Numerical Solutions of the Stokes Equation
by Finite Element Method

Jeremy Knight
December 22, 2011

Premise:

We set out to implement a robust algorithm that will model the Navier-Stokes equations for fluid flows with a small Reynolds number. When the Reynolds number is much less than 1, the two dimensional Navier-Stokes equations become the Stokes equations given by

$$-\Delta\mathbf{u} + \nabla \mathrm{p} = \mathbf{f} \quad \text{in } \Omega \tag{1a}$$
$$\operatorname{div} \mathbf{u} = \mathbf{0} \quad \text{in } \Omega \tag{1b}$$
$$\mathbf{u} = \mathbf{0} \quad \text{on } \Gamma \tag{1c}$$

for velocity $\mathbf{u} = (u_1, u_2)$ and pressure p where Ω us a domain with boundary Γ and $\mathbf{f} = (f_1, f_2)$ is given. We will denote div $A = \frac{\partial A_1}{\partial x_1} + \frac{\partial A_2}{\partial x_2}$, the gradient $\nabla\mathbf{v} = \left(\frac{\partial \mathbf{v}}{\partial x_1}, \frac{\partial \mathbf{v}}{\partial x_2}\right)$, and the Laplacian $\Delta\mathbf{u} = \frac{\partial^2 \mathbf{u}}{\partial x_1^2} + \frac{\partial^2 \mathbf{u}}{\partial x_2^2}$. We will employ a mixed finite element method to discritize the problem and produce an accurate approximate solution.

Plan

To apply the finite element method to the Stokes equations, we will

1) Write a weak formulation of the problem and analyze the discrete analogue,
2) Define appropriate function spaces for the velocity and the pressure spaces,
3) Write basis functions for these spaces,
4) Compute the local stiffness matrices using the local basis functions,
5) Use the local stiffness matrices and a mesh generator to compute the global stiffness matrix S which will be a block matrix containing blocks for both the pressure and velocity,
6) Compute the right-hand vector for the given function $F = \int_\Omega f(x,y) \cdot \mathbf{v}\, dA$ using Gaussian quadrature and form the vector $\mathbf{b} = (F, \mathbf{0})^T$ to represent the right hand of the system $S\mathbf{x} = \mathbf{b}$,
7) Solve the system by computing $\mathbf{x} = S^{-1}\mathbf{b}$, and
8) Plot the values of ξ and θ to represent the solution graphically.

Weak Formulation

We seek u and p in the spaces V and H defined by the Hilbert spaces

$$V = \{\mathbf{v} = (v_1, v_2) : v_i \in H_0^1(\Omega), i = 1,2\}$$
$$H = \left\{q \in L_2(\Omega) : \int_\Omega q\, dx = 0\right\}.$$

Multiplying (1.0) by v and integrating both sides using Green's Theorem gives us

$$-\Delta \mathbf{u} \cdot \mathbf{v} + \nabla p \cdot \mathbf{v} = \mathbf{f}\mathbf{v}$$

$$\int_\Omega -\Delta \mathbf{u} \cdot \mathbf{v}\, dx + \int_\Omega \nabla p \cdot \mathbf{v}\, dx = \int_\Omega \mathbf{f} \cdot \mathbf{v}\, dx$$

$$\left(-\int_\Gamma \mathbf{u} \frac{\partial \mathbf{v}}{\partial n}\, ds + \int_\Omega \nabla \mathbf{u} \cdot \nabla \mathbf{v}\, dx\right) + \left(\int_\Gamma p \frac{\partial \mathbf{v}}{\partial n}\, ds - \int_\Omega p \cdot \operatorname{div} \mathbf{v}\, dx\right) = \int_\Omega \mathbf{f} \cdot \mathbf{v}\, dx$$

$$\int_\Omega \nabla \mathbf{u} \cdot \nabla \mathbf{v}\, dx - \int_\Omega p \cdot \operatorname{div} \mathbf{v}\, dx = \int_\Omega \mathbf{f} \cdot \mathbf{v}\, dx$$

$$\langle \nabla \mathbf{u}, \nabla \mathbf{v} \rangle - \langle p, \operatorname{div} \mathbf{u} \rangle = \langle \mathbf{f}, \mathbf{v} \rangle$$

where $\langle ., . \rangle$ denotes the L_2-inner products. Since we have vector equations, this gives us

$$\langle \nabla \mathbf{u}, \nabla \mathbf{v} \rangle = \sum_{i=1}^{2} \int_\Omega \nabla u_i \cdot \nabla v_i\, dx$$

$$\langle \mathbf{f}, \mathbf{v} \rangle = \sum_{i=1}^{2} \int_\Omega \mathrm{f_i} \cdot v_i\, dx.$$

Discrete problem and the Function Space

We now can discritize the weak formulation by replacing the function spaces V and H by the finite dimensional subspaces V_h and H_h. We thus seek a solution$(u_h, p_h) \in V_h \times H_h$ such that

$$\langle \nabla u_h, \nabla v \rangle - \langle p_h, \operatorname{div} v \rangle = \langle f, v \rangle, \qquad \forall v \in V_h, \tag{2a}$$

$$\langle q, \operatorname{div} u_h \rangle = 0, \qquad \forall q \in H_h. \tag{2b}$$

We will use a mixed finite element analysis on the square finite element space, $T_h = [0,1]^2$. We will divide this region into squares such that the side length of any square K is $h = 1/M$ for a given positive integer M. Our function space for velocity will be

$$V_h = \{v \in V : V|_K \in [Q_2(K)]^2, \forall K \in T_h\},$$

where $Q_2(K)$ is the set of all biquadratic functions on element Ksuch that

$$Q_2(K) = \left\{ v : v(\mathbf{x}) = \sum_{i,j=0}^{2} c_{ij}\, x_1^i x_2^j, \mathbf{x} \in K, a_{ij} \in \mathbb{R} \right\}.$$

Our global degrees of freedom for these functions are

(i) The values of the nodes of T_h,
(ii) The values of the midpoints of the sides of T_h, and
(iii) The values of the midpoints of each rectangle $K \in T_h$.

Our function space for the pressure will be

$$H_h = \{q \in H : q|_K \in P_1(K), \forall K \in T_h\},$$

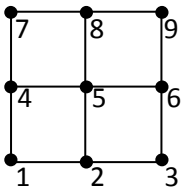
where $P_1(K)$ is the set of all piecewise linear functions on the element Ksuch that

$$P_1(K) = \left\{ q : q(\mathbf{x}) = \sum_{0 \le i+j \le 1} c_{ij} x_1^i x_2^j, \mathbf{x} \in K, a_{ij} \in \mathbb{R} \right\}.$$

Our global degrees of freedom for $P_1(K)$ is the value at a given vertex and the vertices connected by an edge. The fourth vertex of the square region does not contribute to the support of the given node.

