INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 17, 323–348 (1993)

SEGREGATED FINITE ELEMENT ALGORITHMS FOR THE NUMERICAL SOLUTION OF LARGE-SCALE INCOMPRESSIBLE FLOW PROBLEMS

VAHÉ HAROUTUNIAN, MICHAEL S. ENGELMAN AND ISAAC HASBANI
Fluid Dynamics International Inc., 500 Davis Street, Suite 600, Evanston, Illinois 60201, U.S.A.

SUMMARY

This paper presents results of an ongoing research program directed towards developing fast and efficient finite element solution algorithms for the simulation of large-scale flow problems. Two main steps were taken towards achieving this goal. The first step was to employ segregated solution schemes as opposed to the fully coupled solution approach traditionally used in many finite element solution algorithms. The second step was to replace the direct Gaussian elimination linear equation solvers used in the first step with iterative solvers of the conjugate gradient and conjugate residual type. The three segregated solution algorithms developed in step one are first presented and their integrity and relative performance demonstrated by way of a few examples. Next, the four types of iterative solvers (i.e. two options for solving the symmetric pressure type equations and two options for solving the non-symmetric advection–diffusion type equations resulting from the segregated algorithms) together with the two preconditioning strategies employed in our study are presented. Finally, using examples of practical relevance the paper documents the large gains which result in computational efficiency, over fully coupled solution algorithms, as each of the above two main steps are introduced. It is shown that these gains become increasingly more dramatic as the complexity and size of the problem is increased.

KEY WORDS Incompressible flow Finite element method Segregated solution algorithms Iterative solvers Implicit preconditioning

1. INTRODUCTION AND BACKGROUND

Historically, finite element CFD codes have employed fully coupled solution algorithms in conjunction with direct Gaussian elimination type solvers for the numerical solution of the incompressible Navier–Stokes equations. These algorithmic techniques were directly borrowed from the structural analysis community at the time when the finite element method (FEM) was finding its way to the CFD arena. This method of solution, while storage intensive, requires less CPU time compared to most other methods when applied to the solution of 2D (two-dimensional) laminar flow problems of moderate size. However, as the size and physical complexity of the flow problem is increased, the cost of the above solution techniques becomes prohibitively expensive, both in terms of storage and CPU time. This is especially true when large-scale 3D (three-dimensional) flow problems of industrial complexity are modelled. These serious limitations have led developers of finite element CFD codes to search for alternative algorithms based on segregated forms of solution and/or iterative (non-direct) linear equation solvers.

Luckily the search for algorithms based on the segregated form of solution has been relatively easy as it has been possible to borrow freely from the large database of knowledge and experience

0271–2091/93/160323–26$18.00

Received 25 September 1991
Revised 11 March 1993

accumulated by CFD practitioners using traditional finite difference and (in particular) finite volume methods. Given the limited storage capacities of computers in the early days of CFD, a combination of segregated and non-direct solution techniques provided the only viable route towards performing flow simulations of practical interest (these were predominantly 2D laminar flow problems on relatively coarse meshes). These algorithms were initially developed in the context of the finite difference method (see Reference 1–3) and were later adapted to the more general finite volume method[4, 5] (see also Reference 6 for a recent finite volume implementation of the MAC method of Harlow and Welsch[3]). Segregated solution algorithms have undergone many refinements since then and are now the preferred method of solution of most contemporary finite volume CFD codes. The SIMPLE algorithm[7, 8] and its many variants (SIMPLER, SIMPLEC, PISO, etc.) have emerged as being among the most successful algorithms in this area.

As in many other fields of non-linear computational mechanics, the search for robust iterative linear equation solvers has not been easy. It was obvious from the onset that the non-direct solvers employed by the finite volume community could not be used in finite element codes as these are based on line relaxation techniques (ADI schemes coupled with tri-diagonal solvers) and can therefore only be applied on structured meshes. (These solvers are now being slowly abandoned by those finite volume code developers who are adapting the finite volume method to unstructured meshes.)

The next obvious choice of iterative solvers are those derived from the family of conjugate gradient (CG) and conjugate residual (CR) methods; these solvers are also referred to as matrix-free solvers since at the implementation level the global coefficient matrix need not necessarily be stored in its entirety as is the case with direct solvers. As they preserve the stencil structure of the original coefficient matrix—no zero fill—these solvers have minimal storage requirements and can potentially be very fast. However, the problem is that the current state-of-the-art of these solvers has not advanced to the point where these potential gains in speed can be consistently realized over the wide range of matrix characters which are typically encountered in CFD. (There is intense research activity in this field and it is hoped that increasingly more robust and effective versions of these solvers will appear in the near future.)

While current iterative solvers are extremely efficient on simple well-behaved equation systems (i.e. equation systems with diagonally dominant, positive-definite and symmetric coefficient matrices), their performance deteriorates dramatically as the above-mentioned properties are lost from the coefficient matrix. The fully coupled linear equation systems resulting from the discretization by the Galerkin finite element method (GFEM) of the Navier–Stokes equations are particularly ill-suited to iterative linear equation type solvers as was demonstrated in an earlier study by Engelman and Hasbani[9] (see also Reference 10). This study suggested that the source of the difficulty was associated with the inclusion of the continuity constraint in the global coefficient matrix. On the one hand, a penalty approach leads to a non-symmetric definite ill-conditioned matrix with a large condition number, while on the other hand the mixed (u–p) approach leads to an indefinite non-symmetric system. Even the best iterative linear equation solvers with advanced preconditioners were found to be ineffective for these matrices. However, the study found that the same solvers could be used with significant success for the solution of two simpler types of equation systems which are typically encountered in segregated type solution algorithms; namely a Poisson type pressure equation and an advection–diffusion type equation for velocities and transported scalars such as temperature and chemical species. The clear conclusion emerging from that study was that the most promising route to pursue was to first formulate a segregated solution algorithm and then to solve the resulting equation systems with iterative linear equation solvers. This conclusion provided the impetus for the study reported herein.

