Finite Elements for the (Navier) Stokes Equations

ISC 5939: Advanced Graduate Seminar

.....

John Burkardt
Department of Scientific Computing
Florida State University
https://people.sc.fsu.edu/~jburkardt/presentations/...
...stokes_2011_fsu.pdf

02 December 2011

- **Introduction**
- Equations of Fluid Motion
- A Finite Element Formulation
- The Mapping Function
- Computing Basis Functions
- Assembling the Matrix
- IFISS
- Conclusion

INTRO: Equations of Fluid Motion

We have looked at the finite element method on relative simple regions, and with relatively uncomplicated equations. It is time to be brave, and jump into problems where we have complication in the physics...and therefore in the mathematics...and therefore in the computation!

You might not be familiar with the equations that describe fluid flow. We will look at a general version of these equations, which we will quickly whittle down to a more manageable system. But even this fairly simple system has some subtleties not present in our friendly Poisson equation!

We will consider what is involved in defining a fluid flow problem, and describe a simple example.

INTRO: A Finite Element Formulation

Once we've defined the problem in the classical sense, we consider how to set up a finite element model.

The fluid flow equations are more complicated, and involve variables of different types. In particular, we will find out that we have to be careful to use approximations for the velocity and pressure that will guarantee the stability of the calculation. For instance, one legal choice is to use linear elements for pressure, and quadratics for velocity.

Having chosen our basis, we convert the classical PDE's into the recognizable finite element form by multiplying by test functions, integrating, and applying Green's formula where appropriate.

Depending on our choice of flow equations (Stokes or Navier-Stokes), we end up with a linear or nonlinear system, whose coefficients are computed as integrals over the region.

INTRO: Computational Details

Since I enjoy looking at computational details, we will take some time to talk about the relationship between the many physical elements that form the triangulation, and the single reference element where the quadrature rule is defined. I will show you a simple map that can transfer information from one triangle to the other.

We will also consider the actual formulas for the linear and quadratic basis functions. My hope is to convince you that you could think these up for yourself.

I will try to suggest that the matrix assembly for fluid flow problems is more complicated, but still has a certain logic that you can follow.

INTRO: Using Someone Else's Work

Although you should be familiar with the underlying formulas and ideas in a fluid flow calculation, it is far too difficult for you to write your own program, and there's really no need for you to do so!

Instead, you should look for a simple, usable program that can teach you how finite elements can solve the problems you are interested in. Once you understand such a program, you will have enough confidence to try new problems, or to extend the program with some new algorithms, or to ignore my advice and write your own program from scratch.

We will take about a program called IFISS that makes it easy to do some sophisticated computations for PDE's in 2D.

- Introduction
- **Equations of Fluid Motion**
- A Finite Element Formulation
- The Mapping Function
- Computing Basis Functions
- Assembling the Matrix
- IFISS
- Conclusion

EQUATIONS: The Navier Stokes Equations

The **Navier-Stokes** equations are the standard for fluid motion.

Any discussion of fluid flow starts with these equations, and either adds complications such as temperature or compressibility, makes simplifications such as time independence, or replaces some term in an attempt to better model turbulence or other features.

$$\rho \mathrm{v}_t - \mu \Delta \mathrm{v} + \rho(\mathrm{v} \cdot \nabla)\mathrm{v} + \nabla p = f \ \textit{(momentum equations)}$$
$$\rho_t + \nabla \cdot (\rho \mathrm{v}) = 0 \ \textit{(continuity equation)}$$

- v is the velocity vector;
- p is the pressure;
- ρ is the fluid density;
- μ is the dynamic viscosity;
- f represents body forces such as gravity.

EQUATIONS: Cartesian Coordinates

Here is the equivalent, in 2D Cartesian coordinates:

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

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

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

EQUATIONS: Simplifications

We might be interested in *steady state flow*, in which case we can drop the time derivatives.

We might also assume that density is constant. In that case, it is convenient to replace the velocity by the mass velocity.

The pressure can absorb the gravity force, and can be rescaled by the constant density.

The dynamic viscosity can be rescaled by density to yield the kinematic viscosity.

We "abuse notation" by reusing u and v to mean mass velocity, and p to mean the adjusted and rescaled pressure. On the other hand, the kinematic viscosity gets a new symbol, ν.

EQUATIONS: Steady, Incompressible Navier-Stokes

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

The viscosity ν multiplies the "nice" Poisson operator. It represents the tendency of momentum to spread out or diffuse.

EQUATIONS: The Reynolds Number

If we are solving the full Navier Stokes equations, then the relative magnitude of ν measures the balance between diffusion (smoothing) and nonlinear momentum terms (disruptive).

As ν goes to zero, the character of the physical system, the PDE, and the discretized computer model deteriorate. Smooth solutions become irregular. Laminar flows become urbulent.
Computationally, the nonlinear equations become difficult to solve.
A time-dependent problem becomes unstable.

The Reynolds number is a dimensionless quantity that estimates the dominance of momentum over diffusion:

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

where L is a characteristic length.

Mathematicians tend to solve problems with Re=1.

EQUATIONS: Steady, Incompressible Stokes

If the viscosity ν is large enough, the nonlinear terms can be neglected, and we have the *Stokes equations*:

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

Because these equations are linear in u, v and p, they are much easier to work with.

They are also useful even if we want to solve the Navier-Stokes equations, because they can give us a reasonable starting solution.

EQUATIONS: A Problem to Solve

We make take our equations of state to be either the Navier-Stokes or the Stokes equations.

We assume we have the value of the kinematic viscosity ν.

We assume we have been given information about a domain Ω, within which the state equations hold.

Moreover, we have information about Γ, the boundary of Ω, along which boundary conditions have been specified. Typically, these conditions include walls, inlets and outlets at which the velocity or some component of it is specified.