Basis Functions for V_h

We will now designate basis functions $\{\phi_1, \dots, \phi_9\}$ for the velocity space V_h. We will first analyze the reference square $\hat{\tau} = [0,1]^2$ and find the coefficients of the basis functions. Then we will develop the local stiffness matrix using the basis functions.



We begin by numbering the local nodes on the reference square as shown to the right. We then compute the coefficients for the basis functions such that $\phi_i = 1$ at node N_i and $\phi_i = 0$ at all other nodes as described in [3].

This gives us the coefficient matrix for ϕ_i

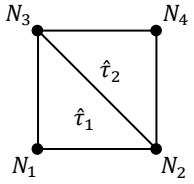
$$\hat{\phi} = \begin{pmatrix} 1 & -3 & 2 & -3 & 9 & -6 & 2 & -6 & 4 \\ 0 & 4 & -4 & 0 & -12 & 12 & 0 & 8 & -8 \\ 0 & -1 & 2 & 0 & 3 & -6 & 0 & -2 & 4 \\ 0 & 0 & 0 & 4 & -12 & 8 & -4 & 12 & -8 \\ 0 & 0 & 0 & 0 & 16 & -16 & 0 & -16 & 16 \\ 0 & 0 & 0 & 0 & -4 & 8 & 0 & 4 & -8 \\ 0 & 0 & 0 & -1 & 3 & -2 & 2 & -6 & 4 \\ 0 & 0 & 0 & 0 & -4 & 4 & 0 & 8 & -8 \\ 0 & 0 & 0 & 0 & 1 & -2 & 0 & -2 & 4 \end{pmatrix}$$

If $\hat{\phi}_{i*}$ represents the i^{th} row of $\hat{\phi}$, then the basis function are then defined as the polynomial

$$\phi_i = \hat{\phi}_{i*}(1 \quad x_2 \quad x_2^2 \quad x_1 \quad x_1x_2 \quad x_1x_2^2 \quad x_1^2 \quad x_1^2x_2 \quad x_1^2x_2^2)^T.$$

Basis Functions for H_h

We will now designate the basis functions and $\{\psi_1, \dots, \psi_4\}$ for the pressure space H_h. We will again analyze the reference square $\hat{\tau} = [0,h]^2$ and find the coefficients of the basis functions. We will break up the $h \times h$ reference square into two triangles $\hat{\tau}_1$ and $\hat{\tau}_2$ as shown to the right.



Now, we will be using piecewise linear functions of the form

$$\psi_i(x_1, x_2) = a + bx_1 + cx_2.$$

We will create the basis functions such that for a given node N_k, $\psi_i = 1$ if $i = k$, and $\psi_i = 0$ when $i \neq k$ with respect to the triangles that contain the node.

First, on $\hat{\tau}_1$we consider the local basis function associated with the node $N_1 = (0,0)$ which is the function $\psi_1(x_1, x_2)$ that satisfies

$\psi_1(0,0) = a + b(0) + c(0) = 1$, $\psi_1(h,0) = a + b(h) + c(0) = 0$, $\psi_1(0,h) = a + b(0) + c(h) = 0$.

This system leads us to the function

$$\psi_1(x_1, y_1) = 1 - \frac{1}{h}x_1 - \frac{1}{h}x_2, \qquad \forall\, (x_1, x_2) \in \hat{\tau}_1$$

Which is 1 at node N_1 and is 0 on the other vertices of $\hat{\tau}_1$.

Likewise, on $\hat{\tau}_2$we consider the local basis function associated with the node $N_4 = (h, h)$ which is the function $\psi_4(x_1, x_2)$ that satisfies
$\psi_4(h,h) = a + b(h) + c(h) = 1\,, \qquad \psi_4(h,0) = a + b(h) + c(0) = 0\,, \qquad \psi_4(0,h) = a + b(0) + c(h) = 0\,.$
This system leads us to the function

$$\psi_4(x_1, y_1) = -1 + \frac{1}{h}x_1 + \frac{1}{h}x_2, \qquad \forall (x_1, x_2) \in \hat{\tau}_2$$

which is 1 at node N_4 and is 0 on the other vertices of $\hat{\tau}_2$.

For the next two basis functions, we need to consider both triangles that includes the associated node.
On $\hat{\tau}_1$ the local basis function associated with the node $N_2 = (h, 0)$ is the function $\psi_2(x_1, x_2)$ which satisfies
$\psi_2(h,0) = a + b(h) + c(0) = 1\,, \qquad \psi_2(0,0) = a + b(0) + c(0) = 0\,, \qquad \psi_2(0,h) = a + b(0) + c(h) = 0\,.$
This system leads us to the function

$$\psi_2(x_1, y_1) = \frac{1}{h}x_1, \qquad \forall (x_1, x_2) \in \hat{\tau}_1$$

which is 1 at node N_2 and is 0 on the other vertices of $\hat{\tau}_1$.

On $\hat{\tau}_2$ the local basis function associated with the node $N_2 = (h, 0)$ is the function $\psi_2(x_1, x_2)$ which satisfies
$\psi_2(h,0) = a + b(h) + c(0) = 1\,, \qquad \psi_2(h,h) = a + b(h) + c(h) = 0\,, \qquad \psi_2(0,h) = a + b(0) + c(h) = 0\,.$
This system leads us to the function

$$\psi_2(x_1, y_1) = 1 - \frac{1}{h}x_2, \qquad \forall (x_1, x_2) \in \hat{\tau}_2$$

which is 1 at node N_2 and is 0 on the other vertices of $\hat{\tau}_2$.

On $\hat{\tau}_1$ the local basis function associated with the node $N_3 = (0, h)$ is the function $\psi_3(x_1, x_2)$ which satisfies
$\psi_3(0,h) = a + b(0) + c(h) = 1\,, \qquad \psi_3(0,0) = a + b(0) + c(0) = 0, \qquad \psi_3(h,0) = a + b(h) + c(0) = 0.$
This system leads us to the function

$$\psi_3(x_1, y_1) = \frac{1}{h}x_2, \qquad \forall (x_1, x_2) \in \hat{\tau}_1$$

which is 1 at node N_3 and is 0 on the other vertices of $\hat{\tau}_1$.