The three segregated solution algorithms developed in the present study are presented in Section 2. Two of these algorithms, the pressure correction (PC) and pressure projection (PP) algorithms, are consistent finite element counterparts of the SIMPLE and SIMPLER algorithms, respectively. The third algorithm, the pressure update (PU) algorithm, is somewhat different in that the continuity equation is satisfied through penalizing the discretized continuity equation on the right-hand side of the discretized pressure equation. The integrity and relative performance of these algorithms are then demonstrated by way of a few numerical examples.

In section 3 the topic of replacing the direct Gaussian type linear equation solvers used in the above algorithms with iterative conjugate gradient and conjugate residual type solvers is considered. Four iterative linear equation solvers and the associated preconditioning strategies used in conjunction with these solvers are first introduced. Next, the relative performance of the various iterative solvers is studied. Finally, the performance of the PP algorithm using iterative solvers is compared both with that of the PP algorithm and a fully coupled algorithm using the direct Gaussian elimination solvers. It is seen that the combined strategy of segregated formulation and iterative linear equation solution leads to an improvement in performance which becomes increasingly more dramatic as the size and complexity of the problem increases.

The iterative solution algorithms presented in this paper have been incorporated in the incompressible finite element CFD code FIDAP (see Reference11). The various solution algorithms and linear equation solvers described in this paper may be optionally invoked by the user for the solution of a wide range of incompressible flow problems. All numerical tests presented in this report were performed with FIDAP on a Convex C220 computer.

2. THE SEGREGATED ALGORITHMIC APPROACH

2.1. Introduction

In a fully coupled formulation, the algebraic equations resulting from the discretization of the continuum flow equations are assembled (after applying an appropriate linearization scheme) and the resultant global equation system is solved for all the nodal unknowns in a simultaneous manner (in principle, any linear equation solver may be used). Various formulations of the continuum equations may be used but the primitive variables, i.e. the (u–p), formulation, is the one most commonly employed — this is especially true for 3D flows. Thus, every solution of the equation system simultaneously updates all the unknown nodal velocities and pressures and any other transported scalar variables which may be present in the problem — such as temperature and/or chemical concentrations. If an advanced turbulence model is being employed, such as the k–ε model, the simulation will involve the additional primary variables of turbulence kinetic energy k and its rate of viscous dissipation ε.

In a segregated formulation, the global discretized equation system is never assembled and solved in its entirety. Many variations are possible ranging from solving separately for each nodal unknown to solving simultaneously for all the nodal unknowns associated with one or more (but not all) of the primary flow variables. In this paper we consider segregated algorithms of the type where separate equation systems are assembled and solved for each of the primary flow variables.

2.2. The fully coupled system of flow equations

We will consider as our starting point the structure of the fully coupled global equation system resulting from the application of GFEM to the incompressible Navier–Stokes equations. We will consider only the primitive variables form of the equations together with the mixed formulation

and employ the fixed point Picard method as the non-linear solution strategy. Additionally, for transient problems, implicit time integration schemes (such as the backward Euler or trapezoidal schemes) will be used. Finally, for the sake of simplicity and brevity of exposition, only the 2D form of the laminar flow equations with an energy equation for heat transfer will be considered.

In GFEM the flow equations and their associated boundary conditions are not solved directly; rather these are replaced by an equivalent weak formulation which essentially satisfies the flow equations in a weighted residual sense. In conventional GFEM, the pressure gradient terms appearing in the momentum equations and diffusion terms appearing in the various conservation equations are integrated by parts. This lowers the order of these differential terms and allows lower order interpolation functions to be used for pressure and the other primary variables. In this context the lowest order of spatial approximation allowed for pressure is piecewise discontinuous, i.e. C^{-1} continuous. Correspondingly, the lowest order of spatial approximation for velocities, temperatures and other scalars present in the problem is piecewise linear or C^0 continuous. Integration by parts also leads to the so-called natural boundary integrals through which flux type boundary conditions are readily (and naturally) incorporated. Finally, an important stability criterion of the conventional GFEM approach is that certain compatibility conditions must be observed when selecting interpolation functions for pressure and velocities. Violation of this Babuska–Brezzi condition (or the inf–sup condition) typically leads to spurious pressure modes and in extreme cases a 'locking' of the velocity field. (There has been intense recent research activity in the FEM community in the development of stabilization schemes which allow the satisfaction of the Babuska–Brezzi condition for most velocity–pressure combinations. However, this very important topic is beyond the scope of the present paper — these techniques may readily be incorporated in the algorithms described herein.)