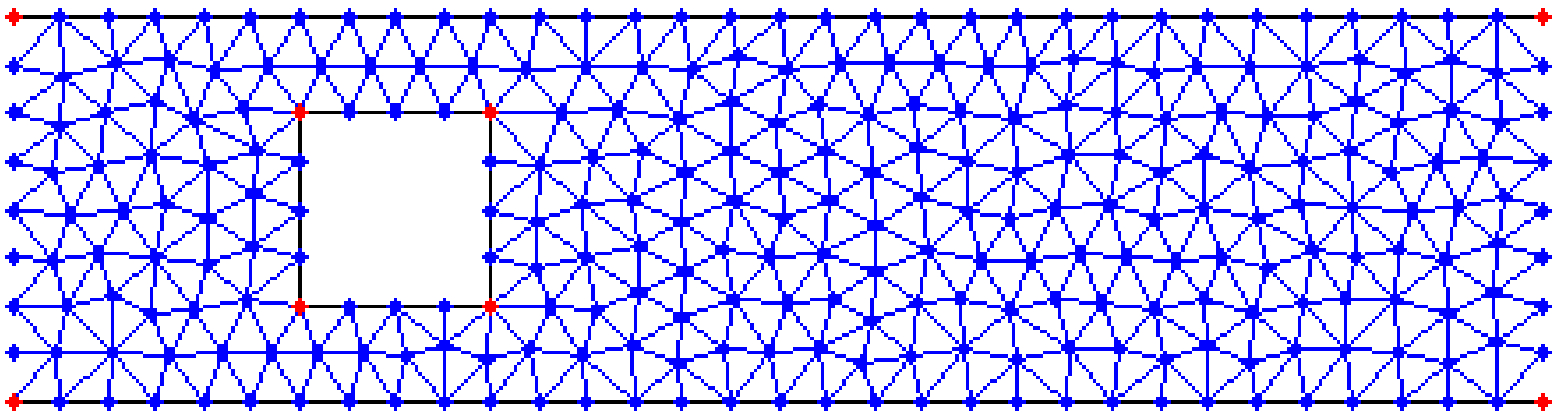
We also need the value of pressure at one point, since it is a potential function, and thus unique only up to an additive constant.

Together, this constitutes the mathematical model of the problem.

EQUATIONS: A Test Problem

Here is an example problem for us to solve, a rectangular channel with a square obstacle. Top and bottom are walls, flow enters from the left and exits on the right.

I've already created a grid using **mesh2d**:



In a finite element approach, we divide the region into small elements that are only influenced by their immediate neighbors. Typically, the solution is sought at the vertices of the elements. For our fluid problem, our approach will be a little more complicated!

EQUATIONS: Remember MESH2D!

mesh2d can set up this grid with some simple commands:

```
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;
      5, 6;   6, 7;   7, 8;   8, 5 ];
      <-- connect vertex pairs to form boundaries

hdata = [];
hdata.hmax = 0.25;     <-- Maximum element size

[ p, t ] = mesh2d ( v, e, hdata );
```

http://people.sc.fsu.edu/~jburkardt/m_src/mesh2d/mesh2d.html

- Introduction
- Equations of Fluid Motion
- **A Finite Element Formulation**
- The Mapping Function
- Computing Basis Functions
- Assembling the Matrix
- IFISS
- Conclusion

FEM: The LBB Condition

You may have heard that, when applying the finite element method to the Navier-Stokes equations for velocity and pressure, you cannot arbitrarily pick the basis functions.

The interaction between the momentum and continuity equations can cause a stability problem; an unwary programmer can try to do everything right, and end up computing garbage.

The problem that is going on is related to the "inf-sup" or "Ladyzhenskaya-Babuska-Brezzi" condition ("LBB"). Everyone in finite element fluid calculations has their favorite way of avoiding the problem. The way we will do it is to use 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.

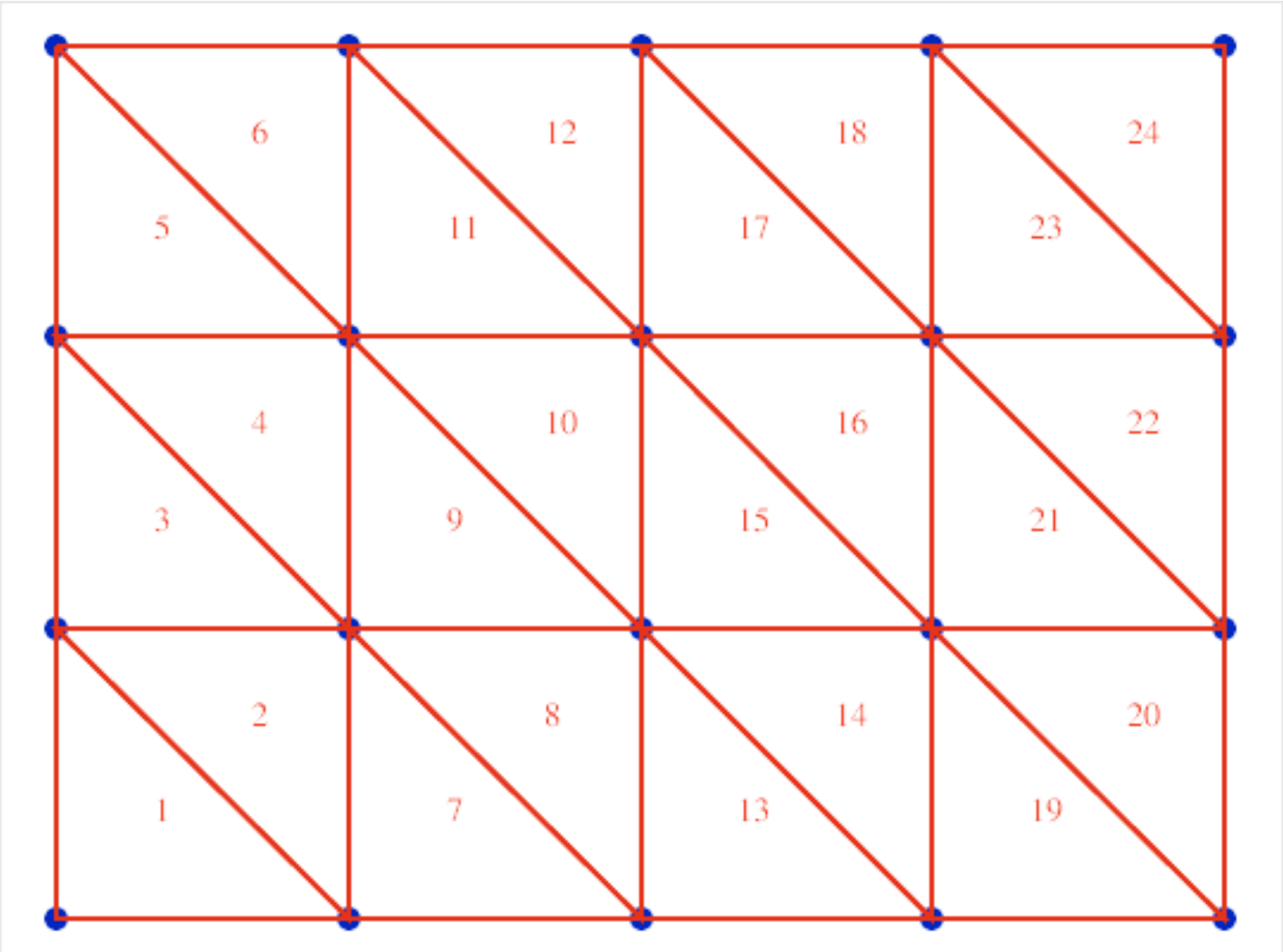
FEM: A Linear Grid

In an earlier talk, we discussed methods for starting with a region Ω, perhaps outline by a set of vertices, and producing a set of *nodes* which can be triangulated so that triplets of nodes define *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: Pressure Grid



