Available at
http://pvamu.edu/aam
Appl. Appl. Math.
ISSN: 1932-9466

Applications and Applied Mathematics:
An International Journal
(AAM)

Vol. 13, Issue 1 (June 2018), pp. 535 - 565

Finite Element Solution of the Two-dimensional Incompressible Navier-Stokes Equations Using MATLAB

[1]*Endalew Getnet Tsega and [2]V.K. Katiyar

Department of Mathematics
Indian Institute of Technology Roorkee
Uttarakhand, India
[1]endalebdumath2016@gmail.com, [2]vktmafma20@gmail.com

*Corresponding Author

Received: July 5, 2017; Accepted: May 28, 2018

Abstract

The Navier–Stokes equations are fundamental in fluid mechanics. The finite element method has become a popular method for the solution of the Navier-Stokes equations. In this paper, the Galerkin finite element method was used to solve the Navier-Stokes equations for two-dimensional steady flow of Newtonian and incompressible fluid with no body forces using MATLAB. The method was applied to the lid-driven cavity problem. The eight-noded rectangular element was used for the formulation of element equations. The velocity components were located at all of 8 nodes and the pressure variable is located at 4 corner of the element. From location of velocity components and pressure, it is obvious that this element consists of 16 unknowns for velocities and 4 unknowns for pressure. As a result, the unknown variables for velocities and pressure are 20 per each element. The quadratic interpolation functions represent velocity components while bilinear interpolation function represents pressure. Finite element codes were developed for implementation. The numerical results were compared with benchmark results from the literature.

Keywords: Navier–Stokes equations; finite element method; steady-state solution; eight node rectangular element; MATLAB

MSC 2010 No.: 35Q30, 35Q35, 76D05, 76M10

1. Introduction

The Navier-Stokes equation is a set of nonlinear partial differential equations that describe the flow of fluids, which represent conservation of linear momentum. It is the cornerstone of fluid mechanics as noted by Cengel et al. (2010). It is solved jointly with continuity equation. These equations cannot be solved exactly. So, approximations and simplifying assumptions are commonly made to allow the equations to be solved approximately. Recently, high speed computers have been used to solve such equations by replacing with a set of algebraic equations using a variety of numerical techniques like finite difference, finite volume, and finite element methods.

Finite element method is the most powerful numerical technique for computational fluid dynamics which is readily applicable to domains of complex geometrical shape and provides a great freedom in the choice of numerical approximations. It reduces a partial differential equation system to a system of algebraic equations that can be solved using traditional linear algebra techniques. In finite element method, the domain of interest is subdivided into small subdomains called finite elements. Over each finite element, the unknown variable is approximated by a linear combination of approximation functions called shape functions which are associated with the node of the element characterize the element. The piecewise approximations for elements are assembled together to obtain a global system to the whole domain. One of the major advantages of the finite element method is that a general purpose computer program can be developed easily to analyze various kinds of problems as noted by Kwon et al. (1997). In particular, any complex shape of problem domain with prescribed boundary conditions can be handled with ease using the finite element method.

Jiajan (2010) discussed the Galerkin finite element formulation of two dimensional unsteady incompressible Navier-Stokes equations using the quadratic triangular element (6-nodes). The pressure variable was located at the corner nodes and the velocity components were located at all of the six nodes. Ghia et al. (1982) studied high Reynolds number solutions for incompressible flow using the Navier-Stokes equation and the multigrid method. Persson (2002) implemented a finite element based solver of the incompressible Navier-Stokes equations on unstructured two dimensional triangular meshes. He solved the lid-driven cavity flow problem for four different Reynold's numbers: 100, 500, 1000 and 2000.

Glaisner et al. (1987) discussed finite-element procedures for the Navier-Stokes equations in the primitive variable formulation and the vorticity stream-function formulations. Steady-state solution of lid-driven cavity flow was obtained by the velocity-pressure formulation using the nine-node rectangular element in their work. Taylor et al. (1981) and Smith et al. (2014) used eight-noded rectangular element mesh to solve two-dimensional incompressible Navier-Stokes Equations with FORTRAN programming language. Rhaman et al. (2014) presented Galerkin finite element method to simulate the motion of fluid particles which satisfies the unsteady Navier-Stokes equations through a programming code developed in FreeFem++.

Simpson (2017) used nine noded rectangular elements with two degree of freedom on each node for finite element simulation of a coupled reaction-diffusion problem using MATLAB. Khennane (2013) developed MATLAB codes for 4-nodded and 8-noded quadrilateral elements for the linear elastic static analysis of a two dimensional problem using finite element method.

2. Governing Equations

For steady Newtonian incompressible fluid with no body forces, the governing equations for two-dimensional flow are:
Continuity equation:

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0. \tag{1}$$

Navier-Stokes equation:

x-component

$$\rho\left(u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}\right)=-\frac{\partial p}{\partial x}+\mu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right), \tag{2}$$

y-component

$$\rho\left(u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}\right)=-\frac{\partial p}{\partial y}+\mu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right). \tag{3}$$

where u and v are the x, y components of the velocity vector, p is static pressure, ρ is density and μ dynamic viscosity of the flowing fluid.

Using L and V as a characteristics (reference) length and velocity respectively, we define the dimensionless variables

$$x^*=\frac{x}{L},\ y^*=\frac{y}{L},\ u^*=\frac{u}{V},\ v^*=\frac{v}{V},\ \text{and}\ p^*=\frac{p}{\rho V^2}.$$

The governing equations, Equation(1), Equation (2),and Equation (3) can be written in their dimensionless form (ignoring the astrix '*') as:

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0, \tag{4}$$

$$u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}=-\frac{\partial p}{\partial x}+\frac{1}{\mathrm{Re}}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right), \tag{5}$$

$$u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}=-\frac{\partial p}{\partial y}+\frac{1}{\mathrm{Re}}\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right), \tag{6}$$

where

$$\mathrm{Re}=\frac{\rho VL}{\mu}$$

is the Reynolds number.

3. Formulation of Element Equations

To describe the structure of finite element programming of a steady-state solution of the Navier–Stokes equations, let us consider a flow confined to a rectangular cavity driven by a uniform horizontal velocity at the top. The velocities at the other three boundaries are set to zero. Eight-noded rectangular elements are used to model the flow. We use all the 8 nodes of each element to model velocities components *u* and *v* and the 4 corner nodes to model the pressure *p*. Meshes are numbered in x-direction. Freedoms numbered are in the order u−p−v as used by Smith et al. (2014) and Taylor et al. (1981).

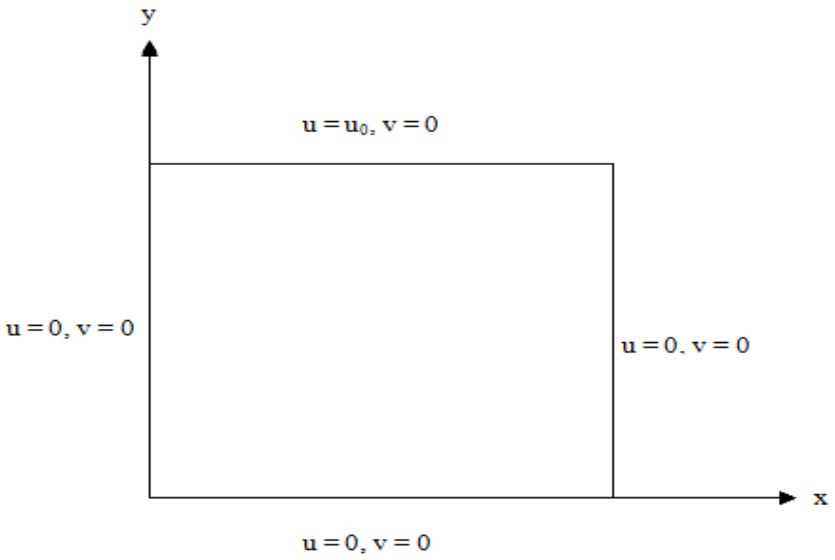


Figure 1. Lid-driven cavity

Let us denote an element by Ω. Shape functions for the rectangular elements are expressed in terms of local coordinates ξ and η where

$$\xi = 2(x-x_c)/\ell_x, \quad \eta = 2(y-y_c)/\ell_y$$

Here, (x_c, y_c) is the centroid of the element, and ℓ_x, ℓ_y represent its length in x and y-direction.

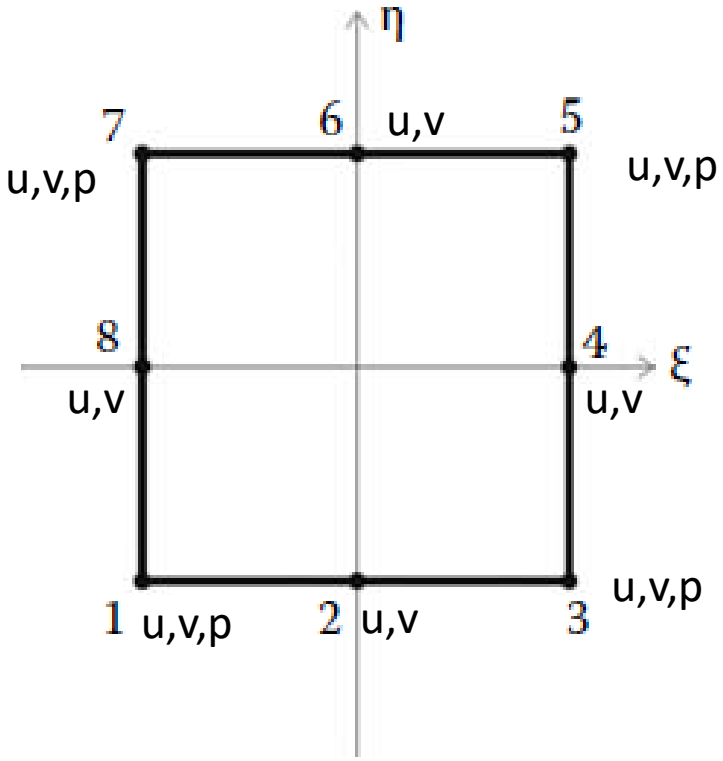


Figure 2. Eight-noded rectangular element

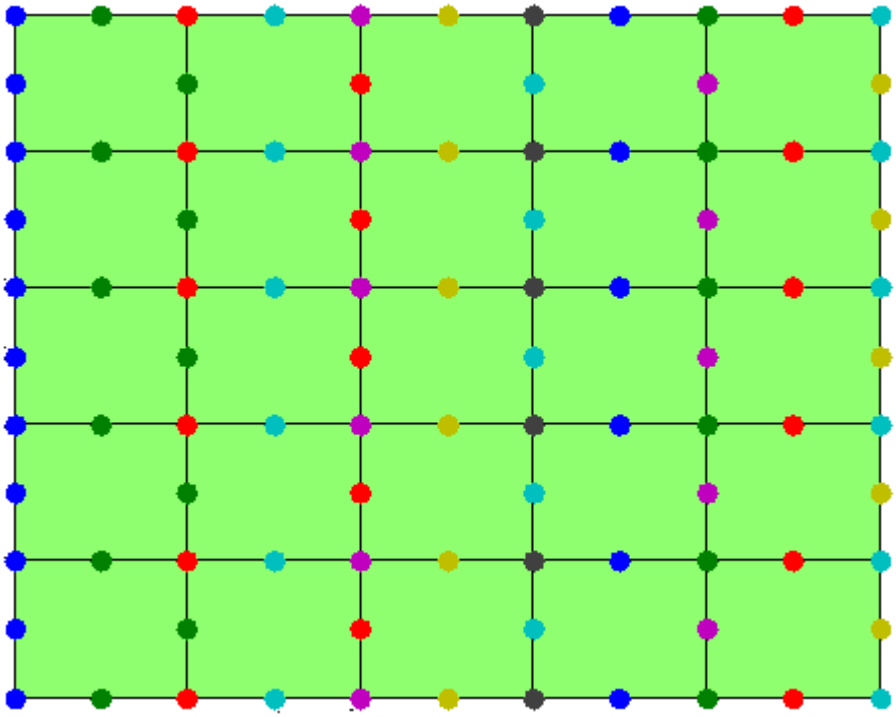


Figure 3. Eight-noded rectangular elements mesh

Suppose the nodes 1, 2, 3, 4, 5, 6, 7, 8 have coordinates (-1, -1), (0, -1), (1, -1), (1, 0), (1, 1), (0, 1), (-1, 1), (-1, 0) in the local coordinate system. Then, the general form of shape functions for 4-noded bilinear rectangular element (considering corner nodes) using local coordinates is

$$M = a + b\xi + c\eta + d\xi\eta. \tag{7}$$

The general form of shape functions for 8-noded quadratic rectangular element (considering all nodes) using local coordinates is

$$N = a + b\xi + c\eta + d\xi^2 + e\xi\eta + f\eta^2 + g\xi^2\eta + h\xi\eta^2. \tag{8}$$

