Design and object-oriented implementation of a
preconditioned-stabilized incompressible
NavierStokes solver using
equal-order-interpolation velocity pressure
elements

Rajiv Sampath and Nicholas Zabaras
Sibley School of Mechanical and Aerospace Engineering
188 Frank H. T. Rhodes Hall
Cornell University
Ithaca, NY 14853-3801
E-mail: zabaras@cornell.edu, rs78@cornell.edu

July 27, 2000
1 Introduction

In incompressible flow finite element computations there are two fundamental sources of instabilities. One is due to the presence of convection terms and can result in node to node oscillations in the velocity field. Such oscillations become more significant for high Reynolds number flows. The other source of instability is produced if an inappropriate combination of interpolation functions for the velocity and pressure is used. These instabilities appear as oscillations in the pressure field.

Tezduyar et. al [1] introduced stabilized finite element formulations using equal-order-interpolation velocity pressure elements for computation of steady and unsteady incompressible flows. It was demonstrated that stabilization is achieved by adding two terms to the standard Galerkin formulation of the problem. The first term is the well known SUPG (streamline-upwind/Petrov-Galerkin) term, which has been applied quite successfully to the simulation of various incompressible flow problems for more than a decade [2]. The second stabilization term is the PSPG (pressure stabilizing/Petrov-Galerkin) term, which was introduced by Tezduyar et al. to accommodate equal-order-interpolation velocity-pressure elements. This numerical approach has proved to be very effective in simulating a number of incompressible flow problems of engineering importance. This numerical methodology is adopted in the current work.

The outline of the rest of this report is as follows: In Section 2, the governing equations of unsteady incompressible flow problems are reviewed. The review of the SUPG and PSPG stabilized incompressible flow formulation is presented in Section 3. Section 4 deals with preconditioning strategies and efficient solution of the discretized finite element equations and also introduces the preconditioned BiCGStab (bi-conjugate stabilized conjugate gradient method) method used in the present implementation. This section also presents brief ideas on the object-oriented implementation. Section 5 presents various numerical experiments and results. Finally, concluding remarks are given in Section 6.

2 The governing equations

Let $\Omega_t \subset \mathbb{R}^{n_d}$ be the spatial domain at time $t \in (0,T)$, where $n_d$ is the number of space dimensions. Let $\Gamma_t$ denote the boundary of $\Omega_t$. We consider the following velocity-pressure formulation of the Navier-Stokes equations governing unsteady incompressible flows:

$$
\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = -\nabla p + \mu \nabla^2 \mathbf{v} \quad \text{on} \quad \Omega_t \quad \forall t \in (0,T),
$$

$$
\nabla \cdot \mathbf{v} = 0 \quad \text{on} \quad \Omega_t \quad \forall t \in (0,T),
$$

where $\rho$ and $\mathbf{v}$ are the density and velocity, and $\sigma$ is the stress tensor given as

$$
\sigma(p,\mathbf{v}) = -p \mathbf{I} + 2\mu \mathbf{v} \quad \text{with} \quad \mu \mathbf{v} = \frac{1}{2} [\nabla \mathbf{v} + (\nabla \mathbf{v})^T]
$$

Here, $p$ and $\mu$ are the pressure and the dynamic viscosity, and $\mathbf{I}$ is the identity tensor. The part of the boundary at which velocity is assumed to be specified is denoted by $(\Gamma_t)_g$:

$$
\mathbf{v} = \mathbf{g} \quad \text{on} \quad (\Gamma_t)_g \quad \forall t \in (0,T)
$$

The “natural” boundary conditions associated with Eq. (1) are the conditions on the stress components, and these are the conditions assumed to be imposed at the remaining part of the boundary:

$$
\mathbf{n} \cdot \sigma = \mathbf{h} \quad \text{on} \quad (\Gamma_t)_b \quad \forall t \in (0,T)
$$
The homogeneous version of Eq. (6), which corresponds to the “traction free” (i.e., zero normal and shear stress) conditions, is often imposed at the outflow boundaries. As initial condition, a divergence-free velocity field $\mathbf{v}_0(x)$ is specified over the domain $\Omega_0$ at $t = 0$:

$$\mathbf{v}(x, 0) = \mathbf{v}_0(x) \quad \text{on} \quad \Omega_0$$  \hspace{1cm} (7)

### 3 Incompressible flow formulations with SUPG and PSPG stabilizations

In this section, the variational formulation with the SUPG and PSPG stabilization terms are described. These formulations are based on finite element discretization in space only.