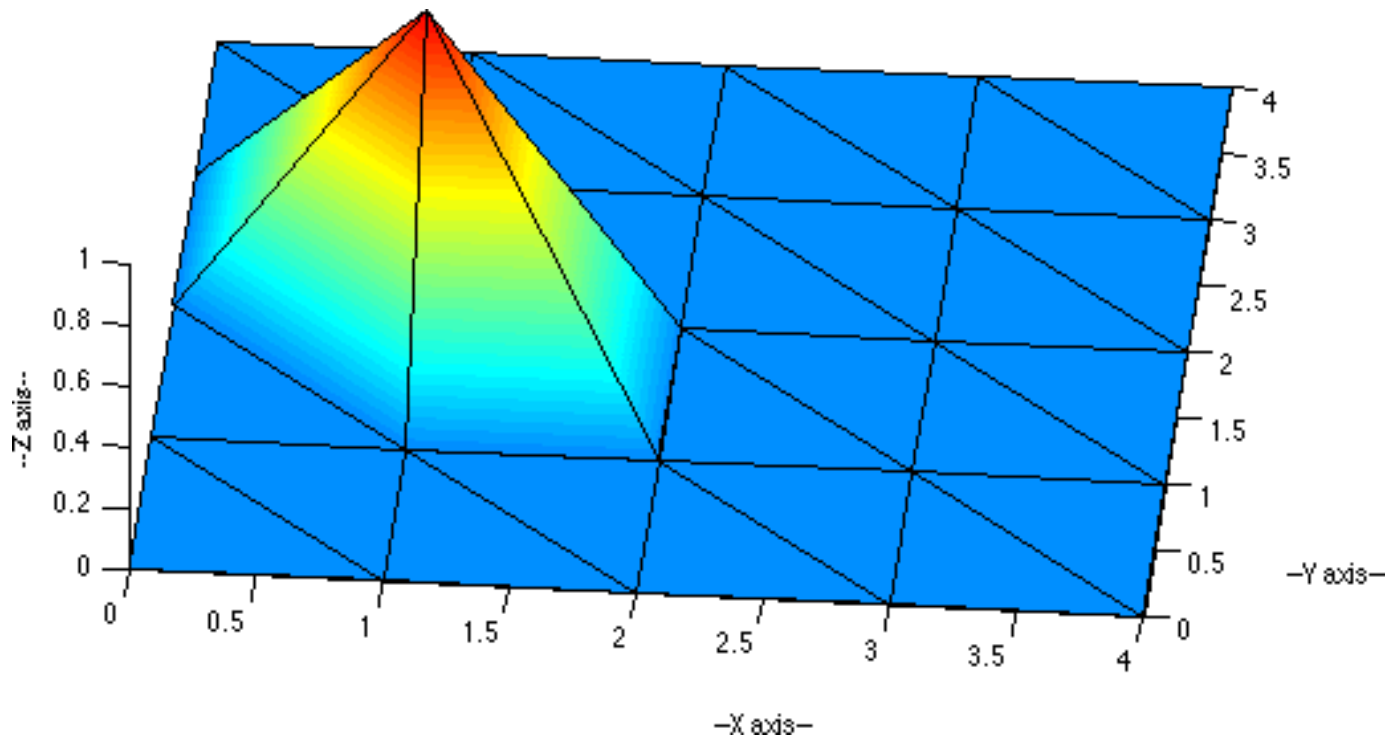
FEM: Pressure Representation

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

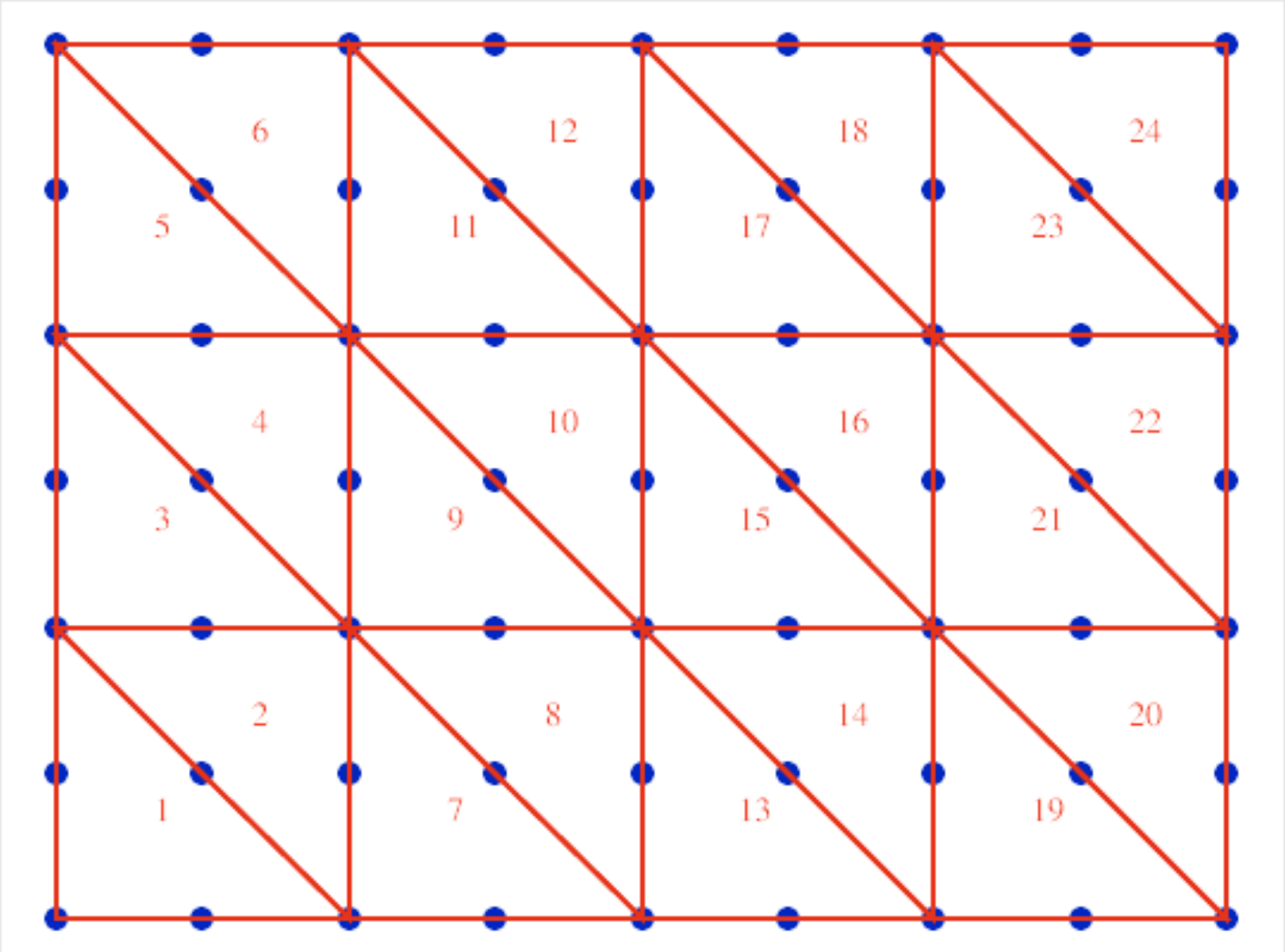
FEM: A Quadratic Grid

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



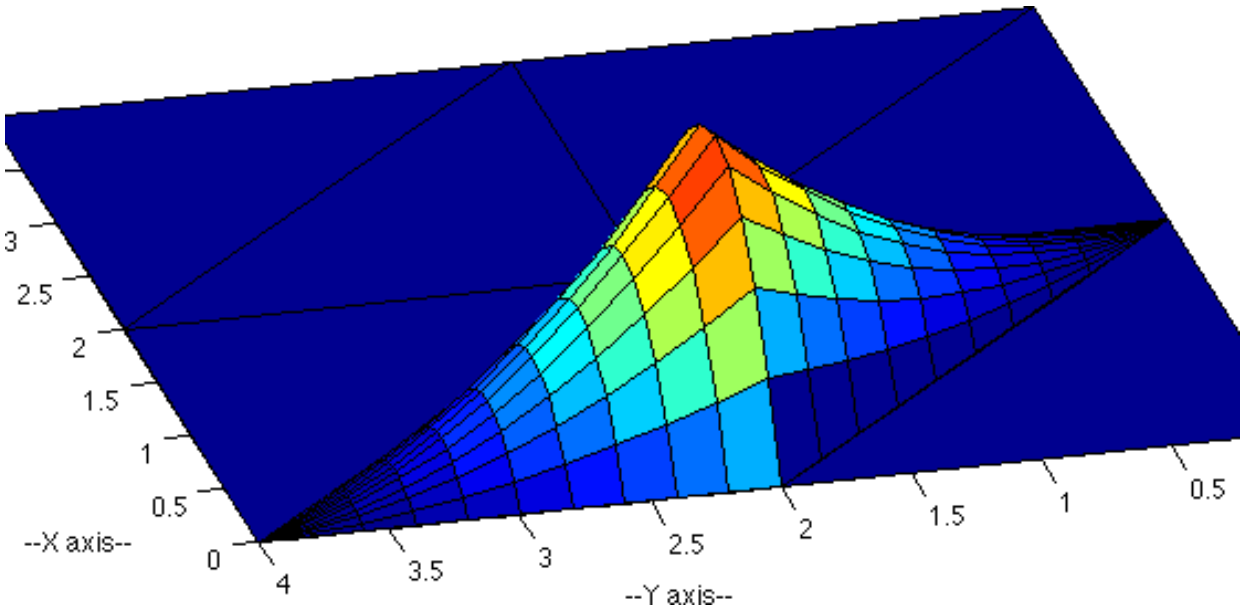
FEM: Velocity Representation

