Finite Element Treatment of the Navier Stokes Equations, Part IV

John Burkardt
School of Computational Science
Florida State University
http://people.sc.fsu.edu/~jburkardt/presentations/fem_ns4.pdf

19 August 2005

1 Introduction

In the previous discussion, we considered the continuous Stokes equations, developed the weak form, constructed a set of basis functions using the finite element method, and arrived at a set of linear discretized equations whose solution would give the finite element approximation to the Stokes equations. Solving the system of linear equations is a relatively straightforward matter.

This laid the groundwork for the current discussion, in which we will rapidly go through the same steps for the continuous Navier Stokes equations. However, the result will be a set of *nonlinear* discretized equations. The solution of a set of nonlinear equations is significantly more complicated.

In particular, Newton's method needs some form of derivative operator. In order to understand how to compute and use the appropriate derivative operator for our problem, we will go back to the continuous Navier Stokes equations and derive the form of Newton's method, and the associated derivative operator. We will then be able to explain what the corresponding derivative operator will be in the discrete case, and apply Newton's method to the nonlinear discretized equations, to arrive at our solution.

We begin by rapidly marching through the steps required to reach the discretized system.

2 Evaluating The Strong Navier-Stokes Equations

Starting from the Stoke equations, the Navier Stokes equations simply add a nonlinear term, with a factor R that determines the strength of the nonlinearity. Briefly, we say a pair $(\mathbf{v}, p)$ of functions is a strong solution of the Navier Stokes equations if:

$$-\Delta \mathbf{v} + R \quad (\mathbf{v} \cdot \nabla)\mathbf{v} + \nabla p = 0 \tag{1}$$
$$\nabla \mathbf{v} = 0 \tag{2}$$

3 The Weak Form of the Navier-Stokes Equations

Multiplying the momentum equations by any velocity basis function ψ_i and the continuity equation by any pressure basis function ϕ_j, and then integrating over the domain Ω, we arrive at the weak form:

$$\int_\Omega (-\Delta \mathbf{v} + R(\mathbf{v} \cdot \nabla)\mathbf{v} + \nabla p)\, \psi_i \, d\Omega = 0 \tag{3}$$
$$\int_\Omega \nabla \mathbf{v} \, \phi_j \, d\Omega = 0 \tag{4}$$

4 Lowering the Order using Integration by Parts

Integration by parts reduces the weak equations to the following form:

$$\int_\Omega \nabla \mathbf{v} \cdot \nabla \psi_i + (R(\mathbf{v} \cdot \nabla)\mathbf{v} + \nabla p)\, \psi_i \, d\Omega = 0 \tag{5}$$
$$\int_\Omega \nabla \mathbf{v} \, \phi_j \, d\Omega = 0 \tag{6}$$

5   The Discretized Weak Navier Stokes Equations

Exactly as in the case of the weak Stokes equations, we construct a set of basis functions for the discretized solution space, and seek solutions of the weak Navier Stokes equations which can be represented in the form:

$$
\begin{aligned}
\mathbf{v}^h(x,y) &= \sum_{j=1}^{N_u} \mathbf{vval}_j \; \psi_j(x,y) \\
p^h(x,y) &= \sum_{j=1}^{N_p} pval_j \; \phi_j(x,y)
\end{aligned}
$$

Thus, our discretized weak system of equations is:

$$
\int_\Omega \nabla \mathbf{v}^h \cdot \nabla \psi_i + (R(\mathbf{v^h} \cdot \nabla)\mathbf{v^h} + \nabla p^h)\, \psi_i \, d\Omega \;=\; 0 \tag{7}
$$

$$
\int_\Omega \nabla \mathbf{v}^h \, \phi_i \, d\Omega \;=\; 0
$$

It may help to write these equations out in some detail, that is, listing the individual components of the velocity, and spelling out the differential operators. This is the set of equations (plus unspecified linearized boundary conditions) that will constitute the Newton function $H(X)$ for which we will be seeking a zero:

$$
\int_\Omega \frac{\partial u^h}{\partial x}\frac{\partial \psi_i}{\partial x} + \frac{\partial u^h}{\partial y}\frac{\partial \psi_i}{\partial y} + (R\,(u^h\frac{\partial u^h}{\partial x} + v^h\frac{\partial u^h}{\partial y}) + \frac{\partial p^h}{\partial x})\,\psi_i\,d\Omega \;=\; 0 \tag{8}
$$

$$
\int_\Omega \frac{\partial v^h}{\partial x}\frac{\partial \psi_i}{\partial x} + \frac{\partial v^h}{\partial y}\frac{\partial \psi_i}{\partial y} + (R\,(u^h\frac{\partial v^h}{\partial x} + v^h\frac{\partial v^h}{\partial y}) + \frac{\partial p^h}{\partial y})\,\psi_i\,d\Omega \;=\; 0 \tag{9}
$$

$$
\int_\Omega (\frac{\partial u^h}{\partial x} + \frac{\partial v^h}{\partial y})\,\phi_i\,d\Omega \;=\; 0 \tag{10}
$$