Using Kronecker-delta property of shape functions, from Equation (7) and Equation (8) shape functions for 4-noded and 8-noded rectangular elements are

$$
\begin{aligned}
M_1 &= \frac{1}{4}(1-\xi-\eta+\xi\eta),\\
M_2 &= \frac{1}{4}(1+\xi-\eta-\xi\eta),\\
M_3 &= \frac{1}{4}(1+\xi+\eta+\xi\eta),\\
M_4 &= \frac{1}{4}(1-\xi+\eta-\xi\eta).
\end{aligned}
\tag{9}
$$

$$
\begin{aligned}
N_1 &= -\tfrac{1}{4}(1-\xi)(1-\eta)(1+\xi+\eta),\\
N_2 &= \tfrac{1}{2}(1-\xi^2)(1-\eta),\\
N_3 &= -\tfrac{1}{4}(1+\xi)(1-\eta)(1-\xi+\eta),\\
N_4 &= \tfrac{1}{2}(1+\xi)(1-\eta^2),\\
N_5 &= -\tfrac{1}{4}(1+\xi)(1+\eta)(1-\xi-\eta),\\
N_6 &= \tfrac{1}{2}(1-\xi^2)(1+\eta),\\
N_7 &= -\tfrac{1}{4}(1-\xi)(1+\eta)(1+\xi-\eta),\\
N_8 &= \tfrac{1}{2}(1-\xi)(1-\eta^2).
\end{aligned}
\tag{10}
$$

Thus, the quadratic interpolation functions are used for velocity components while bilinear interpolation functions for pressure. As a result, the unknown variables for velocities and pressure are 20 per each element. Thus, the dependent variable u, v, and p are expressed as

$$
\begin{aligned}
u &= \sum_{i=1}^{8} N_i u_i\,,\\
v &= \sum_{i=1}^{8} N_i v_i\,,\\
p &= \sum_{i=1}^{4} M_i p_i\,.
\end{aligned}
\tag{11}
$$

where u_i, v_i and p_i are velocity and pressure values at the nodes.

Now, expressing Equation (4), Equation (5), and Equation (6) using these shape functions we get

$$
\sum_{j=1}^{8} \frac{\partial N_j}{\partial x} u_j + \sum_{j=1}^{8} \frac{\partial N_j}{\partial y} v_j = 0\,,
\tag{12}
$$

$$
\begin{aligned}
&\sum_{k=1}^{8} N_k u_k \sum_{j=1}^{8} \frac{\partial N_j}{\partial x} u_j + \sum_{k=1}^{8} N_k v_k \sum_{j=1}^{8} \frac{\partial N_j}{\partial y} u_j = -\sum_{l=1}^{4} \frac{\partial M_l}{\partial x} p_l\\
&\quad + \frac{1}{\mathrm{Re}}\left(\sum_{j=1}^{8} \frac{\partial^2 N_j}{\partial x^2} u_j + \sum_{j=1}^{8} \frac{\partial^2 N_j}{\partial y^2} u_j \right),
\end{aligned}
\tag{13}
$$

$$\sum_{k=1}^{8} N_k u_k \sum_{j=1}^{8} \frac{\partial N_j}{\partial x} v_j + \sum_{k=1}^{8} N_k v_k \sum_{j=1}^{8} \frac{\partial N_j}{\partial y} v_j = -\sum_{l=1}^{4} \frac{\partial M_l}{\partial y} p_l + \frac{1}{\mathrm{Re}} \left(\sum_{j=1}^{8} \frac{\partial^2 N_j}{\partial y^2} v_j + \sum_{j=1}^{8} \frac{\partial^2 N_j}{\partial y^2} v_j \right). \tag{14}$$

Employing Galerkin weighted residual approach on Equation (13), we get

$$\iint_{\Omega} N_i \left[\sum_{k=1}^{8} N_k u_k \sum_{j=1}^{8} \frac{\partial N_j}{\partial x} u_j + \sum_{k=1}^{8} N_k v_k \sum_{j=1}^{8} \frac{\partial N_j}{\partial y} u_j + \sum_{l=1}^{4} \frac{\partial M_l}{\partial x} p_l - \frac{1}{\mathrm{Re}} \left(\sum_{j=1}^{8} \frac{\partial^2 N_j}{\partial x^2} u_j + \sum_{j=1}^{8} \frac{\partial^2 N_j}{\partial y^2} u_j \right) \right] dA = 0 \tag{15}$$

Using Gauss-Divergence Theorem, we have

$$\iint_{\Omega} \frac{1}{\mathrm{Re}} \left(N_i \sum_{j=1}^{8} \frac{\partial^2 N_j}{\partial x^2} u_j + N_i \sum_{j=1}^{8} \frac{\partial^2 N_j}{\partial y^2} u_j \right) dA = -\frac{1}{\mathrm{Re}} \left(\iint_{\Omega} \frac{\partial N_i}{\partial x} \sum_{j=1}^{8} \frac{\partial N_j}{\partial x} u_j dA + \iint_{\Omega} \frac{\partial N_i}{\partial y} \sum_{j=1}^{8} \frac{\partial N_j}{\partial y} u_j dA \right) + \frac{1}{\mathrm{Re}} \oint_{\Gamma} N_i \sum_{j=1}^{8} \frac{\partial N_j}{\partial n} u_j dS$$

where Γ is the boundary of the element Ω, $n = (n_x, n_y)$ is the unit outward normal vector to the element and $\frac{\partial N_j}{\partial n} = \frac{\partial N_j}{\partial x} n_x + \frac{\partial N_j}{\partial y} n_y$ is the directional derivative of N_j in the direction normal to the boundary Γ. Hence, we have

$$\iint_{\Omega} \left[N_i \sum_{k=1}^{8} N_k u_k \sum_{j=1}^{8} \frac{\partial N_j}{\partial x} u_j + N_i \sum_{k=1}^{8} N_k v_k \sum_{j=1}^{8} \frac{\partial N_j}{\partial x} u_j + N_i \sum_{l=1}^{4} \frac{\partial M_l}{\partial x} p_l + \frac{1}{\mathrm{Re}} \left(\frac{\partial N_i}{\partial x} \sum_{j=1}^{8} \frac{\partial N_j}{\partial x} u_j + \frac{\partial N_i}{\partial y} \sum_{j=1}^{8} \frac{\partial N_j}{\partial y} u_j dA \right) \right] dA - \frac{1}{\mathrm{Re}} \oint_{\Gamma} N_i \sum_{j=1}^{8} \frac{\partial N_j}{\partial n} u_j dS = 0.$$

i, *j*=1,2, ..., 8.

For Dirichilet boundary condition, we have

$$\iint_{\Omega} \left[N_i \sum_{k=1}^{8} N_k u_k \sum_{j=1}^{8} \frac{\partial N_j}{\partial x} u_j + N_i \sum_{k=1}^{8} N_k v_k \sum_{j=1}^{8} \frac{\partial N_j}{\partial x} u_j + N_i \sum_{l=1}^{4} \frac{\partial M_l}{\partial x} p_l + \frac{1}{\mathrm{Re}} \left(\frac{\partial N_i}{\partial x} \sum_{j=1}^{8} \frac{\partial N_j}{\partial x} u_j + \frac{\partial N_i}{\partial y} \sum_{j=1}^{8} \frac{\partial N_j}{\partial y} u_j dA \right) \right] dA = 0. \tag{16}$$

i, *j*=1,2, ..., 8.

Applying similar procedure for Equation (14), we get

$$\iint_{\Omega}\left[N_i \sum_{k=1}^{8} N_k u_k \sum_{j=1}^{8} \frac{\partial N_j}{\partial x} v_j + N_i \sum_{k=1}^{8} N_k v_k \sum_{j=1}^{8} \frac{\partial N_j}{\partial x} v_j + N_i \sum_{l=1}^{4} \frac{\partial M_l}{\partial y} p_l + \frac{1}{\mathrm{Re}}\left(\frac{\partial N_i}{\partial x} \sum_{j=1}^{8} \frac{\partial N_j}{\partial x} v_j + \frac{\partial N_i}{\partial y} \sum_{j=1}^{8} \frac{\partial N_j}{\partial y} v_j dA \right)\right] dA = 0. \tag{17}$$

i, j=1,2, ..., 8.

Employing Galerkin weighted residual approach on Equation (12) using the weight functions M_ℓ , we get

$$\iint_{\Omega} M_l \left[\sum_{j=1}^{8} \frac{\partial N_j}{\partial x} u_j + \sum_{j=1}^{8} \frac{\partial N_j}{\partial y} v_j \right] dA = 0,$$

or

$$\iint_{\Omega} \left[M_l \sum_{j=1}^{8} \frac{\partial N_j}{\partial x} u_j + M_l \sum_{j=1}^{8} \frac{\partial N_j}{\partial y} v_j \right] dA = 0. \tag{18}$$

l = 1, 2, 3, 4 .

Due to the nonlinearity, the set of algebraic equations that will be obtained here cannot be solved in a single shot, but an iterative solution is necessary. In such an iterative solution nonlinear terms can be linearized in a number of different ways. The simplest possibility, which will be used in this paper, is known as Picard linearization, in which the nonlinear terms are replaced by

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} \rightarrow \bar{u}\frac{\partial u}{\partial x} + \bar{v}\frac{\partial u}{\partial y},$$
$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} \rightarrow \bar{u}\frac{\partial v}{\partial x} + \bar{v}\frac{\partial v}{\partial y}.$$

where $\bar{u}$ and $\bar{v}$ are approximate values for the velocity components.

We assume starting values $u_{01}, u_{02}, \ldots, u_{08}$ *and* $v_{01}, v_{02}, \ldots, v_{08}$ for the element and

$$\bar{u} = \sum_{k=1}^{8} N_k u_{0k},$$
$$\bar{v} = \sum_{k=1}^{8} N_k v_{0k}.$$

The iteration process continues by replacing u_{0k} and v_{0k} , k =1, 2, ..., 8, by the average of velocity component values from the previous two iterations until tolerance is satisfied.
From Equation (16), Equation (17) and Equation (18), we get a system of equations in matrix form as