On $\hat{\tau}_2$ the local basis function associated with the node $N_3 = (0, h)$ is the function $\psi_2(x_1, x_2)$ which satisfies
$\psi_3(0,h) = a + b(0) + c(h) = 1\,, \qquad \psi_3(h,h) = a + b(h) + c(h) = 0\,, \qquad \psi_3(h,0) = a + b(h) + c(0) = 0\,.$
This system leads us to the function

$$\psi_3(x_1, y_1) = 1 - \frac{1}{h}x_1, \qquad \forall (x_1, x_2) \in \hat{\tau}_2$$

which is 1 at node N_3 and is 0 on the other vertices of $\hat{\tau}_2$.

To summarize, our basis functions for the pressure space are the piecewise linear functions

$$\psi_i(x_1,x_2)=\begin{cases} 1-\frac{1}{h}x_1-\frac{1}{h}x_2, & if\ i=1\ and\ (x_1,x_2)\in\hat{\tau}_1 \\ \frac{1}{h}x_1, & if\ i=2\ and\ (x_1,x_2)\in\hat{\tau}_1 \\ 1-\frac{1}{h}x_2, & if\ i=2\ and\ (x_1,x_2)\in\hat{\tau}_2 \\ \frac{1}{h}x_2, & if\ i=3\ and\ (x_1,x_2)\in\hat{\tau}_1 \\ 1-\frac{1}{h}x_1, & if\ i=3\ and\ (x_1,x_2)\in\hat{\tau}_2 \\ -1+\frac{1}{h}x_1+\frac{1}{h}x_2, & if\ i=4\ and\ (x_1,x_2)\in\hat{\tau}_2 \\ 0, & otherwise \end{cases}$$

Matrix Form and Stiffness Matrices

We have now developed the bases $\{\phi_1,\dots,\phi_9\}$ and $\{\psi_1,\dots,\psi_4\}$ for V_h and H_h respectively. We can now write the discrete problem (2) in matrix form

$$A\xi - B^T\theta = \mathrm{F} \tag{3a}$$
$$-B\xi = 0. \tag{3b}$$

where $A=(a_{ij}), B=(b_{ij}), F=(F_j)$, with

$$a_{ij}=\langle\nabla\phi_i,\nabla\phi_j\rangle, \qquad b_{ij}=\langle\psi_i, div\ \phi_j\rangle, \qquad F_j=\int_\Omega \mathbf{f}\,\phi_j\,dx.$$

Here we have multiplied the second equation (2b) by -1 to allow us to use the block matrix below and we let ξ be the coordinates of u_h and θ be the coordinates of p_h. We can combine this system into a block matrix such that

$$\begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix}\begin{pmatrix} \xi \\ \theta \end{pmatrix}=\begin{pmatrix} F \\ 0 \end{pmatrix}$$

Now for the 2 dimensional problem, the local stiffness matrix A will be a block matrix consisting of two copies of $\bar{A}=(a_{ij})$ on the diagonal and zeros on the off diagonal blocks. For our function space V_h, this will produce an 18×18 matrix for A and a 4×18 matrix for B^T of the form

$$A=\begin{pmatrix} \bar{A} & 0 \\ 0 & \bar{A} \end{pmatrix}, \quad B^T=\begin{pmatrix} \bar{B}^T \\ \bar{B}^T \end{pmatrix}.$$

Velocity Local Stiffness Matrix

To compute the local stiffness matrix $\bar{A}=(a_{ij})$ we need to compute

$$a_{ij}=\langle\hat{\phi}_i,\hat{\phi}_j\rangle=\sum_{\hat{\tau}}\langle\hat{\phi}_i,\hat{\phi}_j\rangle_k=\int_{\hat{\tau}}\nabla\hat{\phi}_i\cdot\nabla\hat{\phi}_j\,dx.$$

We will approximate this integral using the 9 point Gaussian quadrature scheme where $w_0=w_2=5/18, w_1=4/9, \zeta_0=(1-\sqrt{3/5})/2, \zeta_1=1/2, \zeta_2=(1+\sqrt{3/5})/2$, and

$$\int_{\hat{\tau}} \nabla\hat{\phi}_i \cdot \nabla\hat{\phi}_j \, dx \approx \sum_{i,j=0}^{2} w_i w_j \left(\nabla\hat{\phi}_i \cdot \nabla\hat{\phi}_j\right) = \sum_{i,j=0}^{2} w_i w_j \left(\frac{\partial\hat{\phi}_i}{\partial x}\frac{\partial\hat{\phi}_j}{\partial x} + \frac{\partial\hat{\phi}_i}{\partial y}\frac{\partial\hat{\phi}_j}{\partial y}\right)$$

at points (ζ_i, ζ_j) for $i, j = 0,1,2$.

We compute this local stiffness matrix with the code *localstiffnessA.m* and *quadPHI.m* to get

$$S_{ij} = \begin{pmatrix} 0.6222 & -0.2000 & -0.0333 & -0.2000 & -0.3556 & 0.1111 & -0.0333 & 0.1111 & -0.0222 \\ -0.200 & 1.9556 & -0.2000 & -0.3556 & -1.0667 & -0.3556 & 0.1111 & 0 & 0.1111 \\ -0.0333 & -0.2000 & 0.6222 & 0.1111 & -0.3556 & -0.2000 & -0.0222 & 0.1111 & -0.0333 \\ -0.2000 & -0.3556 & 0.1111 & 1.9556 & -1.0667 & 0 & -0.2000 & -0.3556 & 0.1111 \\ -0.3556 & -1.0667 & -0.3556 & -1.0667 & 5.6889 & -1.0667 & -0.3556 & -1.0667 & -0.3556 \\ 0.1111 & -0.3556 & -0.2000 & 0 & -1.0667 & 1.9556 & 0.1111 & -0.3556 & -0.2000 \\ -0.0333 & 0.1111 & -0.0222 & -0.2000 & -0.3556 & 0.1111 & 0.6222 & -0.2000 & -0.0333 \\ 0.1111 & 0 & 0.1111 & -0.3556 & -1.0667 & -0.3556 & -0.2000 & 1.9556 & -0.2000 \\ -0.0222 & 0.1111 & -0.0333 & 0.1111 & -0.3556 & -0.2000 & -0.0333 & -0.2000 & 0.6222 \end{pmatrix}$$

Pressure Local Stiffness Matrix

We must now compute the pressure local stiffness matrix $\bar{B} = (b_{ij})$ where

$$b_{ij} = \langle \psi_i, \operatorname{div} \phi_j \rangle = \sum_{\hat{\tau}} \langle \psi_i, \operatorname{div} \phi_j \rangle_k = \int_{\hat{\tau}} \psi_i \cdot \operatorname{div} \phi_j \, dx.$$

