart-Thomas $H(div)$-conforming linear
- **RT1Ortho** Nédélec $H(curl)$-conforming linear
- **BDM1** Brezzi-Douglas-Marini $H(div)$-conforming linear vector element
- **BDM1Ortho** (Nédélec, type II), $H(curl)$-conforming linear vector
- **TDNNS1** symmetric-matrix $H(div - div)$ conforming linear

Topics

Topics

Steady NSE and Newton's method

- *Nonlinear* NSE in Ω

$$(u \cdot \nabla)u - \nu \Delta u + \nabla p = 0$$
$$\nabla \cdot u = 0$$

- *Could* treat as long-time limit of transient
- Newton's method: Given $F : V \to V$, find $u \in V$ so that $F(u) = 0$

  Choose $u^0 \in V$

  for ($i = 0$; $i <$maximum; $i++$)

    solve $\frac{\partial F}{\partial u}(u^i)\delta u = F(u^i)$

    $u^{i+1} = u^i + \delta u$

    if ($\|\delta u\| < \epsilon$) break

- Must be re-expressed in terms of weak forms.

Newton expressed with weak forms

Weak form of NSE:

$$0 = \int_\Omega (\nu \nabla u \cdot \nabla v + ((u \cdot \nabla)u) \cdot v - p\nabla \cdot v - q\nabla \cdot u - \epsilon\, pq) dx$$

Pick initial guess u^0 *satisfying the essential b.c.*, solve for $(\delta u, \delta p)$

$$\begin{aligned} 0 = \int_\Omega \Big(& \nu \nabla \delta u \cdot \nabla v + ((\delta u \cdot \nabla)u^{k-1}) \cdot v + ((u^{k-1} \cdot \nabla)\delta u) \cdot v \\ & -\delta p \nabla \cdot v - q\nabla \cdot u^{k-1} - \epsilon\, \delta p\, q \Big) dx \\ - \int_\Omega \Big(& \nu \nabla u^{k-1} \cdot \nabla v + ((u^{k-1} \cdot \nabla)u^{k-1}) \cdot v - p^{k-1}\nabla \cdot v - q\nabla \cdot u^{k-1} \\ & -\epsilon\, p^{k-1} q \Big) dx \end{aligned}$$

then update $u^k = u^{k-1} - \delta u$ and $p^k = p^{k-1} - \delta p$ and repeat

“Creeping up” on Reynolds number

- Often cannot solve for desired Reynolds number from initial guess
- Solve a lower Reynolds number first
- Use that as initial guess for higher Reynolds number
- Repeat until reach desired Reynolds number

exmaple33.edp code

```
verbosity=0;

// build the  Mesh
real R = 5, L=15;
border cc(t=0, 2*pi) x=cos(t)/2; y=sin(t)/2; label=1;
border ce(t=pi/2, 3*pi/2)  x=cos(t)*R; y=sin(t)*R; label=1;
border beb(tt=0,1)  real t=tt^1.2; x= t*L; y= -R; label = 1;
border beu(tt=1,0)  real t=tt^1.2; x= t*L; y= R; label = 1;
border beo(t=-R,R)   x= L; y= t; label = 0;
border bei(t=-R/4,R/4)   x= L/2; y= t; label = 0;

mesh Th=buildmesh(cc(-50) + ce(30) + beb(20) + beu(20) + beo(10) + bei(10));
plot(Th, wait=true);
```

exmaple33.edp code cont'd

```
//  FE Spaces
fespace Xh(Th,P2);
fespace Mh(Th,P1);
Xh u1,u2,v1,v2,du1,du2,u1p,u2p;
Mh p,q,dp,pp;
```

exmaple33.edp code cont'd

```
//  FE Spaces
fespace Xh(Th,P2);
fespace Mh(Th,P1);
Xh u1,u2,v1,v2,du1,du2,u1p,u2p;
Mh p,q,dp,pp;

//  macros
macro Grad(u1,u2) [ dx(u1),dy(u1), dx(u2),dy(u2)]//
macro UgradV(u1,u2,v1,v2) [ [u1,u2]' * [dx(v1),dy(v1)] ,
                            [u1,u2]' * [dx(v2),dy(v2)] ]//
macro div(u1,u2)  (dx(u1) + dy(u2))//
```

exmaple33.edp code cont'd

```
//  FE Spaces
fespace Xh(Th,P2);
fespace Mh(Th,P1);
Xh u1,u2,v1,v2,du1,du2,u1p,u2p;
Mh p,q,dp,pp;

//  macros
macro Grad(u1,u2) [ dx(u1),dy(u1), dx(u2),dy(u2)]//
macro UgradV(u1,u2,v1,v2) [ [u1,u2]' * [dx(v1),dy(v1)] ,
                            [u1,u2]' * [dx(v2),dy(v2)] ]//
macro div(u1,u2)  (dx(u1) + dy(u2))//

//  Physical parameter
real nu= 1./50, nufinal=1/200., cnu=0.5;

// stop test for Newton
real eps=1e-6;
```

exmaple33.edp code cont'd

```
//  FE Spaces
fespace Xh(Th,P2);
fespace Mh(Th,P1);
Xh u1,u2,v1,v2,du1,du2,u1p,u2p;
Mh p,q,dp,pp;

//  macros
macro Grad(u1,u2) [ dx(u1),dy(u1), dx(u2),dy(u2)]//
macro UgradV(u1,u2,v1,v2) [ [u1,u2]' * [dx(v1),dy(v1)] ,
                            [u1,u2]' * [dx(v2),dy(v2)] ]//
macro div(u1,u2)  (dx(u1) + dy(u2))//

//  Physical parameter
real nu= 1./50, nufinal=1/200., cnu=0.5;

// stop test for Newton
real eps=1e-6;

// intial guess with B.C.
u1 = ( x^2 + y^2 ) > 2;
u2=0;
```

exmaple33.edp code cont'd

```
//  FE Spaces
fespace Xh(Th,P2);
fespace Mh(Th,P1);
Xh u1,u2,v1,v2,du1,du2,u1p,u2p;
Mh p,q,dp,pp;

//  macros
macro Grad(u1,u2) [ dx(u1),dy(u1), dx(u2),dy(u2)]//
macro UgradV(u1,u2,v1,v2) [ [u1,u2]' * [dx(v1),dy(v1)] ,
                            [u1,u2]' * [dx(v2),dy(v2)] ]//
macro div(u1,u2)  (dx(u1) + dy(u2))//

//  Physical parameter
real nu= 1./50, nufinal=1/200., cnu=0.5;

// stop test for Newton
real eps=1e-6;

// intial guess with B.C.
u1 = ( x^2 + y^2 ) > 2;
u2=0;

while(true){  //  Loop on viscosity
```

exmaple33.edp Newton iteration

```
int n;
real err=0;
for( n=0; n< 15; n++){ // Newton Loop
```

exmaple33.edp Newton iteration

```
int n;
real err=0;
for( n=0; n< 15; n++){ // Newton Loop
  solve Oseen([du1,du2,dp],[v1,v2,q]) =
    int2d(Th) (  nu*( Grad(du1, du2)' * Grad(v1, v2) )
                + UgradV(du1, du2, u1, u2)' * [v1, v2]
                + UgradV( u1, u2, du1, du2)' * [v1, v2]
                - div(du1, du2)*q - div(v1, v2)*dp
                - 1e-8*dp*q // stabilization term
               )
```

exmaple33.edp Newton iteration

```
int n;
real err=0;
for( n=0; n< 15; n++){ // Newton Loop
  solve Oseen([du1,du2,dp],[v1,v2,q]) =
    int2d(Th) (  nu*( Grad(du1, du2)' * Grad(v1, v2) )
               + UgradV(du1, du2, u1, u2)' * [v1, v2]
               + UgradV( u1, u2, du1, du2)' * [v1, v2]
               - div(du1, du2)*q - div(v1, v2)*dp
               - 1e-8*dp*q // stabilization term
              )
    - int2d(Th) (  nu*(Grad(u1, u2)' * Grad(v1, v2) )
                  + UgradV(u1,u2, u1, u2)' * [v1, v2]
                  - div(u1, u2)*q - div(v1, v2)*p
                  - 1e-8*p*q
                 )
```

exmaple33.edp Newton iteration

```
int n;
real err=0;
for( n=0; n< 15; n++){ // Newton Loop
  solve Oseen([du1,du2,dp],[v1,v2,q]) =
    int2d(Th) (  nu*( Grad(du1, du2)' * Grad(v1, v2) )
               + UgradV(du1, du2, u1, u2)' * [v1, v2]
               + UgradV( u1, u2, du1, du2)' * [v1, v2]
               - div(du1, du2)*q - div(v1, v2)*dp
               - 1e-8*dp*q // stabilization term
              )
    - int2d(Th) (  nu*(Grad(u1, u2)' * Grad(v1, v2) )
                  + UgradV(u1,u2, u1, u2)' * [v1, v2]
                  - div(u1, u2)*q - div(v1, v2)*p
                  - 1e-8*p*q
                 )
    + on(1,du1=0,du2=0)
    ;
```

exmaple33.edp Newton iteration

```
int n;
real err=0;
for( n=0; n< 15; n++){ // Newton Loop
  solve Oseen([du1,du2,dp],[v1,v2,q]) =
    int2d(Th) (  nu*( Grad(du1, du2)' * Grad(v1, v2) )
                + UgradV(du1, du2, u1, u2)' * [v1, v2]
                + UgradV( u1, u2, du1, du2)' * [v1, v2]
                - div(du1, du2)*q - div(v1, v2)*dp
                - 1e-8*dp*q // stabilization term
               )
    - int2d(Th) (  nu*(Grad(u1, u2)' * Grad(v1, v2) )
                   + UgradV(u1,u2, u1, u2)' * [v1, v2]
                   - div(u1, u2)*q - div(v1, v2)*p
                   - 1e-8*p*q
                 )
    + on(1,du1=0,du2=0)
    ;

  u1[] -= du1[];
  u2[] -= du2[];
  p[]  -= dp[];

  real Lu1=u1[].linfty,  Lu2 = u2[].linfty , Lp = p[].linfty;
  err= du1[].linfty/Lu1 + du2[].linfty/Lu2 + dp[].linfty/Lp;

  cout << n << " err = " << err << " " << eps << " Re  =" << 1./nu << endl;
```

exmaple33.edp Newton iteration

```
int n;
real err=0;
for( n=0; n< 15; n++){ // Newton Loop
  solve Oseen([du1,du2,dp],[v1,v2,q]) =
    int2d(Th) (  nu*( Grad(du1, du2)' * Grad(v1, v2) )
                + UgradV(du1, du2, u1, u2)' * [v1, v2]
                + UgradV( u1, u2, du1, du2)' * [v1, v2]
                - div(du1, du2)*q - div(v1, v2)*dp
                - 1e-8*dp*q // stabilization term
               )
    - int2d(Th) (  nu*(Grad(u1, u2)' * Grad(v1, v2) )
                   + UgradV(u1,u2, u1, u2)' * [v1, v2]
                   - div(u1, u2)*q - div(v1, v2)*p
                   - 1e-8*p*q
                 )
    + on(1,du1=0,du2=0)
    ;

  u1[] -= du1[];
  u2[] -= du2[];
  p[]  -= dp[];

  real Lu1=u1[].linfty,  Lu2 = u2[].linfty , Lp = p[].linfty;
  err= du1[].linfty/Lu1 + du2[].linfty/Lu2 + dp[].linfty/Lp;

  cout << n << " err = " << err << " " << eps << " Re  =" << 1./nu << endl;
  if(err < eps) break; // converge
```

exmaple33.edp Newton iteration

```
int n;
real err=0;
for( n=0; n< 15; n++){ // Newton Loop
  solve Oseen([du1,du2,dp],[v1,v2,q]) =
    int2d(Th) (  nu*( Grad(du1, du2)' * Grad(v1, v2) )
                + UgradV(du1, du2, u1, u2)' * [v1, v2]
                + UgradV( u1, u2, du1, du2)' * [v1, v2]
                - div(du1, du2)*q - div(v1, v2)*dp
                - 1e-8*dp*q // stabilization term
               )
    - int2d(Th) (  nu*(Grad(u1, u2)' * Grad(v1, v2) )
                   + UgradV(u1,u2, u1, u2)' * [v1, v2]
                   - div(u1, u2)*q - div(v1, v2)*p
                   - 1e-8*p*q
                 )
    + on(1,du1=0,du2=0)
    ;

  u1[] -= du1[];
  u2[] -= du2[];
  p[]  -= dp[];

  real Lu1=u1[].linfty,  Lu2 = u2[].linfty , Lp = p[].linfty;
  err= du1[].linfty/Lu1 + du2[].linfty/Lu2 + dp[].linfty/Lp;

  cout << n << " err = " << err << " " << eps << " Re  =" << 1./nu << endl;
  if(err < eps) break; // converge
  if( n>3 && err > 10.) break; //  Blowup ????
}
```

exmaple33.edp ν iteration

```
if(err < eps){  // if converge  decrease nu (more difficult)
  plot([u1,u2], p, wait=1, cmm=" Re = " + 1./nu , coef=0.3);
```

exmaple33.edp ν iteration

```
if(err < eps){  // if converge  decrease nu (more difficult)
  plot([u1,u2], p, wait=1, cmm=" Re = " + 1./nu , coef=0.3);

  if( nu == nufinal) break;
```

exmaple33.edp ν iteration

```
if(err < eps){  // if converge  decrease nu (more difficult)
  plot([u1,u2], p, wait=1, cmm=" Re = " + 1./nu , coef=0.3);

  if( nu == nufinal) break;

  if( n < 4) cnu=cnu^1.5; // fast converge => change faster
  nu = max(nufinal, nu * cnu); // new viscosity
  u1p=u1;  //  save correct solution ...
  u2p=u2;
  pp=p;
```

exmaple33.edp ν iteration

```
if(err < eps){  // if converge  decrease nu (more difficult)
  plot([u1,u2], p, wait=1, cmm=" Re = " + 1./nu , coef=0.3);

  if( nu == nufinal) break;

  if( n < 4) cnu=cnu^1.5; // fast converge => change faster
  nu = max(nufinal, nu * cnu); // new viscosity
  u1p=u1;  //  save correct solution ...
  u2p=u2;
  pp=p;

} else {  // if blowup,  increase nu (more simple)
  assert(cnu< 0.95); // final blowup ...
```