The structure of the global equation system that results from the application of GFEM to the incompressible flow equations is as follows:

$$\begin{bmatrix} K_{uu} & K_{uv} & -C_x & B_u \\ K_{vu} & K_{vv} & -C_y & B_v \\ C_x^{\mathrm{T}} & C_y^{\mathrm{T}} & 0 & 0 \\ 0 & 0 & 0 & K_\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ p \\ \theta \end{bmatrix} = \begin{bmatrix} F_u \\ F_v \\ b_c \\ f_\theta \end{bmatrix} \tag{1}$$

which represents the discretized algebraic equations resulting from the four conservation equations (i.e. the two momentum equations, the continuity equation and the energy equation). The vectors u, v, p and θ contain the nodal unknown velocities, pressures and temperatures, respectively. In the global coefficient matrix, the diagonal K sub-matrices represent the combined effects of advection and diffusion, and, in transient problems, the effects of the temporal terms. The off-diagonal K sub-matrices in the momentum equations represent coupling terms resulting from diffusion, and, in the case of rotating flows, those from Coriolis forces. In buoyant flows, the B matrices represent the components of the buoyancy force resulting from temperature variations (the Boussinesq approximation is employed for flows involving buoyancy). The C matrices are the pressure gradient operators and their transposes, the C^{T} matrices appearing in the continuity equation, are the velocity divergence operators. The vector b_c on the right-hand side of the continuity equation represents the contributions to this equation from the non-zero Dirichlet velocity boundary conditions. Finally, the right-hand-side vectors F and f contain surface-flux type contributions from the natural boundary conditions, and, in the case of transient flows, history effects from previous time levels.

If, instead of the Picard scheme a gradient type scheme is used for the non-linear terms (such as a Newton type scheme), a different global matrix system results which is similar in structure to

equation (1). The left-hand side global coefficient matrix is replaced by the global Jacobian matrix, the right-hand side force vectors are replaced by the residual vectors and the unknown nodal vectors of the primary variables are replaced by vectors containing the incremental change in the primary nodal variables from one non-linear iteration to the next.

In fully coupled flow algorithms the resultant linear equation systems at each iteration are solved simultaneously using the direct Gaussian elimination type solvers. As stated earlier iterative solvers perform very poorly on such equation systems mainly due to the zero entries appearing on the block diagonal position of the continuity equation.

2.3. The segregated system of flow equations

The global equation system of equation (1) may be decomposed into the following set of decoupled sub-equation systems for the two components of the momentum equation, for the continuity equation and for the energy equation, respectively, viz.

$$K_u u - C_x p = F_u - K_{uv} v - B_u \theta = f_u , \tag{2a}$$

$$K_v v - C_y p = F_v - K_{vu} u - B_v \theta = f_v , \tag{2b}$$

$$C_x^{\mathrm{T}} u + C_y^{\mathrm{T}} v = b_c , \tag{2c}$$

$$K_\theta \theta = f_\theta . \tag{2d}$$

In order to simplify notation, the double subscripts of the diagonal K sub-matrices in the momentum equations (2a) and (2b) have been replaced by the single subscripts (u) and (v), respectively.

The above system of equations forms the basis for the construction of segregated algorithms. However, while clearly decoupled, the form of these equations is not immediately amenable to a segregated solution approach as an explicit discretized equation for pressure is lacking. In principle, such an equation can be obtained by replacing the continuity equation (2c), which implicitly governs the pressure in incompressible flows, with the GFEM form of the consistent pressure equation. This is achieved by first left-multiplying the momentum equations by the inverses of their respective K matrices and then by left-multiplying again by the velocity divergence C^{T} matrices. The equation for pressure that results from adding the two manipulated momentum equations is as follows.

$$(C_x^{\mathrm{T}} K_u^{-1} C_x + C_y^{\mathrm{T}} K_v^{-1} C_y) p = - C_x^{\mathrm{T}} K_u^{-1} f_u - C_y^{\mathrm{T}} K_v^{-1} f_v + C_x^{\mathrm{T}} u + C_y^{\mathrm{T}} v - b_c . \tag{3a}$$

Note that the continuity equation (2c) appears on the right-hand side of equation (3a) and can therefore be set to zero. This enforcement of the discretized continuity constraint finally produces the consistent pressure equation viz.

$$(C_x^{\mathrm{T}} K_u^{-1} C_x + C_y^{\mathrm{T}} K_v^{-1} C_y) p = - C_x^{\mathrm{T}} K_u^{-1} f_u - C_y^{\mathrm{T}} K_v^{-1} f_v \tag{3b}$$

It is very important to note that equation (3b) has not been obtained directly from discretizing the continuum Poisson equation for pressure. Rather, it has been derived at the discretized level from manipulations of the discretized momentum and continuity equations. As a consequence the boundary conditions on pressure, implicitly implied in equation (2) (or equivalently equation (1)) through the boundary conditions on velocity, are automatically and consistently incorporated in equation (3b). Thus solving equations (2a) and (2b) with equation (2c) is equivalent to solving equations (2a) and (2b) with equation (3b). Equation (3b) has therefore been used to replace the continuity equation and in so doing serves as the desired explicit equation governing pressure. As will be seen shortly, the pressure matrix appearing on the left-hand side of equation (3b) is a full

matrix. In this paper we will refer to it as the full consistent pressure matrix (FCPM). Moreover, the pressure equation itself will be referred to as the full consistent pressure equation (FCPE).

The equations forming the basis for a segregated type solution are now equations (2a), (2b), (2d) and (3b). However, these are still rarely employed directly, as the FCPM in equation (3b) is prohibitively expensive to both construct and solve, not only because it is a full matrix, but also because its construction requires inverting the K_u and K_v matrices (the inverse matrices are full matrices even if the K matrices are banded). Moreover, for non-linear problems this matrix has to be constructed and solved once every iteration. Thus, in practice, segregated finite element algorithms employ alternative or simplified forms of the pressure equation which are cheaper to construct and solve. As the continuity equation (2c) can no longer be enforced directly on the right-hand side of the pressure equation (cf. equation (3a)), an alternative strategy of imposing the continuity constraint must also be devised. A large number of alternative approaches are possible, *ranging* from finite element counterparts of the well-established SIMPLE type and UZAWA type algorithms to others that are still under development. In a typical algorithm, the derived pressure equation is used to solve either directly for pressure p or indirectly for a pressure increment Δp.

