

Finite Elements for the (Navier) Stokes Equations

John Burkardt
Department of Scientific Computing
Florida State University

.....

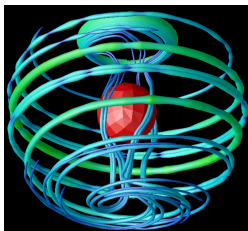
11am, 12 November 2013
CCS, Stanley Thomas Hall

.....

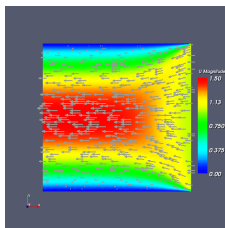
[https://people.sc.fsu.edu/~jburkardt/presentations/...
stokes_2013_tulane.pdf](https://people.sc.fsu.edu/~jburkardt/presentations/stokes_2013_tulane.pdf)



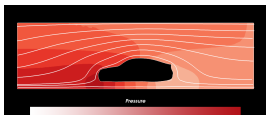
Fluid Flow Problems and Fluid Flow Solvers



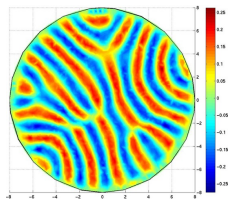
Deal.II



Fenics



FreeFem++



Ifiss



INTRO: Equations of Fluid Motion

If you've taken a course in partial differential equations, you might think all such problems are like the Poisson equation $u_t - \nabla^2 u = f$ on an interval or a rectangle, and can be solved by a sum of sine and cosine functions.

If we try to solve problems in fluid flow, we encounter surprising complications in the equations, the shape of the regions, and the solution methods.

In this talk, we'll stare at the Navier-Stokes equations for fluid flow, then try to simplify them, finding the Stokes equations, a good model for slow-moving fluids.

To compute solutions on "interesting" regions, a technique called finite elements turns the PDE's into a linear system $A^*x=b$.

Many programs are now available to automate this process; we'll look at a MATLAB program called **IFISS**.



- **Equations of Fluid Motion**
- A Finite Element Formulation
- Computing Basis Functions
- Assembling the Matrix
- IFISS



EQUATIONS: The Navier Stokes Equations

Any study of fluid flow starts with the **Navier-Stokes** equations:

$$\rho \mathbf{v}_t - \rho \nu \Delta \mathbf{v} + \rho (\mathbf{v} \cdot \nabla) \mathbf{v} + \nabla p = \mathbf{f} \quad (\text{momentum equations})$$

$$\rho_t + \nabla \cdot (\rho \mathbf{v}) = 0 \quad (\text{continuity equation})$$

We can add complications such as compressibility or heat, makes simplifications such as time independence, or replace some terms in an attempt to better model turbulence or other features.

Notice a version of our friend the heat equation hiding in here:

$$\rho \mathbf{v}_t - \rho \nu \Delta \mathbf{v} = \mathbf{f}$$

- \mathbf{v} is the velocity vector: u , or (u,v) or (u,v,w) ;
- p is the pressure;
- ρ is the fluid density;
- ν is the kinematic viscosity;
- \mathbf{f} represents body forces such as gravity.



EQUATIONS: Unsteady Compressible Navier Stokes

Our first simplification is to work in 2D instead of 3D.

Here are the time-dependent compressible Navier Stokes equations, in 2D Cartesian coordinates, with gravity the only external force (our next simplification).

$$\rho \frac{\partial u}{\partial t} - \rho \nu \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + \rho u \frac{\partial u}{\partial x} + \rho v \frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} = 0$$

$$\rho \frac{\partial v}{\partial t} - \rho \nu \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) + \rho u \frac{\partial v}{\partial x} + \rho v \frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} = -\rho g$$

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} + \frac{\partial \rho v}{\partial y} = 0$$



EQUATIONS: Simplifications

In many flow problems, there is a transient solution that gradually settles down to a long-term unvarying flow.

We can seek this *steady-state* solution by dropping the time derivatives from our equation.

Many fluids, such as water, have an almost constant density. If we assume *incompressibility*, we can divide through by ρ .

We can replace $\frac{p}{\rho}$ by p (rescaling the pressure units);

Gravitational force can be absorbed into pressure.

From now on, we'll just write f_u , f_v for the right hand side velocity functions.



EQUATIONS: Steady, Incompressible Navier-Stokes

Now we have the steady incompressible Navier Stokes equations.

The two momentum equations have the linear diffusion term (multiplied by ν) and nonlinear terms, such as $u \frac{\partial v}{\partial x}$.

$$\begin{aligned} -\nu \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} &= f_u \\ -\nu \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} &= f_v \end{aligned}$$

The continuity equation enforces the incompressibility constraint. Over any small region, the flow in must equal the flow out.

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$



EQUATIONS: The Reynolds Number

The viscosity ν controls the balance between smoothing from diffusion and disruption from nonlinear momentum terms.