6   Newton's Method for the Continuous Navier Stokes Equations

Newton's method for solving a system of nonlinear equations $H(X) = 0$ assumes that we have an approximate solution X_0 and a derivative operator $H'(X)$ which allows us to approximate the behavior of the function H around any point X_0 as a constant plus linear plus nonlinear terms:

$$
H(X) = H(X_0) + H'(X_0)\,(X - X_0) + O(\|X - X_0\|^2). \tag{11}
$$

You may be familiar with Newton's method in the case where X and the function $H(X)$ are scalar-valued. The method can be described formally in the same way for the cases where X and $H(X)$ are vectors or even functions. However, it is important in these more complicated cases to understand what kind of object H' will be, and how it can be computed.

If X and $H(X)$ are scalars, then of course $H'(X)$ is simply the usual derivative function. If X and $H(X)$ are vectors, then $H'(X)$ is the *Jacobian matrix*. But in the case of current interest to us, where we want to consider the Navier Stokes equations in their original, continuous setting, then X is a collection of functions (a vector-valued velocity function $(u,v)(x,y)$ and a scalar-valued pressure function $p(x,y)$), $H(X)$ is the Navier-Stokes equations plus boundary conditions evaluated with those functions, and H' must be some kind of linear operator. In this case, the job of H' is to estimate the value of the residual of the Navier Stokes equation at any new flow field X, based entirely on the value of the current flow field X_0 and its residual $H(X_0)$. If this is a new or unclear concept, then it is best not to worry about it now; continue with the discussion, which may help to clarify the role of H'.

As it stands, Equation (**??**) is true, but useless as far as computation goes! Let us simplify the equation by dropping the nonlinear terms, in which case we have the true, useless, but simpler equation:

$$H(X) \approx H(X_0) + H'(X_0) \, (X - X_0). \tag{12}$$

As long as $H'(X_0)$ is not singular, then in some neighborhood, this linear approximation is actually fairly good. Therefore, we might want to consider a local linear approximation $h(X)$ to $H(X)$, which we define by replacing the approximation sign by an equality:

$$h(X) = H(X_0) + H'(X_0) \, (X - X_0). \tag{13}$$

Equations are so much more useful than approximations! We know this model function $h(X)$ is only valid in a small neighborhood of X_0, but since $h(X)$ is a linear function, we can analyze it completely. One of the most interesting things about a function is to find where it is zero. A formula for the root x^* for which $h(x^*) = 0$ is instantly obtainable:

$$x^* = X_0 - (H'(X_0))^{-1} \, H(X_0). \tag{14}$$

Of course, x^* is just a root of $h(X)$; but if $h(X)$ is a good model of $H(X)$, then it is reasonable to assume that x^* is also a good estimate of the root X^* for which $H(X^*) = 0$.

We are actually going to be generating a sequence of estimates for the root by using this idea of a linear model. Although generated by the linear models, we hope that this sequence tends to converge to the root X^* of the nonlinear equation, and so we will label this sequence using a superscripted capital X.

Thus, we regard our initial estimated value X_0 as the first element of the sequence, and we refere to it as X^0. The next value in the sequence is obtained from Equation (**??**), and is denoted by X^1. If X^1 is a better approximation to the root than X^0 was, then it is presumably a good idea to reconstruct the model linear function base at X_1 and solve for the root of this new linear equation. But that simply means that on our next step, we want to evaluate the right hand side of Equation (**??**) at X^1 now.