In designing these algorithms there are strictly two categories of alternative or simplified pressure equations that one can employ; simplified consistent pressure equations (SCPEs) and non-consistent pressure equations (NCPEs).

Like the FCPE, SCPEs are obtained from direct manipulations of the discretized momentum and continuity equations. However, in their construction the K_u and K_v matrices are replaced by simpler matrices, say $\tilde{K}_u$ and $\tilde{K}_v$, that are easier to invert. The most common approach is to use diagonal matrices. The advantages of this are two fold: firstly, the inversion of the $\tilde{K}$ matrices is trivial, and secondly, the simplified consistent pressure matrix (SCPM) that results (i.e. the $[C_x^T \tilde{K}_u^{-1} C_x + C_y^T \tilde{K}_v^{-1} C_y]$ matrix) is a banded matrix which occupies less storage space and is less expensive to invert. Clearly the $\tilde{K}$ matrices must have the same matrix size as the corresponding K matrices. This means that if the original K matrices are of the size $(N \times N)$ then the $\tilde{K}$ matrices must be of size N. Apart from the above requirement there are, from a mathematical standpoint, no other restrictions on the exact nature of the $\tilde{K}$ matrices. Thus, strictly speaking, any such matrix, even the identity matrix may be used. However, in order to avoid slow rates of convergence, $\tilde{K}$ matrices should be used that are in some manner similar or close to the K matrices. A typical choice is to employ $\tilde{K}$ matrices which contain the diagonal entries of the original K matrices.

NCPEs on the other hand are not obtained from direct manipulations of the discretized momentum and continuity equations. The most common example of a NCPE is one which is obtained by directly discretizing the continuum ∇^2 operator via FEM. While SCPEs, and their associated pressure matrices SCPMs, are in general more difficult to construct compared to NCPEs, it is through the use of such equations that a segregated algorithm can exactly satisfy both the discretized continuity equation (2c) and the consistent boundary conditions on pressure as implied by equation (1). If use is made of NCPEs in a segregated algorithm it will not be possible to apply consistent boundary conditions on pressure and the continuity constraint as defined by equation (2c) will only be satisfied approximately (see References 12 and 13).

In section 2.4, we will outline the segregated algorithms which were developed in the present study. Before that a brief review is presented of various segregated finite element algorithms which have recently appeared in the literature.

As stated in Section 1, a clear trend has emerged in the finite element CFD community towards adopting segregated type solution algorithms and that most of these have been based on either the SIMPLE or UZAWA type algorithms. SIMPLE type finite element algorithms have been developed among others by Benim and Zinser,[14] Kim and Chung,[15] Rice and Schnipke,[16]

Nonino and Giudice,[17] and Shaw.[18] UZAWA type algorithms have been employed among others by Cahouet and Chabard,[19] Fortin and Fortin[20] and Fortin and Boivin.[21]

Of the SIMPLE type algorithms, those of Nonino and Giudice[17] and Kim and Chung[15] are of the transient kind, i.e. the temporal terms in the flow equations act as pivotal terms in the algorithm. Thus, even if only the steady-state solution is of interest, the flow equations need to be integrated in time to reach that state. This approach, while being acceptable (indeed necessary in cases where the final solution is either quasi-steady or dependent on the flow evolution history), may not be cost-effective for most problems where only the steady-state solution is of interest.

In general, it is typically more difficult to develop steady-state versions of segregated type algorithms since one can no longer make direct use of the time step Δt, which is a characteristic and natural time scale of the discretization. However, once such a steady-state algorithm is devised, its extension to a transient version is 'straightforward', as this can be implemented without changing the overall structure of the algorithm.

The algorithms of Benim and Zinser,[14] Rice and Schnipke[16] and Shaw[18] are of the steady-state type. Of these the algorithm of Benim and Zinser[14] is the most consistent finite element implementation (from the standpoint of conventional GFEM methodology) of the SIMPLE algorithm. They employ a SCPE for the pressure correction Δp and, in accordance with standard GFEM practice, use a pressure interpolation which is one order lower than the interpolation functions used for velocities.

The algorithms of Shaw[18] and Rice and Schnipke[16] are somewhat less orthodox in the sense that they are not based on the conventional weak form of the flow equations resulting from GFEM. Moreover, in their algorithms they use a NCPE for pressure (in the case of Rice and Schnipke) or pressure correction (in the case of Shaw). This approach has the very desirable advantage of precluding spurious pressure modes from the pressure solution which consequently allows them to employ the desirable feature of equal-order interpolation for velocities and pressures. However, these advantages are gained at the expense of compromising the exact satisfaction of equation (2c) and the proper imposition of pressure boundary conditions as described earlier.

The basic UZAWA algorithm (see Reference 2) is derived by considering the classical problem of minimizing the dual Lagrangian functional whose saddle points are characterized by the Stokes equations (i.e. the linear momentum and continuity equations). Many variations and extensions are possible both at the continuum and discretized levels. Cahouet and Chabard[19] used an iterative variant of the classical UZAWA algorithm to enforce continuity as part of an overall segregated time-dependent solution algorithm (see also Reference 22). Fortin and Fortin[20] and Fortin and Boivin[21] have used variations of the UZAWA algorithm which are based on minimizing the augmented Lagrangian functional.[23] The equations characterizing the saddle points of the augmented Lagrangian functional are the continuity equation plus a modified momentum equation which is augmented by a penalty term involving the continuity equation. This further enforcement of the continuity constraint has the advantage of accelerating the convergence of the algorithm. However, the accelerated rates of convergence are typically achieved with large values $0(10^6)$ of the associated penalty parameter. This not only introduces a strong coupling between the components of the momentum equations, but also leads to a coefficient matrix with a large condition number. As a result, augmented UZAWA algorithms are less amenable to iterative solvers and the momentum equations are typically solved in a fully coupled manner using direct solvers.

