Incompressible NSE

\[ \frac{\partial u}{\partial t} + u \cdot \nabla u - \nu \Delta u + \nabla p = f \]
\[ \nabla \cdot u = 0 \]

\[ \begin{array}{c}
L \\
T \\
B
\end{array} \]

\[ \begin{array}{c}
L_1 \\
R_1 \\
B_1
\end{array} \]

\[ T \]

\[ R_1 \]

\[ L \]

\[ B \]

**example18.edp** Vortex shedding past a square

**example18.edp** Code outline

1. Generate/plot geometry
2. Generate/plot mesh
3. Specify finite element spaces and instantiate variables
4. Construct weak form with b.c.
5. Time loop
   5.1 Update time-dependent b.c.
   5.2 Solve
   5.3 Plot
Some syntax (C++)

- Comments preceded by `//`
  - Can also use `/* ... */`
  - Good for multiline comments
- Commands mostly end with semicolon
  - No need for a line continuation character
- Extra spaces and indentation don’t matter
- “Curly brackets” (`{}`) used for grouping
- Variables must have a specified type
- Single quotes for characters (‘a’)
- Double quotes for strings ("Hello there")

```cpp
// Example 18: Vortex shedding past a square
real L = 0.0, R = 2.0, B = 0.0, T = 1.0;
real L1 = 0.2, R1 = 0.4, B1 = 0.4, T1 = 0.6;
int nx = 20, ny = 10,
int nx1 = 5, ny1 = 5;

// outer square
border Bottom(t=0,1) {x = L + (R-L)*t; y = B; label = 1;};
border Top(t=0,1) {x = L + (R-L)*t; y = T; label = 3;};
border Left(t=0,1) {x = L; y = B + (T-B)*t; label = 4;};
border Right(t=0,1) {x = R; y = B + (T-B)*t; label = 2;};

// obstacle
border ObsBottom(t=0,1) {x = L1 + (R1-L1)*t; y = B1; label = 5;};
border ObsTop(t=0,1) {x = L1 + (R1-L1)*t; y = T1; label = 7;};
border ObsLeft(t=0,1) {x = L1; y = B1 + (T1-B1)*t; label = 8;};
border ObsRight(t=0,1) {x = R1; y = B1 + (T1-B1)*t; label = 6;};

// plot geometry
plot( Bottom(nx) + Right(ny) + Top(-nx) + Left(-ny)
 + ObsBottom(-nx1) + ObsRight(-ny1) + ObsTop(+nx1)
 + ObsLeft(+ny1), wait=1 );

// generate mesh
mesh Th = buildmesh( Bottom(nx) + Right(ny) + Top(-nx) + Left(-ny)
 + ObsBottom(-nx1) + ObsRight(-ny1) + ObsTop(+nx1) + ObsLeft(+ny1) );

// plot mesh
plot(Th, wait=1, ps="VortexStreetMesh.eps");
```
```
real nu=1./1000., dt=0.05, bndryVelocity;
problem NS ([u1, u2, p] , [v1, v2, q] , init= reuseMatrix) =
  int2d(Th)( 1./dt * ( u1*v1 + u2*v2 )
  + nu * ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
  + dx(u2)*dx(v2) + dy(u2)*dy(v2) )
  + p*q*(0.000001) + dx(u1)*q + dy(u2)*q )
  + int2d(Th) ( -1./dt*convect( [up1, up2] , -dt , up1) * v1
  -1./dt*convect( [up1, up2] , -dt , up2) * v2 )

// b.c.: uniform velocity top, bottom, inlet (left)
// "do nothing" on exit (right)
+ on(1, u1=bndryVelocity, u2=0)
+ on(3, u1=bndryVelocity, u2=0)
+ on(4, u1=bndryVelocity, u2=0)
+ on(5, 6, 7, 8, u1=0, u2=0);
```

**What is convect?**

- Section 9.5.2 of the FreeFem++ book, p. 240ff.
- Note that
\[
\frac{\partial w}{\partial t} + u \cdot \nabla w = \frac{dw(X(t),t)}{dt}
\]
where
\[
\frac{dX(t)}{dt} = u(X(t),t)
\]
- Given a point \(x\), find the solution of the “final value problem”
\[
\frac{dX}{dt} = u^m(X(t)), \text{ with } X(t^{m+1}) = x,
\]
- Then
\[
\frac{dw(X(t),t)}{dt} \approx \frac{w(X(t^{m+1}),t^{m+1}) - w(X(t^m),t^m)}{\Delta t}
= \frac{w^{m+1} - w(X(t^m),t^m)}{\Delta t}
\]
- How to get \(w(X(t^m),t^m)\)?
The “final value problem” for convect?