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:

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

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

FEM: A Quadratic Basis Function

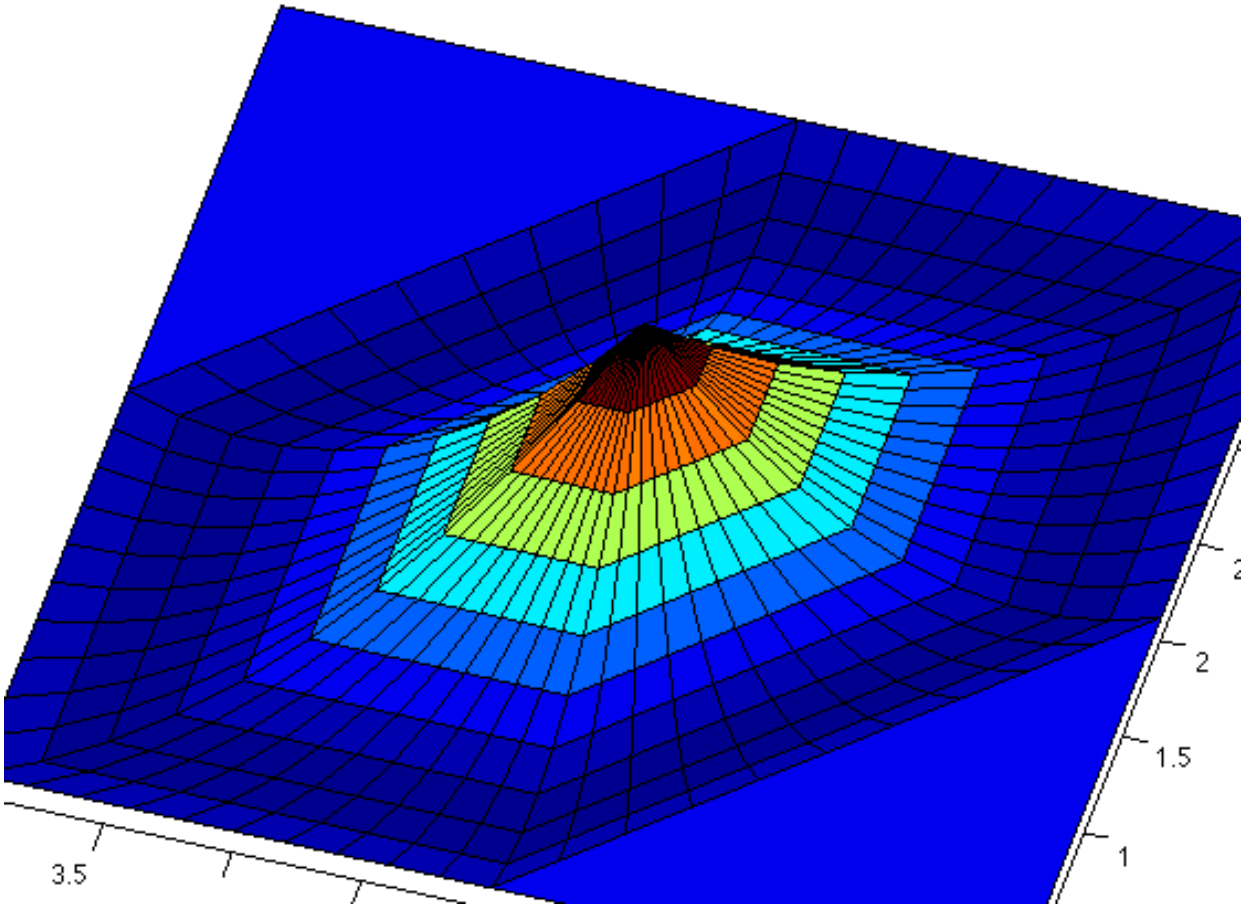


This midside node basis function extends over two elements.