We first compute the coefficient matrix for $\operatorname{div} \phi_j = \frac{\partial \phi_j}{\partial x_1} + \frac{\partial \phi_j}{\partial \phi x_2}$. Now, since $\phi_j \in Q_2$, we have

$$\begin{aligned} \phi_j &= c_{00} + c_{01}x_2 + c_{02}x_2^2 + c_{10}x_1 + c_{11}x_1x_2 + c_{12}x_1x_2^2 + c_{20}x_1^2 + c_{21}x_1^2x_2 + c_{22}x_1^2x_2^2 \\ \frac{\partial \phi_j}{\partial x_1} &= c_{10} + c_{11}x_2 + c_{12}x_2^2 + 2c_{20}x_1 + 2c_{21}x_1x_2 + 2c_{22}x_1x_2^2 \\ \frac{\partial \phi_j}{\partial x_2} &= c_{01} + 2c_{02}x_2 + c_{11}x_1 + 2c_{12}x_1x_2 + c_{21}x_1^2 + 2c_{22}x_1^2x_2. \end{aligned}$$

It is useful to note that we will consider the indices on the coefficients c_{kl} as base-3 (ternary) numbers for programming purposes. This form is useful since we can easily associate c_{kl} as the coefficient for the $x_1^k x_2^l$ term.

We can now write $\operatorname{div} \phi_j = \mathbf{c} \cdot \mathbf{x}$ for the coefficient vector $\mathbf{c}$ and variable vector $\mathbf{x}$ such that

$$\mathbf{c} \cdot \mathbf{x} = \begin{pmatrix} c_{10} + c_{01} \\ c_{11} + 2c_{02} \\ c_{12} \\ 2c_{20} + c_{11} \\ 2c_{21} + 2c_{12} \\ 2c_{22} \\ c_{21} \\ c_{22} \\ 0 \end{pmatrix}^T \begin{pmatrix} 1 \\ x_2 \\ x_2^2 \\ x_1 \\ x_1x_2 \\ x_1x_2^2 \\ x_1^2 \\ x_1^2x_2 \\ x_1^2x_2 \end{pmatrix}$$

Using the algorithm $localstiffnessB.m$ we get the matrix D which contains the coefficients of div ϕ_j in row j.

$$D = \begin{pmatrix} -6 & 13 & -6 & 13 & -24 & 8 & -6 & 4 & 0 \\ 4 & -20 & 12 & -12 & 40 & -16 & 8 & -8 & 0 \\ -1 & 7 & -6 & 3 & -16 & 8 & -2 & 4 & 0 \\ 4 & -12 & 8 & -20 & 40 & -16 & 12 & -8 & 0 \\ -0 & 16 & -16 & 16 & -64 & 32 & -16 & 16 & 0 \\ 0 & -4 & 8 & -4 & 24 & -16 & 4 & -8 & 0 \\ -1 & 3 & -2 & 7 & -16 & 8 & -6 & 4 & 0 \\ 0 & -4 & 4 & -4 & 24 & -16 & 8 & -8 & 0 \\ 0 & 1 & -2 & 1 & -8 & 8 & -2 & 4 & 0 \end{pmatrix}$$

We now use the 9 point Gaussian quadrature scheme from above to compute need to compute the integral

$$b_{ij} = \langle \psi_i, \text{div } \phi_j \rangle = \int_{\hat{\tau}} \psi_i \cdot \text{div } \phi_j \, dx \approx \sum_{k,l=0}^{2} w_k w_l (\psi_i \text{div } \phi_j).$$

The result is the local stiffness matrix from $localstiffnessB.m$ is

$$\bar{B} = \begin{pmatrix} -0.1487 & 0.0475 & -0.0068 & 0.0475 & 0.1041 & -0.0203 & -0.0068 & -0.0203 & 0.0038 \\ 0.0932 & -0.2697 & 0.2846 & -0.9364 & 1.6737 & -0.8686 & 0.2846 & -0.2019 & -0.0594 \\ 0.2043 & -1.1586 & 0.3957 & -0.4920 & 2.1182 & -0.4242 & 0.3957 & -1.0908 & 0.0517 \\ -1.1487 & 2.0475 & -1.3401 & 2.0475 & -6.5626 & 3.3130 & -1.3401 & 3.3130 & -0.3295 \end{pmatrix}.$$

Global Stiffness Matrices

We must now compute the global stiffness matrices for the global nodes. Since the velocity space and pressure space have a different number of degrees of freedom, we must consider different nodes for computing the different blocks of the global stiffness matrix. Setting the boundary nodes to zero due to boundary conditions, if the case of $M = 3$, we have 4 element squares that are $1/3 \times 1/3$ in size and we number the global nodes as follows:

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 21 & 22 & 23 & 24 & 25 & 0 \\ 0 & 16 & 17 & 18 & 19 & 20 & 0 \\ 0 & 11 & 12 & 13 & 14 & 15 & 0 \\ 0 & 6 & 7 & 8 & 9 & 10 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Velocity Nodes

$$\begin{matrix} 0 & - & 0 & - & 0 & - & 0 \\ | & & | & & | & & | \\ 0 & - & 3 & - & 4 & - & 0 \\ | & & | & & | & & | \\ 0 & - & 1 & - & 2 & - & 0 \\ | & & | & & | & & | \\ 0 & - & 0 & - & 0 & - & 0 \end{matrix}$$

Pressure Nodes

We will use the algorithm *localglobalStokes.m* to map the local reference element to the global nodes. We will record the mappings to two matrices $localglobalA$ and $localglobalB$ corresponding to the stiffness matrices A and B. These mappings will then be used to accumulate the contributions of the surrounding nodes to give us the coefficients of our global stiffness matrix.