If ν decreases, the physical system becomes more irregular, the PDE less stable, and the discretized computer model can fail. Experimentally, a smooth flow becomes turbulent and complex.

The **Reynolds number** Re is a dimensionless quantity that estimates this irregularity on the same scale for all flows:

$$Re = \frac{\|v\|L}{\nu}$$

where L is a characteristic length.

Mathematicians tend to solve problems with $Re = 1$; a swimmer in a pool might suggest a value of $Re \approx 1,000,000$. Turbulent flow typically arises at Re values in the thousands or higher.



EQUATIONS: Steady, Incompressible Stokes

Since turbulence will give us problems, it's natural to ask what happens when the viscosity ν is large - perhaps so large that the the nonlinear terms can be completely neglected. In that case, we have the steady *Stokes equations*:

$$\begin{aligned} -\nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + \frac{\partial p}{\partial x} &= f_u \\ -\nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + \frac{\partial p}{\partial y} &= f_v \\ \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} &= 0 \end{aligned}$$

They are a good model for low Reynolds number flow; but they can also be useful if we end up working with the nonlinear problem, because the Stoke equations will at least give us a decent starting estimate for the solution.



EQUATIONS: A Problem to Solve

Let's assume that we've chosen some version of the Navier-Stokes or Stokes equations to work with.

We must specify problem data, such as the value of ν , and body forces.

We must specify the domain Ω where we seek a solution, the boundary of the domain, Γ , and the nature of the solution along this boundary (inlets, walls, sources, sinks).

Together, this constitutes the mathematical model of the problem.

To solve it on a computer, we must now make a **discrete version**.



- Equations of Fluid Motion
- **A Finite Element Formulation**
- Computing Basis Functions
- Assembling the Matrix
- IFISS

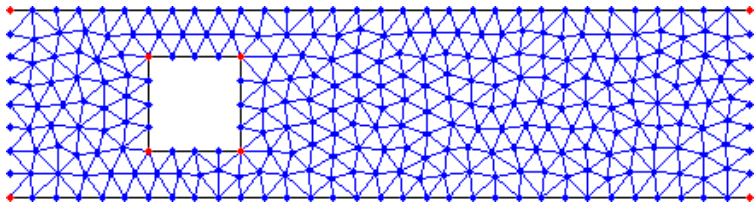


FEM: A Test Problem

The finite element method begins by discretizing the region.

Here is a rectangular channel with a square obstacle. Top and bottom are walls, flow enters from the left and exits on the right.

I've created a grid using a free MATLAB program called **mesh2d**:



Typically, the user simply outlines the region, and a meshing program automatically generates the elements of area.

FEM: MESH2D Can Make a Mesh For Us

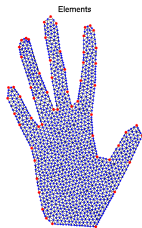
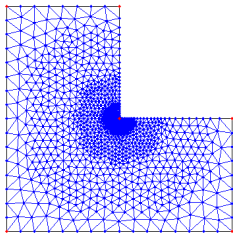
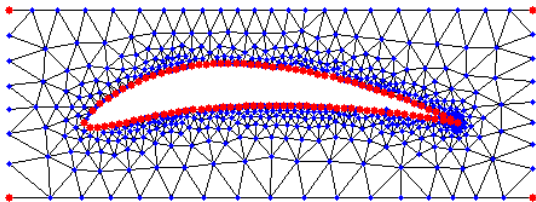
Darren Engwirda's Matlab code **mesh2d** can set up this grid:

```
v = [ 0.0, -1.0; 8.0, -1.0; 8.0, +1.0; 0.0, +1.0;  
      1.5, -0.5; 1.5, +0.5; 2.5, +0.5; 2.5, -0.5 ];  
      <-- vertices  
  
e = [ 1, 2; 2, 3; 3, 4; 4, 1; <-- clockwise  
      5, 6; 6, 7; 7, 8; 8, 5 ]; <-- counter-clockwise  
      <-- vertex pairs form boundary edges  
  
hdata = [];  
hdata.hmax = 0.25; <-- Maximum element size  
  
[ p, t ] = mesh2d ( v, e, hdata ); <-- points, triangles.
```



FEM: Interesting Regions

MESH2D can automatically handle more interesting regions:



FEM: A Linear Grid of Triangles

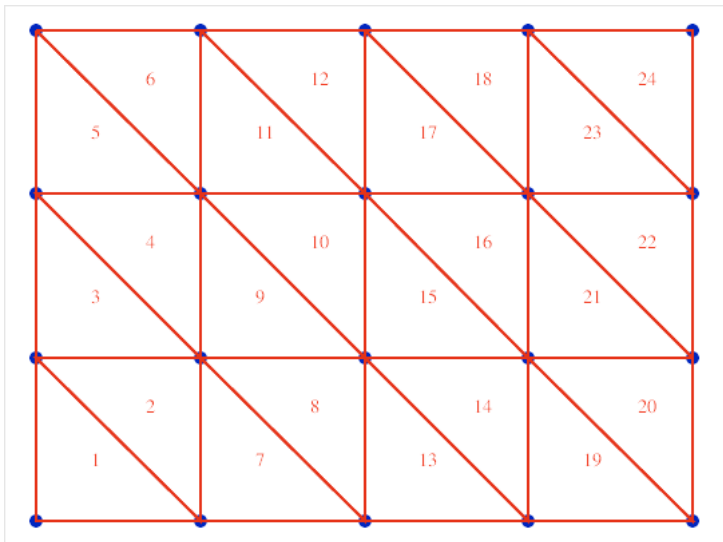
Using a program like MESH2D, we can take a description of a region Ω , perhaps outlined by a set of vertices, and produce a set of *nodes* which can be triangulated so that triplets of nodes define triangular *elements*.

As you have probably seen before, such a triangulation allows us, in a natural way, to define linear basis functions $\phi_i(x, y)$, which are 1 at node i , 0 at all other nodes, and linear over each element.

For reasons I will explain in a minute, let's call the nodes we've just created *pnodes*. This name is meant to suggest that these nodes are associated with the pressure variable.



FEM: A Pressure Grid



FEM: Pressure Representation

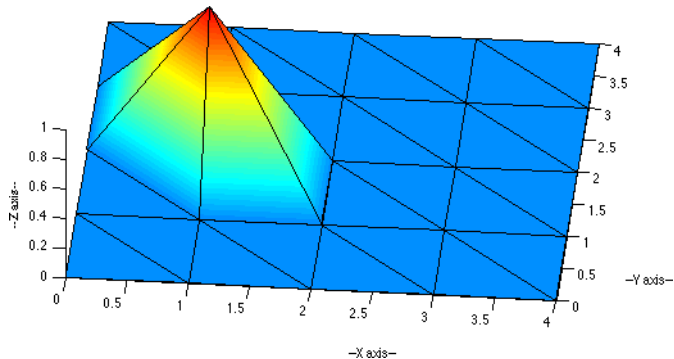
We need to represent our variables as linear combinations of basis functions. The easy case is the pressure p . We can take this to be a linear combination of piecewise linear basis functions $\phi_i(x, y)$,

$$p = \sum_{i=1}^{\text{pnodes}} c_i \phi_i(x, y)$$

where the i -th basis function is associated with the i -th pnode.



FEM: A Linear Basis Function



http://people.sc.fsu.edu/~jburkardt/m_src/fem2d_basis.t3_display/fem2d_basis.t3_display.html



For the Navier-Stokes equations, it turns out that you cannot arbitrarily pick the basis functions.

The problem is related to the “Ladyzhenskaya-Babuska-Brezzi” (“LBB”) or “inf-sup” condition. One way to avoid it uses a **Taylor-Hood** pair of basis functions for the pressure and velocity.

In a typical Taylor-Hood scheme, the polynomial degree of the pressure basis functions is one lower than that used for velocities.

We are using piecewise linear functions for pressure, and so it turns out we should use piecewise quadratic functions for velocity.



FEM: Velocity Grid

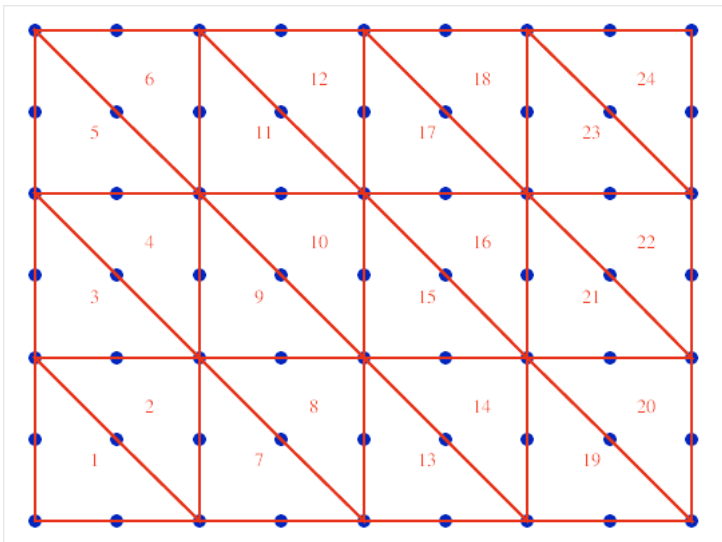
Now we will construct a second grid that is a sort of refinement of the first. The set of nodes in this grid will be called *vnodes*, because they will be associated with velocities. We start by including all the *pnodes*, but we create a new node at the midpoint of every element edge, and add all these nodes as well.

We can look at this procedure as involving two grids, one for pressure and one for velocities. The two grids are nested in an interesting way.

The velocities will “live” on a grid of six-node triangles. These triangles share their vertices with the three-node pressure triangles. But the six-node triangles can be used to define basis functions $\psi_i(x, y)$ which are 1 at node i , zero at all other nodes, and a **quadratic** function over each element.