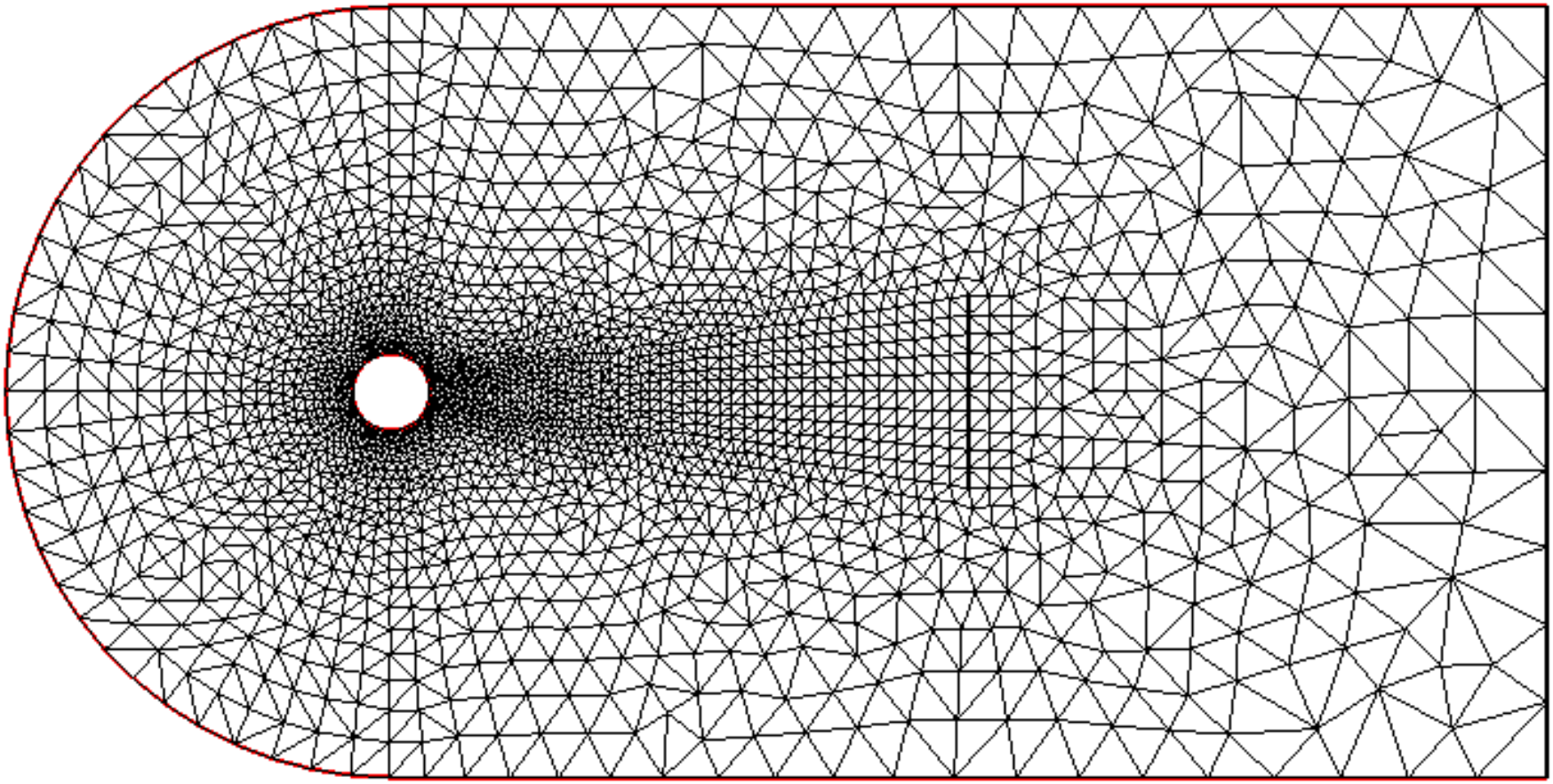
exmaple33.edp ν iteration

```
  if(err < eps){  // if converge  decrease nu (more difficult)
    plot([u1,u2], p, wait=1, cmm=" Re = " + 1./nu , coef=0.3);

    if( nu == nufinal) break;

    if( n < 4) cnu=cnu^1.5; // fast converge => change faster
    nu = max(nufinal, nu * cnu); // new viscosity
    u1p=u1;  //  save correct solution ...
    u2p=u2;
    pp=p;

  } else {  // if blowup,  increase nu (more simple)
    assert(cnu< 0.95); // final blowup ...
    nu = nu/cnu; //  get previous value of viscosity
    cnu= cnu^(1./1.5); // no conv. => change lower
    nu = nu * cnu;  // new vicosity
    cout << " restart nu = " << nu << " Re= "<< 1./nu <<
            "  (cnu = " << cnu << " )" << endl;
    // restore correct solution ..
    u1=u1p;
    u2=u2p;
    p=pp;
  }
} // end while loop
```

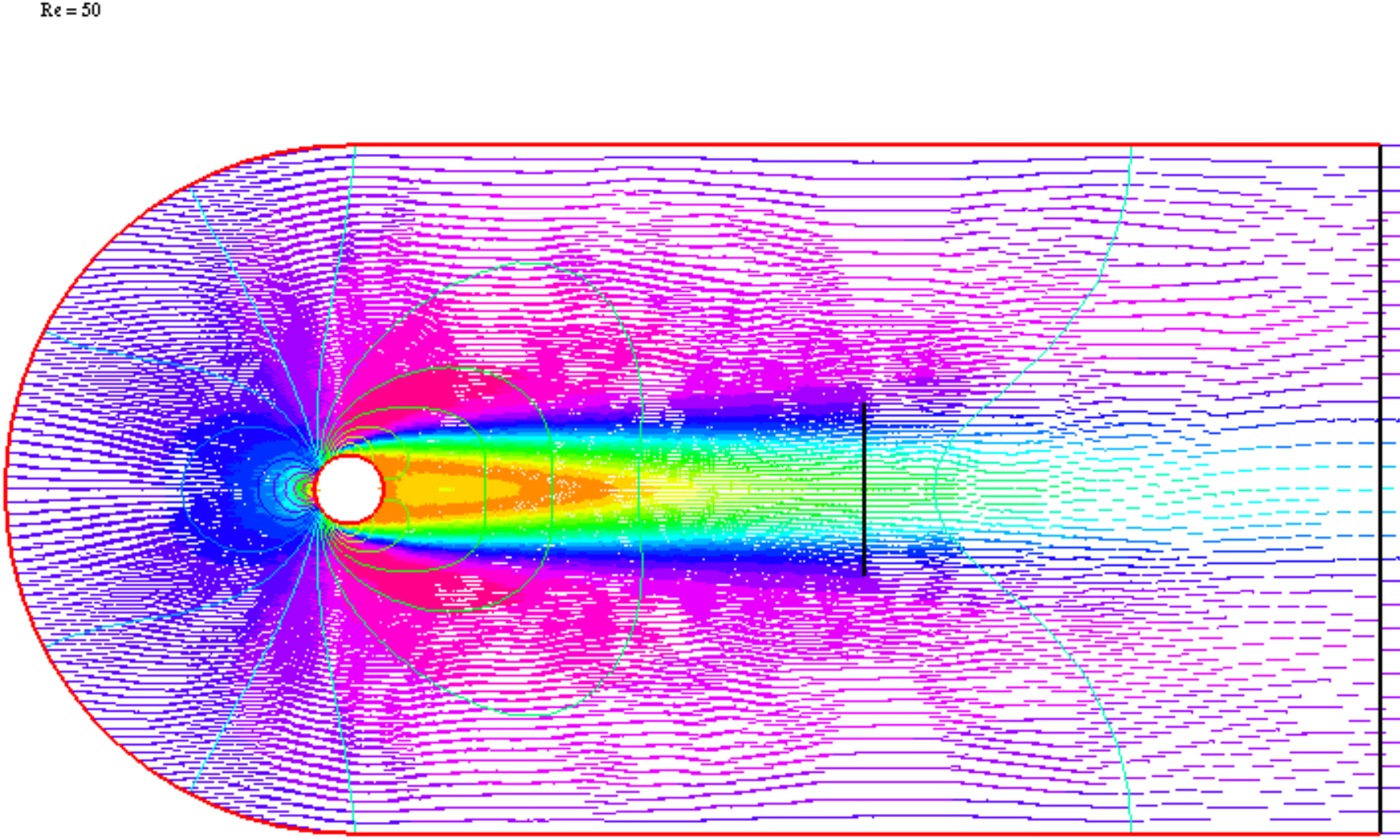
example33.edp printed output

```
  -  mesh:  Nb of Triangles =   4434, Nb of Vertices 2282
0 err = 3 1e-06 Re  =50
1 err = 3.38445 1e-06 Re  =50
2 err = 2.64188 1e-06 Re  =50
3 err = 0.724925 1e-06 Re  =50
4 err = 0.233259 1e-06 Re  =50
5 err = 0.0201813 1e-06 Re  =50
6 err = 0.000100565 1e-06 Re  =50
7 err = 5.11042e-09 1e-06 Re  =50
0 err = 0.481403 1e-06 Re  =100
1 err = 0.163572 1e-06 Re  =100
2 err = 0.0117441 1e-06 Re  =100
3 err = 8.08954e-05 1e-06 Re  =100
4 err = 4.55366e-09 1e-06 Re  =100
0 err = 0.591648 1e-06 Re  =200
1 err = 0.50591 1e-06 Re  =200
2 err = 0.128229 1e-06 Re  =200
3 err = 0.0286099 1e-06 Re  =200
4 err = 0.00125451 1e-06 Re  =200
5 err = 2.2793e-06 1e-06 Re  =200
6 err = 4.16258e-12 1e-06 Re  =200
```