Let us discretize the domain $\Omega$ by subdividing it into elements $\Omega^e, e = 1, 2, 3, \ldots, n_{el}$, where $n_{el}$ is the number of elements. Associated with this discretization, we define the following finite element interpolation function spaces for the velocity and pressure:

$$S_h^v = \{ \mathbf{v}^h | \mathbf{v}^h \in [H^{1b}(\Omega)]^{n_{el}}, \mathbf{v}^h \approx \mathbf{v}^h_{on \Gamma_\delta} \},$$  \hspace{1cm} (8)

$$V_h^v = \{ \mathbf{w}^h | \mathbf{w}^h \in [H^{1b}(\Omega)]^{n_{el}}, \mathbf{w}^h \approx \mathbf{0}^h_{on \Gamma_\delta} \},$$  \hspace{1cm} (9)

$$S_h^p = V_h^p = \{ q^h | q^h \in [H^{1b}(\Omega)] \},$$  \hspace{1cm} (10)

where $H^{1b}(\Omega)$ represents the finite-dimensional function space over the spatial domain $\Omega$. This space is formed by using, over the element domains, first-order polynomials in space. The stabilized Galerkin formulation of (1)–(7) can be written as follows: Find $\mathbf{v}^h \in S_h^v$ and $p^h \in S_h^p$ such that, $\forall \mathbf{w}^h \in V_h^v$ and $\forall q^h \in V_h^p$,

$$\int_{\Omega} \mathbf{w}^h : \rho \left( \frac{\partial \mathbf{v}^h}{\partial t} + \mathbf{v}^h \cdot \nabla \mathbf{v}^h \right) d\Omega + \int_{\Omega} \epsilon(\mathbf{w}^h) : \sigma(p^h, \mathbf{v}^h) d\Omega - \int_{\Gamma_\delta} \mathbf{w}^h : \mathbf{h} + \int_{\Omega} q^h \nabla \cdot \mathbf{v}^h d\Omega + \sum_{e=1}^{n_{el}} \int_{\Omega^e} \left( \delta^h + \epsilon^h \right) \cdot \left[ \rho \left( \frac{\partial \mathbf{v}^h}{\partial t} + \mathbf{v}^h \cdot \nabla \mathbf{v}^h \right) - \nabla \cdot \sigma(p^h, \mathbf{v}^h) \right] d\Omega = 0$$  \hspace{1cm} (11)

As it can be seen from Eq. (11), two stabilizing terms have been added to the standard Galerkin formulation of (1)–(7); the one with $\delta^h$ is the SUPG term, and the one with $\epsilon^h$ is the PSPG (pressure-stabilizing/Petrov-Galerkin) term. The Petrov-Galerkin functions $\delta^h$ and $\epsilon^h$ are defined as

$$\delta^h = \tau_{SUPG} \mathbf{w}^h \cdot \nabla \mathbf{v}^h,$$  \hspace{1cm} (12)

$$\epsilon^h = \tau_{PSPG} \frac{1}{\rho} \nabla q^h,$$  \hspace{1cm} (13)

where

$$\tau_{SUPG} = \frac{h}{2 \| \mathbf{v}^h \|} z(Re \rho),$$

$$\tau_{PSPG} = \frac{h^\#}{2 \| \mathbf{V} \|} z(Re \#).$$  \hspace{1cm} (14)

(15)
Here, $Re_v$ and $Re^\#_V$ are the element Reynolds numbers, which are based, respectively, on the local velocity $v^b$ and a global scaling velocity $V$. That is,

$$Re_v = \frac{\| v^b \|}{2\nu} \quad (16)$$

$$Re^\#_V = \frac{\| V \|}{2\nu} \quad (17)$$

The “element length” $h$ is computed by using the expression

$$h = 2 \left( \sum_{\alpha=1}^{n_{el}} \left| s \cdot \nabla N^e_\alpha \right| \right)^{-1}, \quad (18)$$

where $n_{el}$ is the number of nodes in the element, $N^e_\alpha$ is the basis function associated with the local node $\alpha$, and $s$ is the unit vector in the direction of the local velocity. The “element length” $h^\#$, on the other hand, is defined to be equal to the diameter of the circle which is area-equivalent to the element. The function $z(Re)$ used in Eqs. (14) and (15) is defined as

$$z(Re) = \begin{cases} Re/3, & 0 \leq Re \leq 3, \\ 1, & 3 \leq Re \end{cases} \quad (19)$$