We then use these mappings to compile a sparse matrix using $globalstiffnessStokes.m$ that is a $(2n + m) \times (2n + m)$ in size where n is the number of velocity nodes and m is the number of pressure nodes. The end result is a block matrix of the form

$$S = \begin{pmatrix} A & 0 & B^T \\ 0 & A & B^T \\ B & B & 0 \end{pmatrix}.$$

Here the two A blocks represent the coefficients of the two parts of the system for the functions $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$. The following is the global stiffness matrix for $M = 2$.

$$\begin{pmatrix}
5.6889 & -1.0667 & 0 & -1.0667 & -0.3556 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -6.5626 \\
-1.0667 & 3.9111 & -1.0667 & -0.3556 & -0.4000 & -0.3556 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3.7415 \\
0 & -1.0667 & 5.6889 & 0 & -0.3556 & -1.0667 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4.9392 \\
-1.0667 & -0.3556 & 0 & 3.9111 & -0.4000 & 0 & -1.0667 & -0.3556 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5.1615 \\
-0.3556 & -0.4000 & -0.3556 & -0.4000 & 2.4889 & -0.4000 & -0.3556 & -0.4000 & -0.3556 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.2453 \\
0 & -0.3556 & -1.0667 & 0 & -0.4000 & 3.9111 & 0 & -0.3556 & -1.0667 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.8413 \\
0 & 0 & 0 & -1.0667 & -0.3556 & 0 & 5.6889 & -1.0667 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.6304 \\
0 & 0 & 0 & -0.3556 & -0.4000 & -0.3556 & -1.0667 & 3.9111 & -1.0667 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8526 \\
0 & 0 & 0 & 0 & -0.3556 & -1.0667 & 0 & -1.0667 & 5.6889 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.7170 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5.6889 & -1.0667 & 0 & -1.0667 & -0.3556 & 0 & 0 & 0 & 0 & 0.1041 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.0667 & 3.9111 & -1.0667 & -0.3556 & -0.4000 & -0.3556 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.0667 & 5.6889 & 0 & -0.3556 & -1.0667 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.0667 & -0.3556 & 0 & 3.9111 & -0.4000 & 0 & -1.0667 & -0.3556 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.3556 & -0.4000 & -0.3556 & -0.4000 & 2.4889 & -0.4000 & -0.3556 & -0.4000 & -0.3556 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.3556 & -1.0667 & 0 & -0.4000 & 3.9111 & 0 & -0.3556 & -1.0667 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.0667 & -0.3556 & 0 & 5.6889 & -1.0667 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.3556 & -0.4000 & -0.3556 & -1.0667 & 3.9111 & -1.0667 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.3556 & -1.0667 & 0 & -1.0667 & 5.6889 & 0 \\
-6.5626 & -3.7415 & 4.9392 & 5.1615 & 3.2453 & -0.8413 & 0.6304 & 0.8526 & -0.7170 & 0.1041 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Sample Solution and Accuracy test

We will test the algorithm with an analytic solution to equation (1) on $\Omega = [0,1]^2$.

If we let

$$f_1 = \sin(\pi x)\,(\cos(\pi y) + 1)$$
$$f_2 = -\sin(\pi y)\,(\cos(\pi x) - 1)$$

we have solutions

$$u_1(x,y) = \frac{\sin(\pi x)\cos(\pi y)}{2\pi^2}$$
$$u_2(x,y) = -\frac{\cos(\pi x)\sin(\pi y)}{2\pi^2}$$
$$p(x,y) = \frac{\cos(\pi x) + \cos(\pi y)}{\pi}$$

We will first verify this solution by observing

$$-\Delta u_1 + \nabla \mathrm{p} = -\left(\frac{\partial^2 u_1}{\partial x^2} + \frac{\partial^2 u_1}{\partial y^2}\right) - \frac{\partial p}{\partial x} = \sin(\pi x)\cos(\pi y) + \sin(\pi x) = f_1$$

$$-\Delta u_2 + \nabla \mathrm{p} = -\left(\frac{\partial^2 u_2}{\partial x^2} + \frac{\partial^2 u_2}{\partial y^2}\right) - \frac{\partial p}{\partial y} = -\cos(\pi x)\sin(\pi y) + \sin(\pi y) = f_2.$$

We also have

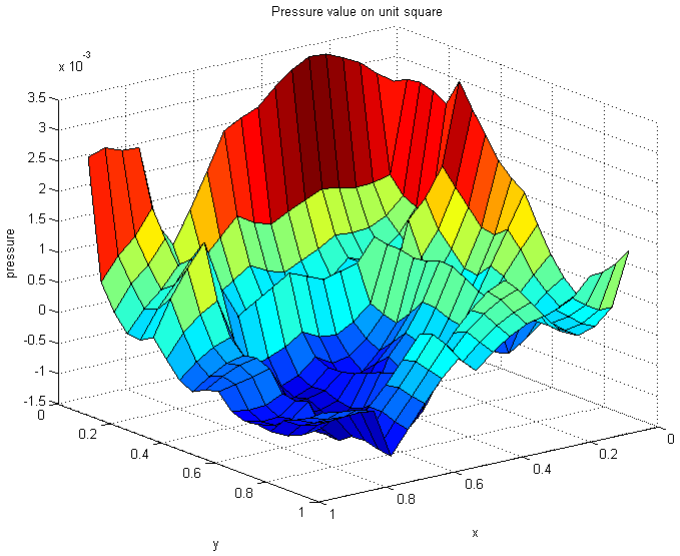
$$\operatorname{div} \mathbf{u} = \frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y} = \frac{\cos(\pi x)\cos(\pi y)}{2\pi} - \frac{\cos(\pi x)\cos(\pi y)}{2\pi} = 0$$

Furthermore, we clearly have $\mathbf{u} = \mathbf{0}$ on Γ.

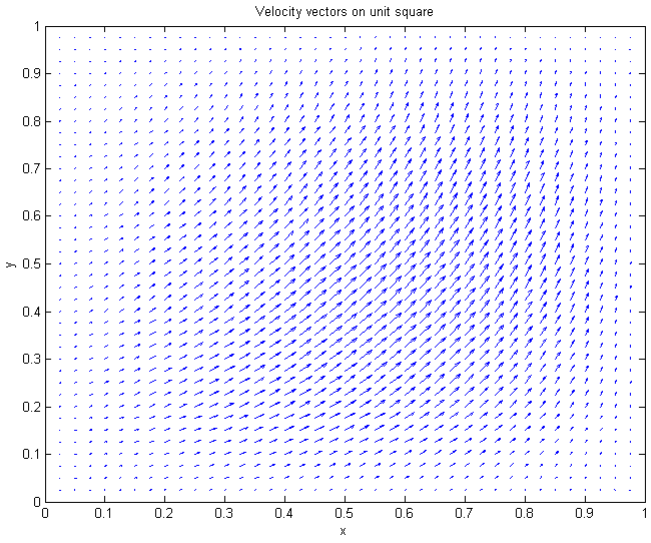
Computing the error for $M = 5, 10, 20,$ and 40 we see that the error is on the order of $h = 1/M$.

M	Velocity Error	Pressure Error	Total Weighted Error
5	0.0030	0.0685	0.0089
10	0.0013	0.0332	0.0045
20	6.2469e-004	0.0163	0.0023
40	2.9999e-004	0.0080	0.0011

When $M = 20$, we get the following pressure plot



and the following velocity direction field for (v_1, v_2)



Conclusion

This algorithm gives us first order convergence which is not as fast as we would like, but it does produce useful and stable results. Future versions of this algorithm will incorporate the ability to watch the plot change as the size of the elements decreases.