UZAWA algorithms also involve, directly or indirectly (depending on the details of implementation), the solution of discretized pressure equations. In the above described UZAWA algorithms use is made of consistent discretized pressure equations throughout so that the continuity constraint and the pressure boundary conditions are correctly applied.

2.4. The present segregated solution algorithms

The design of the segregated solution algorithms reported herein was guided by a series of primary objectives. The two most important of these were the requirements that the algorithms be applicable to both the transient and steady-state version of the flow equations, and that they be equivalent at the discretized level to the original system of discretized equations represented by equation (1), or its equivalent form, equation (2). This last requirement means that solving a given flow problem using the segregated solution algorithm should result in, within the accuracy of the user prescribed convergence tolerance, exactly the same solution as that obtained using a fully coupled algorithm on equation (1). Moreover, the philosophy underpinning the work was to avoid the introduction of any scheme or technique which would restrict the geometric flexibility of the finite element method. Additionally, element dependent techniques were also to be avoided so that use could be made of all types of elements available in the element library.

The three segregated solution algorithms developed in this study are presented in detail below.

2.4.1. The pressure correction (PC) algorithm. The PC algorithm is a direct finite element counterpart of the well-established SIMPLE algorithm. The SIMPLE algorithm and the steps leading to its derivation are discussed in detail by Patankar.[7,8] Here only a brief description of the algorithm is given.

In the SIMPLE algorithm the velocities and pressures are assumed to be decomposed into primary and corrective components. As convergence is approached, the corrective components vanish and the primary components asymptote towards the final solution. The primary components of velocity are obtained from the solution of the momentum equations using the latest pressures and velocities. In general, the updated velocities will not satisfy continuity and when substituted in the discretized continuity equation will produce a non-zero residual. This residual is used to drive an equation for the pressure correction Δp, which is obtained from manipulations of the discretized momentum and continuity equations. The pressure correction is consequently used to update the pressure and to obtain the correction velocities which mass-adjust the velocity field to satisfy continuity. As the above sequence is repeated and convergence is approached, the continuity equation is more closely satisfied which in turn leads to smaller pressure corrections and consequently smaller velocity corrections. At convergence the velocities and pressures simultaneously satisfy the discretized momentum and continuity equations. If other transported scalars (such as temperature) are also present in the flow problem, the discretized conservation equations governing these scalars are solved sequentially with the rest of the flow equations.

The implementation of our version of the SIMPLE algorithm may be summarized in the following algorithmic steps:

Given an initial or guess solution field $(u^0, v^0, p^0, \theta^0)$, for $i=0, 1, 2, 3, \ldots$ until convergence, the following steps should be taken:

(1) solve SCPE for pressure correction Δp

$$[C_x^{\mathrm{T}}(\tilde{K}_u^{-1})^* C_x + C_y^{\mathrm{T}}(\tilde{K}_v^{-1})^* C_y]\Delta p^{i+1/2} = -C_x^{\mathrm{T}}u^i - C_y^{\mathrm{T}}v^i + b_c; \tag{4a}$$

(2) mass-adjust velocity field and increment pressure via

$$u^{i+1/2} = u^i + (\tilde{K}_u^{-1})^* C_x \Delta p^{i+1/2}; \tag{4b}$$

$$v^{i+1/2} = v^i + (\tilde{K}_v^{-1})^* C_y \Delta p^{i+1/2}, \tag{4c}$$

$$p^{i+1} = p^i + (1-\alpha_p)\Delta p^{i+1/2}; \tag{4d}$$

(3) solve x-momentum equation for u

$$\left[\left(\frac{\alpha_u}{1-\alpha_u}\right)\tilde{K}_u + K_u\right]^* u^{i+1} = \hat{f}_u^* + C_x p^{i+1} + \left(\frac{\alpha_u}{1-\alpha_u}\right)\tilde{K}_u^* u^{i+1/2}; \tag{4e}$$

(4) solve y-momentum equation for v

$$\left[\left(\frac{\alpha_v}{1-\alpha_v}\right)\tilde{K}_v + K_v\right]^* v^{i+1} = f_v^* + C_y p^{i+1} + \left(\frac{\alpha_v}{1-\alpha_v}\right)\tilde{K}_v^* v^{i+1/2}; \tag{4f}$$

(5) solve energy equation for θ

$$\left[\left(\frac{\alpha_\theta}{1-\alpha_\theta}\right)\tilde{K}_\theta + K_\theta\right]^* \theta^{i+1} = f_\theta^* + \left(\frac{\alpha_\theta}{1-\alpha_\theta}\right)\tilde{K}_\theta^* \theta^i. \tag{4g}$$

In the above equations the superscripts i, $i+1/2$ and $i+1$ denote previous, intermediate and latest iterate levels, respectively, while the superscript * denotes an expression involving latest available field variables. The $\tilde{K}$ matrices are some convenient approximation to their corresponding K matrices. These are used both for the construction of the SCPM for Δp, and on either side of the momentum and energy equations to relax these equations. The exact definition of the $\tilde{K}$ matrices will be given in Remark 2.4. Finally, the α's appearing in the above equations are relaxation factors which assume values between 0 and 1; the former producing no relaxation and latter producing infinite relaxation (i.e. no change in solution).