The spatial discretization of Eq. (11) leads to the following set of nonlinear differential equations,

$$[M + M_s] a + [N(v) + N_e(v)] v + [K + K_s] v - [G + G_s] p = F + F_\delta,$$  

$$G^T v + M_a + N_e(v) v + K_e v + G_e p = E + E_\delta,$$  

where $v$ is the vector of unknown nodal values of $v^b$, $a$ is the time derivative of $v$, and $p$ is the vector of nodal values of $p^b$. The matrices $M, N(v), K$ and $G$ are derived, respectively, from the time-dependent, advective, viscous, and pressure terms. The vectors $F$ and $E$ are due to boundary contributions. The subscripts $\delta$ and $\epsilon$ identify the SUPG and PSPG contributions, respectively. The various matrices forming the discrete finite element equations are described below, where $\hat{p}$ refers to the global velocity degree of freedom in the final system of equations that corresponds to the $i^{th}$ velocity degree of freedom of the elemental node $\alpha$ ($i = 1, ..., n_{el}$) in the $e^{th}$ element. Similarly, one can define $\hat{q}$. Here, $\delta^e_\alpha$ denotes the SUPG contribution and $\epsilon^e_\alpha$ denotes the PSPG contribution in the $i^{th}$ direction.

$$[M + M_e]_{\hat{q} \hat{q}} = \rho \sum_{\alpha=1}^{n_{el}} \left\{ \int_{\Omega_e} (N^e_\alpha + \delta^e_\alpha) N^e_{\beta,i} d\Omega \right\} , \quad (22)$$

$$[N(v) + N_e(v)]_{\hat{q} \hat{q}} = \rho \sum_{\alpha=1}^{n_{el}} \left\{ \int_{\Omega_e} (N^e_\alpha + \delta^e_\alpha) v^e_k N^e_{\beta,i} d\Omega \right\} , \quad (23)$$

$$[G + G_s]_{\hat{q} q} = \left[ \int_{\Omega_e} N^e_{\alpha,i} N^e_{\beta,i} d\Omega \right] - \left[ \int_{\Omega_e} \delta^e_\alpha N^e_{\beta,i} d\Omega \right] \quad (24)$$

$$[K + K_s]_{\hat{q} \hat{q}} = \left[ \int_{\Omega_e} N^e_{\alpha,i} N^e_{\beta,i} d\Omega \right] + \left[ \int_{\Omega_e} \delta^e_\alpha N^e_{\beta,i} d\Omega \right]$$

$$- \left\{ \sum_{\alpha=1}^{n_{el}} \left[ \int_{\Omega_e} \mu \delta^e_\alpha N^e_{\beta,i} d\Omega \right] \right\} \quad (25)$$

$$- \left\{ \sum_{\alpha=1}^{n_{el}} \left[ \mu \delta^e_\alpha N^e_{\beta,i} d\Omega \right] \right\}$$

\[ 
- \left\{ \sum_{\alpha=1}^{n_{el}} \left[ \mu \delta^e_\alpha N^e_{\beta,i} d\Omega \right] \right\} 
\]
Next, consider the time-integration of Eqs. (20) and (21) by a one step generalized trapezoidal rule; i.e., given \((v^n_t)_n\) and \((p^n_b)_n\), find \((v^{n+1})_n\) and \((p^{n+1})_n\) (T1 formulation). When written in an incremental form, the T1 formulation leads to

\[
\begin{align*}
M^* \Delta a - G^* \Delta p &= R \\
(G^T)^* \Delta a - G_\tau^* \Delta p &= Q
\end{align*}
\]  

where

\[
\begin{align*}
R &= F + F_\delta - \left[ (M + M_b) a + N(v) + N_t(v) + (K + K_b) v - (G + G_t) p \right], \\
Q &= E + E_\delta - \left[ (G^T) v + M_t a + N_t(v) v + (K_t + K\delta) v - (G + G_t) p \right], \\
M^* &= M + M_b + \gamma \Delta t [N(v) + N_t(v) + K + K_b], \\
G^* &= G + G_t, \\
(G^T)^* &= M_t + \gamma \Delta t (N_t + K_t + G^T).
\end{align*}
\]

The parameter \(\gamma\) controls the stability and accuracy of the time integration algorithm.