References

1. Johnson, Claes. Numerical Solution of Partial Differential Equations by the Finite Element Method.
2. Pasiak, Joe. Numerical Solutions of Partial Differential Equation. Lecture Notes for Math 610. 2011.
3. Knight, Jeremy. Computing the Local Stiffness Matrix on Triangulation T_h. http://knightmath.com/tamu/Math696-MathTech/public_html/FEM_project/pages/LocalStiffnessMatrix.pdf
4. Knight, Jeremy. Solving the Poisson Equations with the Finite Element Method. http://knightmath.com/tamu/Math696-MathTech/public_html/FEM_project/pages/poisson&FEM.pdf
5. Duran, Ricardo G. University of Buenos Aires, Argentina. http://mate.dm.uba.ar/~rduran/class_notes/mixed%20methods.pdf
6. Han,Houde and Yan,Ming. A Mixed Finite Element Method on a Staggered Mesh for Navier-Stokes Equations. http://www.math.ucla.edu/~yanm/documents/JCM2008.pdf

Matlab Code

The following contains all the Matlab code used in this project.

```
function [solv1,solv2,solp,xx,yy,xxb,yyb]=StokesSolver(f1,f2,M,varargin)
% This function will solve the Stokes equation for a given intitial
% function f=(f1,f2) with uniform partition M of the unit square [0,1]^2.
%If known solution functions are known for u1, u2, and p, enter these
%as functions for arguments 4,5,and 6 and the error will
% be calculated.


[S,xx,yy,xxb,yyb,nnode,nnodeb]=globalstiffnessStokes(M);
b=StokesConstants(f1,f2,M);
sol=S\b;
sv=0;  %sum for L2 norm for velocity
sp=0; %sum for L2 norm for pressure

solv1=sol(1:nnode);     %solution vector for v1

solv2=sol(nnode+1:2*nnode);       %solution vector for v2

solp=sol(2*nnode+1 : 2*nnode+nnodeb);        %solution vector for pressure

if size(varargin,2)==3
    u1=varargin{1};
    u2=varargin{2};
    p=varargin{3};
    for i=1:nnode
        sv=sv+(solv1(i)-u1(xx(i),yy(i)))^2;
    end
    for i=1:nnode
        sv=sv+(solv2(i)-u2(xx(i),yy(i)))^2;
    end
    for i=1:nnodeb
        sp=sp+(solp(i)-p(xxb(i),yyb(i)))^2;
    end
    ErrorV=(sv^(1/2)/(2*nnode));
    ErrorP=(sp^(1/2))/(nnodeb);
    Error=((sv^(1/2)+sp^(1/2))/(2*nnode+nnodeb));
    disp(ErrorV)
    disp(ErrorP)
    disp(Error)
end

%plot the solutions
    for i=1:M-1
    xmesh(i)=xxb(i);
    ymesh(i)=xxb(i);
    end
    for i=1:M-1
        for j=1:M-1
            ii=(i-1)*(M-1)+j;
            zz(i,j)=solp(ii);
        end
```

```
    end
    surf(xmesh,ymesh,zz);
    xlabel('x');ylabel('y');zlabel('pressure');
    title('Pressure value on unit square');
    pause;
    quiver(xx',yy',solv1,solv2);
    xlabel('x');ylabel('y');
    title('Velocity vectors on unit square');


    %-----------------------------------------------------

function [S,xx,yy,xxb,yyb,nnode,nnodeb]=globalstiffnessStokes(M)
%For Stokes Equations
%Computes the globalstiffness matrix for an MxM mesh on the unit square
%Returns S=Block matrix in the form
%[A 0 B^T]
%[0 A B^T]
%[B B  0]
%Returns xx,yy = coordinates of global nodes


%use localstiffnessA.m to compute PHI, and local stiffness matrix A.
[LSA,ev]=localstiffnessA();
%use localstiffnessB to compute the local stiffness matrix B.
[LSB,PSI]=localstiffnessB();
%compute the values of global nodes, and associate them to local nodes.
[localglobalA,localglobalB,xx,yy,xxb,yyb,nnode,nnodeb,nel,xphys,yphys]=localg
lobalStokes(M);
    %localglobalA: a nel x 9 array that maps the local nodes associated to an
        %element to velocity global nodes.
    %localglobalB: a nel x 4 array that maps the local nodes associated to an
        %element to pressure global nodes.
    %xx & yy: gives x and y values of global nodes for velocity.
    %xxb & yyb: gives x and y values of the global nodes for pressure.
    %nnode,nnodeb: number of velocity and pressure nodes
    %nel: number of elements
    %xphys & yphys: nel x 2 arrays contianing the x&y bounds of the physical
        %element.

%Compute the A block of size nnode x nnode
kkA=0;   %counter for number of nonzero elements in
for i=1:nel %element loop
    for j=1:9  %j,k are the indeces of the local stiffness array
        for k=1:9
            if ((localglobalA(i,j)~=0)&&(localglobalA(i,k)~=0))
                kkA=kkA+1;
                iiA(kkA)=localglobalA(i,j); %global node corresponding to j
(element i)
                jjA(kkA)=localglobalA(i,k); %global node corresponding to k
(element i)
                valA(kkA)=LSA(j,k);
            end
        end
    end
```

```
end
%Record the sparse representation of the A block of our stiffness matrix
SA=sparse(iiA,jjA,valA,nnode,nnode,kkA);
%Compute the B block of size nnodeb x nnode
kkB=0;   %counter for number of nonzero elements in
for i=1:nel %element loop
    for j=1:4  %j,k are the indeces of the local stiffness array
        for k=1:9
            if ((localglobalB(i,j)~=0)&&(localglobalA(i,k)~=0))
                kkB=kkB+1;
                iiB(kkB)=localglobalB(i,j); %global node corresponding to j
(element i)
                jjB(kkB)=localglobalA(i,k); %global node corresponding to k
(element i)
                valB(kkB)=LSB(j,k);
            end
        end
    end
end
%Record the sparse representation of the A block of our stiffness matrix



SB=sparse(iiB,jjB,valB,nnodeb,nnode,kkB);