$$Ah = b, \tag{19}$$

where

$$A = \begin{bmatrix}
a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & a_{14}^{(1)} & a_{15}^{(1)} & a_{16}^{(1)} & a_{17}^{(1)} & a_{18}^{(1)} & a_{11}^{(2)} & a_{12}^{(2)} & a_{13}^{(2)} & a_{14}^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{21}^{(1)} & a_{22}^{(1)} & a_{22}^{(1)} & a_{24}^{(1)} & a_{25}^{(1)} & a_{26}^{(1)} & a_{27}^{(1)} & a_{28}^{(1)} & a_{21}^{(2)} & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} & a_{34}^{(1)} & a_{35}^{(1)} & a_{36}^{(1)} & a_{37}^{(1)} & a_{38}^{(1)} & a_{31}^{(2)} & a_{32}^{(2)} & a_{33}^{(2)} & a_{34}^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{41}^{(1)} & a_{42}^{(1)} & a_{43}^{(1)} & a_{44}^{(1)} & a_{45}^{(1)} & a_{46}^{(1)} & a_{47}^{(1)} & a_{48}^{(1)} & a_{41}^{(2)} & a_{42}^{(2)} & a_{43}^{(2)} & a_{44}^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{51}^{(1)} & a_{52}^{(1)} & a_{53}^{(1)} & a_{54}^{(1)} & a_{55}^{(1)} & a_{56}^{(1)} & a_{57}^{(1)} & a_{58}^{(1)} & a_{51}^{(2)} & a_{52}^{(2)} & a_{53}^{(2)} & a_{54}^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{61}^{(1)} & a_{62}^{(1)} & a_{63}^{(1)} & a_{64}^{(1)} & a_{65}^{(1)} & a_{66}^{(1)} & a_{67}^{(1)} & a_{68}^{(1)} & a_{61}^{(2)} & a_{62}^{(2)} & a_{63}^{(2)} & a_{64}^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{71}^{(1)} & a_{72}^{(1)} & a_{73}^{(1)} & a_{74}^{(1)} & a_{75}^{(1)} & a_{76}^{(1)} & a_{77}^{(1)} & a_{78}^{(1)} & a_{71}^{(2)} & a_{72}^{(2)} & a_{73}^{(2)} & a_{74}^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{81}^{(1)} & a_{82}^{(1)} & a_{83}^{(1)} & a_{84}^{(1)} & a_{85}^{(1)} & a_{86}^{(1)} & a_{87}^{(1)} & a_{88}^{(1)} & a_{81}^{(2)} & a_{82}^{(2)} & a_{83}^{(2)} & a_{84}^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{11}^{(4)} & a_{12}^{(4)} & a_{12}^{(4)} & a_{14}^{(4)} & a_{15}^{(4)} & a_{16}^{(4)} & a_{17}^{(4)} & a_{18}^{(4)} & 0 & 0 & 0 & 0 & a_{11}^{(6)} & a_{12}^{(6)} & a_{13}^{(6)} & a_{14}^{(6} & a_{15}^{(6)} & a_{16}^{(6)} & a_{17}^{(6)} & a_{18}^{(6)} \\
a_{21}^{(4)} & a_{22}^{(4)} & a_{23}^{(4)} & a_{24}^{(4)} & a_{25}^{(4)} & a_{26}^{(4)} & a_{27}^{(4)} & a_{28}^{(4)} & 0 & 0 & 0 & 0 & a_{21}^{(6)} & a_{22}^{(6)} & a_{23}^{(6)} & a_{24}^{(6)} & a_{25}^{(6)} & a_{26}^{(6)} & a_{27}^{(6)} & a_{28}^{(6)} \\
a_{31}^{(4)} & a_{32}^{(4)} & a_{33}^{(4)} & a_{34}^{(4)} & a_{35}^{(4)} & a_{36}^{(4)} & a_{37}^{(4)} & a_{38}^{(4)} & 0 & 0 & 0 & 0 & a_{31}^{(6)} & a_{32}^{(6)} & a_{33}^{(6)} & a_{34}^{(6)} & a_{35}^{(6)} & a_{36}^{(6)} & a_{37}^{(6)} & a_{38}^{(6)} \\
a_{41}^{(4)} & a_{42}^{(4)} & a_{43}^{(4)} & a_{44}^{(4)} & a_{45}^{(4)} & a_{46}^{(4)} & a_{47}^{(4)} & a_{48}^{(4)} & 0 & 0 & 0 & 0 & a_{41}^{(6)} & a_{42}^{(6)} & a_{43}^{(6)} & a_{44}^{(6)} & a_{45}^{(6)} & a_{36}^{(6)} & a_{47}^{(6)} & a_{48}^{(6)} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{11}^{(8)} & a_{12}^{(8)} & a_{13}^{(8)} & a_{14}^{(8)} & a_{11}^{(9)} & a_{12}^{(9)} & a_{13}^{(9)} & a_{14}^{(9)} & a_{15}^{(9)} & a_{16}^{(9)} & a_{17}^{(9)} & a_{18}^{(9)} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{21}^{(8)} & a_{22}^{(8)} & a_{23}^{(8)} & a_{24}^{(8)} & a_{21}^{(9)} & a_{22}^{(9)} & a_{23}^{(9)} & a_{24}^{(9)} & a_{25}^{(9)} & a_{26}^{(9)} & a_{27}^{(9)} & a_{28}^{(9)} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{31}^{(8)} & a_{32}^{(8)} & a_{33}^{(8)} & a_{34}^{(8)} & a_{31}^{(9)} & a_{32}^{(9)} & a_{33}^{(9)} & a_{34}^{(9)} & a_{35}^{(9)} & a_{36}^{(9)} & a_{37}^{(9)} & a_{38}^{(9)} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{41}^{(8)} & a_{42}^{(8)} & a_{43}^{(8)} & a_{44}^{(8)} & a_{41}^{(9)} & a_{42}^{(9)} & a_{43}^{(9)} & a_{44}^{(9)} & a_{45}^{(9)} & a_{46}^{(9)} & a_{47}^{(9)} & a_{48}^{(9)} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{51}^{(8)} & a_{52}^{(8)} & a_{53}^{(8)} & a_{54}^{(8)} & a_{51}^{(9)} & a_{52}^{(9)} & a_{53}^{(9)} & a_{54}^{(9)} & a_{55}^{(9)} & a_{56}^{(9)} & a_{57}^{(9)} & a_{58}^{(9)} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{61}^{(8)} & a_{62}^{(8)} & a_{63}^{(8)} & a_{64}^{(8)} & a_{61}^{(9)} & a_{62}^{(9)} & a_{63}^{(9)} & a_{64}^{(9)} & a_{65}^{(9)} & a_{66}^{(9)} & a_{67}^{(9)} & a_{68}^{(9)} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{71}^{(8)} & a_{72}^{(8)} & a_{73}^{(8)} & a_{74}^{(8)} & a_{71}^{(9)} & a_{72}^{(9)} & a_{73}^{(9)} & a_{74}^{(9)} & a_{75}^{(9)} & a_{76}^{(9)} & a_{77}^{(9)} & a_{78}^{(9)} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{81}^{(8)} & a_{82}^{(8)} & a_{83}^{(8)} & a_{84}^{(8)} & a_{81}^{(9)} & a_{82}^{(9)} & a_{83}^{(9)} & a_{84}^{(9)} & a_{85}^{(9)} & a_{86}^{(9)} & a_{87}^{(9)} & a_{88}^{(9)}
\end{bmatrix}$$

$$h = \begin{bmatrix} u \\ p \\ v \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{bmatrix},$$

$b = \mathbf{0}$ (20X1 zero vector)

Here,

$$a_{ij}^{(1)} = \iint_{\Omega}\left[N_i \bar{u} \frac{\partial N_j}{\partial x} + N_i \bar{v} \frac{\partial N_j}{\partial y} + \frac{1}{\mathrm{Re}}\left(\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} + \frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y} \right)\right] dA,$$

$$i, j = 1, 2, \ldots, 8,$$

$$a_{ij}^{(9)} = a_{ij}^{(1)},$$

$$a_{ij}^{(2)} = \iint_{\Omega} N_i \frac{\partial M_j}{\partial x} dA, \qquad a_{ij}^{(8)} = \iint_{\Omega} N_i \frac{\partial M_j}{\partial y} dA,$$

$$i = 1, 2, \ldots, 8 \quad j = 1, 2, 3, 4 \qquad i = 1, 2, \ldots, 8 \quad j = 1, 2, 3, 4$$

$$a_{ij}^{(4)} = \iint_{\Omega} M_i \frac{\partial N_j}{\partial x} dA, \qquad a_{ij}^{(6)} = \iint_{\Omega} M_i \frac{\partial N_j}{\partial y} dA.$$

$$i = 1, 2, 3, 4, \quad j = 1, 2, \ldots, 8, \qquad i = 1, 2, 3, 4 \quad j = 1, 2, \ldots, 8,$$

4. Derivatives and Integrals Using Local Coordinates

Let N(ξ,η) be a shape function in terms of local coordinates. If x and y are the global coordinates, then

$$\frac{\partial N}{\partial \xi} = \frac{\partial N}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial N}{\partial y}\frac{\partial y}{\partial \xi},$$

$$\frac{\partial N}{\partial \eta} = \frac{\partial N}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial N}{\partial y}\frac{\partial y}{\partial \eta}.$$

$$\begin{bmatrix} \frac{\partial N}{\partial \xi} \\ \frac{\partial N}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \frac{\partial N}{\partial x} \\ \frac{\partial N}{\partial y} \end{bmatrix}.$$

$$\begin{bmatrix} \frac{\partial N}{\partial \xi} \\ \frac{\partial N}{\partial \eta} \end{bmatrix} = J \begin{bmatrix} \frac{\partial N}{\partial x} \\ \frac{\partial N}{\partial y} \end{bmatrix},$$

where

$$J = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix}.$$

is the Jacobian matrix of the transformation of the global coordinate system to local coordinate system.

Then, we have

$$\begin{bmatrix} \frac{\partial N}{\partial x} \\ \frac{\partial N}{\partial y} \end{bmatrix} = J^{-1} \begin{bmatrix} \frac{\partial N}{\partial \xi} \\ \frac{\partial N}{\partial \eta} \end{bmatrix}. \tag{20}$$

5. Computing the Jacobian Matrix

Let $N_1(\xi,\eta)$, $N_2(\xi,\eta)$, . . . , $N_n(\xi,\eta)$, be the shape functions for an element in local coordinates. If (x_1, y_1), (x_2, y_2),. . . , (x_n, y_n) are the global coordinates of the nodes of the element and (x , y) is the global coordinate of a point on the element, then

$$x = N_1(\xi,\eta)x_1+N_2(\xi,\eta)x_2+. \ . \ . + N_n(\xi,\eta)x_n ,$$

$$y = N_1(\xi,\eta)x_1+N_2(\xi,\eta)x_2+. \ . \ . + N_n(\xi,\eta)x_n .$$

Then,

$$\begin{aligned}
\frac{\partial x}{\partial \xi} &= \frac{\partial N_1}{\partial \xi} x_1 + \frac{\partial N_2}{\partial \xi} x_2 + . \ . \ . + \frac{\partial N_n}{\partial \xi} x_n, \\
\frac{\partial y}{\partial \xi} &= \frac{\partial N_1}{\partial \xi} y_1 + \frac{\partial N_2}{\partial \xi} y_2 + . \ . \ . + \frac{\partial N_n}{\partial \xi} y_n, \\
\frac{\partial x}{\partial \eta} &= \frac{\partial N_1}{\partial \eta} x_1 + \frac{\partial N_2}{\partial \eta} x_2 + . \ . \ . + \frac{\partial N_n}{\partial \eta} x_n, \\
\frac{\partial y}{\partial \eta} &= \frac{\partial N_1}{\partial \eta} y_1 + \frac{\partial N_2}{\partial \eta} y_2 + . \ . \ . + \frac{\partial N_n}{\partial \eta} y_n.
\end{aligned}$$

Hence, the Jacobian matrix of the transformation J is

$$J = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \cdots & \frac{\partial N_n}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \cdots & \frac{\partial N_n}{\partial \eta} \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} \frac{\partial N_i}{\partial \xi} x_i & \sum_{i=1}^{n} \frac{\partial N_i}{\partial \xi} y_i \\ \sum_{i=1}^{n} \frac{\partial N_i}{\partial \eta} x_i & \sum_{i=1}^{n} \frac{\partial Ni}{\partial \eta} y_i \end{bmatrix} \tag{21}$$

Formation of discrete finite element system requires evaluation of integrals over elements. Except for simplest of element geometries, this integral cannot be evaluated analytically. Hence, numerical integrations is the only alternatives. Gaussian quadrature is mostly employed. For example, using calculus for coordinate transformation, a typical integral for a two dimensional rectangular element can be evaluated as

$$\iint_{\Omega} f(x,y)dxdy = \iint_{\Omega'} f\left(x(\xi,\eta), y(\xi,\eta)\right)\left|\det(J)\right|d\xi d\eta$$
$$= \int_{-1}^{1}\int_{-1}^{1} f\left(x(\xi,\eta), y(\xi,\eta)\right)\left|\det(J)\right|d\xi d\eta$$
$$= \sum_{i=1}^{m}\sum_{j=1}^{n} w_i w_j \bar{f}(\xi_i,\eta_j), \qquad \bar{f}(\xi,\eta) = f\left(x(\xi,\eta), y(\xi,\eta)\right)\left|\det(J)\right|.$$

where J is the Jacobian matrix of the transformation, ξ_i and η_j are Gaussian quadrature abscissa, and w_i and w_j are corresponding weights.

6. Global System of Equations

After developing governing equations for each element, assembly of the element equations was performed in order to establish global system of equations for the whole domain. In addition to the element equations, Global coordinates to nodes, elements connectivity, and global degree of freedom for nodes are also used develop the global system equations. Applying the boundary conditions, the modified global system of equations is obtained. The MATLAB codes used for solution process are indicated in the appendix.