2.4.2. The pressure projection (PP) algorithm. The PP algorithm is a consistent finite element counterpart of the SIMPLER algorithm (see Reference 7). The important difference between this version of the segregated algorithm and the correction version above is that the pressure is obtained directly from the solution of a SCPE. The algorithm comprises three main steps. At the beginning of a given iteration, an approximation to the pressure is obtained from the solution of a SCPE using the latest available field variables. The various components of the momentum equations and any other conservation equations present in the flow problem are then solved in a sequential manner using the most recent field data. Finally, at the end of the algorithmic sequence the velocity field is mass-adjusted (i.e. forced to satisfy the discretized continuity equation) via an irrotational projection onto a divergence-free sub-space. This last step, which is similar to a least-squares mass adjustment, requires the solution of a further SCPE system for a pseudo-pressure p^s. The SIMPLER algorithm is known to have superior convergence characteristics compared to the SIMPLE algorithm.

The implementation of our version of the SIMPLER algorithm may be summarized in the following algorithmic steps.

Given an initial or guess solution field (u^0, v^0, p^0, θ^0), for $i=0,1,2,\ldots$ until convergence, the following steps should be taken:

(1) solve SCPE for pressure p,

$$[C_x^{\mathrm{T}}(\tilde{K}_u^{-1})^* C_x + C_y^{\mathrm{T}}(\tilde{K}_v^{-1})^* C_y]\, p^{i+1/2} = -C_x^{\mathrm{T}}(\tilde{K}_u^{-1})^*[f_u^* - K_u^* u^i] - C_y^{\mathrm{T}}(\tilde{K}_v^{-1})^*[f_v^* - K_v^* v^i]; \tag{5a}$$

(2) relax pressure via

$$p^{i+1} = \alpha_p p^i + (1-\alpha_p)\, p^{i+1/2}; \tag{5b}$$

(3) solve x-momentum equation for u,

$$\left[\left(\frac{\alpha_u}{1-\alpha_u}\right)\tilde{K}_u + K_u\right]^* u^{i+1/2} = f_u^* + C_x p^{i+1} + \left(\frac{\alpha_u}{1-\alpha_u}\right)\tilde{K}_u^* u^i; \tag{5c}$$

(4) solve y-momentum equation for v,

$$\left[\left(\frac{\alpha_v}{1-\alpha_v}\right)\tilde{K}_v + K_v\right]^* v^{i+1/2} = f_v^* + C_y p^{i+1} + \left(\frac{\alpha_v}{1-\alpha_v}\right)\tilde{K}_v^* v^i; \tag{5d}$$

(5) solve energy equation for θ,

$$\left[\left(\frac{\alpha_\theta}{1-\alpha_\theta}\right)\tilde{K}_\theta + K_\theta\right]^* \theta^{i+1} = f_\theta^* + \left(\frac{\alpha_\theta}{1-\alpha_\theta}\right)\tilde{K}_\theta^* \theta^i; \tag{5e}$$

(6) solve SCPE for p^s,

$$[C_x^T(\tilde{K}_u^{-1})^* C_x + C_y^T(\tilde{K}_v^{-1})^* C_y]p^s = -C_x^T u^{i+1/2} - C_y^T v^{i+1/2} + b_c; \tag{5f}$$

(7) mass adjust velocity field via

$$u^{i+1} = u^{i+1/2} + (\tilde{K}_u^{-1})^* C_x p^s; \tag{5g}$$

$$v^{i+1} = v^{i+1/2} + (\tilde{K}_v^{-1})^* C_y p^s. \tag{5h}$$

In the above equations the definition of the superscripts and $\tilde{K}$ matrices is the same as in the PC algorithm.

2.4.3. The pressure update (PU) algorithm. While in the PP and PC algorithms the incompressibility constraint was being satisfied via mass-adjusting the velocity field using a pressure correction or pseudo-pressure field, in the PU algorithm continuity is satisfied through penalizing the continuity equation on the right-hand side of the SCPE. To the best of our knowledge there is no counterpart of this algorithm in the finite volume context. We will not attempt its detailed derivation here except to say that it was conceived from the notion of a pseudo-temporal approach applied to the momentum equations. The change in the momentum equations from one iteration to the other is considered to be along some type of pseudo-time dimension which advances at different rates in different physical locations of the discretized domain. In a real transient approach the mass matrix, multiplied by the inverse of the time step, appears on both sides of the discretized momentum equations. In this pseudo-temporal approach the above product (i.e. mass matrix divided by Δt) is replaced by the $\tilde{K}$ matrices. The pseudo-momentum equations are not used directly to solve for velocities; rather they are employed to arrive at a SCPE for pressure which contains on its right-hand side the discretized continuity equation at both the current and the advanced iterate levels. Finally, the desired SCPE is obtained by requiring that continuity be satisfied at the advanced iterate level.

The algorithmic steps involved in this version of the segregated algorithm are as follows.

Given an initial or guess solution (u^0, v^0, p^0, θ^0) for $i=0, 1, 2, 3, \ldots$ until convergence, the following steps should be taken:

(1) solve SCPE for pressure p,

$$[C_x^T(\tilde{K}_u^{-1})^* C_x + C_y^T(\tilde{K}_v^{-1})^* C_y]p^{i+1/2} = -C_x^T(\tilde{K}_u^{-1})^*[f_u^* - K_u^* u^i] - C_y^T(\tilde{K}_v^{-1})^*[f_v^* - K_v^* v^i] - \lambda_p(C_x^T u^i + C_y^T v^i - b_c), \tag{6a}$$

(2) relax pressure via

$$p^{i+1} = \alpha_p p^i + (1-\alpha_p)p^{i+1/2}, \tag{6b}$$

(3) solve x-momentum equation for u,

$$\left[\left(\frac{\alpha_u}{1-\alpha_u}\right)\tilde{K}_u + K_u\right]^* u^{i+1} = f_u^* + C_x p^{i+1} + \left(\frac{\alpha_u}{1-\alpha_u}\right)\tilde{K}_u^* u^i; \tag{6c}$$