FEM: Velocity Grid



FEM: Velocity Representation

Our velocities will similarly be represented using the quadratic ψ functions. Since velocity is a vector, we can think of it as having components (u, v) . Our representation can then be written:

$$u = \sum_{i=1}^{\text{vnodes}} a_i \psi_i(x, y)$$

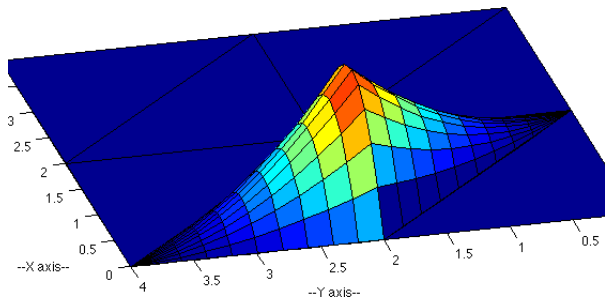
$$v = \sum_{i=1}^{\text{vnodes}} b_i \psi_i(x, y)$$

or

$$\begin{pmatrix} u \\ v \end{pmatrix} = \sum_{i=1}^{\text{vnodes}} \begin{pmatrix} a_i \\ b_i \end{pmatrix} \psi_i(x, y)$$



FEM: A Quadratic Basis Function

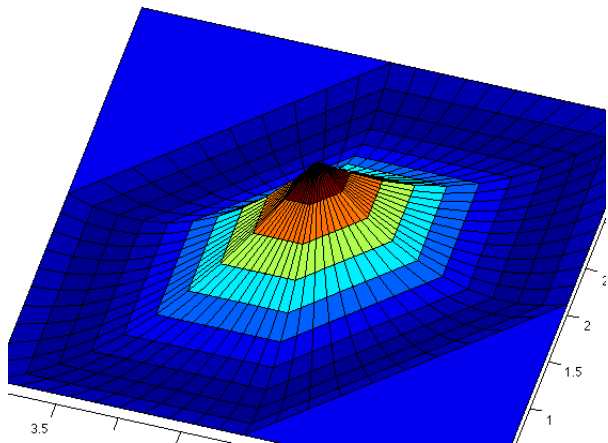


This midside node basis function extends over two elements.

http://people.sc.fsu.edu/~jburkardt/m_src/fem2d_basis_t6_display/fem2d_basis_t6_display.html



FEM: A Quadratic Basis Function



This vertex basis function extends over six elements.

FEM: Multiply by Test Functions

We have represented u , v and p in terms of basis functions $\psi()$ and $\phi()$, and coefficient vectors a , b , c .

To try to determine the coefficients in these representations, we multiply the state equations by the appropriate test functions:

$$\begin{aligned}(-\nu(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) + \frac{\partial p}{\partial x}) \psi_i &= f_u \\(-\nu(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) + \frac{\partial p}{\partial y}) \psi_i &= f_v \\(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}) \phi_i &= 0\end{aligned}$$



FEM: Integrate Over the Region

We integrate each equation over the region Ω :

$$\int_{\Omega} \left(-\nu \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + \frac{\partial p}{\partial x} \right) \psi_i \, dx \, dy = f_u$$
$$\int_{\Omega} \left(-\nu \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) + \frac{\partial p}{\partial y} \right) \psi_i \, dx \, dy = f_v$$
$$\int_{\Omega} \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) \phi_i \, dx \, dy = 0$$



FEM: Integrate By Parts / Green's Theorem

We can solve more general problems if we lower the smoothness requirement on u and v . We can do this using integration by parts:

$$\int_{\Omega} \nu \left(\frac{\partial u}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial u}{\partial y} \frac{\partial \psi_i}{\partial y} \right) + \frac{\partial p}{\partial x} \psi_i \, dx \, dy = \int_{\Omega} f_u \, dx \, dy + \int_{\Gamma} \frac{\partial u}{\partial n} \psi_i \, ds$$
$$\int_{\Omega} f_v + \nu \left(\frac{\partial v}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial v}{\partial y} \frac{\partial \psi_i}{\partial y} \right) + \frac{\partial p}{\partial y} \psi_i \, dx \, dy = \int_{\Omega} f_v \, dx \, dy + \int_{\Gamma} \frac{\partial v}{\partial n} \psi_i \, ds$$
$$\int_{\Omega} \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) \phi_i \, dx \, dy = 0$$

The right hand sides are only “interesting” (nonzero) for nodes on the boundary where a normal inflow or outflow condition is allowed.

We can still recognize the original PDE's, but the integration allows us to think about the average behavior over the area of each element, rather than at particular points.



FEM: We actually have a matrix problem now!

Essentially, these are the discrete equations we want our computer to solve. The unknown quantities are the coefficient vectors a , b , c used to form u , v and p . Because the equations are linear in the variables, they are linear in the coefficients as well.