7. Results and Discussion

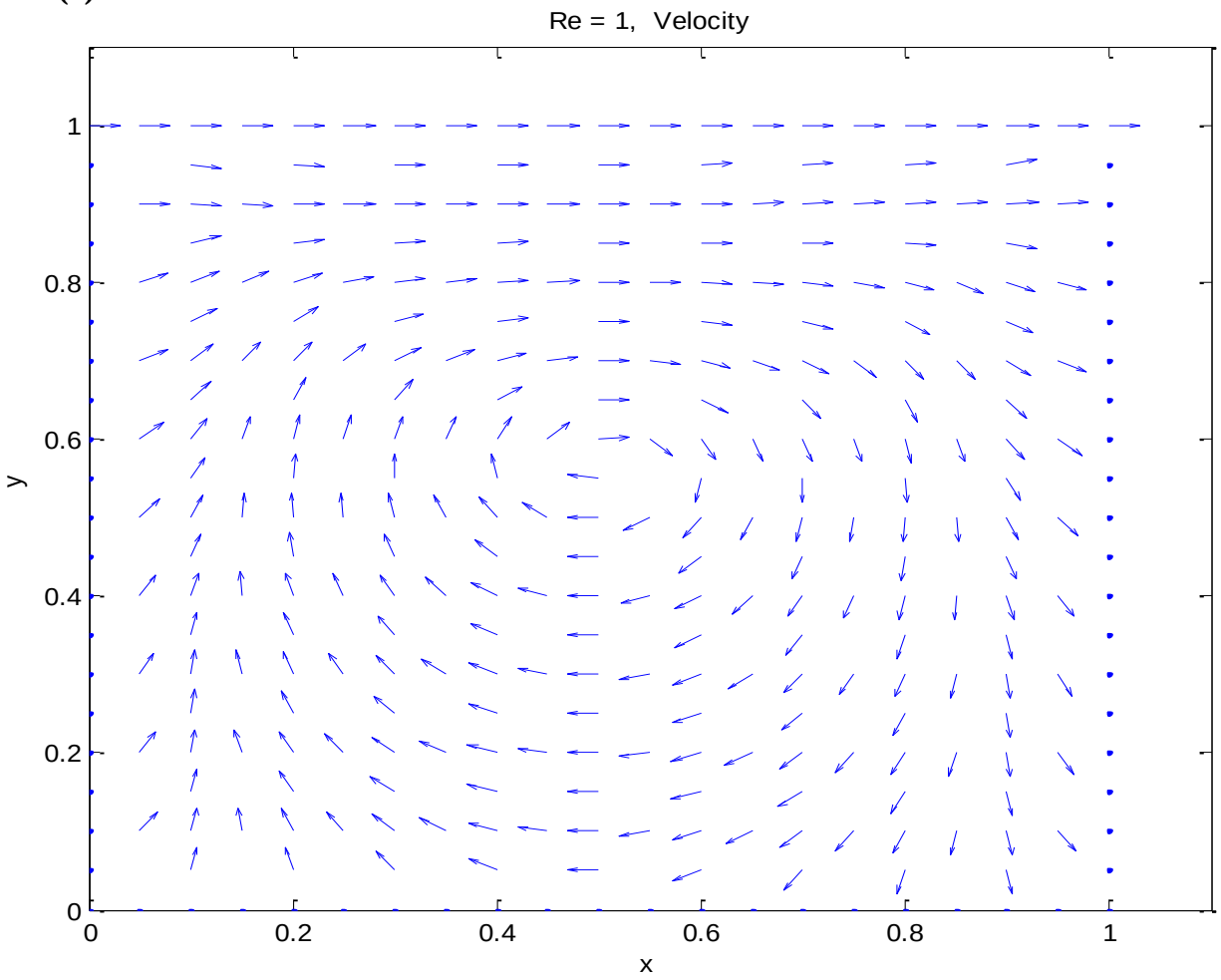
To illustrate the finite element method algorithm discussed in this paper, we considered a square lid-driven cavity flow of length 1 unit. The boundary conditions are such that the flow is driven by a unit horizontal velocity at the top boundary. The velocities at the other three boundaries are set to zero. Eight-node elements are used to model the vector field of velocities, and 4-node elements are used to model the scalar field of pressures. The flow is simulated with Reynolds numbers 1, 10, 50, 100, 200, 500 and 1000 using the same mesh of 100 elements (803 nodes). The numerical results are presented here in terms of velocity quiver plot and pressure contour at the Reynolds numbers. The results in this work were generated by the series of finite element codes we developed. The computations had been carried out with the convergence check of 10^{-6} (tolerance).

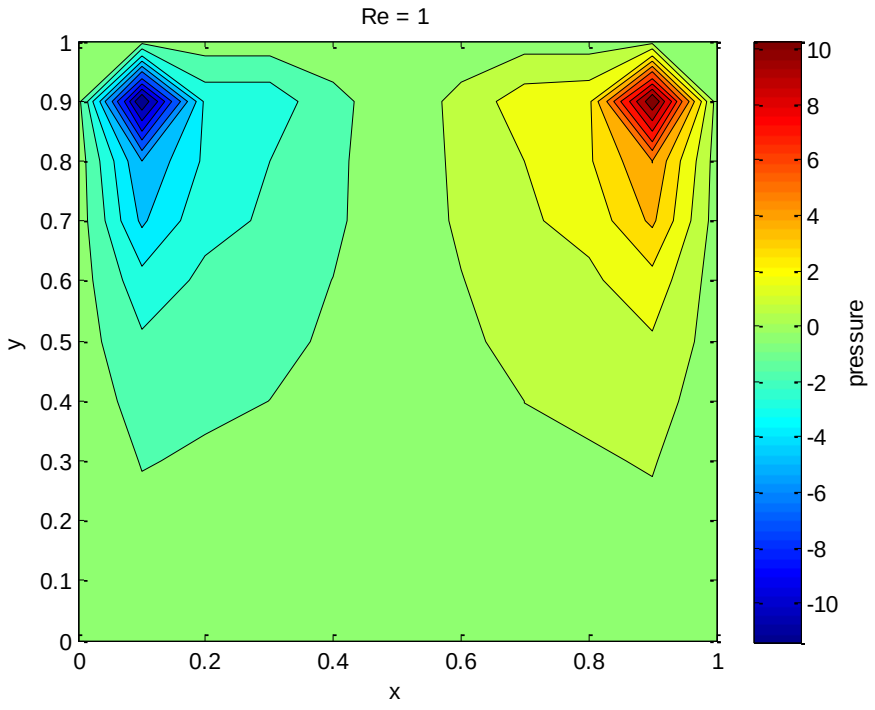
Table 1. Computational performance of five simulations performed for the cavity flow

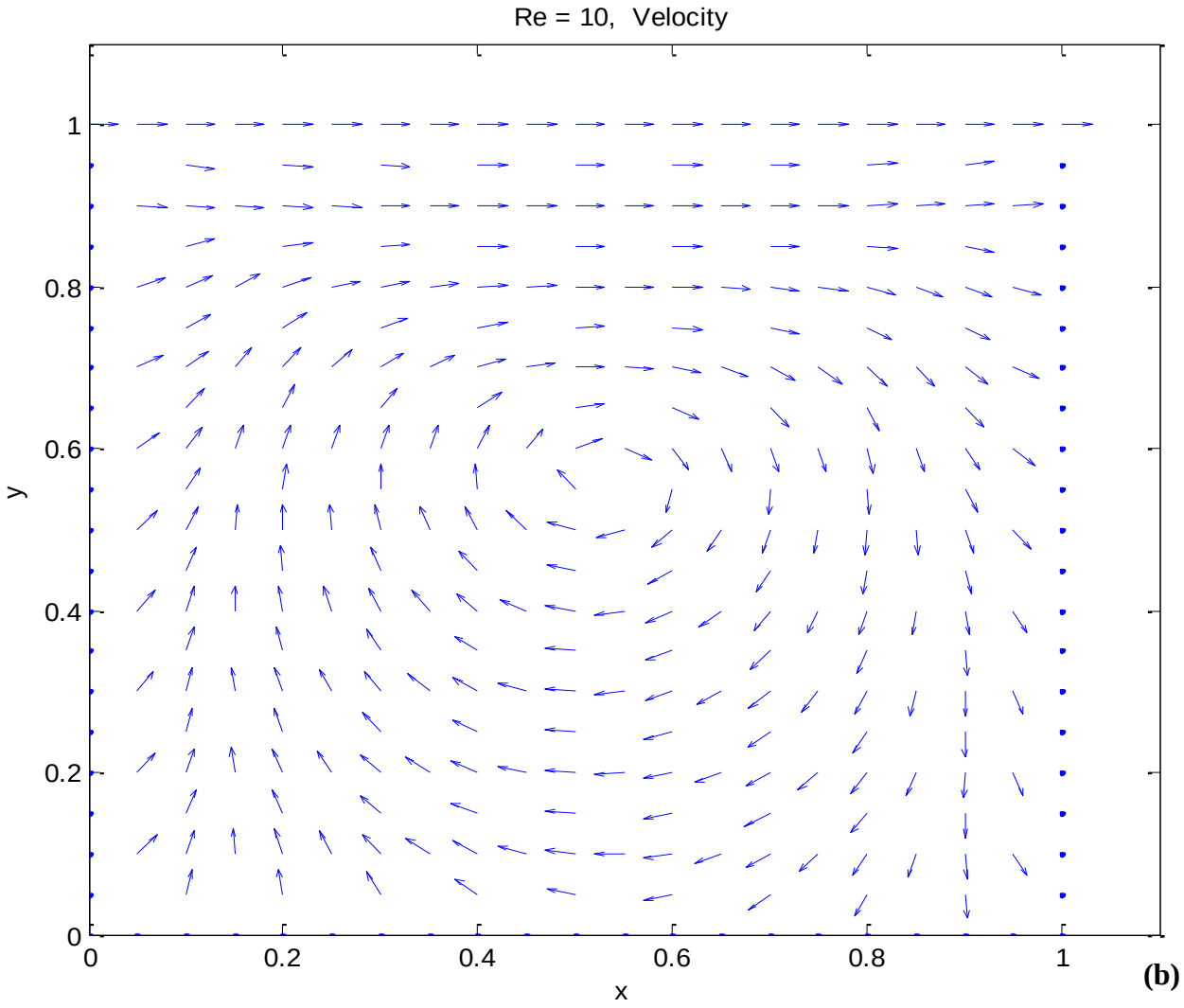
Re	1	10	50	100	200	500	1000
Number of iterations	21	22	26	29	35	47	147
Time Spent for Iterations (sec.)	4.04	4.14	4.72	5.26	6.21	7.62	21.06

As seen from Table1, high Reynolds numbers require more iteration and elapsed time to converge.

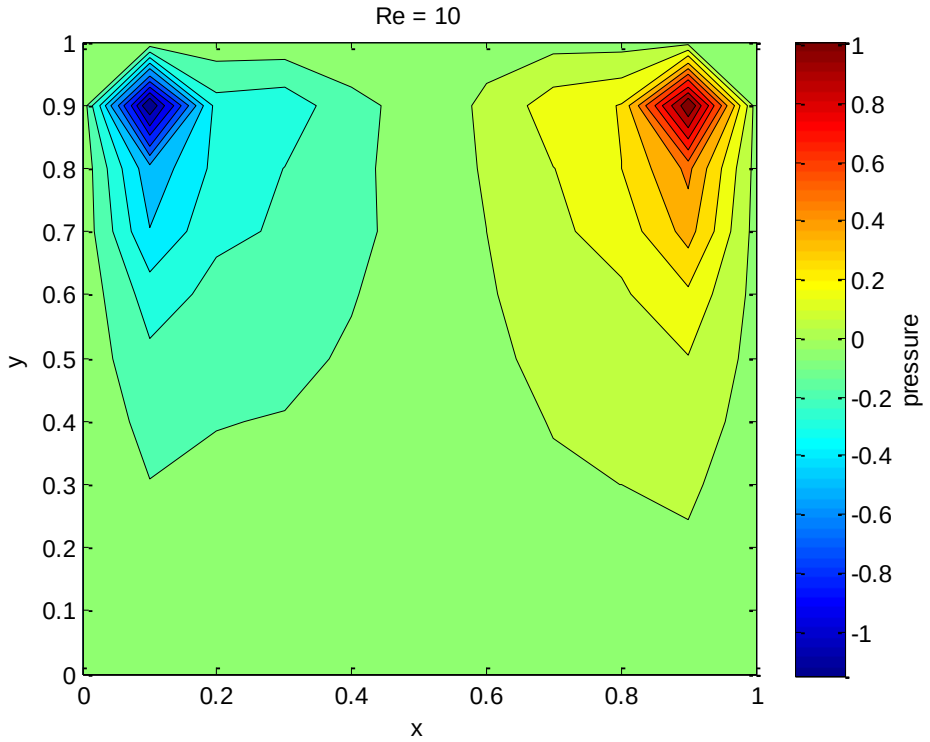
(a)