- Assume \( w^m(x) \) is a known function of \( x \)
- Assume \( u(x, t) \approx u^m(x) \) is constant over the interval \( t^m \leq t < t^{m+1} \)
- \( X(t^{m+1}) - X(t^m) \approx \Delta t u^m(X(t^{m+1})) \)
- Solving, \( x(t^m) = x - u^m(x) \Delta t \)
- \( w(X(t^m), t^m) = w(x - u^m \Delta t) \)
- \( \text{convect}(u, -dt, w)(x) = w(x - u \ dt) \)

That sounds great! Why doesn’t everyone do it?

- Those approximations are \( O(\Delta t) \)
- What if the flow is fast enough so \( x - u \ dt \) is out of the element?
- What if the flow is fast enough so \( x - u \ dt \) is out of \( \Omega \)?
- Hard to prove things about.
- Stability?

More syntax

- Declare a real vector: \( \text{real}[\text{int}] \ v(100); \)
- Looping
  ```
  for (int index=0 ; index < maximum ; i++) {
      ... code ...
  }
  ```
- Can use an already-declared index variable.
- \( \text{index} \) is no longer “in scope” after the loop ends

Scoping, blocking and variable declarations

- Variables can be declared anywhere in the program
- “Blocks” of code are inside curly brackets: \( \{ \ldots \} \)
- Declarations inside a block:
  - Are not known outside the block.
  - “Cover up” previous declarations
Asking questions: if

boolean expression ? true expression : false expression

if ( boolean expression ){
    ... code ...
}

if ( boolean expression ){
    ... code ...
} else if {
    ... code ...
} else {
    ... code ...
}

A single statement does not need to be inside a block.

Boolean expressions

Comparison operators

<table>
<thead>
<tr>
<th>operator</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
</tbody>
</table>

Logical operators

<table>
<thead>
<tr>
<th>operator</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Do not use & or |! They are bitwise operators.

Remark: Logical operators are “short circuit”

Exercise 22 (10 points)

Example 18 presents a vortex-shedding problem using the "convect" function in FreeFem++. Convert the code to use the conventional form of the convection terms. You should observe vortex shedding behavior in your formulation, although details may differ. Explain any differences you think are important between the two solutions.
Comparison with FEniCS code: boundary conditions

- FreeFem++ code goes into the weak form
  + on(1, u1=bdryVelocity, u2=0)
  + on(3, u1=bdryVelocity, u2=0)
  + on(4, u1=bdryVelocity, u2=0)
  + on(5, 6, 7, 8, u1=0, u2=0)

- FEniCS code requires a mesh function
  ```python
  boundaries = MeshFunction("size_t", mesh, mesh.topology().dim()-1)
  ```

- FEniCS code requires new classes
  ```python
  class InflowBoundary(SubDomain):
  def inside(self, x, on_boundary):
      return on_boundary and x[0] < xmin + bmarg
  ```

- FEniCS requires instantiation
  ```python
  inflowBoundary = InflowBoundary()
  g1 = Expression( "4.*Um*(x[1]*(ymax-x[1]))/(ymax*ymax)" , Um=1.5, ymax=ymax)
  bc1 = DirichletBC(W.sub(0), g1, inflowBoundary)
  ```

- FEniCS then has the b.c.s passed to the solver with the matrices.
  ```python
  solve(NSE == LNSE, w, bcs)
  ```

Comparison with FEniCS code: functions and spaces

- FEniCS:
  ```python
  V = VectorFunctionSpace(mesh, "CG", 2)
  Q = FunctionSpace(mesh, "CG", 1)
  W = V * Q
  ```

- FreeFem++
  ```python
  fespace Xh( Th, P2);
  fespace Mh( Th, P1);
  Xh u2, v2, u1, v1, up1, up2;
  Mh p, q;
  ```