In many cases, this procedure, *the Newton iteration*, will produce a sequence of rapidly improving estimates for the root of the original nonlinear function. The Newton iteration consists of repeatedly evaluating the right hand side of Equation (**??**) and using the result as the new estimate of the root:

$$X^{k+1} = X^k - (H'(X^k))^{-1} \, H(X^k). \tag{15}$$

Note that we can also write this equation in the equivalent implicit form:

$$H'(X^k) \, (X^{k+1} - X^k) = -H(X^k). \tag{16}$$

in which case, we have to solve some sort of linear algebraic equation or linear functional equation, for the *Newton increment*

$$\Delta X^k \equiv X^{k+1} - X^k. \tag{17}$$

We will in fact generally want to think of this implicit equation in the form:

$$H'(X^k) \, \Delta X^k = -H(X^k). \tag{18}$$

Depending on the situation, we may prefer the explicit or implicit form of the Newton iteration, and we may want to talk in terms of the next iterate X^{k+1} or the increment ΔX^k.

To apply Newton's method to our flow problem, we need a way of defining the derivative operator H' of the function H. This is a simple task in the typical discretized case, when both H and its argument X are vectors of real numbers. But we need to be able to define H' in the continuous case as well, where H and X are functions, or even sets of functions (horizontal and vertical velocity and the pressure, for instance). In such a case, the simple vector calculus definitions of the derivative as a Jacobian matrix are not useful.

To see what is going on, it really helps to go back to the definition of the derivative as the limit of difference quotients. Let us start with any evaluation point X_0. We pick an arbitrary increment function ΔX. We now want to compare the value of H at X_0 to its values along the one dimensional family of points defined by $X_0 + \epsilon \Delta X$.

This allows us to define a difference quotient whose divisor is a real number. If the resulting expression has a limit, then that limit is termed the *Gateaux derivative* of H in the direction ΔX:

$$H'_G(X_0)(\Delta(X)) \equiv \lim_{\epsilon \to 0} \frac{H(X^0 + \epsilon \Delta X) - H(X^0)}{\epsilon}. \tag{19}$$

Of course, from the definition, the Gateaux derivative only gives information along a particular direction; if the Gateaux derivative exists at X_0 for a particular increment direction ΔX, then it may or may not exist for another increment direction; and if it exists for two distinct increment directions from X_0, there is no guarantee that the two Gateaux derivatives are equal! However, the definition of the Gateaux derivative should start to give you a picture of what it might mean when we speak of the derivative of a function whose argument is itself a function.

In most interesting cases, the Gateaux derivative exists, and is independent of the increment direction, and is continuous in some neighborhood of the point of interest X_0. In this case, it graduates to being a *Frechet derivative*. Although Frechet derivatives are really just the same kind of difference quotient as Gateaux derivatives, their description might seem different, since they are usually described in terms of differences between the value of H at X_0 and a second nearby point X_1:

$$\lim_{\|X_1 - X_0\| \to 0} \frac{\|H(X_1) - H(X_0) - H'_F(X_0)(X_1 - X_0)\|}{\|X_1 - X_0\|} = 0. \tag{20}$$

For our work, the typical argument X_0 will be the triple of state functions (u, v, p); the nonlinear function H will be the Navier Stokes equations, which we will write $F(u, v, p) = f$, and the boundary conditions, which we will write $G(u, v, p) = g$. We will assume that the function H is always Frechet differentiable. Newton's method requires a "suitable" starting estimate $X = (u, v, p)$ of the solution, which can be any functions of (x, y) defined over Ω, and which need not satisfy the boundary conditions.

We may then symbolize the unknowns as:

$$X \equiv \begin{pmatrix} u \\ v \\ p \end{pmatrix} \tag{21}$$

and the system of nonlinear equations as:

$$H(X) \equiv \begin{pmatrix} F(u, v, p) - f \\ G(u, v, p) - g \end{pmatrix}. \tag{22}$$

To work out the linearization operator $H'(X_0)(\Delta X)$ for the Navier Stokes equations, you should simply carry out the usual difference quotient estimation using a Gateaux derivative approach. We will write $X_0 = (u_0, v_0, p_0)$ for the flow values, and $\Delta X = (\Delta u, \Delta v, \Delta p)$ for the increment direction. Now, write down the Navier Stokes equations for the functions (u_0, v_0, p_0), and then write them down again for $(u_0 + \epsilon \Delta u, v_0 + \epsilon \Delta v, p_0 + \epsilon \Delta p)$. (You can probably save some time if you assume you know that will happen to the simple linear terms in the equations.) Subtract the two equations, divide by ϵ, and then let ϵ go to zero to get the value of $H'(X_0)(\Delta X)$.