---

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.

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

FEM: Multiply by Test Functions

We have represented u, v and p in terms of basis functions.

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

$$(-\nu(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) + \frac{\partial p}{\partial x}) * \psi_i = 0$$

$$(-\nu(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) + \frac{\partial p}{\partial y}) * \psi_i = 0$$

$$(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}) * \phi_i = 0$$

FEM: Integrate Over the Region

We integrate each equation over the region Ω:

$$\int_\Omega (-\nu(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) + \frac{\partial p}{\partial x}) * \psi_i \, dx \, dy = 0$$

$$\int_\Omega (-\nu(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) + \frac{\partial p}{\partial y}) * \psi_i \, dx \, dy = 0$$

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

FEM: Integrate By Parts / Green's Theorem

We seek to lower the order of differentiation on u and v:

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

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

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

The right hand sides are only "interesting" (nonzero) for nodes on the boundary where a normal inflow or outflow condition is allowed. Right now, we don't need things to get any more interesting, so we'll assume the right hand sides are all zero!

- Introduction
- Equations of Fluid Motion
- A Finite Element Formulation
- **The Mapping Function**
- Computing Basis Functions
- Assembling the Matrix
- IFISS
- Conclusion

MAP: Reference Triangle $\leftrightarrow$ Physical Triangle

The finite element equations are now a linear system of equations. The coefficients are defined by integrals involving basis functions and their derivatives. We approximate the integrals using a quadrature rule.

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 "physical" triangle.

Miro presented a method in which we approximate the integral by mapping each (x, y) physical triangle to a single reference triangle (ξ, η), where our quadrature rule is defined. There are complications in this method, especially when derivatives occur in our integrand. I will look at going the other way!

MAP: Quadrature in the Physical Triangle

We adjust the quadrature rule for a physical element T, with vertices (x_1, y_1), (x_2, y_2), and (x_3, y_3). A reference abscissa (ξ, η) is transformed to a physical abscissa (x, y):

$$\begin{aligned} x &= \xi x_1 + \eta x_2 + (1 - \xi - \eta) x_3 \\ y &= \xi y_1 + \eta y_2 + (1 - \xi - \eta) y_3 \end{aligned}$$

A reference weight μ becomes a physical weight w:

$$w = \mu \left(x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) \right)$$

which simply multiplies the old weight by the area of T.
The integral I(f,T) is approximated by Q(f,T)

$$I(f, T) \approx Q(f, T) = \sum_i w_i \, f(x_i, y_i)$$

Derivatives in the integrand don't need any special treatment.

MAP: The Mapping Function

Any reference point can be mapped to the physical triangle:

$$\begin{aligned} x =& \xi x_1 + \eta x_2 + (1 - \xi - \eta) x_3 \\ y =& \xi y_1 + \eta y_2 + (1 - \xi - \eta) y_3 \end{aligned}$$

There is also an inverse map, which is easily computed:

$$\begin{aligned} \xi =& \frac{(y_2 - y_3) * (x - x_3) - (x_2 - x_3) * (y - y_3)}{(x_1 - x_3) * (y_2 - y_3) - (y_1 - y_3) * (x_2 - x_3)} \\ \eta =& \frac{(-(y_1 - y_3) * (x - x_3) + (x_1 - x_3) * (y - y_3)}{(x_1 - x_3) * (y_2 - y_3) - (y_1 - y_3) * (x_2 - x_3)} \end{aligned}$$

The denominator will not vanish, because it is a multiple of the area of the triangle.

You can use these facts to construct a two-way linear map between an arbitrary pair of triangles.

- Introduction
- Equations of Fluid Motion
- A Finite Element Formulation
- The Mapping Function
- **Computing Basis Functions**
- Assembling the Matrix
- IFISS
- Conclusion

BASIS: Hard Implementation, Simple Idea

You don't need to memorize a formula for basis functions, but you need to know there's a reasoning behind such formulas. You never want to look at code like the following and say *"I have no way of understanding this, so I'll believe it and use it til it breaks!"*

```
subroutine qbf (x,y,it,in,bb,bx,by,nelemn,nnodes,node,np,xc,yc)
  integer node(nelemn,nnodes)
  real xc(np), yc(np)
  if (in <= 3) then
    in1 = in; in2 = mod(in,3)+1; in3 = mod(in+1,3)+1
    i1 = node(it,in1); i2 = node(it,in2); i3 = node(it,in3)
    d = (xc(i2)-xc(i1))*(yc(i3)-yc(i1))-(xc(i3)-xc(i1))*(yc(i2)-yc(i1))
    t = 1.0+((yc(i2)-yc(i3))*(x-xc(i1))+(xc(i3)-xc(i2))*(y-yc(i1)))/d
    bb = t*(2.0D+00*t-1.0D+00)
    bx = (yc(i2)-yc(i3))*(4.0D+00*t-1.0D+00)/d; by = (xc(i3)-xc(i2))*(4.0D+00*t-1.0D+00)/d
  else
    inn = in-3; in1 = inn; in2 = mod(inn,3)+1; in3 = mod(inn+1,3)+1
    i1 = node(it,in1); i2 = node(it,in2); i3 = node(it,in3); j1 = i2; j2 = i3; j3 = i1
    d = (xc(i2)-xc(i1))*(yc(i3)-yc(i1))-(xc(i3)-xc(i1))*(yc(i2)-yc(i1))
    c = (xc(j2)-xc(j1))*(yc(j3)-yc(j1))-(xc(j3)-xc(j1))*(yc(j2)-yc(j1))
    t = 1.0D+00+((yc(i2)-yc(i3))*(x-xc(i1))+(xc(i3)-xc(i2))*(y-yc(i1)))/d
    s = 1.0D+00+((yc(j2)-yc(j3))*(x-xc(j1))+(xc(j3)-xc(j2))*(y-yc(j1)))/c
    bb = 4.0D+00*s*t
    bx = 4.0D+00*(t*(yc(j2)-yc(j3))/c+s*(yc(i2)-yc(i3))/d)
    by = 4.0D+00*(t*(xc(j3)-xc(j2))/c+s*(xc(i3)-xc(i2))/d)
  end if
  return
end
```

http://people.sc.fsu.edu/~jburkardt/f_src/channel/channel.html

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:

$$\phi_1(x, y) \stackrel{?}{=} (y - y_2)(x3 - x_2) - (y_3 - y_2)(x - x_2)$$

Notice that $\phi_1(x2, y2) = \phi_1(x3, y3) = 0$. What more do we need? Oh yes, we need that $\phi_1(x1, y1) = 1$