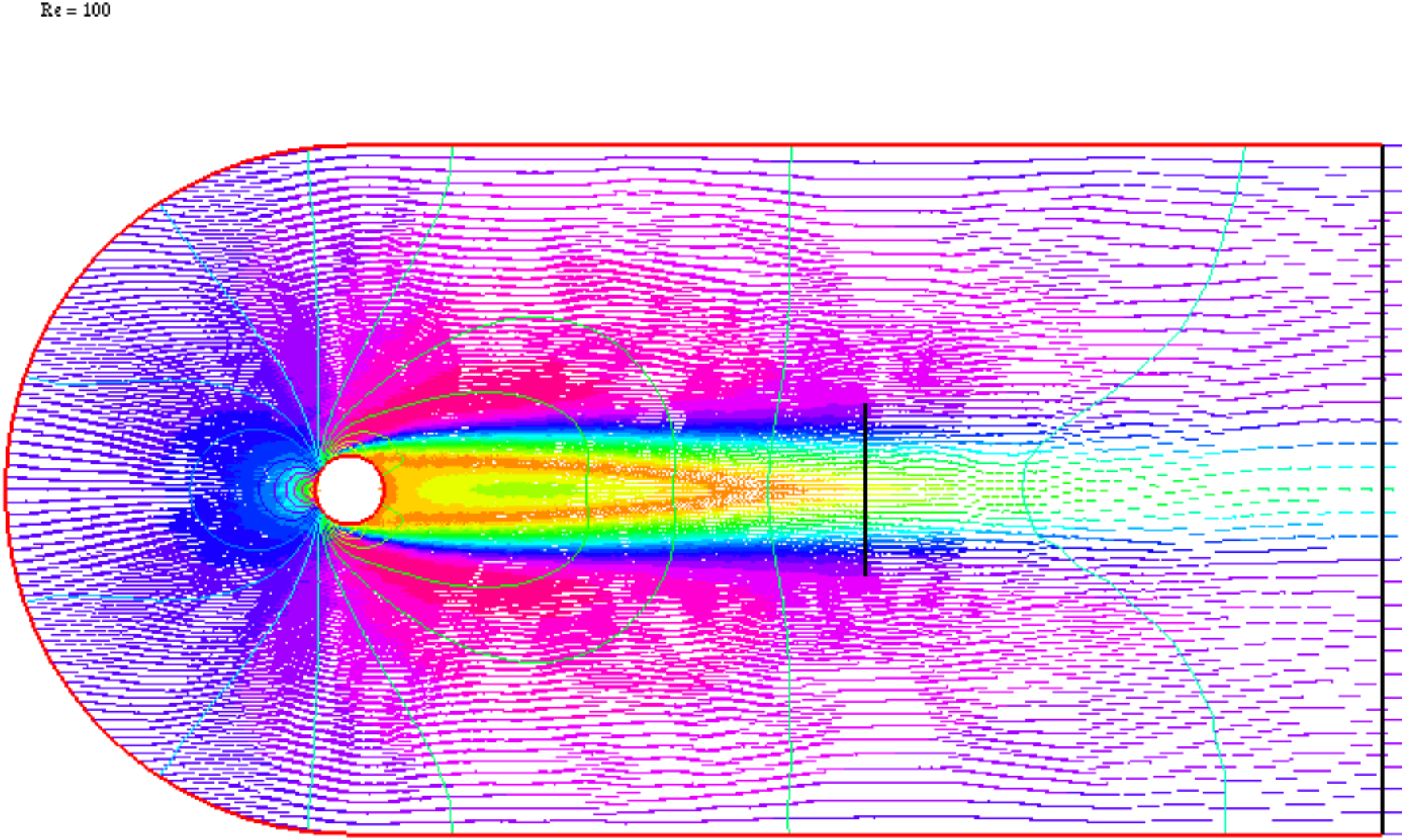
`example33.edp` mesh



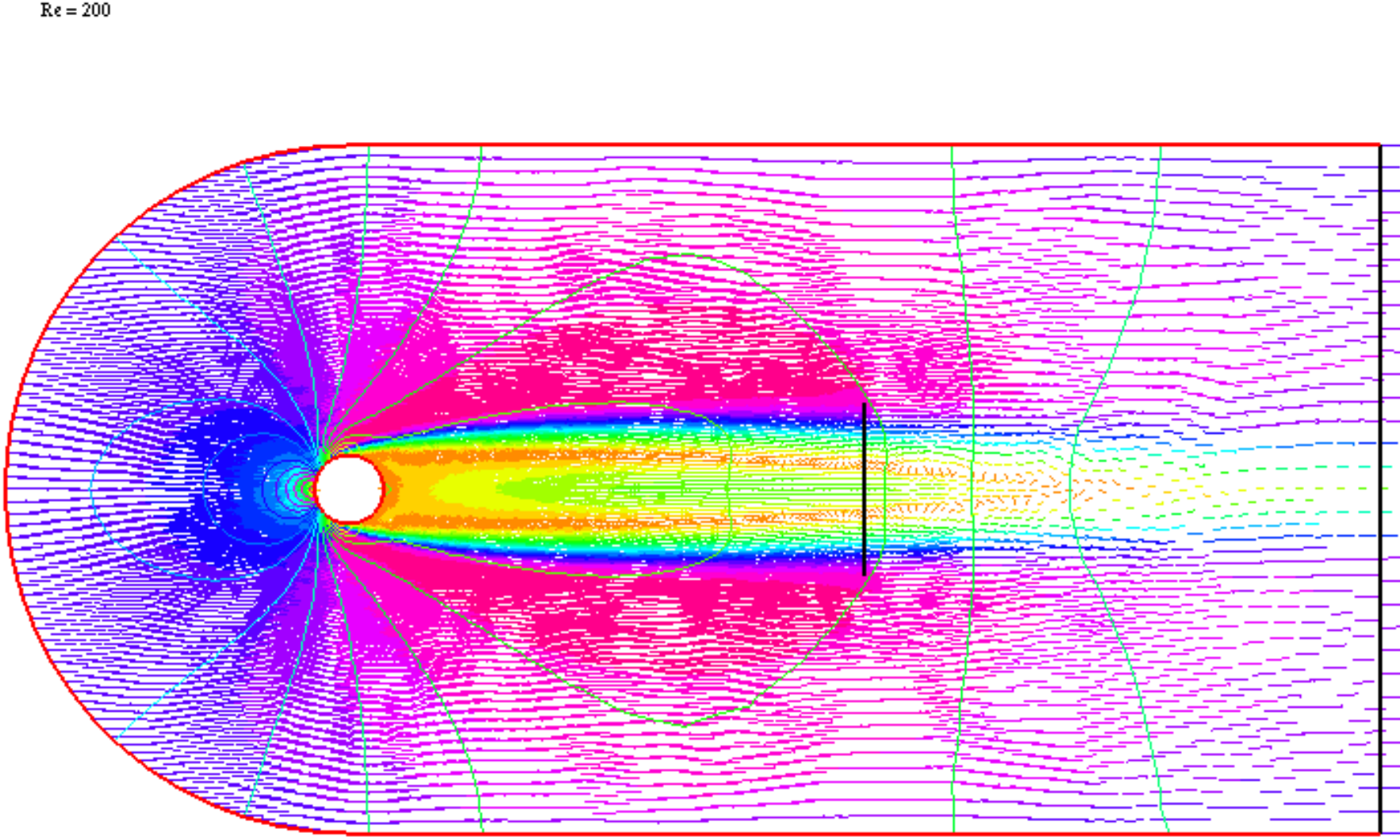
`example33.edp` Reynolds number = 50



`example33.edp` Reynolds number = 100



`example33.edp` Reynolds number = 200



Topics

MPI syntax

- **`mpiComm`** type for MPI communicator
- **`mpiCommWorld`** constant everything communicator
- **`mpisize`** integer = total number processes
- **`mpirank`** integer = id of current process
- Reduce operator keywords **`mpiMAX`**, **`mpiMIN`**, **`mpiSUM`**, **`mpiPROD`**

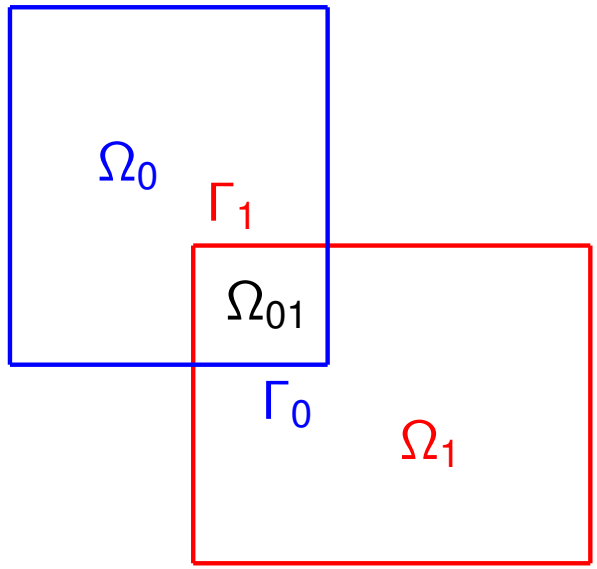
MPI functions

- **processor(**comm**)**
- **mpiBarrier(**comm**)**

Message passing

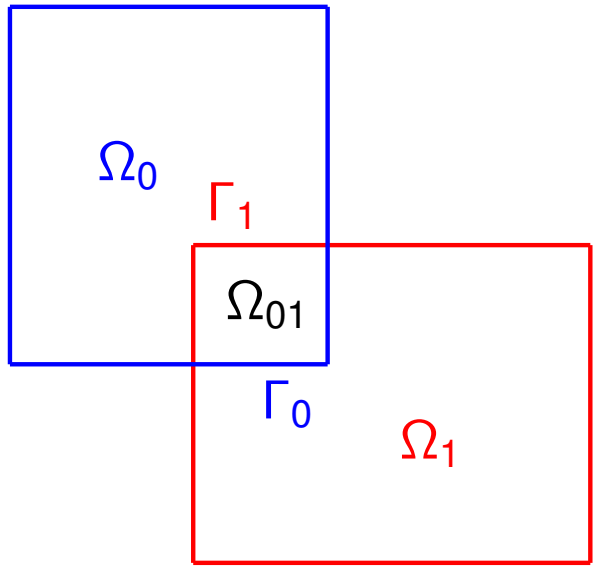
- `processor(10) << a << b;` Send a,b to processor 10
- `processor(10) >> a >> b;` (blocking) Receive a,b from processor 10
- `broadcast(processor(id),a)`
- `mpiReduce(a,b,processor(id),mpiMAX)`
- `mpiAllReduce(a,b,processor(id),mpiMAX)`

Schwarz overlapping method



- Solve on Ω_0, regarding $\Omega_{01} \subset \Omega_0$, with Dirichlet values on Γ_0 taken from solution on Ω_1.
- Solve on Ω_1, regarding $\Omega_{01} \subset \Omega_1$, with Dirichlet values on Γ_1 taken from solution on Ω_0.
- Repeat until converged.

Poisson equation



$$-\Delta u = f \text{ in } \Omega = \Omega_0 \cup \Omega_1, \quad u|_\Gamma = 0$$

$$-\Delta u_0^n = f \text{ in } \Omega_0, \quad u_0^n|_{\Gamma_0} = u_1^{n-1}$$

$$-\Delta u_1^n = f \text{ in } \Omega_1, \quad u_1^n|_{\Gamma_1} = u_0^{n-1}$$

Schwarz in parallel

- Place regions on different processors.
- Only *need* to send data on Γ_0 to processor 1
- Lazily, send all data from Ω_0 to processor 1
- Similarly for Γ_1.
- Could have used “empty mesh”

example34.edp code

```
if ( mpisize != 2 ) {
  cout « " Sorry, Example 34 requires exactly 2 processors. " « endl;
  exit(1);
}
```

example34.edp code

```
if ( mpisize != 2 ) {
  cout « " Sorry, Example 34 requires exactly 2 processors. " « endl;
  exit(1);
}

verbosity = 3;
```

example34.edp code

```
if ( mpisize != 2 ) {
  cout « " Sorry, Example 34 requires exactly 2 processors. " « endl;
  exit(1);
}

verbosity = 3;

int inside = 2;
int outside = 1;
border a( t=1, 2){ x=t; y=0; label=outside;}
border b( t=0, 1){ x=2; y=t; label=outside;}
border c( t=2, 0){ x=t; y=1; label=outside;}
border d( t=1, 0){ x=1-t; y=t; label=inside;}
border e( t=0, pi/2){ x= cos(t); y = sin(t);label=inside;}
border e1( t=pi/2, 2*pi){ x= cos(t); y = sin(t); label=outside;}
```

example34.edp code

```
if ( mpisize != 2 ) {
  cout « " Sorry, Example 34 requires exactly 2 processors. " « endl;
  exit(1);
}

verbosity = 3;

int inside = 2;
int outside = 1;
border a( t=1, 2){ x=t; y=0; label=outside;}
border b( t=0, 1){ x=2; y=t; label=outside;}
border c( t=2, 0){ x=t; y=1; label=outside;}
border d( t=1, 0){ x=1-t; y=t; label=inside;}
border e( t=0, pi/2){ x= cos(t); y = sin(t);label=inside;}
border e1( t=pi/2, 2*pi){ x= cos(t); y = sin(t); label=outside;}

int n=4;
mesh th,TH;

if (mpirank == 0) {
  th = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
} else {
  TH = buildmesh ( e(5*n) + e1(25*n) );
}
```

example34.edp code

```
if ( mpisize != 2 ) {
  cout « " Sorry, Example 34 requires exactly 2 processors. " « endl;
  exit(1);
}

verbosity = 3;

int inside = 2;
int outside = 1;
border a( t=1, 2){ x=t; y=0; label=outside;}
border b( t=0, 1){ x=2; y=t; label=outside;}
border c( t=2, 0){ x=t; y=1; label=outside;}
border d( t=1, 0){ x=1-t; y=t; label=inside;}
border e( t=0, pi/2){ x= cos(t); y = sin(t);label=inside;}
border e1( t=pi/2, 2*pi){ x= cos(t); y = sin(t); label=outside;}

int n=4;
mesh th,TH;

if (mpirank == 0) {
  th = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
} else {
  TH = buildmesh ( e(5*n) + e1(25*n) );
}

// communicate whole mesh instead of only part
broadcast(processor(0), th);
broadcast(processor(1), TH);
```

`example34.edp` code cont'd

```
fespace vh(th, P1);
fespace VH(TH, P1);
vh u=0, v;
VH U=0, V;
real f=1.;
int reuseMatrix=false;
```

example34.edp code cont'd

```
fespace vh(th, P1);
fespace VH(TH, P1);
vh u=0, v;
VH U=0, V;
real f=1.;
int reuseMatrix=false;

problem PB(U, V, init=reuseMatrix, solver=Cholesky) =
    int2d(TH)( dx(U)*dx(V) + dy(U)*dy(V) )
  + int2d(TH)( -f*V) + on(inside, U = u) + on(outside, U = 0 ) ;
```

example34.edp code cont'd

```
fespace vh(th, P1);
fespace VH(TH, P1);
vh u=0, v;
VH U=0, V;
real f=1.;
int reuseMatrix=false;

problem PB(U, V, init=reuseMatrix, solver=Cholesky) =
    int2d(TH)( dx(U)*dx(V) + dy(U)*dy(V) )
  + int2d(TH)( -f*V) + on(inside, U = u) + on(outside, U = 0 ) ;

problem pb(u, v, init=reuseMatrix, solver=Cholesky) =
    int2d(th)( dx(u)*dx(v) + dy(u)*dy(v) )
  + int2d(th)( -f*v) + on(inside ,u = U) + on(outside, u = 0 ) ;
```

`example34.edp` code cont'd

```
for (int i=0; i< 10; i++) {
  cout « mpirank « " looP " « i « endl;
  if (mpirank == 0) {
   PB;
   processor(1) << U[];
   processor(1) >> u[];
  } else {
   pb;
   processor(0) >> U[];
   processor(0) << u[];
  }



}
```