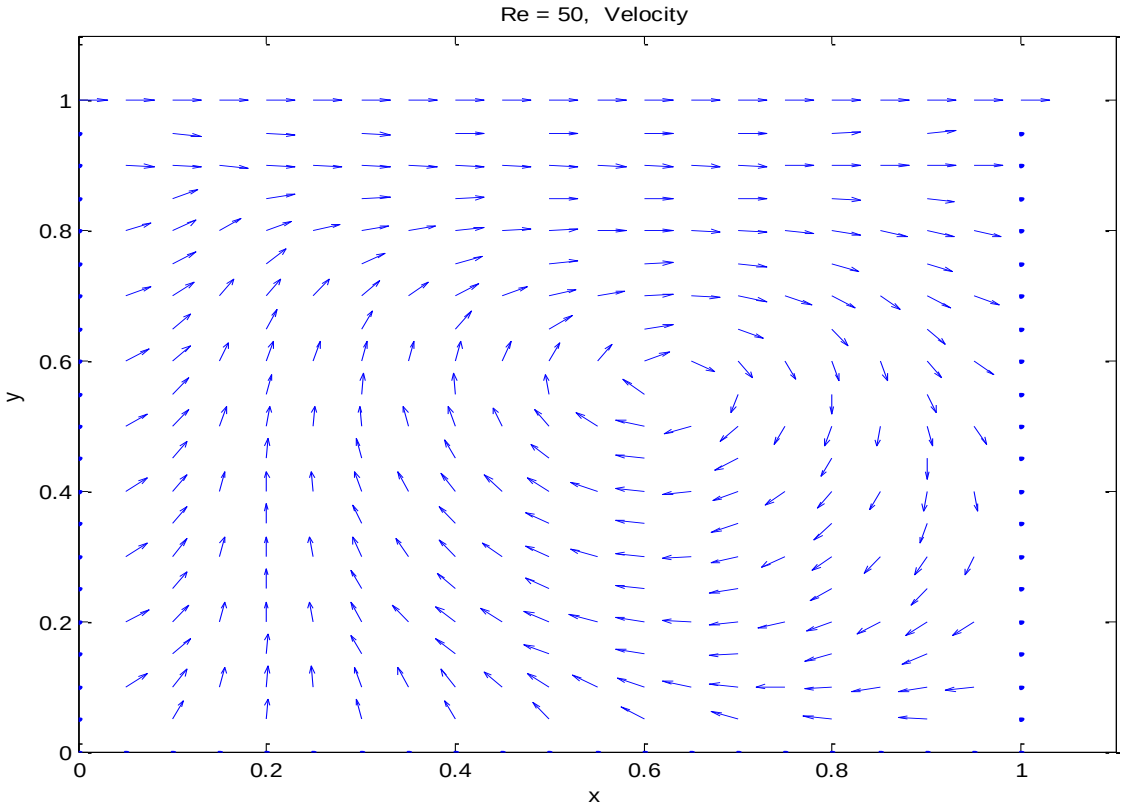




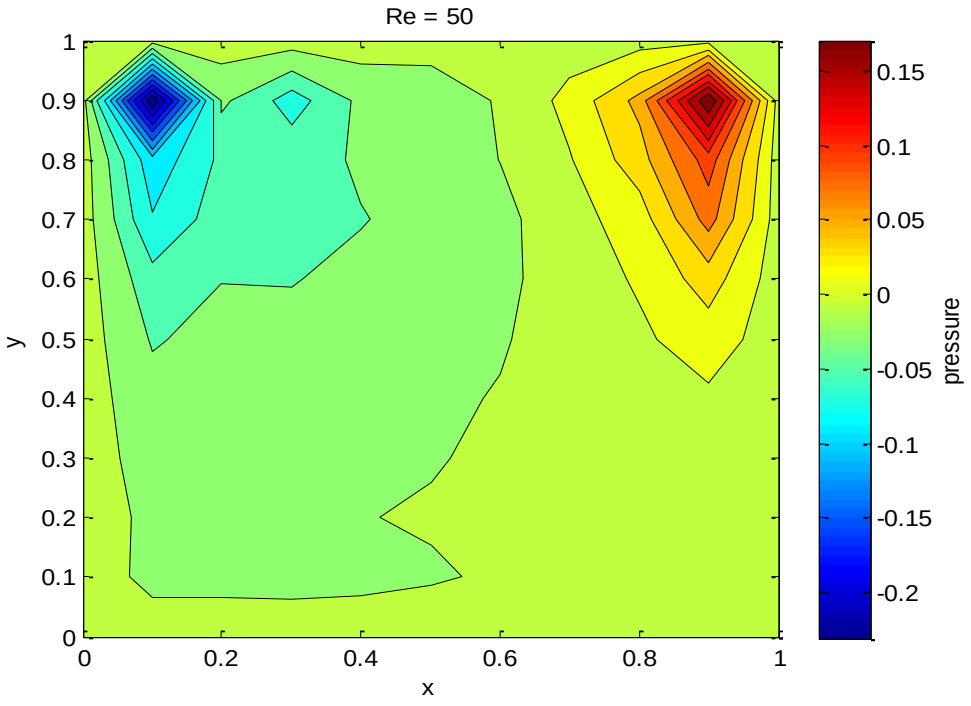

Re = 10, Velocity
y
x
0
0.2
0.4
0.6
0.8
1

(b)

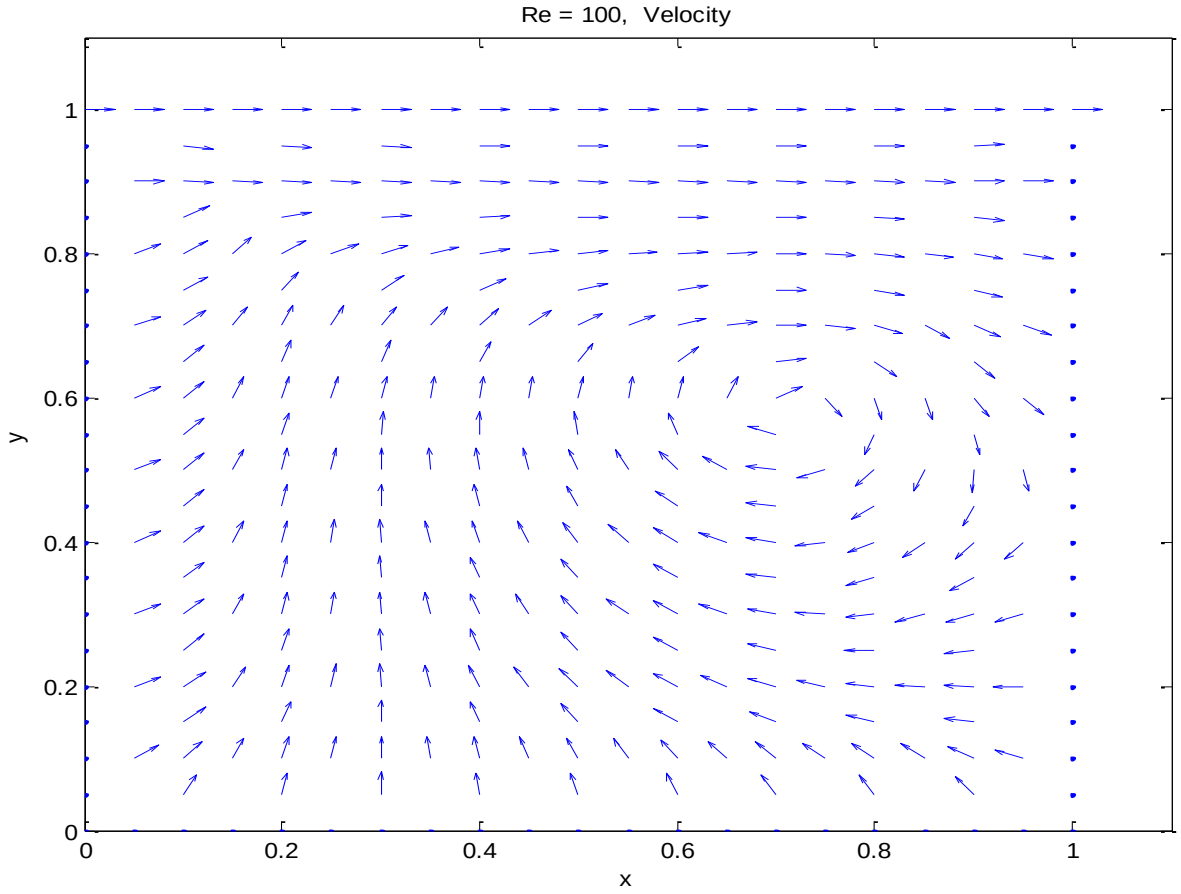

Re = 10
y
x
pressure
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.2
-0.4
-0.6
-0.8
-1

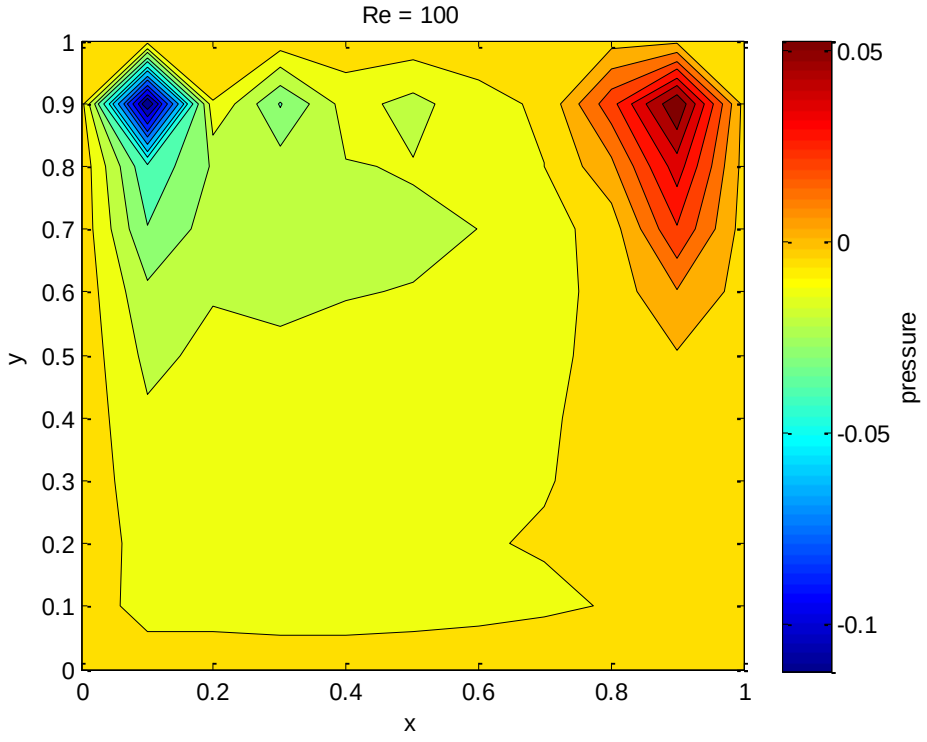

Re = 50, Velocity
y
x
0
0.2
0.4
0.6
0.8
1

(c)