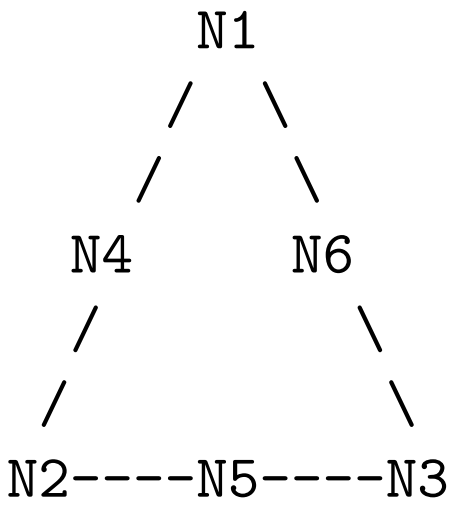
Easy! Just normalize this function by its value at (x_1, y_1):

$$\phi_1(x, y) \stackrel{\checkmark}{=} \frac{(y - y_2)(x3 - x_2) - (y_3 - y_2)(x - x_2)}{(y1 - y_2)(x3 - x_2) - (y_3 - y_2)(x1 - x_2)}$$

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 N12 as well). Another linear function is zero at N23 and N31, 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:

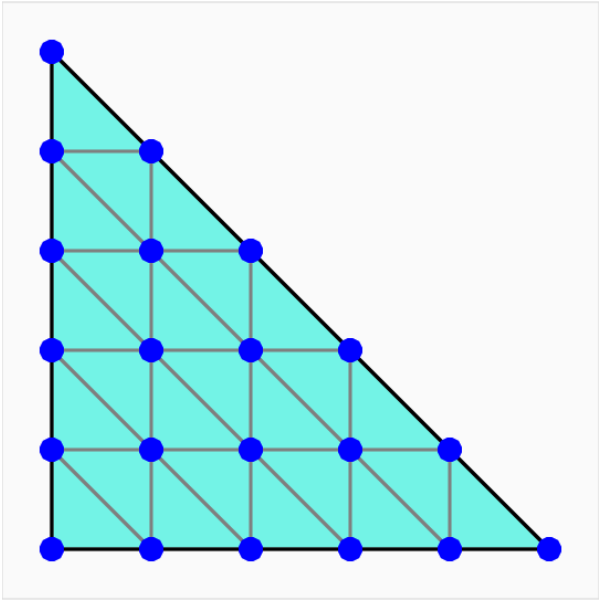
$$\psi_3(x, y) \stackrel{\checkmark}{=} \frac{g(x, y) * h(x, y)}{g(x_3, y_3) * h(x_3, y_3)}$$

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

If we need a 5-th degree polynomial basis, we take a reference triangle and make 6 rows of dots in each direction.

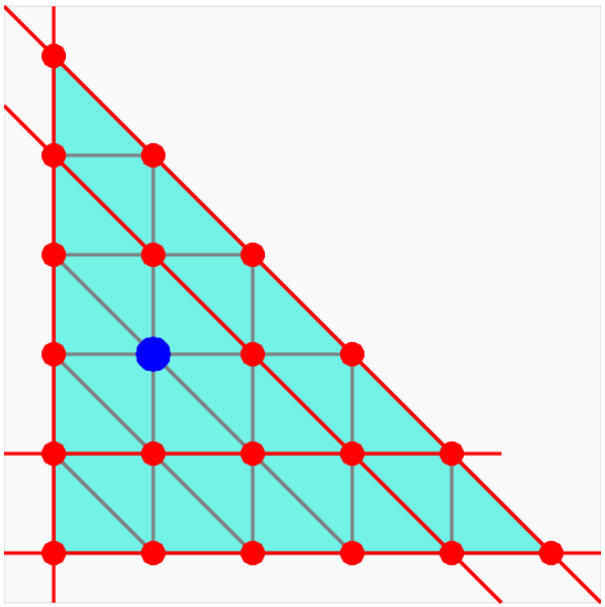


For each node, we must define a degree 5 polynomial in x and y that is 1 at that node, and zero at the other 20. A quintic function could be defined as the product of five linear functions. A linear function covers up points on a line.

Find five straight lines that cover all the other nodes!

BASIS: A Quintic Basis Function

Can we describe the basis function at the blue node?



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

This will be zero at all the red nodes. Of course, we have to normalize the function to get a value of 1 at the blue node.

BASIS: The Quadratic Basis Functions

I don't necessarily want you to ever have to work out the arithmetic involved in computing a quadratic basis function...or its derivatives with respect to x and y.

But I do want to convince you that it's not magic, it doesn't require a special course in analysis, it's really just some carefully thought-out high school geometry!

You should be able to see how to construct:

- cubic elements in a 2D triangle; you'll need 10 nodes;
- quadratic elements in a 3D tetrahedron; again you need 10 nodes; instead of eliminating nodes on 2 lines, you look for 2 planes. There are three cases to consider, a vertex, mid-edge node, or mid-face node.
- quintic elements in a 4D simplex - see how easy it is?

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

FEM STOKES

- Introduction
- Equations of Fluid Motion
- A Finite Element Formulation
- The Mapping Function
- Computing Basis Functions
- **Assembling the Matrix**
- IFISS
- Conclusion

ASSEMBLY

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

We are looking at node I. Node I has a pressure equation associated with it only if it is actually a vertex of the triangle, what we called a *pnode*. Let's assume that this is the case. The pressure equation (continuity equation) is

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

So, oddly enough, pressure itself doesn't show up in the pressure equation! However, node J will contribute to coefficients A(I,J) and B(I,J) for the horizontal and vertical velocities:

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

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

ASSEMBLY: The Horizontal Velocity Equation

Ignoring boundary terms, the horizontal velocity equation is:

$$\int_\Omega \nu\left(\frac{\partial u}{\partial x}\frac{\partial \psi_i}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial \psi_i}{\partial y}\right) + \frac{\partial p}{\partial x} * \psi_i \, dx \, dy = 0$$

So we always get a contribution to A(I,J):

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

and if J is a pressure node, we get a contribution to C(I,J):

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

Do you see that we're just differentiating the equations with respect to a coefficient?