The linearized Navier Stokes equations you will arrive are known as the *Oseen equations*. These are a sort of set of flow equations for the increment functions. Note, in particular, that the velocity increment functions satisfy the same continuity equations as the velocity functions do:

$$-(\frac{\partial^2 \Delta u}{\partial x^2} + \frac{\partial^2 \Delta u}{\partial y^2}) + R \, (\Delta u \frac{\partial u_0}{\partial x} + u_0 \frac{\partial \Delta u}{\partial x} + \Delta v \frac{\partial u_0}{\partial y} + v_0 \frac{\partial \Delta u}{\partial y}) + \frac{\partial \Delta p}{\partial x} = 0 \tag{23}$$

$$-(\frac{\partial^2 \Delta v}{\partial x^2} + \frac{\partial^2 \Delta v}{\partial y^2}) + R \, (\Delta u \frac{\partial v_0}{\partial x} + u_0 \frac{\partial \Delta v}{\partial x} + \Delta v \frac{\partial v_0}{\partial y} + v_0 \frac{\partial \Delta v}{\partial y}) + \frac{\partial \Delta p}{\partial y} = 0 \tag{24}$$

$$\frac{\partial \Delta u}{\partial x} + \frac{\partial \Delta v}{\partial y} = 0 \tag{25}$$

5

You should study these equations, and notice the similarities and differences compared to the Navier Stokes equations. Keep in mind the distinction between the flow quantities and the increment quantities. The flow field quantities are *fixed, known, given functions*; the increment quantities are *freely varying, small, unknown increment functions*; they represent small changes to an existing flow.

Of course, for a given problem, we also have to differentiate whatever side conditions and boundary conditions we have imposed. Formally, the linearized boundary conditions will be part of our operator H', and will have the following form:

$$G'(u_0, v_0, p_0)\ (\Delta u, \Delta v, \Delta p) = G_u(u_0, v_0, p_0)\Delta u + G_v(u_0, v_0, p_0)\Delta v + G_p(u_0, v_0, p_0)\Delta p = 0. \qquad (26)$$

We have had to leave the boundary conditions in this fairly general form, since we wish to be free to choose a variety of different conditions. Usually, the actual boundary equations being employed will be so simple that their linearization will be no challenge at all.

In fact, for our current problem, recall that we assumed the boundary conditions were constant in the unknowns u, v, and p. This implies that the linearized boundary conditions are formally identical to the original boundary conditions on the left hand side, but with a homogeneous right hand side.

For instance, an original boundary condition $u(x, 0) = x^2$ (which is a constant function, in terms of u, v and p) would, after differentiation and evaluation at the increment $(\Delta u, \Delta v, \Delta p)$, become $\Delta u(x, 0) = 0$. The fact that the boundary conditions for the linearized problem don't change much from those for the nonlinear problem will be helpful later.

7   Evaluating a Flow Field Variable

It is going to be convenient to be able to request the value of the state variables and their derivatives at any point in the region. Logically, this is a simple request, but the details of the process can be a bit involved. Thus, it is best to encapsulate this information as a function. We will find this routine usefule when we need to evaluate the residual error associated with an approximate solution of the Navier Stokes equation, but once it is written, it will also be very helpful in plotting, computing tables, and writing out files.

This version of the routine is to be used to evaluate the horizontal or vertical velocity at a point in a given 6-node element. We are given the coordinates of the nodes, the corresponding finite element coefficients, and a number of points within the element at which the (horizontal or vertical) velocity is to be evaluated.

A similar routine can be written for the pressure, but using the three-node element.

```
function [ u, dudx, dudy ] = flowfield_value_t6 ( t6, cu, n, xy );

%% FLOWFIELD_VALUE_T6 evaluates a quantity associated with a quadratic triangle.
%
%  Modified:
%
%    08 July 2005
%
%  Author:
%
%    John Burkardt
%
%  Parameters:
%
%    Input, real T3(2,6), the coordinates of the triangle.
%
%    Input, real CU(6), the coefficients.
%
```

```
%    Input, integer N, the number of evaluation points.
%
%    Input, real XY(2,N), the coordinates of the evaluation points.
%
%    Output, real U(N), DUDX(N), DUDY(N), the value of the pressure and its
%    derivatives at the evaluation points.
%
  [ b, dbdx, dbdy ] = basis_t6 ( t6, n, xy );

   u(1:n)  = cu(1:6) *  b  (1:6,1:n);
  dudx(1:n) = cu(1:6) * dbdx(1:6,1:n);
  dudy(1:n) = cu(1:6) * dbdx(1:6,1:n);
```

8 Code to Assemble the Navier Stokes Residual

A Newton iteration strives to find a zero of a function $f(x)$. The function $\mathbf{f(x)}$ is often called the residual function. In order to apply Newton's method, the user must write a routine that evaluates the residual function for a given argument $\mathbf{x}$. We are working with finite elements; therefore, we may be more comfortable thinking of our unknown as being the vector $\mathbf{c}$ of finite element coefficients.