And that means that we are actually staring at a very fancy form of the usual linear algebra problem

$$Ax = b$$

Before we can say our problem has become trivial, we have some remaining issues:

- How do we evaluate the basis functions ϕ_i and ψ_i ?
- How do we evaluate u , v and p ?
- How do we evaluate the integrals that define A and b ?



- Equations of Fluid Motion
- A Finite Element Formulation
- **Computing Basis Functions**
- Assembling the Matrix
- IFISS



BASIS: Reference Triangle \leftrightarrow Physical Triangle

Our linear system involves integrals of basis functions and their derivatives. We can approximate the integrals using a **quadrature rule**, a set of n points (x_i, y_i) and weights w_i with which we approximate integrals by

$$I(f, \Omega) = \int_{\Omega} f(x, y) dx dy \approx \sum_{i=1}^n w_i f(x_i, y_i) = Q(f, \Omega)$$

The integral approximations can be carried out one element at a time, so we can focus on the problem of estimating an integral over an arbitrary triangle T .

We do this by referring to the unit triangle U formed by vertices $(0, 0)$, $(1, 0)$, $(0, 1)$.



BASIS: The Linear Basis Functions

The basis functions for pressure are defined on the three vertex triangle $T = \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$. Basis $\phi_1(x, y)$ is 1 at vertex 1, 0 at the other two vertices, and linear over T .

Rather than looking up a formula, can we work one out?

If $\phi_1(x, y)$ is linear, and it's zero at nodes 2 and 3, then it's zero on the line between them. The slope of the line through (x_2, y_2) is:

$$s(x_3, y_3) = \frac{y_3 - y_2}{x_3 - x_2}$$

and for an arbitrary point (x, y) , the slope is:

$$s(x, y) = \frac{y - y_2}{x - x_2}$$

We want $\phi_1(x, y)$ to be zero if $s(x, y) = s(x_3, y_3)$. Let's try

$$\phi_1(x, y) \stackrel{?}{=} s(x, y) - s(x_3, y_3) = \frac{y - y_2}{x - x_2} - \frac{y_3 - y_2}{x_3 - x_2}$$



BASIS: The Linear Basis Functions

Let's avoid fractions by multiplying through by the denominators:

$$\phi_1(x, y) \stackrel{?}{=} (y - y_2)(x_3 - x_2) - (y_3 - y_2)(x - x_2)$$

Notice that $\phi_1(x_2, y_2) = \phi_1(x_3, y_3) = 0$. What more do we need?
Oh yes, we need that $\phi_1(x_1, y_1) = 1$

Easy! Just normalize this function by its value at (x_1, y_1) :

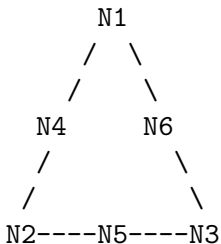
$$\phi_1(x, y) \stackrel{\checkmark}{=} \frac{(y - y_2)(x_3 - x_2) - (y_3 - y_2)(x - x_2)}{(y_1 - y_2)(x_3 - x_2) - (y_3 - y_2)(x_1 - x_2)}$$

Since 1, 2, 3 are “arbitrary”, we also defined $\phi_2(x, y)$ and $\phi_3(x, y)$!



BASIS: The Quadratic Basis Functions

Let's symbolize the six node triangle this way:



Just as for the linear basis functions, we can find a linear function which is zero along any line we choose. Therefore, there is a linear function that is zero at N2 and N1 (and hence at N4 as well). Another linear function is zero at N5 and N6, and so on.

Will this help us find a quadratic basis function?



BASIS: The Quadratic Basis Functions

Suppose we want to find ψ_3 ? There is a linear function $g(x, y)$ that is zero at N1, N4, and N2. There is a linear function $h(x, y)$ that is zero at N5 and N6. Therefore, what about

$$\psi_3(x, y) \stackrel{?}{=} g(x, y) h(x, y)$$

Almost, but we need it to be 1 at (x_3, y_3) . Easy again:

$$\psi_3(x, y) \stackrel{\checkmark}{=} \frac{g(x, y) h(x, y)}{g(x_3, y_3) h(x_3, y_3)}$$

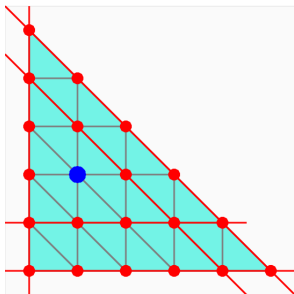
The product of two linear functions is, of course, quadratic.

Pick any node on the six node triangle, and you can cover the other five nodes with two straight lines. End of story!



BASIS: A Quintic Basis For Triangles

By the way, higher order basis functions are easy. To define a 5-th degree basis function, we simply need five linear factors; that's the same as five straight lines that cover all the other nodes!



To find the linear factors, simply “walk” to each boundary, and note the parallel lines you cross.

$$\psi(x, y) = (x)(y)(y - 0.2)(x + y - 1)(x + y - 0.8)$$



- Equations of Fluid Motion
- A Finite Element Formulation
- Computing Basis Functions
- **Assembling the Matrix**
- IFISS



When it's time to assemble the matrix, we have to keep in mind that we have three variables to worry about and two related grids.