ASSEMBLY: MATLAB Code

```
%
%  Add terms to the horizonal 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
```

http://people.sc.fsu.edu/~jburkardt/m_src/fem2d_stokes_sparse/fem2d_stokes_sparse.m

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:

$$A * x = b$$

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

- Introduction
- Equations of Fluid Motion
- A Finite Element Formulation
- The Mapping Function
- Computing Basis Functions
- Assembling the Matrix
- **IFISS**
- Conclusion

IFISS: A MATLAB PDE Solver

I have tried to convince you that 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.

IFISS: Features

IFISS includes built-in software 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.

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.

All these problems can be set up with time dependence.

IFISS: Basis Functions

For flow problems, IFISS offers a variety of mixed finite element bases:

- 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 primary information that the user supplies are lists of

- vertices that outline the boundary and internal holes;
- boundary edges: (v1,v2,1) for Dirichlet, (v1,v2,2) for Neumann;
- obstacles (v1, v2, ..., vn);
- boundary edges (e1, e2, ..., en) where stretching is needed.

IFISS: Customized Problems

Boundary conditions and source terms are specified by creating:

- **myflow_bc(x,y)** which returns specified stream function values;
- **myflow_flow(x,y)** which 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: Features

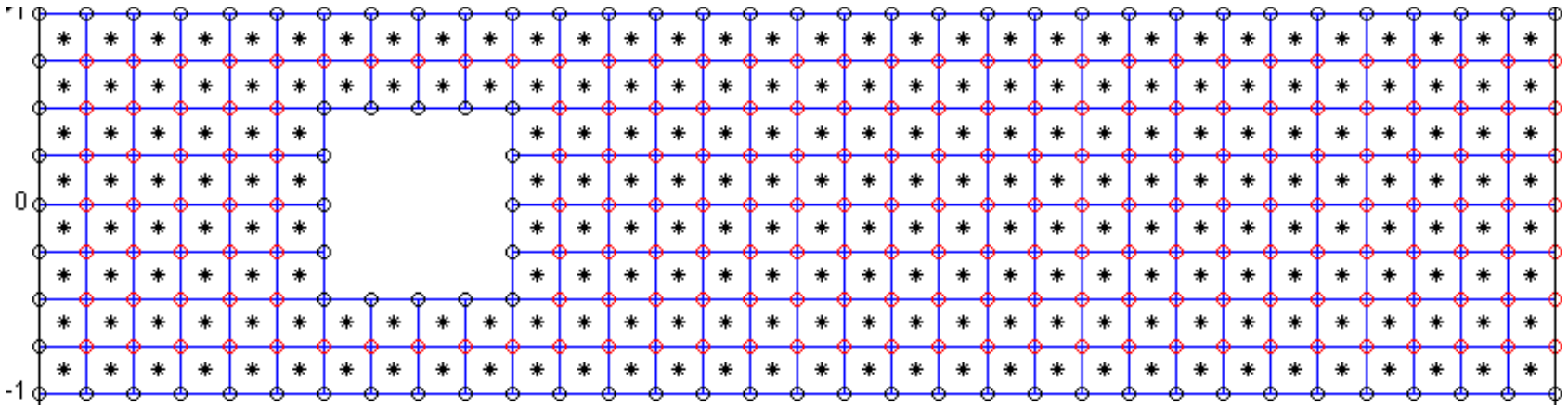
To get you started, IFISS includes example test problems. For the Navier Stokes solver, these include:

1. **NS1**: Poiseuille channel flow
2. **NS2**: flow over a step
3. **NS3**: the driven cavity
4. **NS4**: Blasius flow, a flat plate moving through a liquid
5. **NS5**: channel flow with a square obstacle

Problems 2, 3 and 5 are also available as time dependent problems.

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).

(The "piecewise linear" statement is not exactly true, since quadrilateral elements allow an 'xy' term.)

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
```

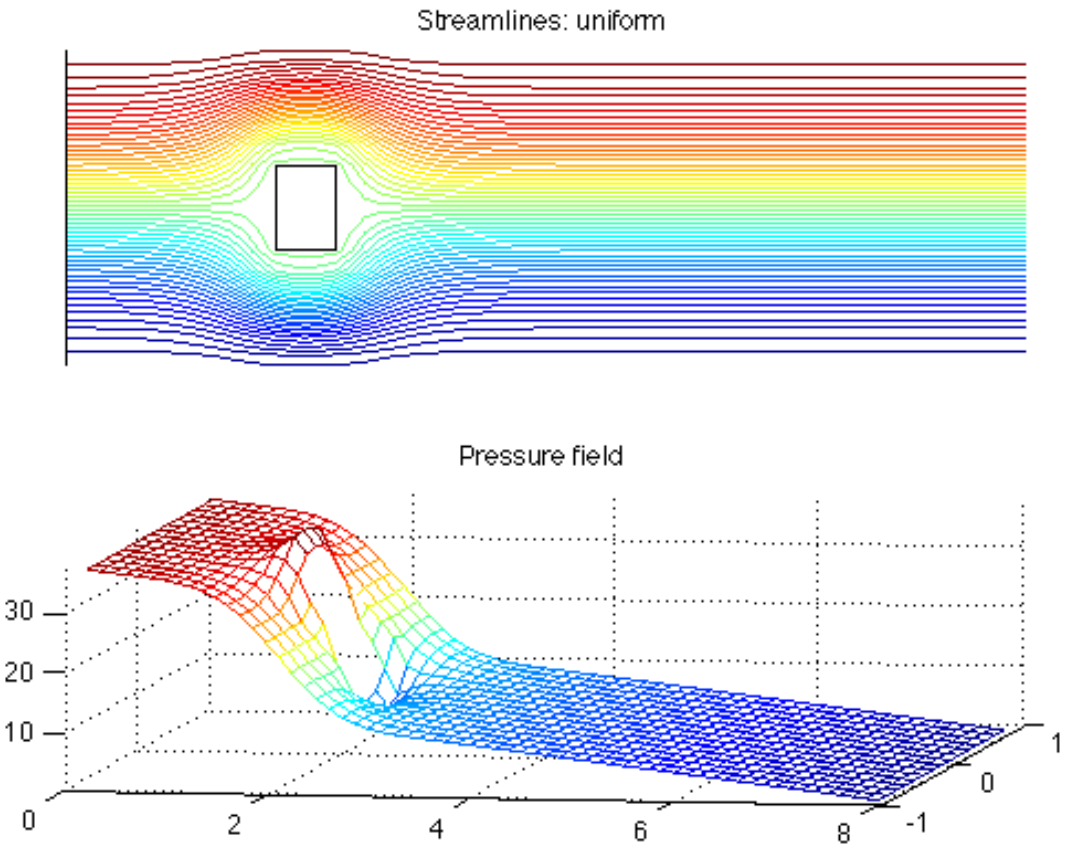
IFISS: Channel Flow With Obstacle (More Choices)

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
```