Comparison with FEniCS code: Weak form

- FEniCS
  ```python
  LNSE = inner(u0,v)*dx
  NSE = (inner(u,v) + dt*(inner(grad(u)*u0,v) + nu*inner(grad(u),grad(v)) - div(v)*p + q*div(u) ) + dx
  ```

- FreeFem++
  ```python
  problem NS ([u1, u2, p], [v1, v2, q] =
  int2d(Th) (1./dt*(u1*v1 + u2*v2) +
  u1*inner(grad(u),grad(v)) - div(v)*p + q*div(u) ) + dx
  ```

I'd like to print some stuff.

- Printing uses cout and `<<`
- Add lines inside the loop
  ```python
  cout << "t" << times[i] << " umax="
  << max(u1[].max, u2[].max)
  << " vertical velocity=" << u2(0.5, 0.5) << endl;
  ```
Can I put some stuff into a file?
▶ Place file definition and use inside a block
▶ File is closed when it goes out of scope

```cpp
ofstream vels("vels.txt");
for (i=0 ; i < numTSteps ; i++) {
    // ramp up velocity from 0.0
    bndryVelocity = i*dt;
    if (bndryVelocity >= 1.0){
        bndryVelocity = 1.0;
    }
    up1 = u1;
    up2 = u2;
    // solve the problem
    NS;
    // reuse the matrix in the rest of the iterations
    reuseMatrix = true;
    // write a 2-column file (t,velocity)
    vels << i*dt << " " << u2(0.5, 0.5) << endl;
}
} // file is closed
```

---

**example19.edp** Poisson's equation on a circle

\[ -\Delta u(x, y) = f(x, y) \text{ inside } \Omega \]
\[ u(x, y) = 0 \text{ on } \partial \Omega \]

where \( \Omega \) is the unit circle in \( \mathbb{R}^2 \).

---

**example19.edp**

```cpp
// defining the boundary
border C(t=0,2*pi){x=cos(t); y=sin(t);}
// the triangulated domain Th is on the left side of its boundary
// because boundary parameterization is CCW

// mesh based on 50 t-increments
mesh Th = buildmesh (C(50));
```

**Remark:** Mesh boundary is polygonal, not curved. \( P^2 \) elements will have extra boundary nodes off the circle.
Some more syntax

- Book Chapter 4
- Variables: real, int, bool, complex
- Variable names: letters and numbers, no _
- File variables: ofstream, ifstream
- Global variables: cout, cin, true, false, pi, i
- Arrays: real[int]

Global variables (used mainly in weak forms)

- For current point
  - x, y, z
  - label (boundary point label, 0 if not boundary)
  - region
    - P, P.x, P.y, P.z
    - N, N.x, N.y, N.z
- lenEdge length of current edge
- hTriangle size of current triangle
- area area of current triangle
- volume volume of current triangle
- nuTriangle number (int) of current triangle
- nuEdge number (int) of current edge
- nuTonEdge number (int) of adjacent triangle

Operations and functions

- Usual arithmetic operators
- Raise to power ^
- Usual elementary functions (sin, acosh, etc.)
- randres53 generates uniform reals in [0, 1) with 53-bit resolution

Functions

- func declares a function
- Simple example
  - func u = x^2 + sin(pi*y)
- With declarations
  - func real u( int k){
    return 3.0*k*k;
  }
- With arrays
  - func real[int] sqarr( int[int] L, int n){
    real[int] ans(n);
    for (int k=0; k<n ; k++){
      ans[k]=L[k]^2;
    }
    return ans;
  }
- Examples → tutorial/func.edp
Exercise 23 (5 points)

The following example is presented in Chapter 4.

```plaintext
mesh Th=square(20,20,[-pi+2*pi*x,-pi+2*pi*y]); // [-pi,pi] X [-pi,pi]
fespace Vh(Th,P2);
func z=x+y*1i; // x+iy
func g=abs(sin(z/10)*exp(z^2/10)); // complex arguments
Vh fh = f;
plot(fh); // contours of f
Vh gh = g;
plot(gh); // contours of g
```

- Copy this to a file, run FreeFem++, and send me the plots.
- Change the mesh so that the base of the square is oriented at an angle of $\pi/4$ instead of being horizontal. Send me the changed code and the resulting plots.