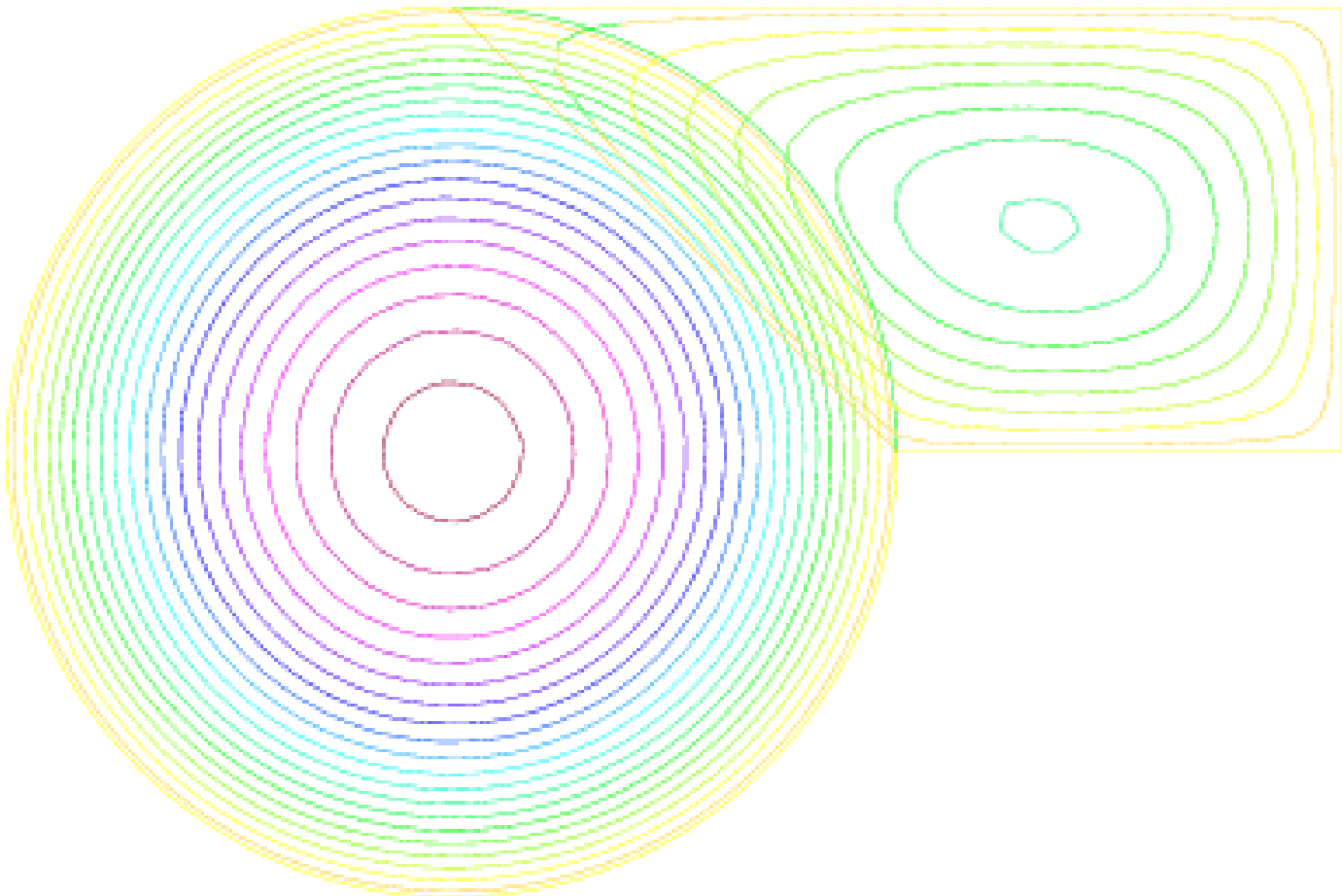
`example34.edp` code cont'd

```
for (int i=0; i< 10; i++) {
  cout « mpirank « " looP " « i « endl;
  if (mpirank == 0) {
   PB;
   processor(1) << U[];
   processor(1) >> u[];
  } else {
   pb;
   processor(0) >> U[];
   processor(0) << u[];
  }
  if (false && mpirank==0){
    plot(U,u,wait=true,ps="Uu34-"+i+".eps");
  }
}
if (mpirank==0){
  plot(U,u,ps="Uu34.eps",wait=true);
  plot(U,ps="U34.eps",wait=true);
  plot(u,ps="u34.eps",wait=true);
}
```

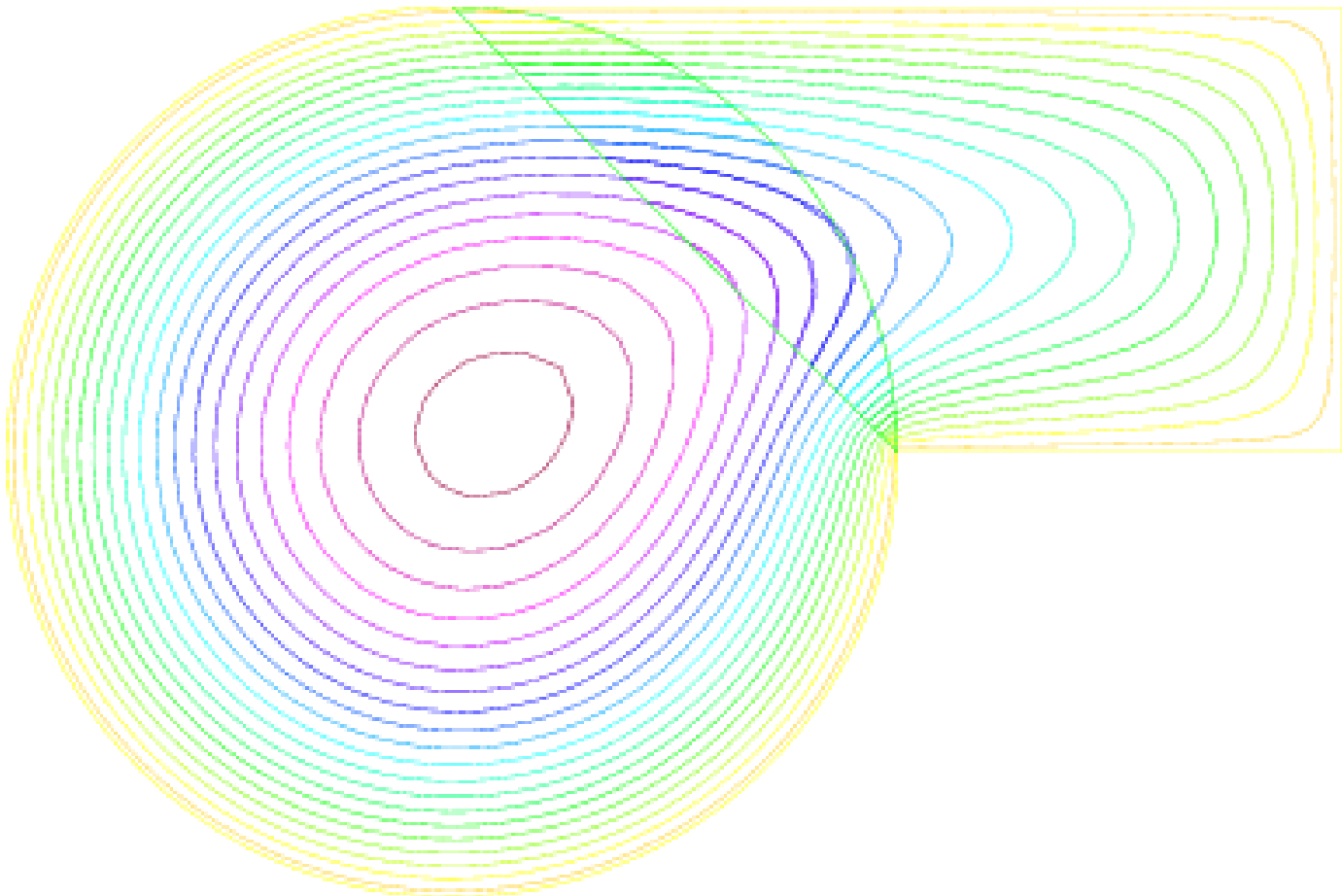
`example34.edp` code cont'd

```
for (int i=0; i< 10; i++) {
  cout « mpirank « " looP " « i « endl;
  if (mpirank == 0) {
   PB;
   processor(1) << U[];
   processor(1) >> u[];
  } else {
   pb;
   processor(0) >> U[];
   processor(0) << u[];
  }
  if (false && mpirank==0){
    plot(U,u,wait=true,ps="Uu34-"+i+".eps");
  }
}
if (mpirank==0){
  plot(U,u,ps="Uu34.eps",wait=true);
  plot(U,ps="U34.eps",wait=true);
  plot(u,ps="u34.eps",wait=true);
}

// Want both processors to terminate together.
mpiBarrier(mpiCommWorld);
```

Example 34, first iteration



Example 34, final



Topics

Non-overlapping Schwartz method

- $-\Delta u = f$ in $\Omega = \Omega_1 \cup \Omega_2$, with $u|_\Gamma = 0$
- $\Gamma_1 = \Gamma_2 = \Omega_1 \cap \Omega_2$ is a line
- $\Gamma_e^i = \Gamma \setminus \Gamma_i$
- Find λ so that $\lambda = u_1|_{\Gamma_1} = u_2|_{\Gamma_2}$, where

$$-\Delta u_i = f \text{ in } \Omega_i, \qquad u_i|_{\Gamma_i} = \lambda, \quad u_i|_{\Gamma_e^i} = 0$$

Two affine maps

- Define two affine mappings

$$S_i : \lambda \mapsto u_i|_{\Gamma_i}$$

- Problem is to find λ satisfying $S_1(\lambda) = S_2(\lambda)$

SPD matrix

- This is an affine problem with a SPD matrix
- Let A_i be the matrix of subproblem i
- Let P_i be the projection matrix $u_i \mapsto \lambda$
- The system can be written as

$$\begin{pmatrix} A_1 & 0 & P_1^T \\ 0 & A_2 & P_2^T \\ P_1 & P_2 & -2I \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \lambda \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ 0 \end{pmatrix}$$

- This can be reduced to

$$\begin{pmatrix} A_1 & 0 & P_1^T \\ 0 & A_2 & P_2^T \\ 0 & 0 & -2I - P_1 A_1^{-1} P_1^T - P_2 A_2^{-1} P_2^T \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \lambda \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ -P_1 A_1^{-1} b_1 - P_2 A_2^{-1} b_2 \end{pmatrix}$$

Solve iteratively, use CG

To compute the result $(-2I - P_1 A_1^{-1} P_1^T - P_2 A_2^{-1} P_2^T)\lambda$:

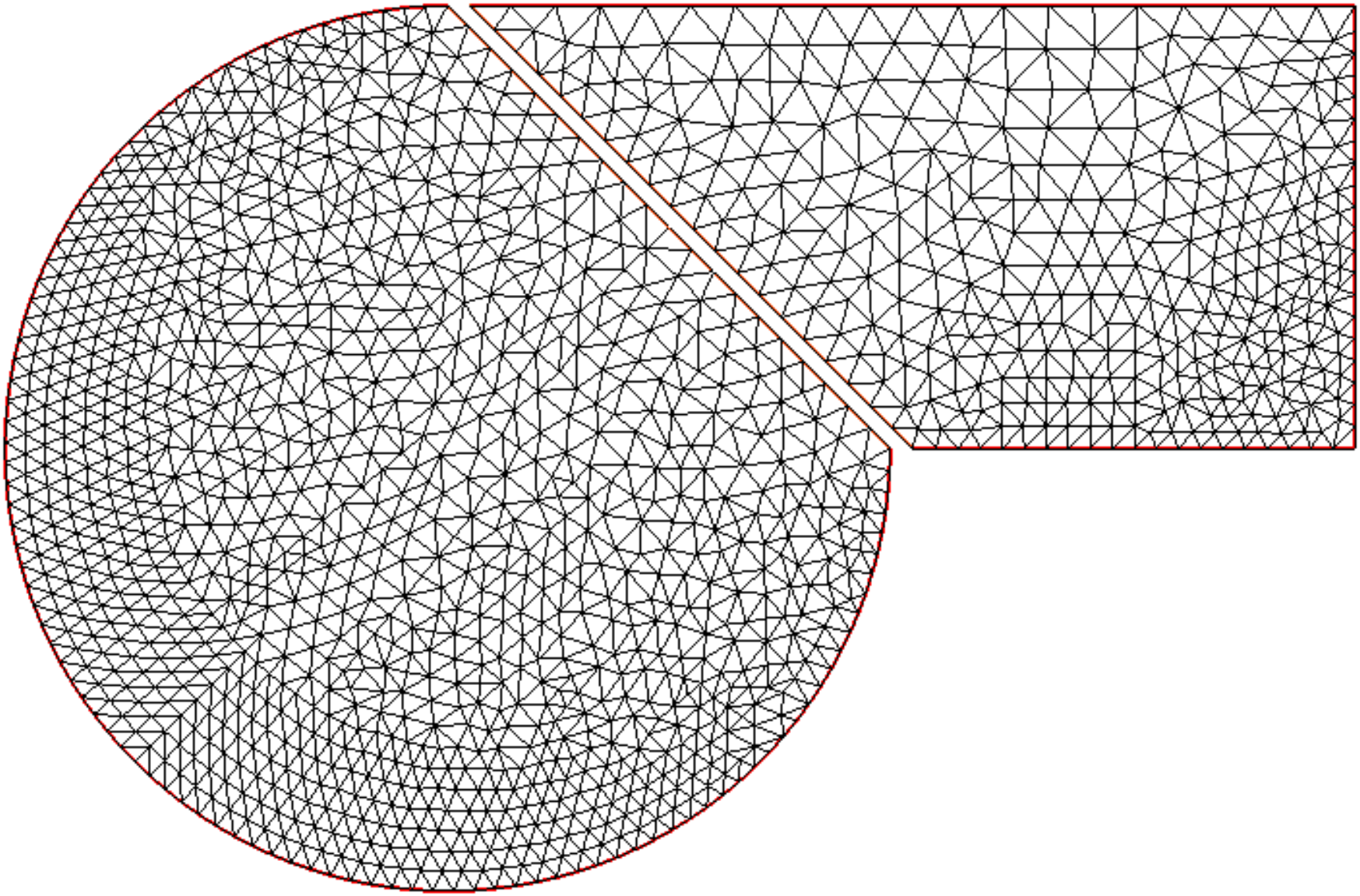
1. Solve for u_1 and u_2
2. Compute the difference $S_1(\lambda) - S_2(\lambda)$

Note: the number of DOFs in λ is small. Should converge quickly.