**Remark 1:** The above formulation has been tested with only bilinear and trilinear elements. In these cases the contributions at the element interiors from the viscous term is neglected, i.e., \(K_t\) and \(K_\delta\) are neglected. This avoids the calculation of second order derivatives. A numerical experiment including the second order derivatives was done to validate our assumption. The results confirmed that the terms \(K_t\) and \(K_\delta\) could be neglected in such cases.

**Remark 2:** The system (31) and (32) are solved by treating the velocity explicitly. The results are typically obtained with two passes per time step. In such computations, \(M^*, G^*\) and \((G^T)^*\) are replaced with

\[
\begin{align*}
M^* &= M_L, \\
G^* &= G, \\
(G^T)^* &= \gamma \Delta t G^T,
\end{align*}
\]

where \(M_L\) is the lumped version of the mass matrix \(M\).
4 Preconditioning strategies for efficient solution of the discretized FE equations and details on object oriented implementation

4.1 Efficient solution of FE discretized equations and preconditioning strategies

The system of Eqs. (31) and (32) are greatly simplified due to the replacement described in Eqs. (38-40). This is due to the fact that the matrices $M^*$, $G^*$ and $(G^T)^*$ are independent of the actual convective velocity and hence are constant for all times (if time step is constant) in the case of fixed domain problems. Thus, for fixed domain problems, the matrices can be formed and LU-factorized once and for all, before entering the loops of time step and iterations. This improves the computational efficiency tremendously.

In the case of problems involving a deforming domain, as for example a solidification problem, the problem doesn’t involve constant matrices. Hence, if a direct solver is used, one would require to LU-factorize the matrices at every time step. This would be computationally taxing for moderately large problems in 2D and even small problems in 3D. This necessitates the use of iterative solvers.

In this work we employ a stabilized bi-conjugate gradients (BiCGSTAB) method for the solution of the non-symmetric system of linear equations ($Ax = b$, $A$ non-symmetric). The algorithm is listed in Box I for completeness. However, as is well known, iterative methods have a very slow convergence rate for problems having a very high condition number. This implies that, to obtain a satisfactory performance, some preconditioning strategies needs to employed. Many effective preconditioners are currently available in literature (see for e.g. [6], [4], [7]) However, even the best preconditioner for a general linear system cannot match the performance of a preconditioner specific to a particular system. In the present context, this approach is used. In most problems involving fluid flow of interest to us (such as solidification problems) the domain changes very slowly (since we typically use small time steps). This means that the stiffness matrix at the previous time step is very close to the one at the current step. Hence LU-decomposition at the previous step is very close to the LU decomposition at the current step. This idea is used in the development of the LU-preconditioned BiCGSTAB method elucidated below.

**Box I**

Stabilized bi-conjugate gradients algorithm (BiCGSTAB)

- Given $x_0$, compute $r_0 = b - Ax_0$ and set $p_0 = r_0$
- Choose $r_k = \frac{(r_{k-1}, r_{k-1})}{(Ap_{k-1}, p_{k-1})}$ such that $(r_k, r_k) \neq 0$. For $k = 1, 2, ...$
- Compute $Ap_k$
- Set $x_{k-1/2} = x_{k-1} + \frac{1}{\alpha_{k-1}} p_{k-1}$, where $\alpha_{k-1} = \frac{(r_k, r_k)}{(Ap_k, p_k)}$
- Compute $r_{k-1/2} = r_{k-1} - \frac{1}{\alpha_{k-1}} Ap_{k-1}$
- Compute $\beta_k = \frac{(r_k, r_k)}{(r_{k-1/2}, r_{k-1/2})}$
- Set $x_k = x_{k-1/2} + \beta_k r_{k-1/2}$, where $\omega_k = \frac{(r_k, r_k)}{(Ap_{k-1}, Ap_{k-1})}$.
- Compute $r_k = r_{k-1/2} - \omega_k Ap_{k-1}$
- Compute $p_k = r_k + h_k (p_{k-1} - \omega_k Ap_{k-1})$, where $h_k = \frac{\alpha_{k-1} (r_k, r_k)}{(r_{k-1}, r_{k-1})}$.