In some ways, this code is similar to the code used to assemble the Stokes equations. However, the Stokes equations are linear, and the assembly routine there was required to produce the matrix $\mathbf{A}$ for which $A \cdot c = rhs$ and therefore, the residual for the Stokes equations would be computed as $f = A \cdot c - rhs$

Since the Stokes equations are linear, we don't actually evaluate the state functions directly, but work with the coefficients c. For the Navier Stokes equations, we need to use the coefficients c to evaluate the state variables u, v, and p, and their derivatives, in order to evaluate the more complicated residual function.

Despite the extra complications, you should still be able to go to the final lines of this function, and identify the weak form of the momentum equations and the continuity equation, which constitute the residual!

The following function demonstrates how one might compute the appropriate residual function for the Navier Stokes equations.

```
function r = residual_navier_stokes ( node_num, node_xy, triangle_num, ...
  triangle_node, quad_num, node_u_variable, node_v_variable, ...
  node_p_variable, variable_num, reynolds, c )

%% RESIDUAL_NAVIER_STOKES evaluates the Navier Stokes residual.
%
%  Discussion:
%
%    The residual terms due to boundary conditions must be applied separately.
%
%    The Navier Stokes equations in weak form are:
%
%      Integral ( ( dBdx(I) * dUdx + dBdy(I) * dUdy )
%        + B(I) * ( R * ( U * dUdx + V * dUdy ) + dPdx - U_RHS ) ) = 0
%
%      Integral ( ( dBdx(I) * dVdx + dBdy(I) * dVdy )
%        + B(I) * ( R * ( U * dVdx + V * dVdy ) + dPdy - V_RHS ) ) = 0
%
%      Integral ( Q(I) * ( dUdx + dVdy - P_RHS ) ) = 0
%
```

```
%  Modified:
%
%    19 August 2005
%
%  Author:
%
%    John Burkardt
%
%  Parameters:
%
%    Input, integer NODE_NUM, the number of nodes.
%
%    Input, real NODE_XY(2,NODE_NUM), the coordinates of the nodes.
%
%    Input, integer TRIANGLE_NUM, the number of triangles.
%
%    Input, integer TRIANGLE_NODE(6,TRIANGLE_NUM), the nodes of each triangle.
%
%    Input, integer QUAD_NUM, the order of the quadrature rule.
%
%    Input, integer NODE_P_VARIABLE(NODE_NUM),
%    is the index of the pressure variable associated with the node,
%    or -1 if there is no associated pressure variable.
%
%    Input, integer NODE_U_VARIABLE(NODE_NUM),
%    is the index of the horizontal velocity variable associated with the node.
%
%    Input, integer NODE_V_VARIABLE(NODE_NUM),
%    is the index of the vertical velocity variable associated with the node.
%
%    Input, integer VARIABLE_NUM, the number of unknowns.
%
%    Input, real REYNOLDS, the value of the Reynolds number.
%
%    Input, real C(VARIABLE_NUM), the finite element coefficients of an
%    approximate solution of the Navier Stokes equations.
%
%    Output, real R(VARIABLE_NUM), the Navier Stokes residual.
%

%
%  Initialize the residual to 0.
%
  r(1:variable_num) = 0.0;
%
%  Get the quadrature weights and nodes.
%
  [ quad_w, quad_xy ] = quad_rule ( quad_num );
%
%  Consider all quantities associated with a given TRIANGLE.
%
```