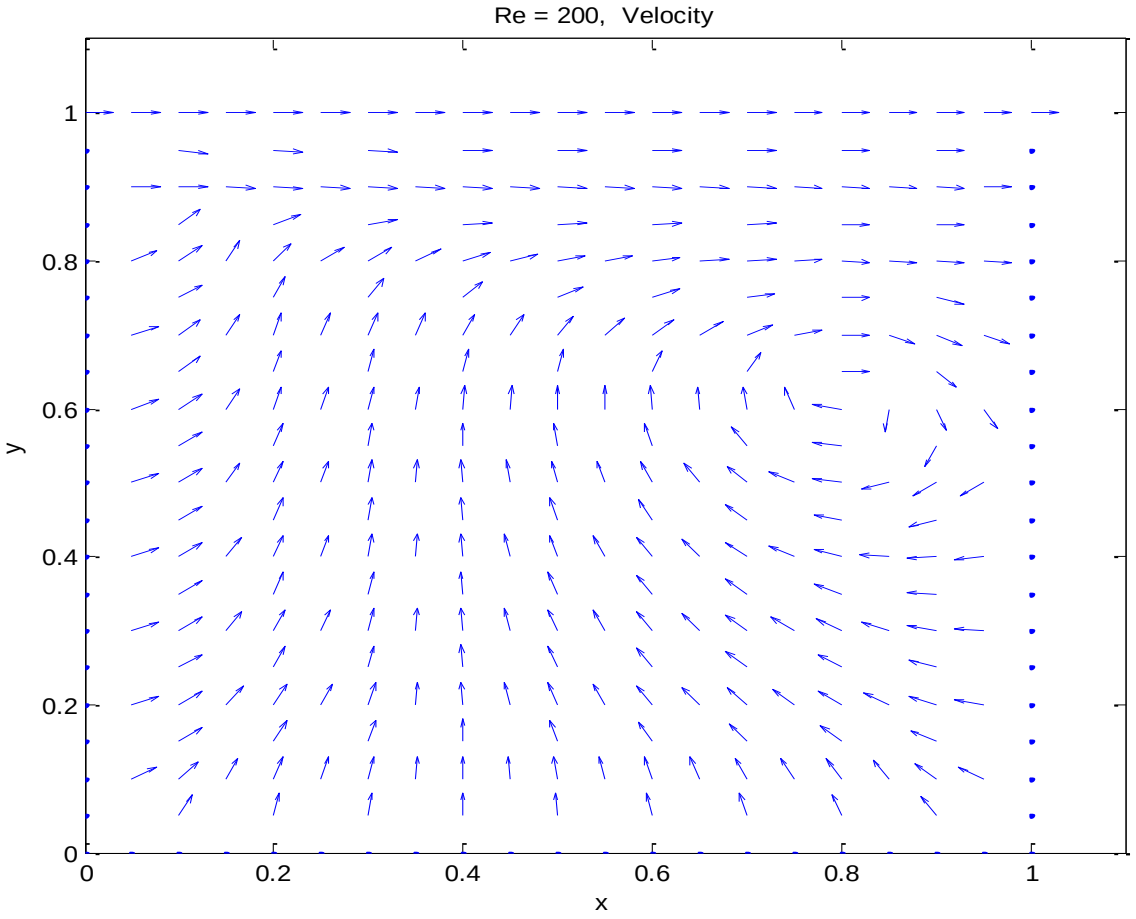

Re = 50
y
x
pressure
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2

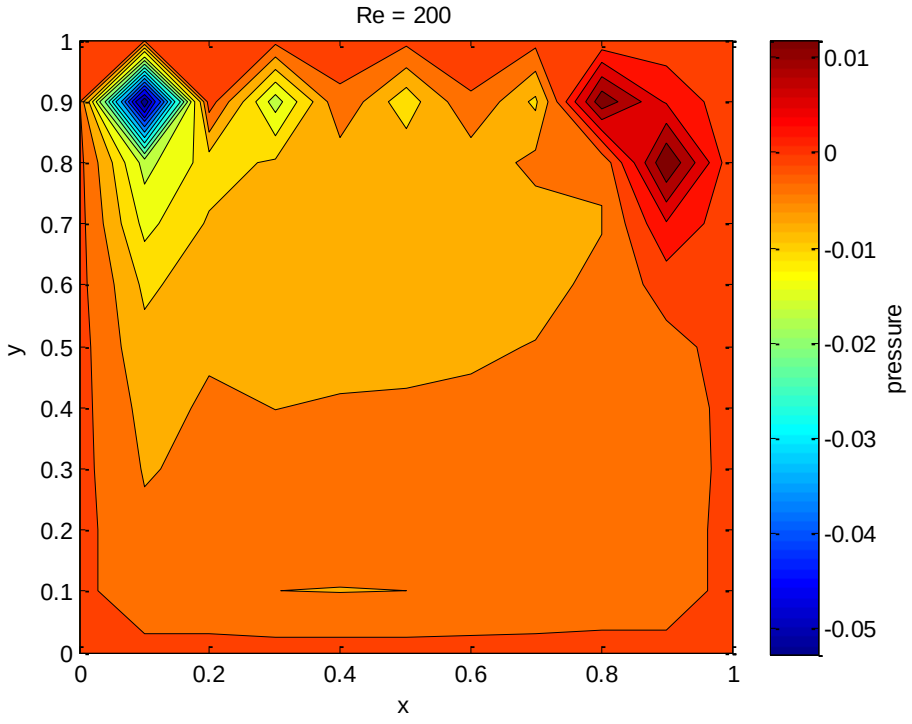
(d)



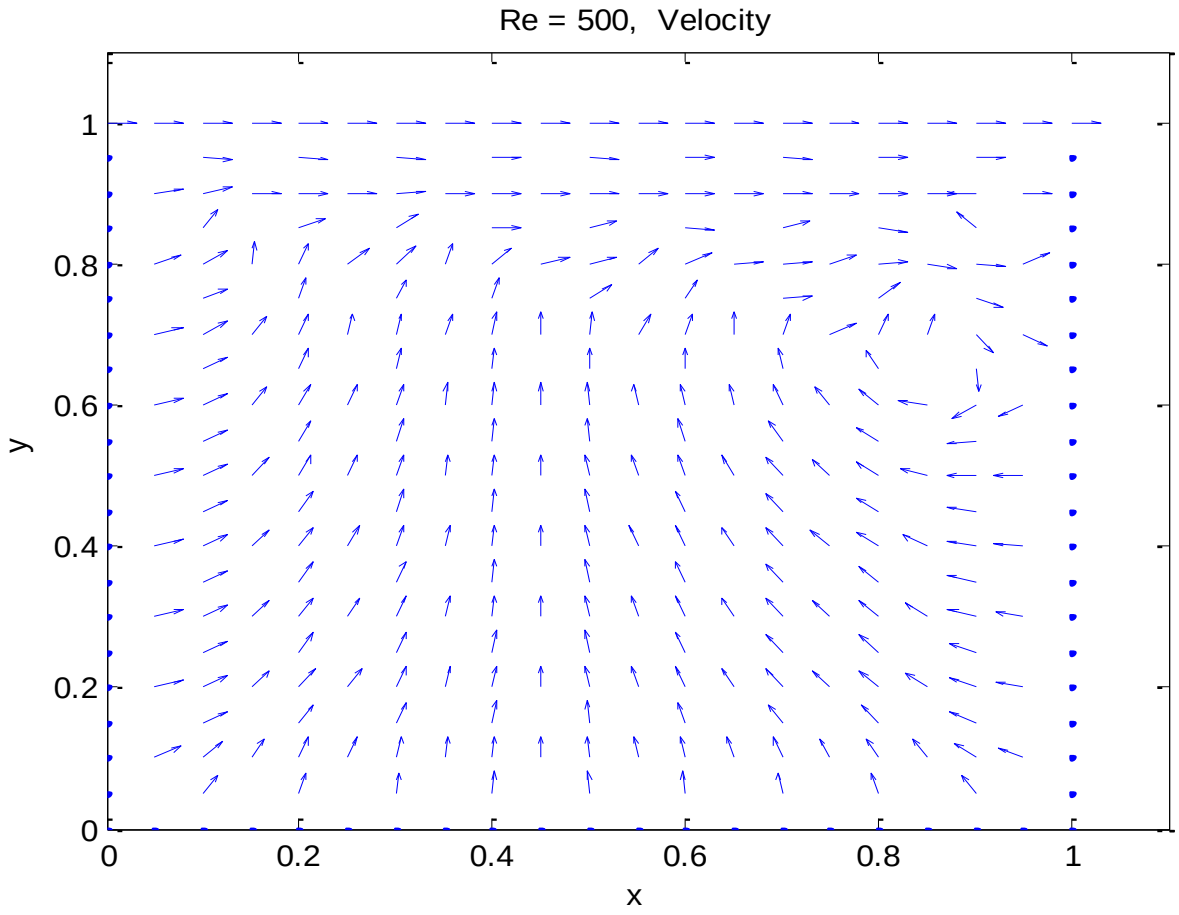


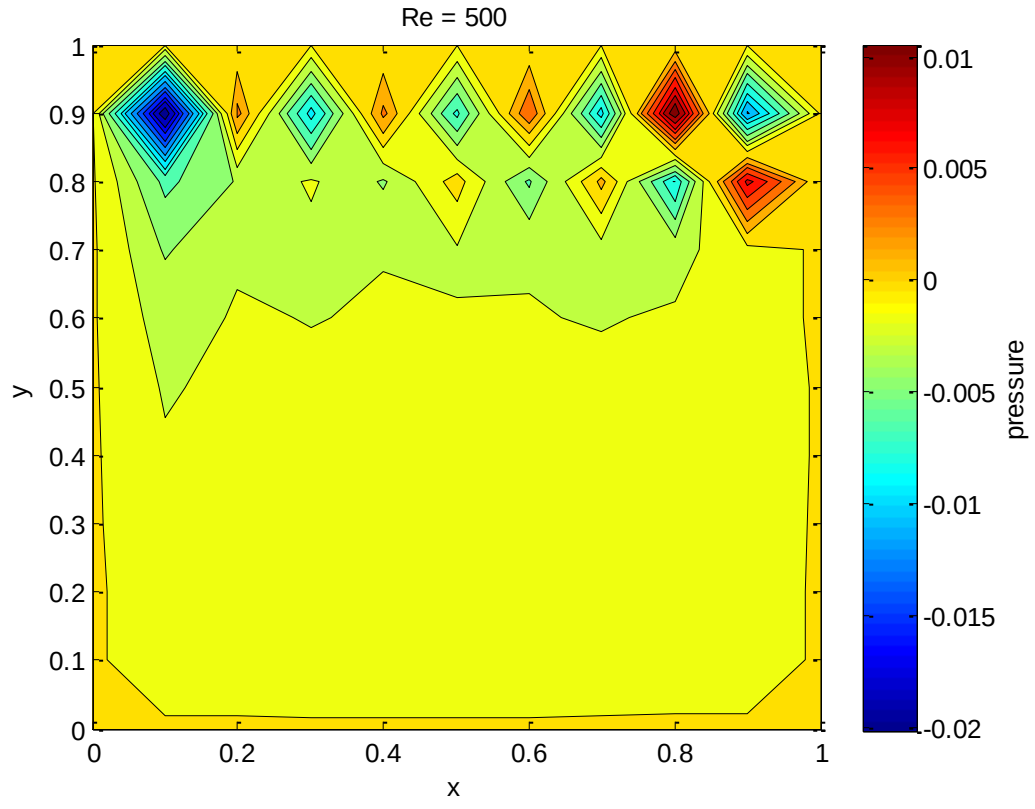
(e)


Re = 200, Velocity
y
x
0
0.2
0.4
0.6
0.8
1


Re = 200
y
x
pressure
0.01
0
-0.01
-0.02
-0.03
-0.04
-0.05

(f)


Re = 500, Velocity
y
x
0
0.2
0.4
0.6
0.8
1


Re = 500
y
x
pressure
0.01
0.005
0
-0.005
-0.01
-0.015
-0.02

(g)