Let us say that the system we are interested to solve is represented by:

$$Ax = b$$

(41)

Let us say we have an approximation to the 'stiffness matrix' $\hat{A} = \hat{L}\hat{U}$ which is LU-factorized as shown. This LU-decomposition could be formed after a periodic interval in the whole simulation, which ensures that
$\tilde{A} \approx A$. The period after which the LU-factorization is performed is a compromise between computational cost and improving the condition number $\kappa(A) = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}$ of the resulting system.

Now, using the approximate stiffness matrix $\tilde{A}$ as a preconditioner, we form the preconditioned system

$$\tilde{A}\tilde{x} = \tilde{b}$$

(42)

where,

$$\tilde{A} = \tilde{U}^{-1}\tilde{L}^{-1}A$$

(43)

$$\tilde{x} = x$$

(44)

$$\tilde{b} = \tilde{U}^{-1}\tilde{L}^{-1}b$$

(45)

Now, this preconditioned system is solved using the BICGSTAB. However, as it is the algorithm cannot be used as this requires computing the inverse of $L$ and $U$ which would be computationally taxing. Moreover this approach would also require matrix-matrix multiplication which would be computationally taxing. In order, to simplify the computation and hence the flop count, we have simplified the solution of the preconditioned system to the algorithm shown in Box II. This algorithm has been obtained by substituting the preconditioned system into the algorithm described in Box I and simplifying the computations. A slight variant of this algorithm is available in [12]

**Box II**

Preconditioned stabilized bi-conjugate gradients algorithm (BICGSTAB)

- Given $x_o$, compute $r_o = b - Ax_o$
- Choose $F_o$ such that $\langle F_o, r_o \rangle \neq 0$, e.g. $F_o = r_o$
- Set $\rho_o = \alpha = \omega = 1$
- Set $\nu_o = p_o = 0$
- For $k = 1, 2, ...,$
  - $\rho_k = \langle F_o, r_{k-1} \rangle$
  - $\beta = \rho_k \alpha / (\rho_{k-1} \omega_{k-1})$
  - $p_k = r_{k-1} + \beta (p_{k-1} - \omega_{k-1} \nu_{k-1})$
  - Solve $\tilde{L} \tilde{U}y = p_k$
  - $\nu_k = A y$
  - $\alpha = \rho_k / \langle F_o, \nu_k \rangle$
  - $s = r_{k-1} - \alpha \nu_k$
  - Solve $\tilde{L} \tilde{U}z = s$
  - $t = Az$
  - $\omega_k = \langle t, s \rangle / \langle t, t \rangle$
  - $x_k = x_{k-1} + \alpha y + \omega_k z$
  - $r_k = s - \omega_k t$
4.2 Object-oriented design

The various algorithms detailed earlier are coded using the \textit{diffpack} [8] library of C++ routines. The SUPG and PSPG weighting functions are independent of the weighted residual formulation and hence would need to be separate from the main class. These functions are encoded into a class ‘TezduyarUpwindFE’. The main members of this class are provided in Box III.

\textit{Box III}

\textbf{Main members of the class ‘TezduyarUpwindFE’ defined to manage the SUPG and PSPG functions}

class TezduyarUpwindFE {
private:

Vec(real) SUPG_weight_func; // discontinuous SUPG weighting function
Mat(real) PSPG_weight_func; // discontinuous PSPG weighting function
real h; // "element length" h
real h_hash; // "element length" h#
real tan_SUPG; // scalar parameter in SUPG weights
real tan_PSPG; // scalar parameter in PSPG weights

protected:

void calcSUPGPrm
()
FiniteElement& fe,
const Ptv(real)& u, // local velocity
real nu; // viscosity

void calcPSPGPrm
()
FiniteElement& fe,
const Ptv(real)& U, // global scaling velocity
real nu; // viscosity

public:

void calcSUPGWeightingFunction
()
FiniteElement& fe,
const Ptv(real)& u, // local velocity
real nu; // viscosity

void calcPSPGWeightingFunction
()
FiniteElement& fe,
const Ptv(real)& U, // global scaling velocity
real rho, // density
real nu; // viscosity

real SUPG (int i) { return SUPG_weight_func(i); }
real PSPG (int i, int j) { return PSPG_weight_func(i,j); }
};
The discretized finite element equations are cast in the form of a system of linear equations which can be solved either using a direct solver (e.g. GaussElimination) or using preconditioned iterative solvers. A class ‘LinearSolver’ is developed which provides for making and solving linear systems in the nonlinear solution process. This class provides for two different solution methods. One using a direct banded GaussElimination and second using a preconditioned BiCGStab method. The preconditioning matrix is LU-factorized and passed on as input to the preconditioned solver. The LU-factorization is done at periodic intervals specified by the user. The class is so designed that it can easily accommodate new solvers if needed. The main details of this class are provided in Box IV.

Box IV
Main members of the class ‘LinearSolver’ defined to make and solve linear systems

```cpp
class LinearSolver: public FEM {
    protected:
        Handle<GridFE> grid; // finite element mesh
        Handle<DegFreeFE> dof; // matrix dof => u, p dof
        Handle<MatBand<real>> A_assembled; // assembled stiffness matrix
        Handle<Vec<real>> b_assembled; // assembled force vector
        Handle<Vec<real>> linear_solution; // solution of linear subsystem
        MatBand<real> precondMatrix; // LU factorized preconditioning matrix
        Boolean direct_solver; // whether to use direct or iterative solvers
        PrecondBiCGStab precond_solver; // preconditioned iterative solver
        CPUclock cpu_solve; // calculates cpu time
        double cpu_time_solve; // cpu time to solve linear system

    public:
        virtual void define(MenuSystem& menu, int level = MAIN);
        virtual void scan(MenuSystem& menu);

    protected:
        virtual void makeAndSolveLinearSystem (Boolean A_has_changed,
                                               Boolean calcPreconditioner);
        virtual void calcPreconditioners ();//
        virtual void makeSystem (DegFreeFE & dof, Matrix(NUMT)& mt,
                                  Vector(NUMT)& vc, Boolean matrix_has_changed);
        virtual void cpuSolveTime(); // gets cpu time and writes report
}
```

Finally, the main routines dealing with the various field values \( u, p \) are coded in a class ‘StabNavierStokes’. This class is derived from the generic class ‘LinearSolver’ so that it can retrieve information about the grid and linear systems. This class basically provides for the main time loop and the nonlinear solution process. This class provides the problem specific ‘integrands’ routines. The main members of this class are shown in Box V.
5 Numerical experiments and results

In this section, numerical examples are presented for testing the SUPG/PSPG stabilized NavierStokes solver developed for flow simulation at low and medium Reynolds number. Four numerical examples are considered. We consider the lid-driven cavity problem at Re = 400 as a first example. This problem has been investigated several times earlier. The lid-driven cavity problem involves singularities in the pressure field and, therefore, is regarded as a stringent test case. The next example that we consider, is the classic flow past a circular cylinder problem. This problem has also been investigated very many times and is considered as a benchmark problem. Next, we consider 3D lid-driven cavity problem at Re = 100 and finally the problem flow past a tube bundle. In all computations, either bilinear or trilinear velocity-pressure elements are used depending on the
dimension of the problem. In all examples, a banded Gauss elimination technique without LU-factorization (see subsection on computational issues) was used for solving linear systems.

5.1 The lid-driven cavity flow at Reynolds number 400

In this problem the lid of the cavity has unit velocity; based on this velocity and the dimensions of the cavity the Reynolds number is 400. We choose a uniform grid of 64 × 64 Q1Q1(bilinear velocity-pressure element) elements. The time step was chosen to be 0.02 for most of the duration of the simulation with smaller steps at early times.