(4) solve y-momentum equation for v,

$$\left[\left(\frac{\alpha_v}{1-\alpha_v}\right)\tilde{K}_v + K_v\right]^* v^{i+1} = f_v^* + C_y p^{i+1} + \left(\frac{\alpha_v}{1-\alpha_v}\right)\tilde{K}_v^* v^i; \tag{6d}$$

(5) solve energy equation for θ,

$$\left[\left(\frac{\alpha_\theta}{1-\alpha_\theta}\right)\tilde{K}_\theta + K_\theta\right]^* \theta^{i+1} = f_\theta^* + \left(\frac{\alpha_\theta}{1-\alpha_\theta}\right)\tilde{K}_\theta^* \theta^i. \tag{6e}$$

The notation convention used in the above equations is the same as that used in the PC and PP algorithms above. However, note the appearance of the dimensionless coefficient λ_p on the right-hand side of the pressure equation. This may be thought of as a penalty parameter which controls the penalization of the continuity constraint in the pressure equation. The value of λ_p should be chosen large enough to adequately enforce the continuity constraint. However, too large a value may lead to large pressure perturbations and destabilize the convergence behaviour. Our testing has shown the range $0{\cdot}1 < \lambda_p < 1$ to work well for most problems and we have consequently adopted a default value of $\lambda_p = 0{\cdot}2$.

Remarks 2.4

1. Recall that the above three algorithms are variations based on equations (2a), (2b), (2d) and (3b). These alternative approaches need to be adopted since it is prohibitively expensive to construct and solve equation (3b).
2. In the above algorithms use is made of SCPEs throughout for the pressure-like variables. An inspection of the above algorithms shows that at convergence they satisfy all the discretized equations in equation (1) and their associated boundary conditions, including the implied boundary conditions on pressure, to within the convergence tolerance. Therefore, the solution which is obtained from these algorithms is formally identical to the one that would be obtained from solving the system of equations (1) or equations (2), in a fully coupled manner. We have verified this through numerous numerical tests.
3. The $\tilde{K}$ matrices in the above algorithms are diagonal matrices and are obtained from the following expressions:

$$(\tilde{K}_u)_{ii} = \sum_e \sum_j |(K_u)_{ij}|, \tag{7a}$$

$$(\tilde{K}_v)_{ii} = \sum_e \sum_j |(K_v)_{ij}|, \tag{7b}$$

$$(\tilde{K}_\theta)_{ii} = \sum_e \sum_j |(K_\theta)_{ij}|, \tag{7c}$$

where $\sum_e$ denotes a sum or assembly of element-level contributions.

Since the above $\tilde{K}$ matrices are diagonal, their inversion is trivial and inexpensive. As a result the SCPMs appearing in the above algorithms are substantially less expensive to construct and solve compared to the FCPM in equation (3b). However, note that the

construction of these SCPMs is still non-trivial as they can only be formed at the global level from the global C and $\tilde{K}$ matrices. As these SCPMs need to be formed repeatedly during the course of the computations (typically once per iteration), it is important to devise fast and efficient procedures for their construction.

4. It can be readily shown that the converged solution in the above algorithms is entirely independent of the exact nature of the $\tilde{K}$ matrices as long as the dimension of the $\tilde{K}$ matrices is the same as their corresponding K matrices. Thus, strictly speaking, any matrix could be used; even the identity matrix. However, in order to obtain reasonable rates of convergence, the $\tilde{K}$ matrices should be close to the K matrices in some sense. Our choice of the $\tilde{K}$ matrices as defined by equation (7) provides a reasonable compromise between closeness to the K matrices and simplicity. A more sophisticated choice which could lead to faster rates of convergence would be to employ banded $\tilde{K}$ matrices which are incomplete versions of the full K matrices. For instance, the $\tilde{K}$ matrices may only contain entries in locations corresponding to the original K matrices. Many variations are possible, however, these more sophisticated choices will significantly increase the cost of forming the SCPMs and will only be effective if they dramatically increase the rate of convergence.
5. There is an important distinction between the pressure correction variable Δp in the PC algorithm and the pseudo-pressure p^s in the PP algorithm. While both variables are used in the same manner to perform the mass adjustments, they have different physical significances. In the PC algorithm, Δp is a true physical pressure which is also used to update the pressure field. In the PP algorithm, p^s although having dimensions of pressure, is in reality a Lagrangian multiplier through which the velocity field is mass adjusted. p^s is discarded after the mass adjustment process and is not used to update the pressure field.
6. The order (or sequence) of the various steps appearing in the above algorithms was determined partly by heuristic notions of optimality of rate of convergence and partly by the need to produce a computer code that was efficient and well organized.
7. Relaxation of the equations in the above algorithms is achieved in two entirely different ways. The pressures are relaxed explicitly (cf. equation (5b)). On the other hand, inertial or implicit relaxation is used for the other primary variables governed by transport equations. This has the effect of increasing the weight of the diagonal terms of the coefficient matrix of the equations governing these primary variables. That is, in the above equations the $[(\alpha/(1-\alpha))\tilde{K}+K]$ matrices are more diagonally dominant than the K matrices. It is well known that iterative matrix solvers perform more favourably on diagonally dominant systems. As our ultimate intention was to use iterative solvers in our algorithms, we introduced implicit relaxation (i.e. implicit preconditioning) in the equations to render them more amenable to these solvers (see Section 3.4).