---

Array operations

```plaintext
int i;
real [int] tab(10), tab1(10); // 2 array of 10 real
complex [int] ctab(10), ctab1(10); // 2 array of 10 complex

tab = 1; // set all the array to 1
tab[1] = 2;
tab = 1+2i; // set all the array to 1+2i
tab[1] = 2;
tab <<< tab[1] <<< " " <<< tab[9] <<< " size = " <<< tab.n <<< endl;
```

Yields as output

```plaintext
tab: 2 1 size = 10
ctab: (2,0) (1,2) size = 10
```

---

Array operations, cont’d

```plaintext
ctab1 = ctab;
tab = tab + tab1;
tab = 2 *tab + tab1*5;
tab1 = 2*tab - tab1*5;
tab += tab;
```

Yields as output

```plaintext
whole array tab = 10
18 36 18 18 18
18 18 18 18 18
```

---

Array operations, cont’d

```plaintext
tab1 = tab;
tab = tab + tab1;
tab = 2 *tab + tab1*5;
tab1 = 2*tab - tab1*5;
tab += tab;
```

Yields as output

```plaintext
whole array ctab = 10
(18,36) (36,0) (18,36) (18,36) (18,36)
(18,36) (18,36) (18,36) (18,36) (18,36)
ctab[1] = (36,0) ctab[9] = (18,36)
```
real [string] map; // a dynamically-sized array
map["1"] = 2.0;
map[2] = 3.0; // 2 is automatically cast to the string "2"
cout << " map["1"] = " << map["1"] << " == 2.0 ; " << endl;
assert( abs(map["1"] - 2.0) < 1.e-6);
assert( abs(map[2] - 3.0) < 1.e-6);
Yields as output
map["1"] = 2 == 2.0 ;
map[2] = 3 == 3.0

real [int] tab2 = [1,2,3,3.14];
int [int] itab2 = [1,2,3,5];
cout << "Length of array tab2 = " << tab2.n << endl;
cout << "Whole array tab2 = " << tab2 << endl;
cout << "Whole array itab2 = " << itab2 << endl;

tab2.resize(10);
for (int i=4; i<tab2.n; i++){
    tab2[i]=i;
}
cout << "Whole resized array tab2 = " << tab2 << endl;
tab2 /= 2;
cout << "Whole array tab2/2 = " << tab2 << endl;
tab2 *= 2;
cout << "Whole array tab2*2 = " << tab2 << endl;

real [int, int] mat(5,5), mmat(5,5);
mat=0;
for(int i=0; i< mat.n; i++){
    for(int j=0; j< mat.m; j++){
        mat(i,j) = i + 100*(j + 1);
    }
}
cout << "Expanded mat array = " << mat << endl;

for(int i=5; i<mat.n ;i++){
    for(int j=0; j<mat.m ;j++){
        mat(i,j) = i + 100*(j + 1);
    }
}

mmat=mat;
cout << "mmat 2D array = " << mmat << endl;

mat.resize(10,10);
// add new rows
for(int i=5; i<mat.n ;i++){
    for(int j=0; j<mat.m ;j++){
        mat(i,j) = i + 100*(j + 1);
    }
}

// add new columns
for(int i=0;i<mat.n;i++){
    for(int j=5;j<mat.m;j++){
        mat(i,j) = i + 100*(j + 1);
    }
}
cout << "Expanded mat array = " << mat << endl;
Example 20: Section 3.1

\[ -\Delta \phi = f \text{ in } \Omega \]