%Store transpose of B
SBT=sparse(jjB,iiB,valB,nnode,nnodeb,kkB);

%Now we will build the Global sparce matrix which will be a square
%matrix with (2*nnode+nnodeb) rows and columns

ii=[iiA,iiA+nnode,iiB+2*nnode,iiB+2*nnode,jjB,jjB+nnodeb];
jj=[jjA,jjA+nnode,jjB,jjB+nnodeb,iiB+2*nnode,iiB+2*nnode];
val=[valA,valA,valB,valB,valB,valB];
n=(2*nnode+nnodeb);
kk=2*kkA+4*kkB;
S=sparse(ii,jj,val,n,n,kk);

%-----------------------------------------------------------------

function [localglobalA,localglobalB,x,y,xb,yb,nnode,nnodeb,nel,xphys,yphys] =
localglobalStokes(M)
% M is the number of elements in each direction.
% brute force structure for quadratic velocity and linear pressure,
%localglobalA is a nelx9 array mapping the global velocity nodes for each
element
%localglobalB is a nelx4 array mapping the global pressure nodes points for
each
% homogeneous Dirichlet Stokes problem
% first the nodes
h2=.5/M;    %Distance between vel nodes
h=1./M;      %Distance between pres nodes
tm=2*M-1;   %Number of nodes in one row not on boundary for Velocity
tmb=M-1;    %Number of nodes in one row not on boundary for Pressure
nel=M*M;    %Total Number of Square Elements
nnode=tm*tm; %Total Number of vel nodes
```

```
nnodeb=tmb*tmb; %Total number of pres nodes
for j=1:tm
    for i=1:tm
        k=i+(j-1)*tm;
        x(k)=i*h2; %X-coordinate of vel node k
        y(k)=j*h2; %Y-coordinate of vel node k
    end
end
for j=1:tmb
    for i=1:tmb
        k=i+(j-1)*tmb;
        xb(k)=i*h; %X-coordinate of vel node k
        yb(k)=j*h; %Y-coordinate of vel node k
    end
end
k=1;    %Element counter

for j=1:M   %Vertical Element row number (count from bottom left)
jj=2*j-1;   %Vert distance of top vel node in the element k
kk=j-1;      %Vert distance of top pres node in the element k
    for i=1:M
        ii=2*i; %Horiz ditance of right hand vel node in element k
        ll=i;   %Horiz distance of right hand vel node in element k
        localglobalA(k,1)=ii-2+(jj-2)*tm;
        localglobalA(k,2)=ii-1+(jj-2)*tm;
        localglobalA(k,3)=ii+(jj-2)*tm;
        localglobalA(k,4)=ii-2+(jj-1)*tm;
        localglobalA(k,5)=ii-1+(jj-1)*tm;
        localglobalA(k,6)=ii+(jj-1)*tm;
        localglobalA(k,7)=ii-2+jj*tm;
        localglobalA(k,8)=ii-1+jj*tm;
        localglobalA(k,9)=ii+jj*tm;
        localglobalB(k,1)=ll-1+(kk-1)*tmb;
        localglobalB(k,2)=ll+(kk-1)*tmb;
        localglobalB(k,3)=ll-1+kk*tmb;
        localglobalB(k,4)=ll+kk*tmb;
        if (i==1)   %implies left side
            localglobalA(k,1)=0;
            localglobalA(k,4)=0;
            localglobalA(k,7)=0;
            localglobalB(k,1)=0;
            localglobalB(k,3)=0;
        end
        if(i==M)    %implies right side
            localglobalA(k,3)=0;
            localglobalA(k,6)=0;
            localglobalA(k,9)=0;
            localglobalB(k,4)=0;
            localglobalB(k,2)=0;
        end
        if(j==1)    %implies bottom
            localglobalA(k,1)=0;
            localglobalA(k,2)=0;
            localglobalA(k,3)=0;
            localglobalB(k,1)=0;
            localglobalB(k,2)=0;
```

```
        end
        if (j==M)   %implies top
            localglobalA(k,7)=0;
            localglobalA(k,8)=0;
            localglobalA(k,9)=0;
            localglobalB(k,3)=0;
            localglobalB(k,4)=0;
        end
        k=k+1;
    end
end
%Compute and record physical boundaries of element k
k=0;
for j=1:M
    for i=1:M k=k+1;
        xphys(k,1)=(i-1)*h;
        xphys(k,2)=i*h;
        yphys(k,1)=(j-1)*h;
        yphys(k,2)=j*h;
    end
end
return

%-----------------------------------------------------------------

function [b]=StokesConstants(f1,f2,M)
%computes the constants for the right hand side of the Finite Element
%Method solution to Stokes equations using quadrature approximation.
%f1,f2=functions given in the PDE for velocity functions v1 and v2
%M=number of subintervals

[LSA,ev]=localstiffnessA(); %compute phi matrix ev
[localglobalA,localglobalB,x,y,xb,yb,nnode,nnodeb,nel,xphys,yphys] =
localglobalStokes(M);
b=zeros(2*nnode+nnodeb,1);    %initialize  b-vector
h=1/M;

%Calculate coefficients for Guassian Quadrature
w(1)=5/18;
w(2)=4/9;
w(3)=5/18;
%
z(1)=(1-sqrt(3/5))/2;
z(2)=1/2;
z(3)=(1+sqrt(3/5))/2;

%Create quadrature vectors for the X, Y, and ww=w_i*w_j
for i=0:2
    for j=0:2
        ii=3*i+j+1;
        X(ii)=z(j+1);
        Y(ii)=z(i+1);
        ww(ii)=w(i+1)*w(j+1);
    end
end
```

```
%Map the quadrature node (x_a,y_a) on the reference element to
%(x_a(k),y_a(k)); this maps the quadrature nodes on Omega proportionally
%into the physical elements
for i=1:9
    for k= 1:nel
        x(i,k)=xphys(k,1)*(1-X(i))+xphys(k,2)*X(i);
        y(i,k)=yphys(k,1)*(1-Y(i))+yphys(k,2)*Y(i);
    end
end


for k=1:nel
    for j=1:9
        s1=0;
        s2=0;
        for i=1:9
            s1=s1+ww(i)*ev(i,j)*f1(x(i,k),y(i,k));
            s2=s2+ww(i)*ev(i,j)*f2(x(i,k),y(i,k));
        end
        n=localglobalA(k,j);
        if (n~=0)
            b(n)=b(n)+s1*h*h;
            b(n+nnode)=b(n+nnode)+s2*h*h;
        end
    end
end

%-------------------------------------------------------------

function [LSA,ev]=localstiffnessA()