8

```
   for triangle = 1 : triangle_num
%
%  Make a copy of the triangle.
%
     t3(1:2,1:3) = node_xy(1:2,triangle_node(1:3,triangle));
     t6(1:2,1:6) = node_xy(1:2,triangle_node(1:6,triangle));
%
%  Map the quadrature points QUAD_XY to points XY in the physical triangle.
%
     xy(1:2,1:quad_num) = reference_to_physical_t3 ( t3, quad_num, quad_xy );

     w(1:quad_num) = quad_w(1:quad_num) * triangle_area ( t3 );
%
%  Evaluate the basis functions at the quadrature points.
%
     [ b, dbdx, dbdy ] = basis_t6 ( t6, quad_num, xy );
     [ q, dqdx, dqdy ] = basis_t3 ( t3, quad_num, xy );
%
%  Extract the indices of the finite element coefficients for this triangle.
%
     iu(1:6) = node_u_variable(triangle_node(1:6,triangle));
     iv(1:6) = node_v_variable(triangle_node(1:6,triangle));
     ip(1:3) = node_p_variable(triangle_node(1:3,triangle));
%
%  Extract the finite element coefficients for this triangle.
%
     cu(1:6) = c(iu(1:6));
     cv(1:6) = c(iv(1:6));
     cp(1:3) = c(ip(1:3));
%
%  Evaluate the flowfield at each quadrature point.
%
      u  (1:quad_num) = cu(1:6) *  b  (1:6,1:quad_num);
     dudx(1:quad_num) = cu(1:6) * dbdx(1:6,1:quad_num);
     dudy(1:quad_num) = cu(1:6) * dbdx(1:6,1:quad_num);

      v  (1:quad_num) = cv(1:6) *  b  (1:6,1:quad_num);
     dvdx(1:quad_num) = cv(1:6) * dbdx(1:6,1:quad_num);
     dvdy(1:quad_num) = cv(1:6) * dbdx(1:6,1:quad_num);

      p  (1:quad_num) = cp(1:3) *  q  (1:3,1:quad_num);
     dpdx(1:quad_num) = cp(1:3) * dqdx(1:3,1:quad_num);
     dpdy(1:quad_num) = cp(1:3) * dqdx(1:3,1:quad_num);
%
%  Evaluate the right hand side at each quadrature point.
%
     [ u_rhs, v_rhs, p_rhs ] = rhs ( quad_num, xy );
%
%  Consider the contribution to the integrals from the QUAD-th quadrature point.
%
     for quad = 1 : quad_num
```

```
      for i = 1 : 6

        r(iu(i)) = r(iu(i)) + w(quad) * ( ...
          ( dbdx(i,quad) * dudx(quad) + dbdy(i,quad) * dudy(quad) ) ...
          + b(i,quad) * ( ...
          reynolds * ( u(quad) * dudx(quad) + v(quad) * dudy(quad) ) ...
          + dpdx(quad) - u_rhs(quad) ) );

        r(iv(i)) = r(iv(i)) + w(quad) * ( ...
          ( dbdx(i,quad) * dvdx(quad) + dbdy(i,quad) * dvdy(quad) ) ...
          + b(i,quad) * ( ...
          reynolds * ( u(quad) * dvdx(quad) + v(quad) * dvdy(quad) ) ...
          + dpdy(quad) - v_rhs(quad) ) );

      end

      for i = 1 : 3

        r(ip(i)) = r(ip(i)) + w(quad) * q(i,quad) * ( ...
          dudx(quad) + dvdy(quad) - p_rhs(quad) );

      end

    end

  end
```

9 Newton's Method for the Discretized Weak Navier Stokes Equations

We assume that by now, you can fill in the steps that are necessary to transform the continuous Oseen equations to a discretized weak form, including multiplication by a test function, integration over the region, and integration by parts.

The resulting set of equations (plus unspecified linearized boundary conditions) will constitute the linear system that must be repeatedly solved for the increments $(\Delta u, \Delta v, \Delta p)$ that allow us to update our estimate (u^0, v^0, p^0) for the solution of the discretized weak Navier-Stokes equations:

$$\int_{\Omega} \frac{\partial \Delta u}{\partial x}\frac{\partial \psi_i}{\partial x} + \frac{\partial \Delta u}{\partial y}\frac{\partial \psi_i}{\partial y} + (R\,(\Delta u \frac{\partial u^0}{\partial x} + u^0 \frac{\partial \Delta u}{\partial x} + \Delta v \frac{\partial u^0}{\partial y} + v^0 \frac{\partial \Delta u}{\partial y}) + \frac{\partial \Delta p}{\partial x})\,\psi_i\,d\Omega \quad = \quad 0 \quad (27)$$

$$\int_{\Omega} \frac{\partial \Delta v}{\partial x}\frac{\partial \psi_i}{\partial x} + \frac{\partial \Delta v}{\partial y}\frac{\partial \psi_i}{\partial y} + (R\,(\Delta u \frac{\partial v^0}{\partial x} + u^0 \frac{\partial \Delta v}{\partial x} + \Delta v \frac{\partial v^0}{\partial y} + v^0 \frac{\partial \Delta v}{\partial y}) + \frac{\partial \Delta p}{\partial y})\,\psi_i\,d\Omega \quad = \quad 0 \quad (28)$$

$$\int_{\Omega} (\frac{\partial \Delta u}{\partial x} + \frac{\partial \Delta v}{\partial y})\,\phi_i\,d\Omega \quad = \quad 0 \quad (29)$$