- Elastic membrane \( \Omega \)
- Rigid support \( \Gamma \), may have vertical displacement
- \( \Gamma \) is ellipse
- Load \( f \)
- Solving for vertical displacement, \( \phi \)
- Membrane glued to \( \Gamma \): Dirichlet b.c.
- Membrane free at \( \Gamma \): Neumann b.c.
- New:
  - Both Dirichlet and Neumann b.c.
  - Accessing values from mesh and solution
  - Write a plot file for gnuplot

```
// example20.edp
// original file: membrane.edp
real theta = 4.*pi/3.;
real a = 2., b = 1.; // semimajor and semiminor axes
func bndryelev = x; // elevation of Gamma1 boundary
border Gamma1(t=0, theta) { x = a * cos(t); y = b*sin(t); }
border Gamma2(t=theta, 2*pi) { x = a * cos(t); y = b*sin(t); }
mesh Th = buildmesh( Gamma1(100) + Gamma2(50) ); // construct mesh
fespace Vh(Th,P2); // P2 conforming triangular FEM
Vh phi,w,f=1; // phi is shape function, w is test function, f is load
// problem definition
solve Laplace(phi, w) = int2d(Th)( dx(phi)*dx(w) + dy(phi)*dy(w) ) - int2d(Th)( f*w ) + on(Gamma1, phi= bndryelev);
plot(phi, wait=true, ps="solution20.eps"); //Plot solution
plot(Th, wait=true, ps="mesh20.eps"); //Plot mesh
```
gnuplot file

- Gnuplot file has groups of 4 lines, each with a vertex location and value \( x_j, y_j, \phi_j \) for \( j = 0, 1, 2, 0 \) followed by a blank line.
- Commands for gnuplot are
  
  ```
  set palette rgbformulae 30,31,32
  splot "graph.txt" with lines palette
  ```

example20.edp code cont’d

```cpp
// to build a gnuplot data file
ofstream ff("graph20.txt");
for (int i=0; i<Th.nt; i++){
  for (int j=0; j < 3; j++){
    ff << Th[i][j].x << " " << Th[i][j].y << " " << phi[Vh(i,j)] << endl;
  }
  ff << Th[i][0].x << " " << Th[i][0].y << " " << phi[Vh(i,0)] << endl << endl;
}
```

\( \text{Th.nt} \) is number of triangles in \( \mathcal{T}_h \).
\( \text{Th[i][j].x} \) is \( x \)-coordinate of vertex \( j \) of triangle \( i \) in the mesh.
\( \phi[Vh(i,j)] \) is the value of dof of \( \phi \) located at vertex \( j \) of triangle \( i \) in fespace \( V_h \). The extra dofs have numbers 3 and higher.
Demonstration (for Example20)

$ gnuplot

gnuplot> set palette rgbformulae 30,31,32

gnuplot> splot "graph.txt" with lines palette

Example 21: errors

▶ from membranerror.edp
▶ Like Example 20
▶ Change to have exact solution
▶ Look at errors and convergence
▶ New:
  ▶ Turning off extraneous output
  ▶ Plot Th and phi together
  ▶ Loading extra elements

example21.edp code

verbosity=0;

load "Element_P3"

real theta = 4.*pi/3.;
real a=1., b=1.; // ellipse is a circle here
border Gamma1(t=0.0,theta) { x = a * cos(t); y = b*sin(t); }
border Gamma2(t=theta,2.0*pi) { x = a * cos(t); y = b*sin(t); }

func f=-4.0*(cos(x^2+y^2-1.0) -(x^2+y^2)*sin(x^2+y^2-1.0));
func phiexact=sin(x^2+y^2-1.0);

int meshes=3, mshdensity=1;

int[int] L2error(meshes);

for ( int n=0; n<meshes; n++)
    mesh Th = buildmesh( Gamma1(40*mshdensity) + Gamma2(20*mshdensity) );

mshdensity *= 2;
fespace Vh(Th,P2);
Vh phi,w;

solve laplace(phi, w, solver=UMFPACK) =
    int2d(Th)( dx(phi) * dx(w) + dy(phi) * dy(w) )
        - int2d(Th)( f*w )
        + on(Gamma2, phi=0)+ on(Gamma1, phi=0);

phi = (phi-phiexact);
plot(Th, phi, wait=true, fill=true); //Plot Th and phi

// compute error
L2error[n]= sqrt( int2d(Th)( (phi)^2 ) );

example21.edp code cont'd

// print errors
for (int n=0; n<meshes; n++){
    cout << " L2error " << n << " = "<< L2error[n] <<endl;
}

// print convergence rates
for (int n=1; n<meshes; n++){
    cout <<" convergence rate = "<< log( L2error[n-1] / L2error[n] )/log(2.)
        <<endl;
}
Output

L2error 1 = 0.000816958
L2error 2 = 0.000203404
L2error 3 = 5.07361e-05
convergence rate = 2.01175
convergence rate = 2.00592
convergence rate = 2.00326

Error rates too slow!