%%Use Gaussian quadrature scheme to compute the
% local stiffness matrix(LSA)
%for quadratics with homogonous Dirichlet problem.
w(1)=5/18;
w(2)=4/9;
w(3)=5/18;
%
z(1)=(1-sqrt(3/5))/2;
z(2)=1/2;
z(3)=(1+sqrt(3/5))/2;

%Create vectors for the X, Y, and W=w_i*w_j
for i=0:2
    for j=0:2
        ii=3*i+j+1; %Convert the ternary number ij to decimal integer
        X(ii)=z(j+1);
        Y(ii)=z(i+1);
        W(ii)=w(i+1)*w(j+1);
    end
end

[ev,evx,evy]=QuadPHI(X,Y,2);

for j=1:9
```

```
    for k=1:9
        s1=0;
        for iq=1:9
            s1=s1+W(iq)*(evx(iq,j)*evx(iq,k)+ evy(iq,j)*evy(iq,k));
        end
        LSA(j,k)=s1;
    end
end

%-----------------------------------------------------------------

function [LSB,PSI]=localstiffnessB()
%%Use Gaussian quadrature scheme to compute the the local stiffness
matrix(LS)
%the Stokes problem with b=<q,div v..
w(1)=5/18;
w(2)=4/9;
w(3)=5/18;
%
z(1)=(1-sqrt(3/5))/2;
z(2)=1/2;
z(3)=(1+sqrt(3/5))/2;

%Create vectors for the X, Y, and W=w_i*w_j for quadrature
for i=0:2
    for j=0:2
        ii=3*i+j+1; %Convert the ternary number ij to decimal integer
        X(ii)=z(j+1);
        Y(ii)=z(i+1);
        W(ii)=w(i+1)*w(j+1);
    end
end

%Compute the coefficients of div(phi_j) at node i .
D=divPHI();
%Compute divergence for each point in Gaussian Quadrature
evB=zeros(9,9);
for i=1:9
    for j=1:9
         s1=0;
         for l=0:2
            for k=0:2
                jj=3*k+l+1;
                s1=s1+D(j,jj)*X(i)^k*Y(i)^l;
            end
         end
         %evB(i,j) is the value of div(phi_j) at point i
        evB(i,j)=s1;
    end
end

%
[PSI]=LinPSI(X,Y,2);
LSB=zeros(4,9);
for i=1:4
    for j=1:9
```

```
        s1=0;
        for iq=1:9
            s1=s1+W(iq)*PSI(iq,i)*evB(iq,j);
        end
        LSB(i,j)=s1;
    end
end

%-----------------------------------------------------------------

function [ev,evx,evy]=QuadPHI(X,Y,M)
%To determine matrices phi_j(x_i,x_i)
%d(phi_j)/dx  and
%d(phi_j)/dy

% Create the coefficient matrix for Phi(x,y)
% Elements a_(ij,kl) with ij and kl being base-3 numbers
% ii and jj are the decimal equivalent
for i=0:M
    for j=0:M
        for k=0:M
            for l=0:M
                ii=3*i+j+1;
                jj=3*k+l+1;
                A(ii,jj)=(i/M)^k*(j/M)^l;
            end
        end
    end
end
A1=inv(A);
phi=A1';

%Loop to create matrices ev,evx,evy, again using base-3 notation for (ij,kl)
for i=1:(M+1)^2
    for j=1:(M+1)^2
         s1=0;
         s2=0;
         s3=0;
         for k=0:M
            for l=0:M
                jj=3*k+l+1;
                s1=s1+phi(j,jj)*X(i)^l*Y(i)^k;
                if l~=0
                    s2=s2+phi(j,jj)*l*X(i)^(l-1)*Y(i)^k;
                end
                if k~=0
                    s3=s3+phi(j,jj)*X(i)^l*k*Y(i)^(k-1);
                end
            end
         end
        ev(i,j)=s1;          %ev(i,j) is the value of phi_j at node i
        evx(i,j)=s2;         %evx(i,j) is the value of (d/dx)phi_j at node i
        evy(i,j)=s3;         %evy(i,j) is the value of (d/dy)phi_j at node i
    end
end
return
```

```
%-------------------------------------------------------------

function [DIV]=divPHI()
% To compute the coefficient matrix of div(phi) for a
% biquadratic function phi(x_1,x_2)
% phi=9x9 coefficient matrix for phi basis functions.
%Base 3 indices will be translated to decimal equivalent+1
%00=1,01=2,02=3,10=4,11=5,12=6,20=7,21=8,22=9

% Create the coefficient matrix for Phi(x,y)
M=2;
for i=0:M
    for j=0:M
        for k=0:M
            for l=0:M
                ii=3*i+j+1;
                jj=3*k+l+1;
                A(ii,jj)=(i/M)^k*(j/M)^l;
            end
        end
    end
end
A1=inv(A);
phi=A1';

DIV=zeros(9,9);
for i=1:9
    DIV(i,1)=phi(i,4)+phi(i,2);
    DIV(i,2)=phi(i,5)+2*phi(i,3);
    DIV(i,3)=phi(i,6);
    DIV(i,4)=2*phi(i,7)+phi(i,5);
    DIV(i,5)=2*phi(i,8)+2*phi(i,6);
    DIV(i,6)=2*phi(i,9);
    DIV(i,7)=phi(i,8);
    DIV(i,8)=phi(i,9);
    DIV(i,9)=0;
end

%-------------------------------------------------------------

function [PSI]=LinPSI(X,Y,M)
%To compute the value of psi_j at each of the nodes
%(X,Y) = coordinates of nodes; M=number of subdivisions
%PSI gives the value of psi_j at point i
nnodes=length(X);
PSI=zeros(nnodes,4);
for j=1:4
    for i=1:nnodes
        PSI(i,j)=EvalPsi(j,X(i),Y(i),M);
    end
end
return

%-------------------------------------------------------------
```

```
function [y]=EvalPsi(i,x1,x2,M)
%Given coordinates x1,x2, the psi function number i,
%and the number of subdivisions of the element
%this function will compute the value of y=psi(x1,x2)
%on the for piecewise linear function on [0,h]^2
%which is subdivided into two triangles
%T1={(0,0),(0,h),(h,0)} ; T2={(h,0),(0,h),(h,h)}

h=1./M;
%determine the triangle that contains (x1,x2)
if x2<=h-x1
    T=1;
else
    T=2;
end

if (i==1)&&(T==1)
    y=1-(1/h)*x1-(1/h)*x2;
elseif i==2
    if T==1
        y=(1/h)*x1;
    else
        y=1-(1/h)*x2;
    end
elseif i==3
    if T==1
        y=(1/h)*x2;
    else
        y=1-(1/h)*x1;
    end
elseif i==4&&T==2
    y=-1+(1/h)*x1+(1/h)*x2;
else
    y=0;
end
```