As initial condition, we imposed a zero velocity everywhere. The problem was run up to a nondimensional time τ = 20. Steady state was achieved around τ = 15. The velocity components along the vertical and horizontal center lines, pressure along the vertical centerline and pressure and stream function contours are shown in Fig. 1. The results are compared with those of Tezduyar et al. in [3]. The results compare very well. The maximum stream function value in the present simulation is seen to be 0.113 while Tezduyar et al. report 0.115 for the same finite element discretization. The mid-plane velocity profiles reported in [3] are in excellent agreement with the present results.

5.2 Unsteady flow past a circular cylinder at Reynolds number 100

The problem consists of a circular cylinder immersed in a flowing viscous liquid. At Reynolds number below 40, a pair of symmetrical eddies form on the downstream side of the cylinder. At higher Reynolds numbers, the symmetrical eddies become unstable and periodic vortex shedding occurs. The eddies are transported downstream, resulting in the well-known Karman vortex street.

This problem is of engineering interest and has been studied extensively by various authors [1, 2, 9].

A Reynolds number of 100 is considered to be the standard for testing FEM algorithms on the cylinder problem. It is high enough for vortex shedding to occur, but low enough that the boundary layers can be easily resolved.

The domain and boundary conditions along with the finite element mesh are shown in Fig. 2. The Reynolds number based on the inlet velocity and the cylinder diameter is 100. The mesh has been graded to efficiently resolve the developing boundary layer. In the downstream region, elements are sized to capture the vortex street. The time step used is 0.03.

The initial condition was that of zero velocity everywhere except at the boundary nodes. This causes a initial discontinuity but the first time step produces a smooth velocity field which becomes the effective initial condition.

The problem was run a total of 4800 time steps, corresponding to 144 time units. Initially, a pair of symmetric attached eddies grew behind the cylinder, reaching a steady state by about t = 36. Streamlines, stationary streamlines and pressure contours corresponding to this state are shown in Fig. 3.

The results after 1800 times steps showed growing oscillations which finally lead to vortex shedding (see Fig. 4). Steady periodic shedding was achieved after about 6 shedding cycles (see Fig. 5). The observed shedding period, τ, was 6 time units (200 time steps), giving a dimensionless shedding frequency, or Strouhal number (S = D/uoτ), of 0.167. This compares well with the results in [2, 5].
Figure 1: Steady state solution for the lid-driven cavity flow at Reynolds number 400: velocity components along the vertical and horizontal center lines, pressure along vertical centerline, pressure and stream function contours at $\tau = 20$. 
Figure 2: Flow past a circular cylinder: problem domain, boundary conditions and finite element mesh
Figure 3: “Steady-state” solution for flow past a cylinder at Reynolds number 100: stream function, stationary stream function and pressure contours at \( \tau = 45 \).
Figure 4: Developing vortex shedding: streamlines
Figure 5: Periodic solution for flow past a circular cylinder at Reynolds number 100: stream function, stationary stream function and pressure contours at $\tau = 144$. 
5.3 3D lid-driven cavity flow at Reynolds number 100

This example is basically an extension of the first. The top lid of the cubical cavity has a prescribed unit velocity. Based on this velocity the Reynolds number is 100. Fig. 6 shows the problem domain and boundary conditions. Using the symmetry about the midplane only half of the domain is modeled using $8 \times 16 \times 16$ trilinear elements (8-noded brick element). The non-dimensional time step was 0.01 for almost the entire simulation time except at early times ($\tau < 0.2$) when a time step of 0.002 was used. The entire simulation extended over a non-dimensional time of 12.0 when the flow had almost reached steady-state.

The variation of the horizontal velocity profile at $x = 0.5, y = 0.5$ as a function of the z-coordinate is shown in Fig. 7 and the midplane pressure distribution ($y = 0.5$) is shown in Fig. 8. The horizontal velocity profile along the central axis compares very well with the results reported in [10].

5.4 Viscous flow around a tube-bundle at Re = 500

This final example was obtained from [11]. The problem consists of a viscous fluid flowing through a equally spaced bundle of cylindrical tubes. Due to the symmetry, only a part of the domain is modeled as shown in Fig. 9. The finite element mesh shown in the same figure consists of 275 bilinear-elements and 336 nodes. The non dimensional time step was chosen to be 0.01. The flow was simulated until $\tau = 20$ by which time the flow had reached a steady state. The corresponding velocity vectors, streamlines and pressure contours are shown in Fig. 10. They compare well with results reported in [11].