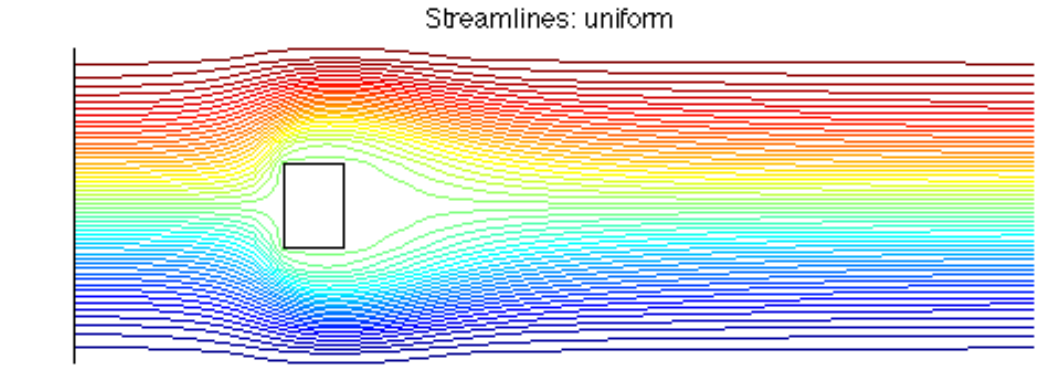
IFISS: Stokes Flow

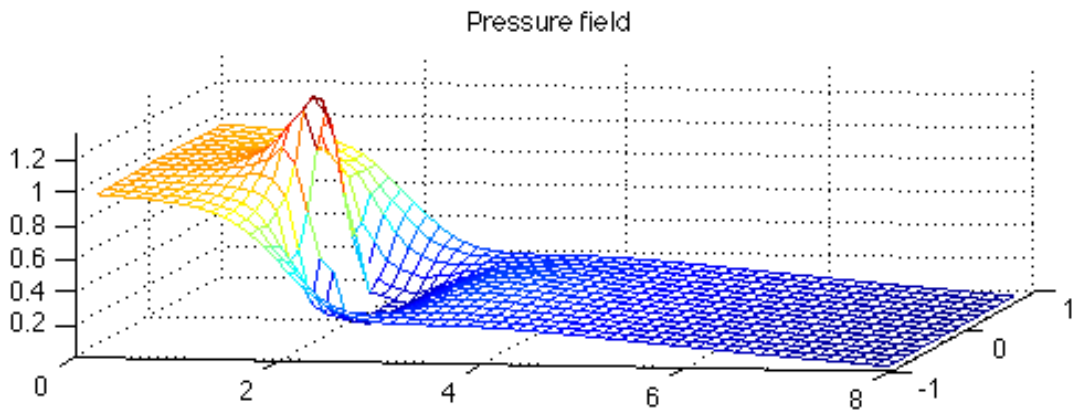
IFISS displays the Stokes flow used for initialization.



IFISS: Navier Stokes Flow

The final Navier-Stokes solution shows significant differences.





IFISS: Overview

If you are interested in flow problems, IFISS is a great place to start.

You can learn a lot just by looking at how it is put together.

You can easily set up new 2D problems (new geometry, boundary conditions, source terms)

You can also use it as a starting point for new algorithms you are interested in.

- Introduction
- Equations of Fluid Motion
- A Finite Element Formulation
- The Mapping Function
- Computing Basis Functions
- Assembling the Matrix
- IFISS
- **Conclusion**

CONCLUSION: The Big Picture

I hope I have shown you that the finite element ideas we have talked about this semester, applied to simple ODE's or boundary value problems, can be extended to somewhat-scary-looking PDE's associated with complicated physical processes.

I emphasize that we encountered a complicated grid, a pair of variables, one a vector and the other a scalar, two kinds of basis functions, and a real nightmare trying to keep track of how to assemble the matrix. But I hope that you saw that we were always doing finite element kinds of things, just on a more complicated system.

To get us there quickly, I left out many details, such as boundary conditions, compressibility, temperature and time dependence, and nonlinearity. Each of these features adds more power to the model, but makes it harder to see the underlying structures.

CONCLUSION: More Things I Left Out

Many of the things I've told you about the quadratic elements and their basis functions are only true in the case where the quadratic elements have straight sides.

It's possible to define a quadratic finite element triangulation in which the elements have curved sides. This allows a better matching of unusual boundary shapes, but it makes everything much more complicated!

The Navier-Stokes equations are handled similarly to the Stokes equations. However, when we form the finite element system, it is nonlinear. Therefore, some kind of iterative method is needed to solve the system. The matrix associated with Newton's method for the Navier-Stokes case is quite similar to the matrix employed in the Stokes equations.

CONCLUSION: Experiments

I have some Stokes and Navier-Stokes programs on my web site. While they might be useful for learning how the equations can be handled, they are not very sophisticated, can't solve big problems, and are not extensively documented.

You're probably much better off trying out the IFISS program, which includes several different PDE's, standard test problems, built in iterative solvers, and some sophisticated error analysis. And since it's written in MATLAB, you can take apart and examine the pieces that interest you.

This will be our last presentation in the finite element seminar. I hope you have had a taste of the many aspects of the finite element method; you probably don't have to write your own finite element program, but you will surely run across problems in your scientific computing work in which some of this material will come back to help you!

CONCLUSION: References:

- http://www.cs.umd.edu/∼elman/ifiss.html
- Howard Elman, Alison Ramage, David Silvester, **Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics**, Oxford, 2005, ISBN: 978-0198528678, LC: QA911.E39.
- Howard Elman, Alison Ramage, David Silvester, *Algorithm 866: IFISS, A Matlab Toolbox for Modelling Incompressible Flow*, ACM Transactions on Mathematical Software, Volume 33, Number 2, June 2007, Article 14.
- Howard Elman, Alison Ramage, David Silvester, *Incompressible Flow Iterative Solution Software (IFISS), Installation and Software Guide*, Version 3.1, released 25 January 2011, http://www.cs.umd.edu/∼elman/ifiss/ifiss_guide_3.1.pdf