Mortar methods

- No reason λ has to be in a FE space related to u_i!
- Define a third FE space on the interface line
- Solve for λ in that space

Example 35, same problem as Example 34



FreeFem++ limitation

- FreeFem++ does not support one-dimensional FE spaces in this context.
- Allow λ to be in one of the 2D pieces put with all zeros except on Γ_i
- Could have used **`emptymesh`**

example35.edp code

```
// example35.edp from schwarz-gc.edp, Section 9.28, p. 258
// Schwarz without overlapping (Shur complenement Neumann -> Dirichet)
verbosity=2;
real cpu=clock();
```

example35.edp code

```
// example35.edp from schwarz-gc.edp, Section 9.28, p. 258
// Schwarz without overlapping (Shur complenement Neumann -> Dirichet)
verbosity=2;
real cpu=clock();
int inside = 2;
int outside = 1;
border Gamma1(t=1,2){x=t; y=0; label=outside;}
border Gamma2(t=0,1){x=2; y=t; label=outside;}
border Gamma3(t=2,0){x=t; y=1; label=outside;}
border GammaInside(t=1,0){x = 1-t; y = t; label=inside;}
border GammaArc(t=pi/2, 2*pi){ x= cos(t); y = sin(t); label=outside;}
```

example35.edp code

```
// example35.edp from schwarz-gc.edp, Section 9.28, p. 258
// Schwarz without overlapping (Shur complenement Neumann -> Dirichet)
verbosity=2;
real cpu=clock();
int inside = 2;
int outside = 1;
border Gamma1(t=1,2){x=t; y=0; label=outside;}
border Gamma2(t=0,1){x=2; y=t; label=outside;}
border Gamma3(t=2,0){x=t; y=1; label=outside;}
border GammaInside(t=1,0){x = 1-t; y = t; label=inside;}
border GammaArc(t=pi/2, 2*pi){ x= cos(t); y = sin(t); label=outside;}

int n=4;
mesh Th1 = buildmesh( Gamma1(5*n) + Gamma2(5*n) + GammaInside(5*n) +
                      Gamma3(5*n));
mesh Th2 = buildmesh ( GammaInside(-5*n) + GammaArc(25*n) );

plot(Th1,Th2);
```

example35.edp code

```
// example35.edp from schwarz-gc.edp, Section 9.28, p. 258
// Schwarz without overlapping (Shur complenement Neumann -> Dirichet)
verbosity=2;
real cpu=clock();
int inside = 2;
int outside = 1;
border Gamma1(t=1,2){x=t; y=0; label=outside;}
border Gamma2(t=0,1){x=2; y=t; label=outside;}
border Gamma3(t=2,0){x=t; y=1; label=outside;}
border GammaInside(t=1,0){x = 1-t; y = t; label=inside;}
border GammaArc(t=pi/2, 2*pi){ x= cos(t); y = sin(t); label=outside;}

int n=4;
mesh Th1 = buildmesh( Gamma1(5*n) + Gamma2(5*n) + GammaInside(5*n) +
                      Gamma3(5*n));
mesh Th2 = buildmesh ( GammaInside(-5*n) + GammaArc(25*n) );

plot(Th1,Th2);

fespace Vh1(Th1,P1);
Vh1 u1=0,v1;
Vh1 lambda=0;
fespace Vh2(Th2,P1);
Vh2 u2=0,v2;
```

example35.edp code, cont'd

```
problem Pb2(u2,v2,init=reuseMatrix,solver=Cholesky) =
    int2d(Th2)( dx(u2)*dx(v2)+dy(u2)*dy(v2) )
  + int2d(Th2)( -v2)
  + int1d(Th2,inside)(-lambda*v2) +    on(outside,u2= 0 ) ;

problem Pb1(u1,v1,init=reuseMatrix,solver=Cholesky) =
    int2d(Th1)( dx(u1)*dx(v1)+dy(u1)*dy(v1) )
  + int2d(Th1)( -v1)
  + int1d(Th1,inside)(+lambda*v1) +    on(outside,u1 = 0 ) ;
```

`example35.edp` code, cont'd

```
problem Pb2(u2,v2,init=reuseMatrix,solver=Cholesky) =
    int2d(Th2)( dx(u2)*dx(v2)+dy(u2)*dy(v2) )
  + int2d(Th2)( -v2)
  + int1d(Th2,inside)(-lambda*v2) +    on(outside,u2= 0 ) ;

problem Pb1(u1,v1,init=reuseMatrix,solver=Cholesky) =
    int2d(Th1)( dx(u1)*dx(v1)+dy(u1)*dy(v1) )
  + int2d(Th1)( -v1)
  + int1d(Th1,inside)(+lambda*v1) +    on(outside,u1 = 0 ) ;

varf b(u2,v2,solver=CG) =int1d(Th1,inside)(u2*v2);
matrix B= b(Vh1,Vh1,solver=CG);
```

example35.edp code, cont'd

```
problem Pb2(u2,v2,init=reuseMatrix,solver=Cholesky) =
    int2d(Th2)( dx(u2)*dx(v2)+dy(u2)*dy(v2) )
  + int2d(Th2)( -v2)
  + int1d(Th2,inside)(-lambda*v2) +    on(outside,u2= 0 ) ;

problem Pb1(u1,v1,init=reuseMatrix,solver=Cholesky) =
    int2d(Th1)( dx(u1)*dx(v1)+dy(u1)*dy(v1) )
  + int2d(Th1)( -v1)
  + int1d(Th1,inside)(+lambda*v1) +    on(outside,u1 = 0 ) ;

varf b(u2,v2,solver=CG) =int1d(Th1,inside)(u2*v2);
matrix B= b(Vh1,Vh1,solver=CG);

//  map lambda -> S1(lambda)-S2(lambda)
func real[int] BoundaryProblem(real[int] &l) {
   lambda[]=l;  // lambda[]= DOF vector of lambda
```

`example35.edp` code, cont'd

```
problem Pb2(u2,v2,init=reuseMatrix,solver=Cholesky) =
    int2d(Th2)( dx(u2)*dx(v2)+dy(u2)*dy(v2) )
  + int2d(Th2)( -v2)
  + int1d(Th2,inside)(-lambda*v2) +    on(outside,u2= 0 ) ;

problem Pb1(u1,v1,init=reuseMatrix,solver=Cholesky) =
    int2d(Th1)( dx(u1)*dx(v1)+dy(u1)*dy(v1) )
  + int2d(Th1)( -v1)
  + int1d(Th1,inside)(+lambda*v1) +    on(outside,u1 = 0 ) ;

varf b(u2,v2,solver=CG) =int1d(Th1,inside)(u2*v2);
matrix B= b(Vh1,Vh1,solver=CG);

//  map lambda -> S1(lambda)-S2(lambda)
func real[int] BoundaryProblem(real[int] &l) {
   lambda[]=l;  // lambda[]= DOF vector of lambda
   Pb1;
   Pb2;
   reuseMatrix=true;
   v1=-(u1-u2);
   lambda[]=B*v1[];
   return lambda[];
}
```

example35.edp code, cont'd

```
Vh1 p=0,q=0;

//  solve the problem with Conjugate Gradient
LinearCG(BoundaryProblem, p[], q[], eps=1.e-6, nbiter=100);

//  compute the final solution, because CG works with increment
BoundaryProblem(p[]); // solve again  to have right u1,u2

cout << " - CPU time  Example 35:" <<  clock()-cpu « endl;

real[int] viso=[0,.02,.04,.06,.08,.10,.12,.14,.16,.18,
                .19,.20,.21,.22,.23,.24,.26,.28];

plot(u1, u2, viso=viso, value=true);
```

Topics

Complex values: waves in a microwave oven

- The field in a microwave oven satifies a Helmholz equation

$$\beta v + \Delta v = 0$$

- Within the food to be cooked, heat source is proportional to v^2.
- At equilibrium

$$-\Delta\theta = v^2 I_B$$

  where I_B is the characteristic function of the food.
- $\beta = 1/(1 - 0.5i)$ in the air and $\beta = 2/(1 - 0.5i)$ in the food
- The power source is along the left wall

example36.edp code

```
real a=20, b=20, c=15, d=8, e=2, l=12, f=2, g=2;
border a0(t=0,1) {x=a*t;      y=0;               label=1;}
border a1(t=1,2) {x=a;        y= b*(t-1);        label=1;}
border a2(t=2,3) {x=a*(3-t); y=b;               label=1;}
border a3(t=3,4) {x=0;        y=b-(b-c)*(t-3); label=1;}
border a4(t=4,5) {x=0;        y=c-(c-d)*(t-4); label=2;}
border a5(t=5,6) {x=0;        y=d*(6-t);       label=1;}

border b0(t=0,1) {x=a-f+e*(t-1); y=g;               label=3;}
border b1(t=1,4) {x=a-f;         y=g+l*(t-1)/3;     label=3;}
border b2(t=4,5) {x=a-f-e*(t-4); y=l+g;             label=3;}
border b3(t=5,8) {x=a-e-f;       y= l+g-l*(t-5)/3; label=3;}

int n=2;
mesh Th = buildmesh(a0(10*n) + a1(10*n) + a2(10*n) + a3(10*n)
     + a4(10*n) + a5(10*n) + b0(5*n) + b1(10*n) + b2(5*n) + b3(10*n));
plot(Th, wait=1);
```

example36.edp code

```
fespace Vh(Th,P1);

real meat = Th(a-f-e/2,g+l/2).region, air= Th(0.01,0.01).region;

Vh R=(region-air)/(meat-air);
```

example36.edp code

```
fespace Vh(Th,P1);

real meat = Th(a-f-e/2,g+1/2).region, air= Th(0.01,0.01).region;

Vh R=(region-air)/(meat-air);

Vh<complex> v, w;
solve muwave(v,w) = int2d(Th)(v*w*(1+R)
                -(dx(v)*dx(w) + dy(v)*dy(w))*(1-0.5i) )
                + on(1, v=0) + on(2, v=sin(pi*(y-c)/(c-d)) );
```

example36.edp code

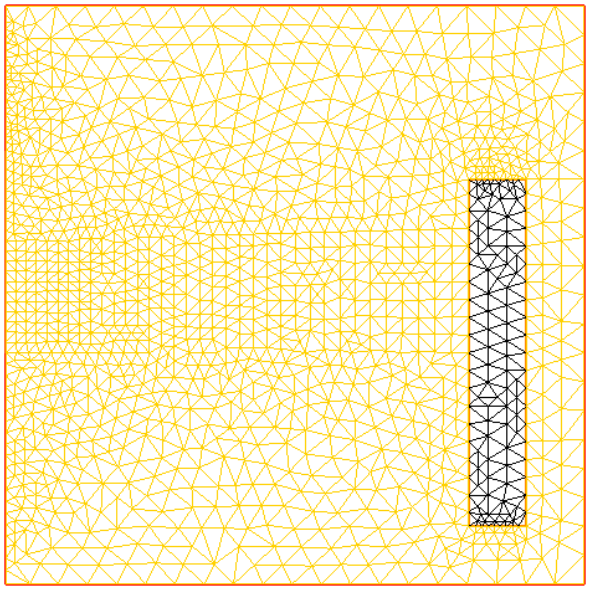
```
fespace Vh(Th,P1);

real meat = Th(a-f-e/2,g+1/2).region, air= Th(0.01,0.01).region;

Vh R=(region-air)/(meat-air);

Vh<complex> v, w;
solve muwave(v,w) = int2d(Th)(v*w*(1+R)
                -(dx(v)*dx(w) + dy(v)*dy(w))*(1-0.5i) )
                + on(1, v=0) + on(2, v=sin(pi*(y-c)/(c-d)) );

Vh vr=real(v), vi=imag(v);
plot(vr, wait=true, fill=true);
plot(vi, wait=true, fill=true);
```

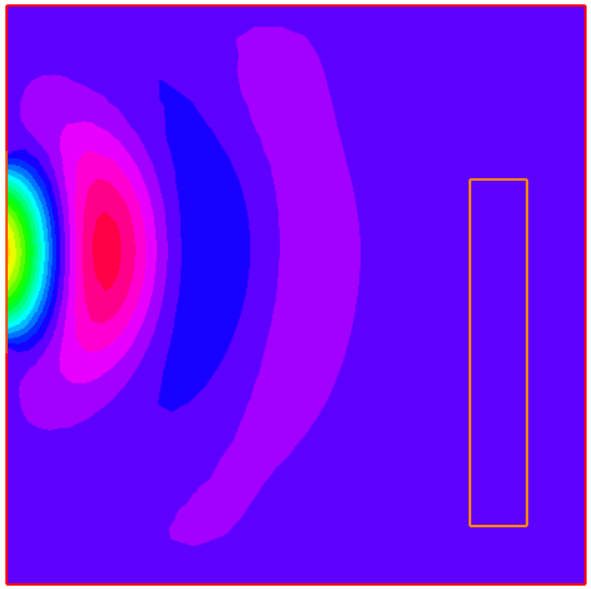
example36.edp code

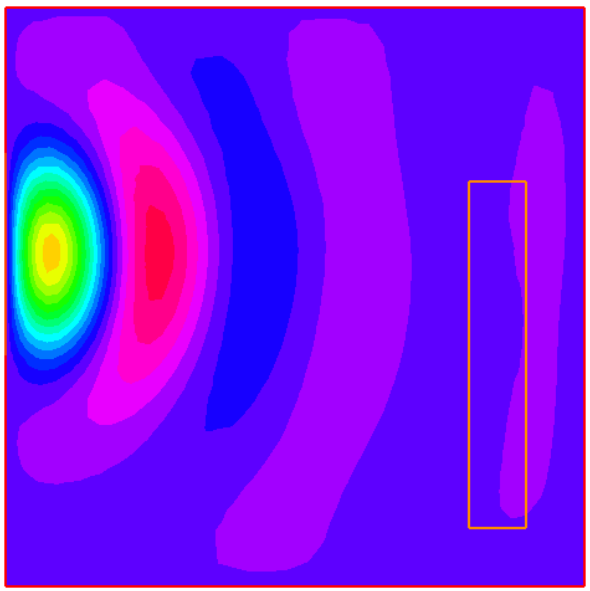
```
fespace Vh(Th,P1);

real meat = Th(a-f-e/2,g+l/2).region, air= Th(0.01,0.01).region;

Vh R=(region-air)/(meat-air);

Vh<complex> v, w;
solve muwave(v,w) = int2d(Th)(v*w*(1+R)
                -(dx(v)*dx(w) + dy(v)*dy(w))*(1-0.5i) )
                + on(1, v=0) + on(2, v=sin(pi*(y-c)/(c-d)) );

Vh vr=real(v), vi=imag(v);
plot(vr, wait=true, fill=true);
plot(vi, wait=true, fill=true);

fespace Uh(Th,P1);
Uh u,uu, ff=1e5*(vr^2 + vi^2)*R;

solve temperature(u,uu)= int2d(Th)( dx(u)*dx(uu) + dy(u)*dy(uu))
                       - int2d(Th)(ff*uu) + on(1,2,u=0);

plot(u,wait=true, fill=true);
```