It's easy to become confused by all the details, so let's recall what is happening here. We are describing what has to happen during one step of the Newton iteration. So we assume we have an initial estimate for the solution, (u^0, v^0, p^0), or, in general, the current k-th estimate (u^k, v^k, p^k). We set up the above linear system to get the increments or corrections, $(\Delta u, \Delta v, \Delta p)$, in order to arrive at an improved estimate

$$(u^{k+1}, v^{k+1}, p^{k+1}) = (u^k, v^k, p^k) + (\Delta u, \Delta v, \Delta p) \tag{30}$$

10 Code for the Jacobian Matrix

The jacobian is computed by differentiating the residual with respect to each unknown. In this case, the unknowns are the finite element coefficients; differentiation of a term like **dudx(quad)** by the coefficient associated with the horizontal velocity coefficient associated with basis node **j** results in a differentiated term of **dbdx(j,quad)**, for instance.

You should be able to derive the following jacobian routine by "differentiating" the residual routine line by line!

```
function a = jacobian_navier_stokes ( node_num, node_xy, triangle_num, ...
  triangle_node, quad_num, node_u_variable, node_v_variable, ...
  node_p_variable, variable_num, reynolds, c )

%% JACOBIAN_NAVIER_STOKES evaluates the Navier Stokes jacobian.
%
%  Discussion:
%
%    The residual terms due to boundary conditions must be applied separately.
%
%    The Navier Stokes equations in weak form are:
%
%       Integral ( ( dBdx(I) * dUdx + dBdy(I) * dUdy )
%          + B(I) * ( R * ( U * dUdx + V * dUdy ) + dPdx - U_RHS ) ) = 0
%
%       Integral ( ( dBdx(I) * dVdx + dBdy(I) * dVdy )
%          + B(I) * ( R * ( U * dVdx + V * dVdy ) + dPdy - V_RHS ) ) = 0
%
%       Integral ( Q(I) * ( dUdx + dVdy - P_RHS ) ) = 0
%
%  Modified:
%
%    19 August 2005
%
%  Author:
%
%    John Burkardt
%
%  Parameters:
%
%    Input, integer NODE_NUM, the number of nodes.
%
%    Input, real NODE_XY(2,NODE_NUM), the coordinates of the nodes.
%
%    Input, integer TRIANGLE_NUM, the number of triangles.
%
%    Input, integer TRIANGLE_NODE(6,TRIANGLE_NUM), the nodes of each triangle.
%
%    Input, integer QUAD_NUM, the order of the quadrature rule.
%
%    Input, integer NODE_P_VARIABLE(NODE_NUM),
%    is the index of the pressure variable associated with the node,
%    or -1 if there is no associated pressure variable.
```

```
%
%    Input, integer NODE_U_VARIABLE(NODE_NUM),
%    is the index of the horizontal velocity variable associated with the node.
%
%    Input, integer NODE_V_VARIABLE(NODE_NUM),
%    is the index of the vertical velocity variable associated with the node.
%
%    Input, integer VARIABLE_NUM, the number of unknowns.
%
%    Input, real REYNOLDS, the value of the Reynolds number.
%
%    Input, real C(VARIABLE_NUM), the finite element coefficients of an
%    approximate solution of the Navier Stokes equations.
%
%    Output, real A(VARIABLE_NUM,VARIABLE_NUM), the Navier Stokes jacobian.
%

%
%  Initialize the jacobian to 0.
%
  a(1:variable_num,1:variable_num) = 0.0;
%
%  Get the quadrature weights and nodes.
%
  [ quad_w, quad_xy ] = quad_rule ( quad_num );
%
%  Consider all quantities associated with a given TRIANGLE.
%
  for triangle = 1 : triangle_num
%
%  Make a copy of the triangle.
%
    t3(1:2,1:3) = node_xy(1:2,triangle_node(1:3,triangle));
    t6(1:2,1:6) = node_xy(1:2,triangle_node(1:6,triangle));