To assemble the equation associated with a variable at a given node, we have to consider all the elements that include that node, all the nodes in those elements, and all the variables associated with those nodes. You can see there can be a lot of bookkeeping!

But at some point, we're looking at the equation for node I , and considering contributions from variables defined at node J . These contributions get added to the (I, J) matrix element, and if we want to, we can call this element $A(I, J)$ for horizontal velocity, $B(I, J)$ for vertical velocity, and $C(I, J)$ for pressure.



ASSEMBLY: The Pressure Equation

The i -th pressure equation (continuity equation) is

$$\int_{\Omega} \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) \phi_i \, dx \, dy = 0$$

Every node J neighboring node I contributes to the matrix:

$$A(I, J) = A(I, J) + \int_{\Omega} \frac{\partial \psi_j}{\partial x} \phi_i \, dx \, dy$$

$$B(I, J) = B(I, J) + \int_{\Omega} \frac{\partial \psi_j}{\partial y} \phi_i \, dx \, dy$$



ASSEMBLY: The Horizontal Velocity Equation

The left hand side of the i -th horizontal velocity equation is:

$$\int_{\Omega} \nu \left(\frac{\partial u}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial u}{\partial y} \frac{\partial \psi_i}{\partial y} \right) + \frac{\partial p}{\partial x} \psi_i \, dx \, dy$$

Every node J neighboring node I contributes to $A(I,J)$ and $B(I,J)$:

$$A(I, J) = A(I, J) + \int_{\Omega} \nu \left(\frac{\partial \psi_j}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y} \frac{\partial \psi_i}{\partial y} \right) \, dx \, dy$$

$$B(I, J) = B(I, J) + \int_{\Omega} \nu \left(\frac{\partial \psi_j}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y} \frac{\partial \psi_i}{\partial y} \right) \, dx \, dy$$

and if J is a pressure node, node J also contributes to $C(I,J)$:

$$C(I, J) = C(I, J) + \int_{\Omega} \frac{\partial \phi_j}{\partial x} \psi_i \, dx \, dy$$



ASSEMBLY: MATLAB Code

```
%  
% Add terms to the horizontal momentum equation.  
%  
    a(iu,ju) = a(iu,ju) + w(quad) * nu ...  
        * ( dbidx(test) * dbjdx(basis) + dbidy(test) * dbjdy(basis) );  
  
    if ( 0 < jp )  
        a(iu,jp) = a(iu,jp) + w(quad) * bi(test) * dqjdx(basis);  
    end  
  
%  
% Add terms to the vertical momentum equation.  
%  
    a(iv,jv) = a(iv,jv) + w(quad) * nu ...  
        * ( dbidx(test) * dbjdx(basis) + dbidy(test) * dbjdy(basis) );  
  
    if ( 0 < jp )  
        a(iv,jp) = a(iv,jp) + w(quad) * bi(test) * dqjdy(basis);  
    end  
  
%  
% Add terms to the continuity equation.  
%  
    if ( 0 < ip )  
        a(ip,ju) = a(ip,ju) + w(quad) * qi(test) * dbjdx(basis);  
        a(ip,jv) = a(ip,jv) + w(quad) * qi(test) * dbjdy(basis);  
    end
```



ASSEMBLY: It All Ends Up as a Linear System

Of course, we don't have separate matrices called A , B and C , so we have to store all these coefficients in one big matrix, and we store the coefficients of the representations for u , v and p in one big vector.

Because we have multiple equations and variables, and a pair of grids, a lot of the programming involves simply figuring out where to put things and how to get them back!

We still have some boundary conditions to take care of, but that's another side issue. In the end, we wind up with a sparse linear system:

$$Ax = b$$

that we solve for the finite element coefficients that give us functional representations of the state variables.



ASSEMBLY: Time Dependent Problems

Our simplified system dropped the time derivatives. If we are interested in the time-dependent behavior of the system, we can approximate time derivatives by finite differences like $\frac{\Delta u}{\Delta t}$.

Our solution scheme must now keep track of two versions of the state variables, u, v, p , namely the “old” or current values, and the “new” values that we must solve for.

In the simplest case, we evaluate most of the equation at the current time, so that at time step k we have a system like:

$$\frac{x^{k+1} - x^k}{\Delta t} + Ax^k = b$$

This is easily rewritten as an explicit formula for x^{k+1} .

The implicit version allows larger time steps, but requires a linear solve:

$$\frac{x^{k+1} - x^k}{\Delta t} + Ax^{k+1} = b$$



ASSEMBLY: Nonlinear Problems

To work with the real Navier-Stokes equations, we need to restore the nonlinear term and solve it. We no longer have a linear system for the discrete variables, but rather something like this:

$$Ax + g(x) = b$$

Our natural response is to think of the equations as a nonlinear function, and compute the Jacobian:

$$F(x) = b - Ax - g(x)$$
$$F'_{i,j} = \frac{\partial f_i}{\partial x_j} = -A - \frac{\partial g_i}{\partial x_j}$$

Given a starting point (perhaps by solving the Stokes equations), we can then try Newton's method to get a solution. Note that, as the Reynolds number increases, the problem becomes "more nonlinear" and the Newton iteration may fail to converge.