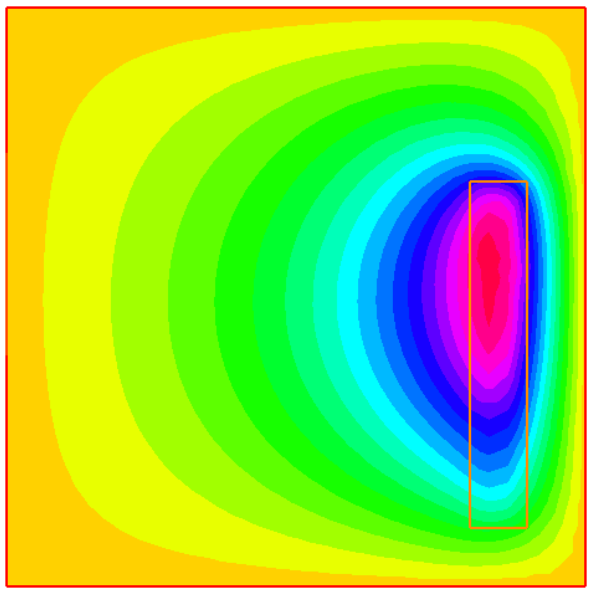
`example36.edp` results

mesh, real part of wave, imaginary part of wave, temperature



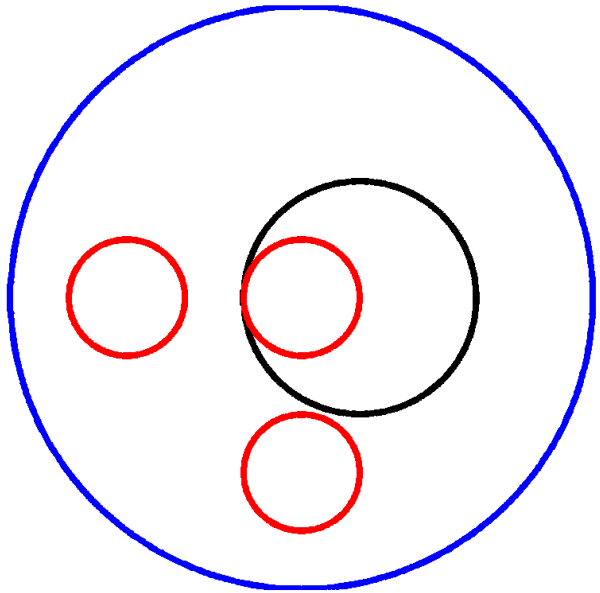






Topics

Example 37, Control problem



$$\min_{b,c,d\in\mathbb{R}} J = \int_E (u - u_d)$$
$$-\nabla(\kappa(b,c,d)\nabla u) = 0$$
$$u|_\Gamma = u_\Gamma$$
$$\kappa(x) = 1 + bI_B(x) + cI_C(x) + dI_D(x)$$

Need derivatives

$$\frac{\partial J}{\partial b} = 2\int_E (u - u_d)$$
$$\nabla(\kappa\nabla\delta u) + \nabla(I_b\nabla u) = 0$$
$$\delta u|_\Gamma = 0$$

example37.edp code

```
border aa(t=0, 2*pi) {    x = 5*cos(t);    y = 5*sin(t); }
border bb(t=0, 2*pi) {    x = cos(t);      y = sin(t);   }
border cc(t=0, 2*pi) {    x = -3+cos(t);   y = sin(t);   }
border dd(t=0, 2*pi) {    x = cos(t);      y = -3+sin(t);}
mesh th = buildmesh(aa(70) + bb(35) + cc(35) + dd(35));
```

example37.edp code

```
border aa(t=0, 2*pi) {    x = 5*cos(t);    y = 5*sin(t); }
border bb(t=0, 2*pi) {    x = cos(t);      y = sin(t);   }
border cc(t=0, 2*pi) {    x = -3+cos(t);   y = sin(t);   }
border dd(t=0, 2*pi) {    x = cos(t);      y = -3+sin(t);}
mesh th = buildmesh(aa(70) + bb(35) + cc(35) + dd(35));

fespace Vh(th,P1);

Vh Ib=((x^2+y^2)<1.0001),
   Ic=(((x+3)^2+ y^2)<1.0001),
   Id=((x^2+(y+3)^2)<1.0001),
   Ie=(((x-1)^2+ y^2)<=4),
   ud,u,uh,du;
```

example37.edp code

```
border aa(t=0, 2*pi) {    x = 5*cos(t);    y = 5*sin(t); }
border bb(t=0, 2*pi) {    x = cos(t);      y = sin(t);   }
border cc(t=0, 2*pi) {    x = -3+cos(t);   y = sin(t);   }
border dd(t=0, 2*pi) {    x = cos(t);      y = -3+sin(t);}
mesh th = buildmesh(aa(70) + bb(35) + cc(35) + dd(35));

fespace Vh(th,P1);

Vh Ib=((x^2+y^2)<1.0001),
   Ic=(((x+3)^2+ y^2)<1.0001),
   Id=((x^2+(y+3)^2)<1.0001),
   Ie=(((x-1)^2+ y^2)<=4),
   ud,u,uh,du;

real[int] z(3);
```

example37.edp code

```
border aa(t=0, 2*pi) {    x = 5*cos(t);    y = 5*sin(t); }
border bb(t=0, 2*pi) {    x = cos(t);      y = sin(t);   }
border cc(t=0, 2*pi) {    x = -3+cos(t);   y = sin(t);   }
border dd(t=0, 2*pi) {    x = cos(t);      y = -3+sin(t);}
mesh th = buildmesh(aa(70) + bb(35) + cc(35) + dd(35));

fespace Vh(th,P1);

Vh Ib=((x^2+y^2)<1.0001),
   Ic=(((x+3)^2+ y^2)<1.0001),
   Id=((x^2+(y+3)^2)<1.0001),
   Ie=(((x-1)^2+ y^2)<=4),
   ud,u,uh,du;

real[int] z(3);

problem A(u,uh) = int2d(th)((1 + z[0]*Ib + z[1]*Ic + z[2]*Id)*
                ( dx(u)*dx(uh) + dy(u)*dy(uh)) ) + on(aa,u=x^3-y^3);
```

example37.edp code

```
border aa(t=0, 2*pi) {    x = 5*cos(t);    y = 5*sin(t); }
border bb(t=0, 2*pi) {    x = cos(t);      y = sin(t);   }
border cc(t=0, 2*pi) {    x = -3+cos(t);   y = sin(t);   }
border dd(t=0, 2*pi) {    x = cos(t);      y = -3+sin(t);}
mesh th = buildmesh(aa(70) + bb(35) + cc(35) + dd(35));

fespace Vh(th,P1);

Vh Ib=((x^2+y^2)<1.0001),
   Ic=(((x+3)^2+ y^2)<1.0001),
   Id=((x^2+(y+3)^2)<1.0001),
   Ie=(((x-1)^2+ y^2)<=4),
   ud,u,uh,du;

real[int] z(3);

problem A(u,uh) = int2d(th)((1 + z[0]*Ib + z[1]*Ic + z[2]*Id)*
                ( dx(u)*dx(uh) + dy(u)*dy(uh)) ) + on(aa,u=x^3-y^3);

// construct target to aim at
z[0]=2;   // target value z[0]=b
z[1]=3;   // target value z[1]=c
z[2]=4;   // target value z[2]=d
A;
ud = u;   // fix the target function
```

example37.edp code, cont'd

```
ofstream f("J.txt");
```

example37.edp code, cont'd

```
ofstream f("J.txt");
func real J(real[int] & Z){
    for (int i=0; i<z.n; i++){
      z[i]=Z[i];  // copy to global variable
    }
    A;
    real s= int2d(th)( Ie * (u-ud)^2 );
    f<<s<<"   ";  // save results for later examination
    return s;
}
```

`example37.edp` code, cont'd

```
ofstream f("J.txt");
func real J(real[int] & Z){
    for (int i=0; i<z.n; i++){
      z[i]=Z[i];  // copy to global variable
    }
    A;
    real s= int2d(th)( Ie * (u-ud)^2 );
    f<<s<<"   ";  // save results for later examination
    return s;
}

real[int] dz(3), dJdz(3);

problem B(du, uh) = int2d(th)( (1 + z[0]*Ib + z[1]*Ic + z[2]*Id)*
                               (dx(du)*dx(uh) + dy(du)*dy(uh)) )
  +int2d(th)( (dz[0]*Ib + dz[1]*Ic + dz[2]*Id)*
              (dx(u)*dx(uh) + dy(u)*dy(uh)) )
  +on(aa, du=0);
```

`example37.edp` code, cont'd

```
ofstream f("J.txt");
func real J(real[int] & Z){
    for (int i=0; i<z.n; i++){
      z[i]=Z[i];  // copy to global variable
    }
    A;
    real s= int2d(th)( Ie * (u-ud)^2 );
    f<<s<<"   ";  // save results for later examination
    return s;
}

real[int] dz(3), dJdz(3);

problem B(du, uh) = int2d(th)( (1 + z[0]*Ib + z[1]*Ic + z[2]*Id)*
                               (dx(du)*dx(uh) + dy(du)*dy(uh)) )
  +int2d(th)( (dz[0]*Ib + dz[1]*Ic + dz[2]*Id)*
              (dx(u)*dx(uh) + dy(u)*dy(uh)) )
  +on(aa, du=0);

func real[int] DJ(real[int] &Z) {
      for(int i=0; i<z.n; i++) {
          for(int j=0; j<dz.n; j++){
            dz[j]=0;
          }
          dz[i]=1;
          B;
          dJdz[i]= 2*int2d(th)( Ie*(u-ud)*du );
      }
      return dJdz;
}
```

`example37.edp` code, cont'd

```
real[int] Z(3);
for(int j=0; j<z.n; j++){
   Z[j]=1;  // initial guess
}
```

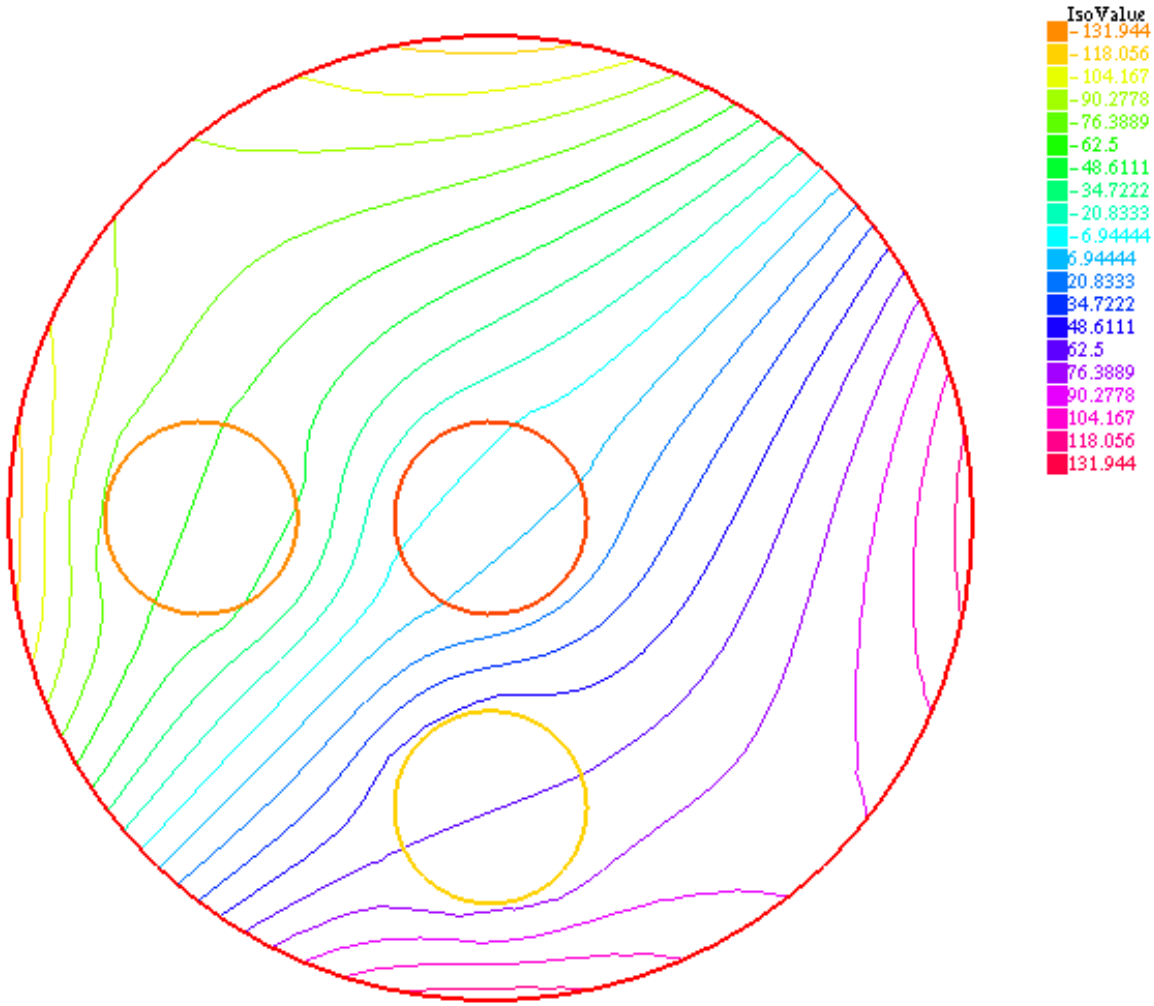
`example37.edp` code, cont'd

```
real[int] Z(3);
for(int j=0; j<z.n; j++){
   Z[j]=1;  // initial guess
}
BFGS(J, DJ, Z, eps=1.e-6, nbiter=15, nbiterline=20);
```