Error on coarsest mesh

Maximum errors are near boundary because of geometry errors.

Change boundary condition

Change from
on(Gamma2, phi=0)

to
on(Gamma2, phi=phiexact)

Output:

L2error 1 = 8.05871e-08
L2error 2 = 5.4554e-09
L2error 3 = 3.30556e-10
convergence rate = 4.31729
convergence rate = 3.88479
convergence rate = 4.04472

WARNING: special elements need FreeFem++-cs
LD_LIBRARY_PATH needs proper definition
Example 22: Heat exchanger

- Circular enclosure \( C_0 \) containing two rectangular thermal conductors \( C_1 \) and \( C_2 \)
- \( C_1 \) held at constant temperature
- \( C_2 \) has higher thermal conductivity.
- \( \nabla \cdot (\kappa \nabla u) = 0 \) on \( \Omega \) with \( u|_{\Gamma} = g \)
- New stuff
  - More complex geometry
  - Saving and retrieving the mesh

\[
\nabla \cdot (\kappa \nabla u) = 0 \quad \text{on} \quad \Omega \quad \text{with} \quad u|_{\Gamma} = g
\]

Example 22 mesh

- \( C_1 \) hot: boundary condition, \( C_2 \) is part of mesh

---

**example22.edp code**

```
// Either build mesh or retrieve mesh
mesh Th;
int C0, C1, C2;
if (true) {
    C0= 99; C1= 98; C2= 97; // could be anything
    border C00( t=0,2*pi){ x=5*cos(t); y=5*sin(t); label=C0; } \ outer
    border C11(t=0,1){ x=1+t; y=3; label=C1; } \ heated blade
    border C12(t=0,1){ x=2; y=3-6*t; label=C1; }
    border C13(t=0,1){ x=2-t; y=-3; label=C1; }
    border C14(t=0,1){ x=1; y=-3+6*t; label=C1; }
    border C21(t=0,1){ x=-2+t; y=3; label=C2; } \ cooling blade
    border C22(t=0,1){ x=-1; y=3-6*t; label=C2; }
    border C23(t=0,1){ x=-1-t; y=-3; label=C2; }
    border C24(t=0,1){ x=-2; y=-3+6*t; label=C2; }
    plot( C00(50) + C11(5) + C12(20) + C13(5) + C14(20) + C21(-5) + C22(-20) + C23(-5) + C24(-20), wait=true);
    Th = buildmesh( C00(50) + C11(5) + C12(20) + C13(5) + C14(20) + C21(-5) + C22(-20) + C23(-5) + C24(-20), wait=true);
}
else {
    Th = readmesh("example22.msh");
    C0= 99; C1= 98; C2= 97; // Numbers are in file, not labels
}

fespace Vh(Th,P1);
Vh u,v;
Vh kappa = 1 + 4*(x<-1) * (x>-2) * (y<3) * (y>-3);
solve a(u,v) = int2d(Th) ( kappa*( dx(u)*dx(v) + dy(u)*dy(v) ) ) + on(C0, u=20) + on(C1, u=100);
plot(u,value=true,wait=true,fill=false);
```

---

**example22.edp code, cont'd**

```
else {
    Th = readmesh("example22.msh");
    C0= 99; C1= 98; C2= 97; // Numbers are in file, not labels
}

fespace Vh(Th,P1);
Vh u,v;
Vh kappa = 1 + 4*(x<-1) * (x>-2) * (y<3) * (y>-3);
solve a(u,v) = int2d(Th) ( kappa*( dx(u)*dx(v) + dy(u)*dy(v) ) ) + on(C0, u=20) + on(C1, u=100);
plot(u,value=true,wait=true,fill=false);
```
Example 22 is clearly symmetric about the $x$-axis. Modify the example so that it only solves *half* the problem, with a symmetry boundary (homogeneous Neumann condition) on the $x$-axis. Check your work visually by comparing the solutions. Pay particular attention to level curves that pass through the cooling blade (C2).