- Equations of Fluid Motion
- A Finite Element Formulation
- Computing Basis Functions
- Assembling the Matrix
- **IFISS**



IFISS: A MATLAB PDE Solver

It's important to find and use good software tools that other people have written.

It helps you to start solving interesting problems right away, it lets you see how someone has worked out the solution of the underlying software issues, and it gives you a good base from which to add new software features for your own research.

IFISS = Incompressible Flow Iterative Solution Solver is a MATLAB package that is a very useful tool for people interested in learning about solving PDE's.

<http://www.cs.umd.edu/~elman/ifiss.html>



IFISS includes built-in solvers for 2D versions of:

- the Poisson equation
- the convection-diffusion equation
- the Stokes equations
- the Navier-Stokes equations

The user can specify the geometry and the boundary conditions, and time dependence.

The package uses MATLAB's sparse storage structure; it can use MATLAB's sparse direct solver, but also can invoke iterative solvers, including GMRES and multigrid methods.

IFISS offers a variety of mixed finite element bases for flow:

- Stable rectangular: $Q_2 - Q_1$ or $Q_2 - P_{-1}$;
- Stabilized rectangular: $Q_1 - P_0$ or $Q_1 - Q_1$;



IFISS: Customized Problems

IFISS comes with sample problems, which can guide the user in designing a new problem.

The domain, and its gridding, are defined by a function such as **grids/myflow_domain.m**.

The user supplies lists of:

- vertices that outline the boundary and internal holes;
- boundary edges for Dirichlet or Neumann conditions;
- obstacles (v_1, v_2, \dots, v_n);

Boundary conditions and sources are specified by:

- **myflow_bc(x,y)** returns specified stream function values;
- **myflow_flow(x,y)** returns specified flow values;



IFISS: Customized Problems

The user can define the PDE's to be solved as well. Actually, this means writing the code to assemble the system matrix.

Here is part of the code for the Stokes equations, which should start to look familiar now!

The INVJAC and JAC factors arise because these equations are integrated in the reference element, not in their “home” element.

```
for j = 1:9

  for i = 1:9
    ae(:,i,j) = ae(:,i,j) + wght*dpsidx(:,i).*dpsidx(:,j).*invjac(:);
    ae(:,i,j) = ae(:,i,j) + wght*dpsidy(:,i).*dpsidy(:,j).*invjac(:);
    re(:,i,j) = re(:,i,j) + wght*psi(:,i).*psi(:,j).*jac(:);
    bbxe(:,i,j) = bbxe(:,i,j) - wght*psi(:,i) .*dpsidx(:,j);
    bbye(:,i,j) = bbye(:,i,j) - wght*psi(:,i) .*dpsidy(:,j);
  end

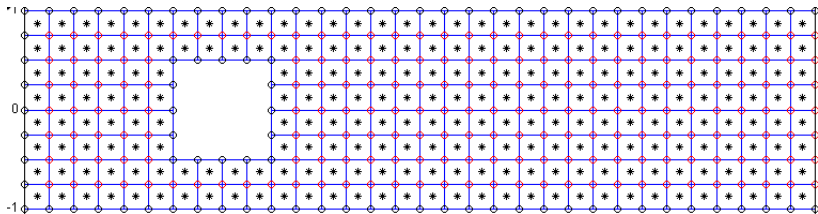
  for i=1:3
    bxe(:,i,j) = bxe(:,i,j) - wght*chi(:,i) .* dpsidx(:,j);
    bye(:,i,j) = bye(:,i,j) - wght*chi(:,i) .* dpsidy(:,j);
  end

end
```



IFISS: Channel Flow With Obstacle

Here is an IFISS grid for problem **NS5**. Top and bottom are walls, flow enters from the left and leaves on the right, and there's a square obstacle.



Yes, IFISS uses **quadrilateral** elements, not triangles!

The pressures are piecewise constant (asterisks at centers) and the velocities are piecewise linear (vertices of quadrilaterals).



IFISS: Channel Flow With Obstacle

Here is how to run IFISS with default data for the obstacle problem:

```
>> setpath                <-- sets up MATLAB path
>> navier_testproblem     <-- request a Navier-Stokes test
```

specification of reference Navier-Stokes problem.

choose specific example (default is cavity)

- 1 Channel domain
- 2 Flow over a backward facing step
- 3 Lid driven cavity
- 4 Flow over a plate
- 5 Flow over an obstacle

: 5



IFISS: Channel Flow With Obstacle (More Choices)

Now we set the grid size and shape, the velocity and pressure basis functions, and the viscosity:

Grid generation for domain with obstacle.

grid parameter: 3 for underlying 8x20 grid

(default is 4) : **return**

uniform/stretched grid (1/2) (default is uniform) : **return**

Q1-Q1/Q1-P0/Q2-Q1/Q2-P1: 1/2/3/4? (default Q1-P0) : **return**

setting up Q1-P0 matrices... done

system matrices saved in obstacle_stokes_nobc.mat ...