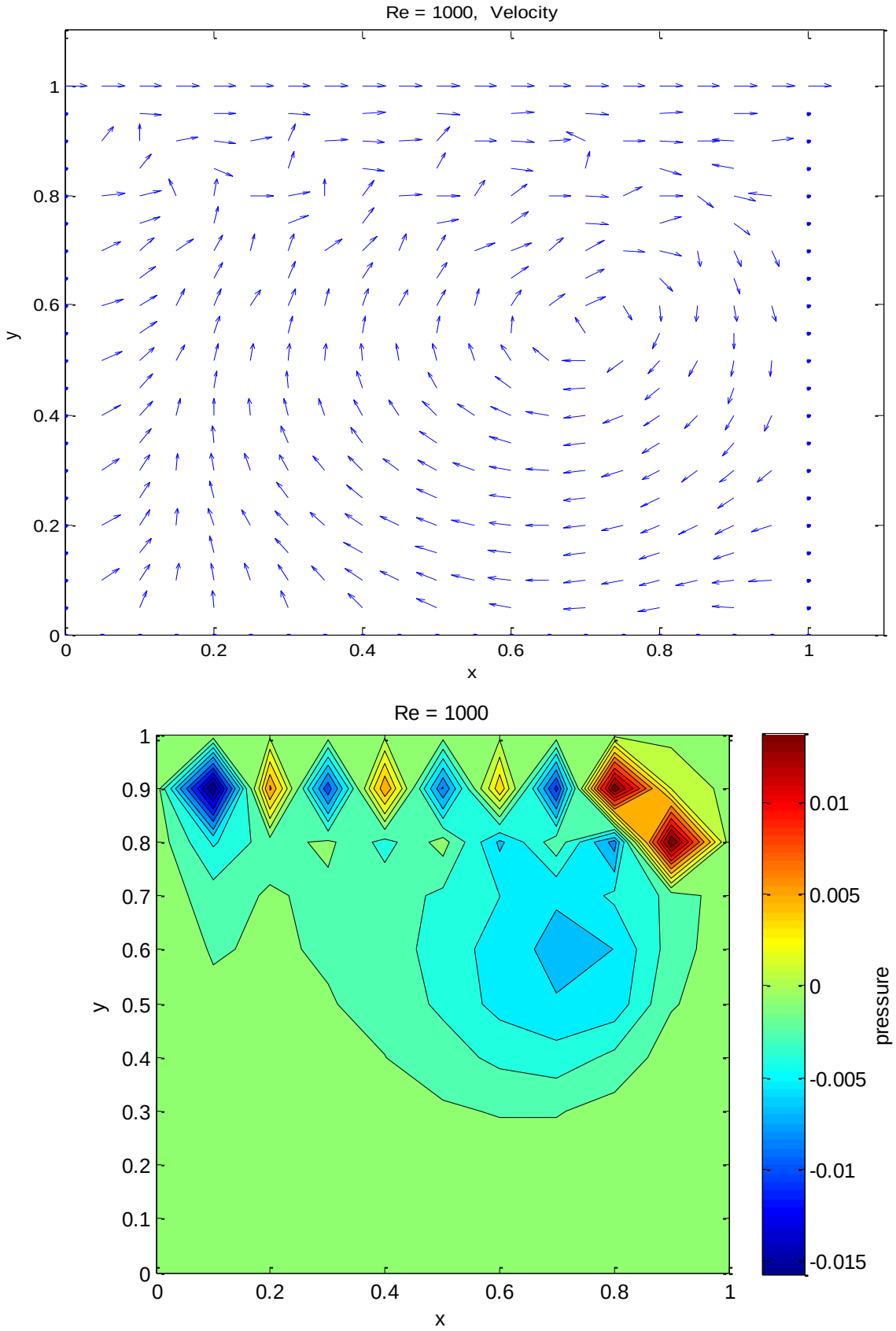


Figure 4. (a) - (g) Velocity quiver and pressure contour plots at different Reynolds numbers

8. Conclusion

In this paper, we discussed finite element solution of the two-dimensional incompressible Navier-Stokes equations by the benchmark of square lid-driven cavity. Dirichlet boundary conditions were imposed on every boundary of the domain. The finite element programming codes were constructed to solve these equations. These programming codes are written using MATLAB 7.10.0 (R2010a). The finite element programs consist of one main program and nine sub programs. These programs with modification can be used to solve related fluid flow problems. The codes for the geometry and the boundary conditions are original and very efficient. The numerical results from finite element programming agreed with the numerical results obtained from finite element analysis done by Ghia et al. (1982).

Acknowledgement

The authors would like thank the reviewers for their valuable comments to improve our paper. We also want to express our sincere thanks and appreciation to the chief editor for his prompt reply, careful work and clear instructions.

REFERENCES

Cengel, Yunus A. and Cimbala, John M. (2014). Fluid Mechanics: Fundamentals and Applications, Third edition, McGraw-Hill.

Chung, T. J. (2010), Computational Fluid Dynamics, Second edition, Cambridge University press.

Ghia U., Ghia, K. N., and Shin, C. T. (1982). High Reynolds number solutions for Incompressible Flow Using the Navier-Stokes Equation and the Multigrid Method, Computational Physics, vol. 48, pp. 387–411.

Glaisner, F. and Tezduyar, T.E. (1987). Finite Element Techniques for the Navier-Stokes Equations in the Primitive Variable Formulation and the Vorticity Stream-function Formulation, Department of Mechanical Engineering, University of Houston.

Jiajan, Wanchai (2010). Solution to incompressible Navier- Stokes equations by using finite element method, MSc thesis, University of Texas at Arlington.

Khennane, Amar (2013). Introduction to Finite Element Analysis Using MATLAB and Abaqus, Taylor & Francis Group, LLC.

Papadopoulos, Panayiotis (2010). Introduction to the Finite Element Method.

Per-Olof Persson (2002). Implementation of Finite Element-Based Navier-Stokes Solver, 2.094 – Project, MIT.

Reddy, J.N. (2006). An Introduction to the Finite Element Method, Third edition, Texas A&M University, Texas USA .

Rhaman, M. M. and Helal, K. M. (2014).Numerical Simulations of Unsteady Navier-Stokes Equations for Incompressible Newtonian Fluids using FreeFem++ based on Finite Element Method, Annals of Pure and Applied Mathematic Vol. 6, No. 1, 70-84.

Sert, Cüneyt (2012). Finite Element Analysis in Thermofluids, Middle East Technical University, Department of Mechanical Engineering.

Simpson, Guy (2017). Practical Finite Element Modelling in Earth Science using Matlab.

Smith, I.M., Griffiths, D. V. and Margets, L. (2014). Programming the Finite Element Method, John Wiley & Sons Ltd

Taylor, C. and Hughes, T.G. (1981). Finite Element Programming of Navier-Stokes Equations, Swansea, U.K., Pineridge Press Ltd.

Zienkiewicz, O. C., Taylor and R. L., Nithiarasu, P. (2014). The Finite Element Method for Fluid Dynamics, Seventh edition, Elsevier Ltd.

APPENDIX

The Finite Element Programming Codes

1. The Main program

```
%Finite element program for 2D steady incompressible
%Navier-stokes equation
clear all;
close all;
clc;
%Nx=3;Ny=3;x0=0;xf=3;y0=0;yf=0.6;
Nx=10;Ny=10;x0=0;xf=1;y0=0;yf=1;
%Nx=5;Ny=5;x0=0;xf=1;y0=-1;yf=0;
Re=[1 10 50 100 200  500 1000];
for j=1:length(Re)
tol=1e-6;
maxit=1000;
erru=1;errv=1;
vnd=(Nx+1)*(Ny+1);
mnd=(Nx+1)*Ny+Nx*(Ny+1);
nd=vnd+mnd;
neq=2*nd+vnd;
u0=ones(nd,1);
v0=zeros(nd,1);
celem=connective(Nx,Ny);
elem8=[celem(:,1:8)];
elem4=[celem(:,9:12)];
gdof=fungdof(Nx,Ny,x0,xf,y0,yf);
nel=length(elem8(:,1));
node=node48(Nx,Ny,x0,xf,y0,yf);
ne=length(celem(:,1));
iter =0;
tic
while(or(erru>tol,errv>tol) & iter<=maxit)
    iter =iter+1
    %Step5. Assembely
    %Assembly matrix
    A=assemblymatrix(Nx,Ny,x0,xf,y0,yf,u0,v0,Re(j));
    %Assembly vector
    b=assemblyfluidvector(Nx,Ny,x0,xf,y0,yf,Re(j));
    dx=(xf-x0)/Nx;dy=(yf-y0)/Ny;
    bdof=funbdof(Nx,Ny);
    nb=10*(Nx+Ny);
    bv=zeros(nb,1);
    bv(2*Nx+4*Ny:4*(Nx+Ny))=1;
```

```
%     bv(4*(Nx+Ny)+1)=1;
    %Step6.Applying boundary conditions
    upv=zeros(neq,1);
    nc=length(bdof);
    for i=1:nc
        id=bdof(i);
        upv(id)=bv(i);
    end
    freeNodes=setdiff(1:neq,bdof);
    b=b-A*upv;
    %Step7.Solve the system
    upv(freeNodes)=A(freeNodes,freeNodes)\b(freeNodes);
    %Step8.Post processing
    u=upv(1:nd);
    p=upv(nd+1:nd+vnd);
    v=upv(nd+vnd+1:neq);
    erru=max(abs(u-u0))
    errv=max(abs(v-v0))
     uvel=(u+u0)/2;
     vvel=(v+v0)/2;
%      invel(1:nd)=uvel;
%      invel(nd+vnd+1:neq)=vvel;
%      u0=invel(1:nd);
%      v0=invel(nd+vnd+1:neq)
%  u0=u;
%  v0=v;
    u0=uvel;
    v0=vvel;
end

fprintf('u-velocity     v-velocity\n')
velocity=[u v];
disp(velocity);
fprintf('pressure\n')
disp(p)
figure
%  subplot(1,2,1)
node=node48(Nx,Ny,x0,xf,y0,yf);
nodep=node4(Nx,Ny,x0,xf,y0,yf);
x=node(:,1);
y=node(:,2);
L = sqrt(u.^2+v.^2);
quiver(x,y,u./L,v./L,0.4);
axis([0 1.1 0 1.1]);
xlabel('x');
title(['Re = ',num2str(Re(j)) ',  Velocity'])
ylabel('y')
view(0,90);
```

```
x=x0:dx:xf;
y=y0:dy:yf;
[X,Y]=meshgrid(x,y);
for i=1:Ny+1
    Mp(i,:)=p((Nx+1)*i-Nx:(Nx+1)*i);
end
%     figure
%     surf(X,Y,Mp,'FaceColor','interp','EdgeColor','none');
%     title('Re=100');
%     h=colorbar;
%     set(get(h,'title'),'String','pressure');
%     xlabel('x');
%     ylabel('y')
%     view(0,90);
figure
% subplot(1,2,2)
contourf(X,Y,Mp,20);
title(['Re = ',num2str(Re(j))])
h=colorbar;
set(get(h,'ylabel'),'String','pressure');
xlabel('x');
ylabel('y')
view(0,90);

end
toc
```

2. Mesh Generating Code

```
% Step1: mesh generating
close all;
clear all;
clc
Nx=5;Ny=5;x0=0;xf=1;y0=0;yf=1;
dx=(xf-x0)/Nx;dy=(yf-y0)/Ny;
x1=x0:dx/2:xf;
x2=x0:dx:xf;
y1=y0:dy:yf;
y2=dy/2:dy:yf;
x3=x0:dx:xf;
y3=y0:dy:yf;
[X1,Y1]=meshgrid(x1,y1);
[X2,Y2]=meshgrid(x2,y2);
[X3,Y3]=meshgrid(x3,y3);
u1=zeros(length(x1),length(y1));
u2=zeros(length(x2),length(y2));
u3=zeros(length(x3),length(y3));
figure
surf(x3,y3,u3','FaceColor','interp');
view(0,90)
hold on
p1=plot3(X2,Y2,u2','.','markersize',25);
p2=plot3(X1,Y1,u1','.','markersize',25);
axis([x0 xf y0 yf 0 0.01]);
```

3. Global coordinates to nodes

```
function node=node48(Nx,Ny,x0,xf,y0,yf)
%Step3:Assigning global coordinates to nodes
%Nx=Number of elments in x-direction
%Ny=Number of elements in y direction
%Lx=Length of rectanglar domain in x direction
%Ly=Length of rectanglar domain in y direction
%node4= coordinates of 4 node elements
%node8= coordinates of 8 node elements
dx=(xf-x0)/Nx;dy=(yf-y0)/Ny;
x1=x0:dx/2:xf;
y1=y0:dy/2:yf;
x2=dx/2:dx:xf;
y2=dy/2:dy:yf;
[X1,Y1]=meshgrid(y1,x1);
node1=[Y1(:) X1(:)];
[X2,Y2]=meshgrid(y2,x2);
node2=[Y2(:) X2(:)];
node1=num2str(node1);
node2=num2str(node2);
node=sortrows(setdiff(node1,node2,'rows'),2);
node=str2num(node);
node=sortrows(node,2);
node=sortrows(node,1);
node=sortrows(node,2);
```