%
%  Map the quadrature points QUAD_XY to points XY in the physical triangle.
%
    xy(1:2,1:quad_num) = reference_to_physical_t3 ( t3, quad_num, quad_xy );

    w(1:quad_num) = quad_w(1:quad_num) * triangle_area ( t3 );
%
%  Evaluate the basis functions at the quadrature points.
%
    [ b, dbdx, dbdy ] = basis_t6 ( t6, quad_num, xy );
    [ q, dqdx, dqdy ] = basis_t3 ( t3, quad_num, xy );
%
%  Extract the indices of the finite element coefficients for this triangle.
%
    iu(1:6) = node_u_variable(triangle_node(1:6,triangle));
    iv(1:6) = node_v_variable(triangle_node(1:6,triangle));
    ip(1:3) = node_p_variable(triangle_node(1:3,triangle));
```

```
%
%  Extract the finite element coefficients for this triangle.
%
     cu(1:6) = c(iu(1:6));
     cv(1:6) = c(iv(1:6));
     cp(1:3) = c(ip(1:3));
%
%  Evaluate the flowfield at each quadrature point.
%
      u  (1:quad_num) = cu(1:6) *  b  (1:6,1:quad_num);
   dudx(1:quad_num) = cu(1:6) * dbdx(1:6,1:quad_num);
   dudy(1:quad_num) = cu(1:6) * dbdx(1:6,1:quad_num);

      v  (1:quad_num) = cv(1:6) *  b  (1:6,1:quad_num);
   dvdx(1:quad_num) = cv(1:6) * dbdx(1:6,1:quad_num);
   dvdy(1:quad_num) = cv(1:6) * dbdx(1:6,1:quad_num);

      p  (1:quad_num) = cp(1:3) *  q  (1:3,1:quad_num);
   dpdx(1:quad_num) = cp(1:3) * dqdx(1:3,1:quad_num);
   dpdy(1:quad_num) = cp(1:3) * dqdx(1:3,1:quad_num);
%
%  Evaluate the right hand side at each quadrature point.
%
     [ u_rhs, v_rhs, p_rhs ] = rhs ( quad_num, xy );
%
%  Consider the contribution to the integrals from the QUAD-th quadrature point.
%
     for quad = 1 : quad_num
%
%  Derivatives of all 6 momentum equations with respect to all 6 U and V
%  coefficients.  Only here do we have nonlinear terms.
%
        for i = 1 : 6
          for j = 1 : 6

            a(iu(i),iu(j)) = a(iu(i),iu(j)) + w(quad) * ( ...
              ( dbdx(i,quad) * dbdx(j,quad) ...
              + dbdy(i,quad) * dbdy(j,quad) ) ...
              + b(i,quad) * reynolds * ( ...
                b(j,quad) * dudx(quad) + u(quad) * dbdx(j,quad) ...
              + v(quad) * dbdy(j,quad) ) );

            a(iu(i),iv(j)) = a(iu(i),iv(j)) + w(quad) * ...
              reynolds * b(i,quad) * b(j,quad) * dudy(quad);

            a(iv(i),iu(j)) = a(iv(i).iu(j)) + w(quad) * ...
              reynolds * b(i,quad) * b(j,quad) * dvdx(quad);

            a(iv(i),iv(j)) = a(iv(i),iv(j)) + w(quad) * ( ...
              ( dbdx(i,quad) * dbdx(j,quad) ...
              + dbdy(i,quad) * dbdy(j,quad) ) ...
```

```
                  + b(i,quad) * reynolds * ( ...
                    u(quad) * dbdx(j,quad) + b(j,quad) * dvdy(quad) ...
                  + v(quad) * dbdy(j,quad) ) );

          end
        end
%
%  Derivatives of the six momentum equations with respect to the three
%  pressure coefficients.
%
      for i = 1 : 6
        for j = 1 : 3

          a(iu(i),ip(j)) = a(iu(i),ip(j)) + w(quad) * b(i,quad) * dqdx(j,quad);
          a(iv(i),ip(j)) = a(iv(i),ip(j)) + w(quad) * b(i,quad) * dqdy(j,quad);

        end
      end
%
%  Derivatives of the three continuity equations with respect to the
%  six velocity coefficients.
%
      for i = 1 : 3
        for j = 1 : 6

          a(ip(i),iu(j)) = a(ip(i),iu(j)) + w(quad) * q(i,quad) * dbdx(j,quad);
          a(ip(i),iv(j)) = a(ip(i),iv(j)) + w(quad) * q(i,quad) * dbdy(j,quad);

        end
      end

    end

  end
```