5.5 Computational issues

In all problems solved above the domain is not deforming and hence the linear system to be solved at every iteration in every time step is constant. This fact helps us save a lot of computational time. The stiffness matrix can be LU factored once and for all at the very first iteration of the first time step and this factored form can be used again and again. This implies that at every iteration only forward and backward substitution
Figure 7: 3D lid-driven cavity problem: horizontal velocity profile at \((x = 0.5, y = 0.5)\) as function of z-coordinate.

Figure 8: 3D lid-driven cavity problem: mid-plane \((y = 0.5)\) pressure distribution.
Figure 9: Viscous flow around a ‘tube bundle’: finite element mesh and boundary conditions

Figure 10: Viscous flow around a ‘tube bundle’: velocity vectors, streamlines and pressure contours
components of Gauss elimination need to be executed. The corresponding CPU statistics is shown in Table I. In comparison to this, the preconditioned BiCGStab algorithm (see Box II) is computationally taxing. However, in problems where the domain deforms, the stiffness matrix would not be constant and hence would change at each time step. This would imply that the LU-factorization would have to be computed at every time step. This would imply tremendous increase in computational costs when a Gauss elimination procedure is used. However, if the domain is deforming slowly, then using a preconditioned BiCGStab algorithm would be a judicious choice. This is due to the fact that the preconditioner calculated periodically, would be a close approximation to the stiffness matrix at the current step and hence the condition number of the system would be very close to unity. Hence the BiCGStab algorithm would converge in 2-3 iterations. Table II compares the computational cost in using (i) a Gauss elimination procedure with LU decomposition (ii) a Gauss elimination procedure without LU decomposition (iii) a preconditioned BiCGStab algorithm for the various examples considered above. The comparison shows that in problems where the domain is not changing method (ii) would be the best choice while for the problems involving a deforming domain method (iii) would probably be a better choice.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Bandwidth</th>
<th>Number of dof</th>
<th>Number of time steps</th>
<th>Simulation time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lid driven cavity (2D)</td>
<td>67</td>
<td>12675</td>
<td>2160</td>
<td>170</td>
</tr>
<tr>
<td>flow past a circular cylinder</td>
<td>84</td>
<td>4530</td>
<td>4500</td>
<td>142</td>
</tr>
<tr>
<td>lid driven cavity (3D)</td>
<td>164</td>
<td>10404</td>
<td>1460</td>
<td>707</td>
</tr>
<tr>
<td>viscous flow past tube bundle</td>
<td>19</td>
<td>825</td>
<td>2000</td>
<td>13</td>
</tr>
</tbody>
</table>

**Table II**

Computational cost (secs) per time step for various linear solvers

<table>
<thead>
<tr>
<th>Problem</th>
<th>Gauss elimination (LU)</th>
<th>Gauss elimination (no LU)</th>
<th>Preconditioned BiCGStab</th>
</tr>
</thead>
<tbody>
<tr>
<td>lid driven cavity (2D)</td>
<td>53.56</td>
<td>0.98</td>
<td>2.82</td>
</tr>
<tr>
<td>flow past a circular cylinder</td>
<td>29.52</td>
<td>0.4</td>
<td>1.22</td>
</tr>
<tr>
<td>lid driven cavity (3D)</td>
<td>451.92</td>
<td>6.9</td>
<td>17.58</td>
</tr>
<tr>
<td>viscous flow past tube bundle</td>
<td>0.3</td>
<td>0.02</td>
<td>0.06</td>
</tr>
</tbody>
</table>

6 Conclusions

In this report, few ideas on developing a object-oriented NavierStokes solver based on equal-order-interpolation technique was presented. The method proposed by Tezduyar et. al was implemented in the framework of **diffpack**. A few ideas on efficient solution of FE discrete equations were elucidated and in particular the effectiveness of using a preconditioned BiCGStab algorithm was considered in detail. Four numerical examples were considered for testing the accuracy and correctness of the code. The results match very well with reported data. CPU comparisons were done and results showed that preconditioned methods would play a dominant role in deforming domain problems. Gauss elimination is the preferred choice in fixed domain problems.
References