4. Global node numbers Code

```
function celem=connective(Nx,Ny)
%Nx=Number of elments in x-direction
%Ny=Number of elements in y direction
%elem4=4 noded elements with global nodes
%elem8=8 noded elements with global nodes
elem4=zeros(Nx*Ny,4);
for i=1:Nx*Ny
    r(i)=mod(i,Nx);
    q(i)=(i-r(i))/(Nx);
    if r(i)==0;
        s(i)=i+q(i)-1;
    else
        s(i)=i+q(i);
    end
end
for k=1:Nx
    elem4(k,1)=2*k-1;
    elem4(k,2)=2*k+1;
    elem4(k,3)=elem4(k,2)+3*Nx+2;
    elem4(k,4)=elem4(k,1)+3*Nx+2;
end
  for k=Nx+1:Nx*Ny
    elem4(k,1:4)=elem4(k-Nx,1:4)+3*Nx+2;
  end
elem8=zeros(Nx*Ny,8);
```

```
for k=1:Nx
    elem8(k,1)=2*k-1;
    elem8(k,2)=2*k;
    elem8(k,3)=2*k+1;
    elem8(k,8)=elem8(k,1)+2*(Nx+1)-k;
    elem8(k,5)=elem8(k,3)+3*Nx+2;
    elem8(k,6)=elem8(k,2)+3*Nx+2;
    elem8(k,7)=elem8(k,1)+3*Nx+2;
    elem8(k,4)=elem8(k,8)+1;
end
  for k=Nx+1:Nx*Ny
    elem8(k,1:8)=elem8(k-Nx,1:8)+3*Nx+2;
  end
celem=[elem8(:,:) elem4(:,:)];
end
```

5. Create global degree of freedom

```
function gdof=fungdof(Nx,Ny,x0,xf,y0,yf)
mnd=(Nx+1)*Ny+Nx*(Ny+1);
vnd=(Nx+1)*(Ny+1);
mnd=(Nx+1)*Ny+Nx*(Ny+1);
NN=vnd+mnd;
celem=connective(Nx,Ny);
elem8=[celem(:,1:8)];
pnode=zeros(Nx*Ny,4);
for k=1:Nx
    pnode(k,1)=NN+k;
    pnode(k,2)=NN+k+1;
    pnode(k,3)=pnode(k,2)+Nx+1;
    pnode(k,4)=pnode(k,3)-1;
end
  for k=Nx+1:Nx*Ny
   pnode(k,1:4)=pnode(k-Nx,1:4)+Nx+1;
  end
 for e = 1:length(celem(:,1))
    dofCounter = 0;
    % u velocity DOFs
    for i = 1:8
        dofCounter = dofCounter + 1;
        gdof(e,dofCounter) = elem8(e,i);
    end
    % pressure DOFs
    for i = 1:4
        dofCounter = dofCounter + 1;
        gdof(e,dofCounter) = pnode(e,i);
    end

    % v velocity DOFs
    for i =1:8
        dofCounter = dofCounter + 1;
        gdof(e,dofCounter) =2*NN-mnd + elem8(e,i);
    end

end
```

6. Calculate local matrix for each element

```
function Aelem=fluidelemmatrix(coord8,coord4,u0,v0,Re)
Aelem=zeros(20,20);
A1=zeros(8,8);
A2=zeros(8,4);
A3=zeros(8,8);
A4=zeros(4,8);
A5=zeros(4,4);
A6=zeros(4,8);
A7=zeros(8,8);
A8=zeros(8,4);
A9=zeros(8,8);
Nx=5;Ny=5;x0=0;xf=1;y0=0;yf=1;
celem=connective(Nx,Ny);
elem8=[celem(:,1:8)];
elem4=[celem(:,9:12)];
nel=length(elem8(:,1));
u0=ones(8,1);
v0=zeros(8,1);
xp=[-sqrt(0.6) 0 sqrt(0.6)];
wp=[5/9 8/9 5/9];
[g1,g2]=meshgrid(xp,xp);
gp=[g2(:) g1(:)];
[w1,w2]=meshgrid(wp,wp);
w=w2(:).* w1(:);
ngp=length(gp);
for k=1:ngp
    ksi=gp(k,1);
    eta=gp(k,2);
    N(1,k)=-0.25*(1-ksi)*(1-eta)*(1+ksi+eta);
    N(2,k)=0.5*(1-ksi.^2)*(1-eta);
    N(3,k)=-0.25*(1+ksi)*(1-eta)*(1-ksi+eta);
    N(4,k)=0.5*(1+ksi)*(1-eta.^2);
    N(5,k)= -0.25*(1+ksi)*(1+eta)*(1-ksi-eta);
    N(6,k)=0.5*(1-ksi.^2)*(1+eta);
    N(7,k)= -0.25*(1-ksi)*(1+eta)*(1+ksi-eta);
    N(8,k)= 0.5*(1-ksi)*(1-eta.^2);
    dN(1,1,k)=0.25*(1-eta)*(2*ksi+eta);
    dN(1,2,k)=-ksi*(1-eta);
    dN(1,3,k)=0.25*(eta - 1)*(eta - 2*ksi);
    dN(1,4,k)=0.5*(1-eta.^2);
    dN(1,5,k)=0.25*(eta + 1)*(eta + 2*ksi);
    dN(1,6,k)=-ksi*(eta + 1);
    dN(1,7,k)=-0.25*(1+eta)*(eta-2*ksi);
    dN(1,8,k)=0.5*(-1+eta.^2);
```

```
    dN(2,1,k)=0.25*(1-ksi)*(2*eta+ksi);
    dN(2,2,k)=0.5*(-1+ksi.^2);
    dN(2,3,k)=0.25*(1+ksi)*(2*eta-ksi);
    dN(2,4,k)= -eta*(1+ksi);
    dN(2,5,k)=0.25*(1+ksi)*(ksi+2*eta);
    dN(2,6,k)=0.5*(1-ksi.^2);
    dN(2,7,k)=0.25*(1-ksi)*(2*eta-ksi);
    dN(2,8,k)= eta*(ksi-1);
    M(1,k)=0.25*(1-ksi)*(1-eta);
    M(2,k)=0.25*(1+ksi)*(1-eta);
    M(3,k)=0.25*(1+ksi)*(1+eta);
    M(4,k)=0.25*(1-ksi)*(1+eta);

    dM(1,1,k)=-0.25*(1-eta);
    dM(1,2,k)=0.25*(1-eta);
    dM(1,3,k)=0.25*(1+eta);
    dM(1,4,k)=-0.25*(1+eta);

    dM(2,1,k)=-0.25*(1-ksi);
    dM(2,2,k)= -0.25*(1+ksi);
    dM(2,3,k)= 0.25*(1+ksi);
    dM(2,4,k) =0.25*(1-ksi);
end


for k=1:ngp
    ubar = 0;
    vbar = 0;
    for i = 1:8
        ubar = ubar + N(i,k)*u0(i);
        vbar = vbar + N(i,k)*v0(i);
    end
    Jacob8(:,:)= dN(:,:,k)*coord8(:,:);%Jacobian at kth gp
    detJacob8=abs(det(Jacob8));
    gdN=Jacob8(:,:)\dN(:,:,k);%2X8 change of variable in der. at kth gp
    gdN1=gdN(1,:);
    gdN2=gdN(2,:);
    for i=1:8
        for j=1:8
            A1(i,j)=A1(i,j)+w(k)*(N(i,k)*ubar*gdN1(j)...
                +N(i,k)*vbar*gdN2(j)...
                +(1/Re)*(gdN1(i)*gdN1(j)+gdN2(i)*gdN2(j)))*detJacob8;
        end
    end
    A9=A1;
    Jacob4(:,:)=dM(:,:,k)*coord4(:,:);
    gdM=Jacob4(:,:)\dM(:,:,k);
```

```
    gdM1=gdM(1,:);
    gdM2=gdM(2,:);
    detJacob4 = abs(det(Jacob4));
    for i=1:8
        for j=1:4
            A2(i,j)=A2(i,j)+w(k)*N(i,k)*gdM1(j)*detJacob4;
            A8(i,j)=A8(i,j)+w(k)*N(i,k)*gdM2(j)*detJacob4;
        end
    end
    Jacob4(:,:)=dM(:,:,k)*coord4(:,:);
    gdM=Jacob4(:,:)\dM(:,:,k);
    gdM1=gdM(1,:);
    gdM2=gdM(2,:);
    detJacob4=abs(det(Jacob4));
    for i=1:4
        for j=1:8
            A4(i,j)=A4(i,j)+w(k)*M(i,k)*gdN1(j)*detJacob4;
            A6(i,j)=A6(i,j)+w(k)*M(i,k)*gdN2(j)*detJacob4;
        end
    end
end
Aelem=[A1 A2 A3;A4 A5 A6;A7 A8 A9];
```

7. Calculate local vector for each element

```
function belem=localfluidvector(celem)
n=length(celem(1,:));
belem =zeros(2*n-4,1);
```

8. Assemble local matrices into the global

```
function A=assemblymatrix(Nx,Ny,x0,xf,y0,yf,u0,v0,Re)
vnd=(Nx+1)*(Ny+1);
mnd=(Nx+1)*Ny+Nx*(Ny+1);
nd=vnd+mnd;           %velocity nodes
neq=2*nd+vnd;          % total number of degree of freedom
celem=connective(Nx,Ny);
elem4=[celem(:,9:12)];
elem8=[celem(:,1:8)];
node=node48(Nx,Ny,x0,xf,y0,yf);
gdof=fungdof(Nx,Ny,x0,xf,y0,yf);
n=2*length(celem(1,:))-length(elem4(1,:));
ne=length(celem(:,1));%number of elements
```

```
A=zeros(neq,neq);
% invel=zeros(neq,1);
% invel(1:nd)=1;
% v0=invel(nd+vnd+1:neq);
for e=1:ne
   Aelem=fluidelemmatrix(node(elem8(e,:),:),...
       node(elem4(e,:),:),u0(elem8(e,:),:),v0(elem8(e,:),:),Re);
   for i = 1:n
    for j = 1:n
     A(gdof(e,i),gdof(e,j))= A(gdof(e,i),gdof(e,j))+Aelem(i,j);
        end
    end
end
```

9. Assemble local vectors into the global

```
function b=assemblyfluidvector(Nx,Ny,x0,xf,y0,yf,Re)
vnd=(Nx+1)*(Ny+1);
vnd=(Nx+1)*(Ny+1);
mnd=(Nx+1)*Ny+Nx*(Ny+1);
nd=vnd+mnd;
neq=2*nd+vnd;
celem=connective(Nx,Ny);
elem4=[celem(:,9:12)];
elem8=[celem(:,1:8)];
node=node48(Nx,Ny,x0,xf,y0,yf);
n=2*length(celem(1,:))-4;
ne=length(celem(:,1));
gdof=fungdof(Nx,Ny,x0,xf,y0,yf);
b=zeros(neq,1);
for e = 1:ne
    belem=localfluidvector(celem);
    for i = 1:n
        b(gdof(e,i))= b(gdof(e,i))+belem(i);
    end
end
```

10. Identify boundary nodes

```
function bdof=funbdof(Nx,Ny)
nb=10*(Nx+Ny);
bdof=zeros(nb,1);
for k=1:2*Nx+2
    bdof(k)=k;
end
for k=(2*Nx+3):2:(2*Nx+3)+4*(Ny-1)

    if mod(Nx,2)==1
    if mod((k-1)/2,2)==0
        bdof(k)=bdof(k-2)+Nx+1;
    else
        bdof(k)=bdof(k-2)+2*Nx+1;
    end
    bdof(k+1)=bdof(k)+1;
    else
      if mod((k-1)/2,2)==1
        bdof(k)=bdof(k-2)+Nx+1;
    else
        bdof(k)=bdof(k-2)+2*Nx+1;
    end
    bdof(k+1)=bdof(k)+1;
    end
end
for k=2*Nx+1+4*Ny-2:4*(Nx+Ny)
    bdof(k)=Ny*(3*Nx+2)-(2*Nx+1+4*Ny-2)+k;
end


for k=4*(Nx+Ny)+1:4*(Nx+Ny)+Nx+2
    bdof(k)= bdof(k-1)+1;
end
for k=(4*(Nx+Ny)+Nx+2+1):2:(4*(Nx+Ny)+Nx+2+1+2*(Ny-2))
    bdof(k)= bdof(k-2)+Nx+1;
end

for k=(4*(Nx+Ny)+Nx+2+2):2:(4*(Nx+Ny)+Nx+2+1+2*(Ny-2))
    bdof(k)= bdof(k-1)+1;
end


 for k=((4*(Nx+Ny)+Nx+2+1+2*(Ny-2))+1):((4*(Nx+Ny)+Nx+2+1+2*(Ny-2))+1)+Nx
     bdof(k)=bdof(k-1)+1;
 end
bdof(((4*(Nx+Ny)+Nx+2+1+2*(Ny-2))+1)+Nx+1:10*(Nx+Ny))= bdof(1:4*(Nx+Ny))+...
    bdof(((4*(Nx+Ny)+Nx+2+1+2*(Ny-2))+1)+Nx);
```