Incompressible flow problem on obstacle domain ...

viscosity parameter (default 1/50) : **return**



IFISS: Channel Flow With Obstacle (More Choices)

Now we specify some solver options.

Picard/Newton/hybrid linearization 1/2/3

(default hybrid) : **return**

number of Picard iterations (default 6) : **return**

number of Newton iterations (default 5) : **return**

nonlinear tolerance (default 1.d-8) : **return**

stokes system ...

Stokes stabilization parameter (default is 1/4) : **return**

setting up Q1 convection matrix... done.

uniform/exponential streamlines 1/2

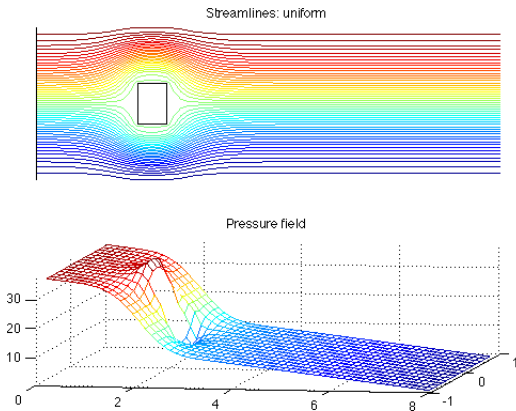
(default uniform) : **return**

number of contour lines (default 50) : **return**



IFISS: Stokes Flow

IFISS displays the Stokes flow used for initialization.

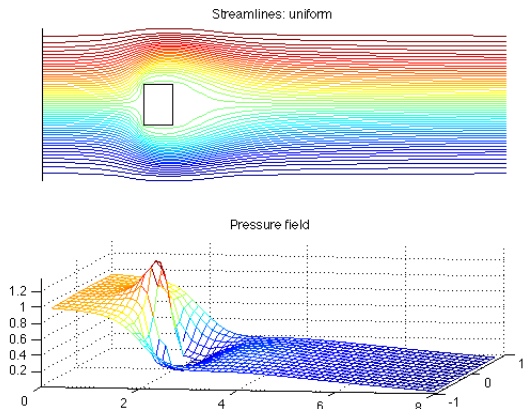


Notice that the flow seems symmetric before and after the obstruction. It's actually reversible!



IFISS: Navier Stokes Flow

The final Navier-Stokes solution shows significant differences.



The flow before and after the obstruction is quite different.



CONCLUSION: The Big Picture

Navier-Stokes equations govern blood flow and the Gulf Stream.

We can talk about the (almost) **automatic** solution of the Navier Stokes equations because of new, powerful, free software:

- DEAL.II (C++ based)
- FEniCS (C++ or Python)
- FreeFem++ (C++)
- IFISS (Matlab)

We can:

- see a flow change as we vary the shape of the region, or place obstructions in its path;
- compute the lift of a wing over a range of angles and speeds;
- design streamlined cars or rockets;
- investigate cooling systems for homes or computer centers.

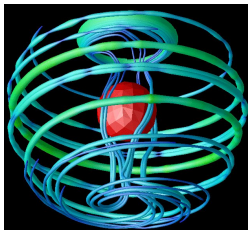


CONCLUSION: Finite Element Flow Solvers

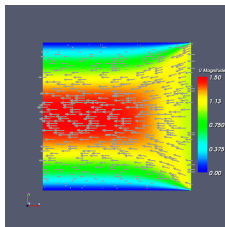
- Howard Elman, Alison Ramage, David Silvester, *IFISS, A Matlab Toolbox for Modeling Incompressible Flow*, ACM Transactions on Mathematical Software, Volume 33, Number 2, June 2007, Article 14.
- Frederic Hecht, *New development in FreeFem++*, Journal of Numerical Mathematics, Volume 20, Number 3-4, 2012, pages 251-265.
- Anders Logg, Kent-Andre Mardal, Garth Wells, *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*, Lecture Notes in Computational Science and Engineering, Springer, 2011.
- Wolfgang Bangerth, Ralf Hartmann, Guido Kanschat, *DEAL.II - a general-purpose object-oriented finite element library*, ACM Transactions on Mathematical Software, Volume 33, Number 4, article 24, August 2007.



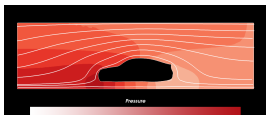
CONCLUSION: The Big Picture



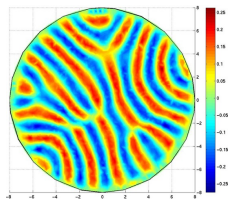
Deal.II



Fenics



FreeFem++



Ifiss