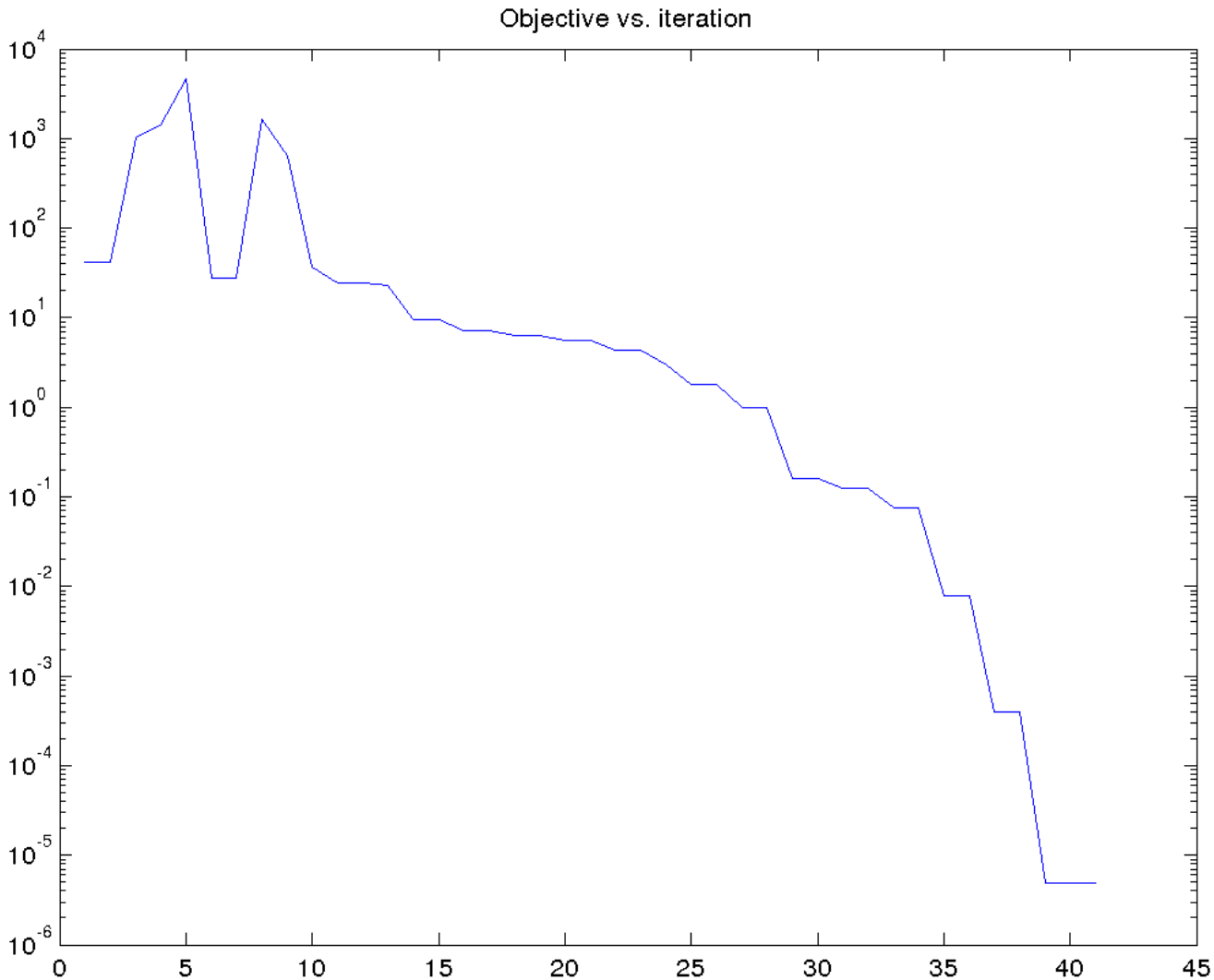
`example37.edp` code, cont'd

```
real[int] Z(3);
for(int j=0; j<z.n; j++){
   Z[j]=1;  // initial guess
}
BFGS(J, DJ, Z, eps=1.e-6, nbiter=15, nbiterline=20);

cout « "BFGS: J(z) = " « J(Z) «  endl;
for(int j=0; j<z.n; j++){
   cout<< z[j] <<endl;
}
plot(ud,value=true);
```

`example37.edp` final u_d



`example37.edp`, iteration history of J



Topics

Over-simplified compressible flow

- Conservation of mass and conservation of momentum (NSE)

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u + \nabla \rho = 0$$

$$\frac{\partial \rho}{\partial t} + (u \cdot \nabla)\rho + \nabla \cdot u = 0$$

$$u|_{\Gamma_D} = u_0$$

$$\rho|_{\Gamma_D} = \rho_0$$

- Supersonic flow generates shocks
- Shock resolution generally involves special techniques
- Mesh refinement can do the job

Example 38, compressible flow with mesh refinement

- Solve compressible flow system several time steps
- Remesh based on ρ solution
- Repeat several times
- Restart if necessary

example38.edp code

```
verbosity = 1;
bool anew = true;  // fresh start (false means restart)
bool savefiles = false;
real x0=0.5, y0=0, rr=0.2;
```

example38.edp code

```
verbosity = 1;
bool anew = true;  // fresh start (false means restart)
bool savefiles = false;
real x0=0.5, y0=0, rr=0.2;

border ccc(t=0,2){ x=2-t; y=1;}
border ddd(t=0,1){ x=0; y=1-t;}  // second border is no. 2
border aaa1(t=0,x0-rr){ x=t; y=0;}
border circle(t=pi,0){ x=x0+rr*cos(t); y=y0+rr*sin(t);}
border aaa2(t=x0+rr,2){ x=t; y=0;}
border bbb(t=0,1){ x=2; y=t;}
```

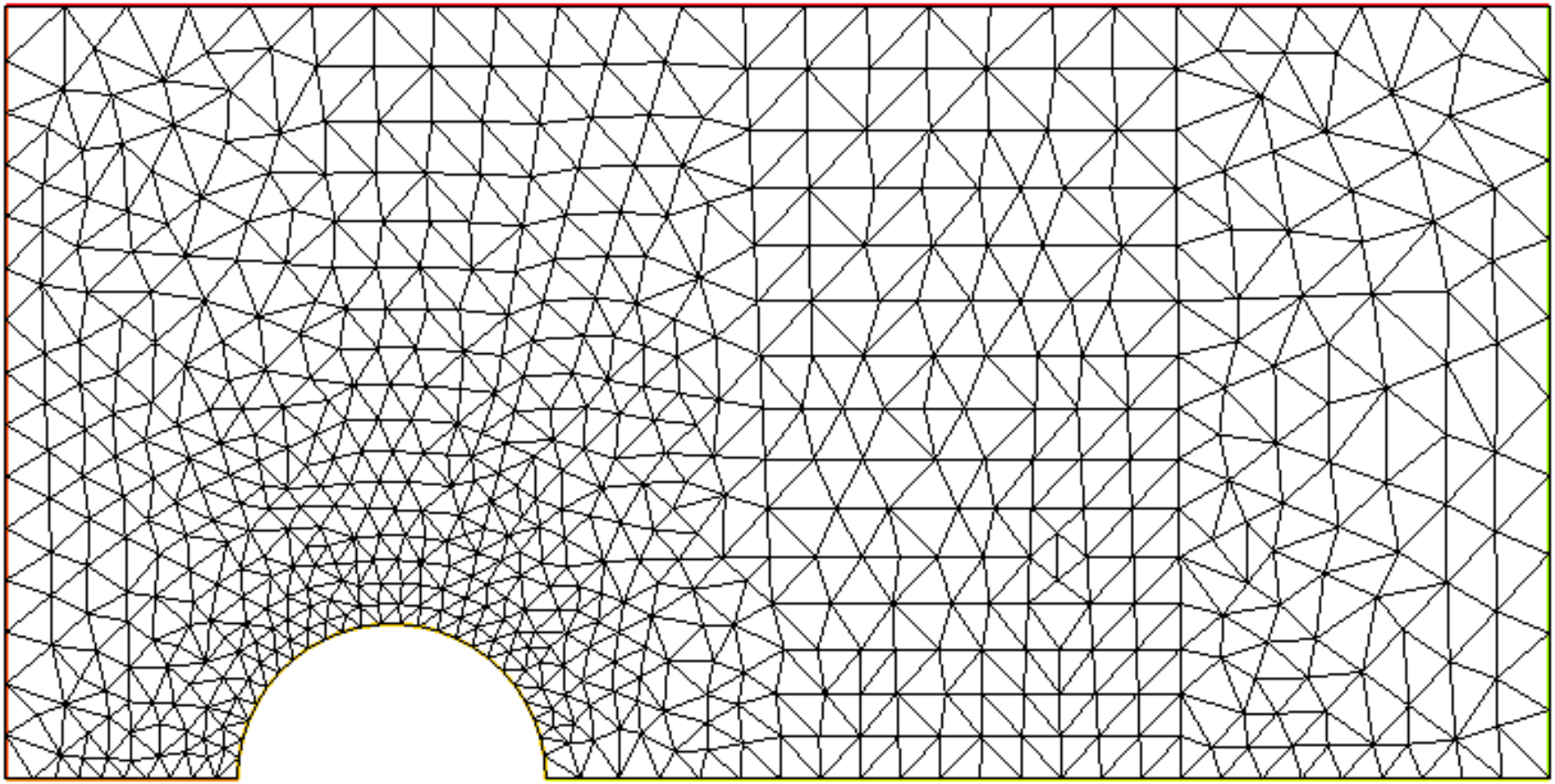
example38.edp code

```
verbosity = 1;
bool anew = true;  // fresh start (false means restart)
bool savefiles = false;
real x0=0.5, y0=0, rr=0.2;

border ccc(t=0,2){ x=2-t; y=1;}
border ddd(t=0,1){ x=0; y=1-t;}  // second border is no. 2
border aaa1(t=0,x0-rr){ x=t; y=0;}
border circle(t=pi,0){ x=x0+rr*cos(t); y=y0+rr*sin(t);}
border aaa2(t=x0+rr,2){ x=t; y=0;}
border bbb(t=0,1){ x=2; y=t;}

int m=5;
mesh Th;
if(anew){
  Th = buildmesh (ccc(5*m) + ddd(3*m) + aaa1(2*m) + cercle(5*m)
                + aaa2(5*m) + bbb(2*m) );
} else {
  Th = readmesh("Th_circle.mesh");
  plot(Th,wait=0);
}
```

example38.edp code

```
verbosity = 1;
bool anew = true;  // fresh start (false means restart)
bool savefiles = false;
real x0=0.5, y0=0, rr=0.2;

border ccc(t=0,2){ x=2-t; y=1;}
border ddd(t=0,1){ x=0; y=1-t;}  // second border is no. 2
border aaa1(t=0,x0-rr){ x=t; y=0;}
border circle(t=pi,0){ x=x0+rr*cos(t); y=y0+rr*sin(t);}
border aaa2(t=x0+rr,2){ x=t; y=0;}
border bbb(t=0,1){ x=2; y=t;}

int m=5;
mesh Th;
if(anew){
  Th = buildmesh (ccc(5*m) + ddd(3*m) + aaa1(2*m) + cercle(5*m)
                + aaa2(5*m) + bbb(2*m) );
} else {
  Th = readmesh("Th_circle.mesh");
  plot(Th,wait=0);
}

real dt=0.01, u0=2, err0=0.00625;
fespace Wh(Th,P1);
fespace Vh(Th,P1);
Wh u, v, u1, v1, uh, vh;
Vh r, rh, r1;
```

`example38.edp` initial mesh



`example38.edp` code, cont'd

```
macro dn(u) (N.x*dx(u) + N.y*dy(u) ) //  def the normal derivative
```

example38.edp code, cont'd

```
macro dn(u) (N.x*dx(u) + N.y*dy(u) ) //  def the normal derivative

if(anew){
  u1= u0;
  v1= 0;
  r1 = 1;
} else {
  ifstream g("u.txt");
  g » u1[];
  ifstream gg("v.txt");
  gg » v1[];
  ifstream ggg("r.txt");
  ggg » r1[];
  plot(u1, value=true ,wait=true);
  err0 = err0/10;
  dt = dt/10;
}
```

example38.edp code, cont'd

```
macro dn(u) (N.x*dx(u) + N.y*dy(u) ) //  def the normal derivative

if(anew){
  u1= u0;
  v1= 0;
  r1 = 1;
} else {
  ifstream g("u.txt");
  g » u1[];
  ifstream gg("v.txt");
  gg » v1[];
  ifstream ggg("r.txt");
  ggg » r1[];
  plot(u1, value=true ,wait=true);
  err0 = err0/10;
  dt = dt/10;
}

problem  eul(u,v,r,uh,vh,rh)
   = int2d(Th)(  (u*uh + v*vh + r*rh)/dt
               + ((dx(r)*uh+ dy(r)*vh) - (dx(rh)*u + dy(rh)*v)) )
   + int2d(Th)(-(rh*convect([u1,v1],-dt,r1) +
        uh*convect([u1,v1],-dt,u1) + vh*convect([u1,v1],-dt,v1))/dt)
   + int1d(Th,6)(rh*u)
   + on(2,r=0) + on(2,u=u0) + on(2,v=0);
```

`example38.edp` code, cont'd

```
int remesh=80;
for(int k=0;k<3;k++) {
  if(k==20){
    err0 = err0/10;
    dt = dt/10;
    remesh = 5;
  }
```

example38.edp code, cont'd

```
int remesh=80;
for(int k=0;k<3;k++) {
  if(k==20){
    err0 = err0/10;
    dt = dt/10;
    remesh = 5;
  }
  for(int i=0;i<remesh;i++){
     eul;
     u1=u;
     v1=v;
     r1=abs(r);
     cout«"k="« k «"  E="« int2d(Th)(u^2+v^2+r)«endl;
     plot(r, wait=false, value=true);
  }
```

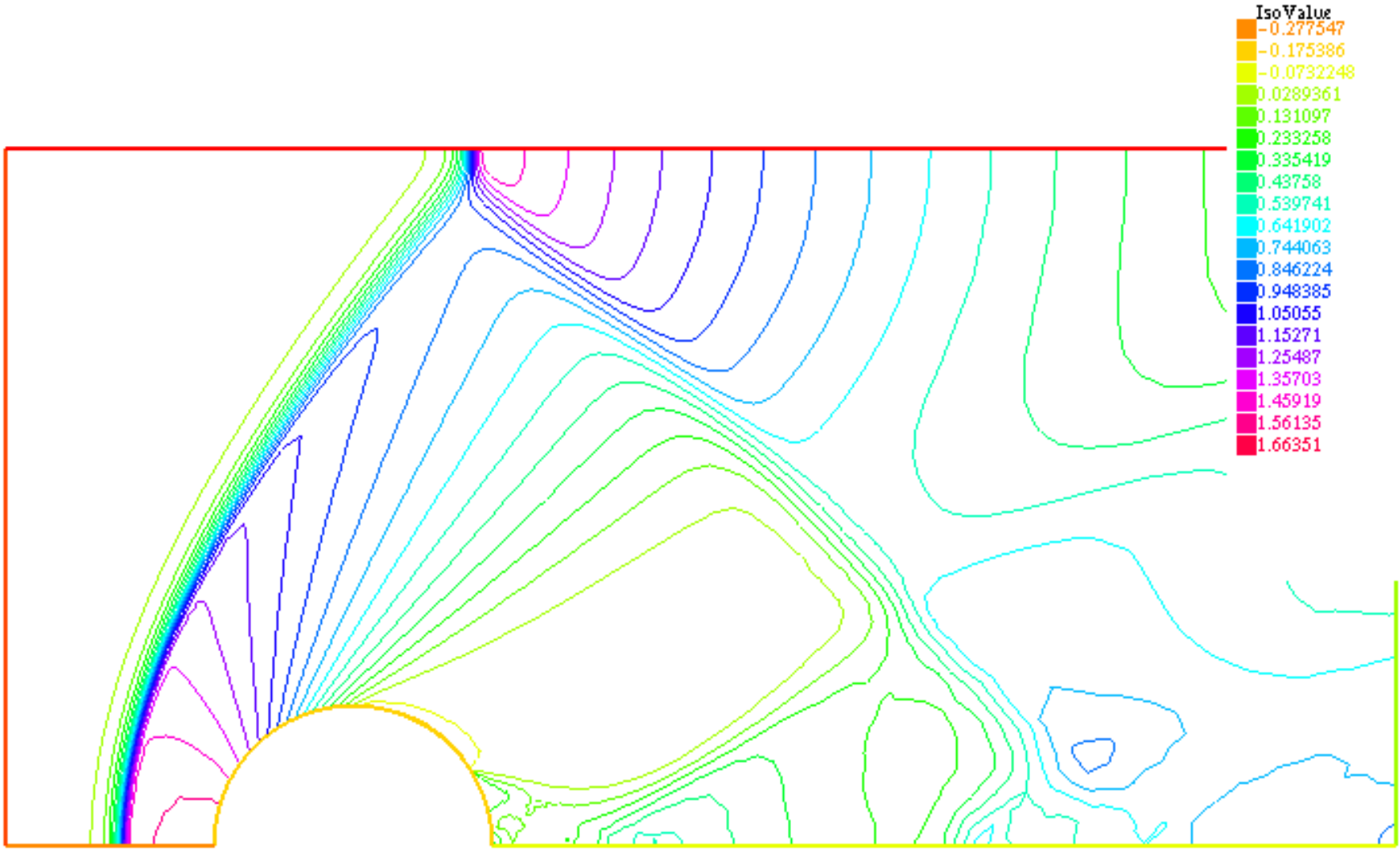
`example38.edp` code, cont'd

```
int remesh=80;
for(int k=0;k<3;k++) {
  if(k==20){
    err0 = err0/10;
    dt = dt/10;
    remesh = 5;
  }
  for(int i=0;i<remesh;i++){
     eul;
     u1=u;
     v1=v;
     r1=abs(r);
     cout«"k="« k «"  E="« int2d(Th)(u^2+v^2+r)«endl;
     plot(r, wait=false, value=true);
  }
  Th = adaptmesh (Th, r, nbvx=40000, err=err0, abserror=true,
       nbjacoby=2, omega=1.8, ratio=1.8, nbsmooth=3,
       splitpbedge=true, maxsubdiv=5, rescaling=true) ;
  plot(Th, wait=0);
  // interpolate solution from old to new mesh
  u = u;
  v = v;
  r = r;
```

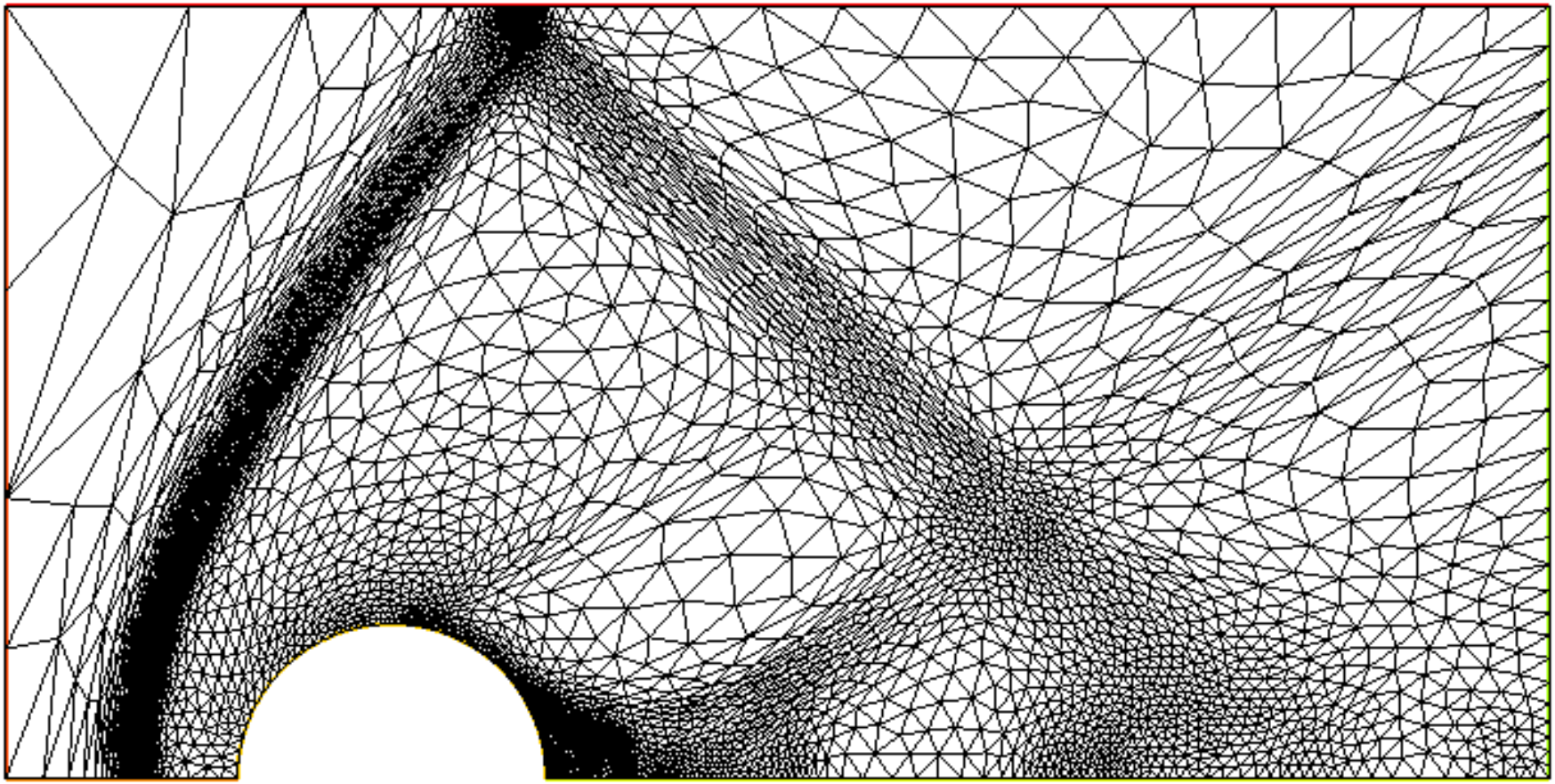
example38.edp code, cont'd

```
  if (savefiles) {
    savemesh(Th,"Th_circle.mesh");

    ofstream f("u.txt");
    f << u[];

    ofstream ff("v.txt");
    ff << v[];

    ofstream fff("r.txt");
    fff << r[];

    r1 = sqrt(u*u+v*v);
    plot(r1, value=true);
    r1 = r;
  }
}
```

`example38.edp` final density



`example38.edp` final mesh



Topics

Mixed formulation, RT elements

- Poisson problem: find p

$$\begin{aligned} -\Delta p &= f \text{ in } \Omega \\ p &= g_d \text{ on } \Gamma_D \\ \frac{\partial p}{\partial n} &= g_n \text{ on } \Gamma_N \end{aligned}$$

- Reformulate by defining $u = \nabla p$
- New: find p and *vector* u

$$\begin{aligned} -\nabla p + u &= 0 \text{ in } \Omega \\ \nabla \cdot u &= f \text{ in } \Omega \\ p &= g_d \text{ on } \Gamma_D \\ u \cdot n &= g_n \text{ on } \Gamma_N \end{aligned}$$

Weak form

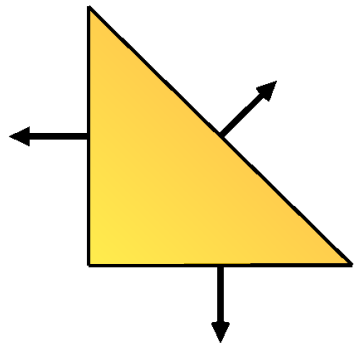
$$\int_\Omega p\nabla \cdot v + u \cdot v = \int_{\Gamma_D} g_d v \cdot n$$

$$\int_\Omega q\nabla \cdot u = \int_\Omega qf$$

$$u \cdot n = g_n \text{ on } \Gamma_N$$

- p needs to be in L^2: P0 element
- u needs to be in $H(\text{div})$: RT0 element

RT0 element



- FEniCS calls this RT1, FreeFem++ calls it RT0
- Vector-valued shape functions of the form $v(x) = \vec{\alpha} + \beta\vec{x}$
- $\vec{\alpha}$ is a vector, β is a scalar
- Not continuous, but mid-side normal is.

example39.edp code

```
mesh Th=square(100,100);
fespace Vh(Th,RT0);
fespace Ph(Th,P0);

Vh [u1,u2],[v1,v2];
Ph p,q;
```

example39.edp code

```
mesh Th=square(100,100);
fespace Vh(Th,RT0);
fespace Ph(Th,P0);

Vh [u1,u2],[v1,v2];
Ph p,q;

func gd = 1.;
func gn = 1.;
func f = 1.;
```

example39.edp code

```
mesh Th=square(100,100);
fespace Vh(Th,RT0);
fespace Ph(Th,P0);

Vh [u1,u2],[v1,v2];
Ph p,q;

func gd = 1.;
func gn = 1.;
func f = 1.;

problem laplaceMixed([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
 + on(4, u1 = gn*N.x, u2 = gn*N.y);
```

example39.edp code

```
mesh Th=square(100,100);
fespace Vh(Th,RT0);
fespace Ph(Th,P0);

Vh [u1,u2],[v1,v2];
Ph p,q;

func gd = 1.;
func gn = 1.;
func f = 1.;

problem laplaceMixed([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
 + on(4, u1 = gn*N.x, u2 = gn*N.y);
```

example39.edp code

```
mesh Th=square(100,100);
fespace Vh(Th,RT0);
fespace Ph(Th,P0);

Vh [u1,u2],[v1,v2];
Ph p,q;

func gd = 1.;
func gn = 1.;
func f = 1.;

problem laplaceMixed([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
 + on(4, u1 = gn*N.x, u2 = gn*N.y);

laplaceMixed;
```

example39.edp code

```
mesh Th=square(100,100);
fespace Vh(Th,RT0);
fespace Ph(Th,P0);

Vh [u1,u2],[v1,v2];
Ph p,q;

func gd = 1.;
func gn = 1.;
func f = 1.;

problem laplaceMixed([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
 + on(4, u1 = gn*N.x, u2 = gn*N.y);

laplaceMixed;

plot([u1,u2], coef=0.1, wait=true, value=true);
plot(p, fill=true, wait=true, value=true);
```

`example39.edp` pressure