2.5. *Numerical tests and discussion of results*

In this section some selected results are presented from numerous numerical tests which were conducted with the above segregated algorithms. These demonstrate the integrity of the algorithms and their performance relative to each other and a fully coupled solution algorithm. All the numerical tests reported in this section were performed using the direct Gaussian elimination type linear equation solvers for the various equation sub-systems.

First the relative performance of the three algorithms is shown by considering the two flow test cases identified in Table I.

In Case 1, a mesh of 943 nodes is employed consisting of 212 9-node quadratic quadrilateral elements. In case 2, a mesh of 1305 nodes is employed consisting of 1232 4-node linear

Table I. Definition of test Cases 1 and 2

Case 1	2D laminar flow over two ribs in a channel at $Re = 100$
Case 2	2D turbulent flow in axisymmetric U-bend at $Re = 100\,000$

Table II. Relative performance of the PC, PU and PP algorithms

	Case 1			Case 2		
Solution algorithm	PC	PU	PP	PC	PU	PP
No. of iterations	98	39	24	≈ 1000	171	135
CPU time (seconds)	250	100	62	≈ 10000	1976	1570

quadrilateral elements. The standard k–ε turbulence model (Reference 24) is used in Case 2 and in all other turbulent flow problems reported in this paper. Moreover, in turbulent flows involving solid boundaries, specialized near-wall finite elements are used to model the viscous and buffer sublayers (see Reference 25).

Selected performance data from the above two test cases are presented in Table II.

The information contained in Table II and some additional information relating to our experience with the above three algorithms is summarized below.

Remarks 2.5.1

1. The above runs were made with finely tuned relaxation factors so that the data contained in the above table representatively reflects the relative performance characteristics of the above three algorithms.
2. It is seen from Table II that of the three algorithms the PP algorithm is the most efficient; it is closely followed by the PU algorithm, and the PC algorithm is the slowest by a substantial margin.
3. The above performance characteristics are quite representative over a much wider range of flow problems.
4. The PP algorithm requires the least amount of relaxation for stability and the PC algorithm requires the most.
5. Owing to its superior efficiency, we have employed the PP algorithm far more extensively than the PU and PC algorithms. For this reason we will only present numerical results from the PP algorithm in the remainder of this section and in Section 3, where the topic of iterative solution techniques is considered.
6. As alluded to in Section 1, for the small-scale problems considered above, a segregated scheme is not necessarily the most efficient approach of solution. For example, with a fully coupled solution algorithm using the Picard method and a (u–p) formulation (cf. equation (1)) Case 1 can be solved in 31 s using about five times more CPU memory. Moreover, even more efficient fully coupled algorithms may be used. The same problem when solved with a fully coupled algorithm using the Newton–Raphson scheme and the penalty formulation only required 14 s of CPU time and less than three times more storage compared to the segregated algorithms. However, the Newton–Raphson schemes have a much smaller radius of convergence compared to the Picard scheme and are therefore not generally robust enough to be applied to a wide range of flow problems. This is especially

true of turbulent flow problems where the k–ε turbulence model is employed in the simulation.

Next the performance of the PP algorithm is shown relative to the fully coupled algorithm using the Picard scheme and the (u–p) formulation — the base fully coupled algorithm denoted hereafter as the FC algorithm. For this purpose two additional test problems are introduced which are identified in Table III.

In the above examples, test runs are made with both a coarse and a fine mesh. This also allows relative performances to be studied as a function of mesh refinement.

Selected performance data from the above two test cases are presented in Table IV.

The information contained in Table IV and some additional information relating to experience we have gained in comparing the performances of the PP algorithm and the FC algorithm is summarized below.

Remarks 2.5.2

1. The PP algorithm always requires significantly less matrix storage compared to the FC algorithm. (The storage requirements of the PP, PU and PC algorithms are identical.)
2. The theoretical matrix storage ratio between the FC and PP algorithms is NDOF^2, where NDOF is equal to the primary active degrees of freedom, i.e. u, v, w, p, etc. (This ratio is typically realized on large meshes where the effects of the Dirichlet boundary conditions on matrix size become negligible.) Thus, the PP algorithm will require proportionately less matrix storage as the dimensionality and the number of primary variables involved in the flow problem (e.g. temperature, chemical species concentration, k and ε) are increased.
3. The CPU time ratio between the FC and PP algorithms is less predictable. It can be shown that the theoretical matrix solution time ratio between the FC and PP algorithms is NDOF^2 per iteration. Again this ratio is typically realized on very fine meshes where the overhead costs associated with the other parts of the algorithms (i.e. cost of martix formation and

Table III. Definition of test Cases 3 and 4

Case 3	2D laminar flow in square driven cavity at $Re = 1000$
Case 4	3D laminar flow in square driven cavity at $Re = 100$

Table IV. Relative performance of the PP and FC algorithms

No. of nodes	Case 3 (51 × 51)		Case 3 (101 × 101)		Case 4 (15 × 7 × 15)		Case 4 (21 × 11 × 21)	
Solution algorithm	FC	PP	FC	PP	FC	PP	FC	PP
No. of iterations	12	67	12	141	8	10	8	14
Solution time (CPU seconds)	275	889	2912	12506	450	255	6736	2437
Time/iteration (CPU seconds)	22·9	13·2	242·6	88·6	56·2	25·5	842	174·1
Matrix storage (megawords)	1·47	0·27	11·9	2·07	2·47	0·335	21·7	2·25

assembly, which is linearly proportional to the number of elements) become negligible. Thus, the PP algorithm requires less CPU time per iteration than the FC algorithm and this difference becomes larger with increasing NDOF.

4. Total CPU time is, of course, also a function of the number of iterations needed to reach convergence. The PP algorithm (and the other segregated algorithms), being decoupled in nature, typically requires more iterations to converge compared to the FC algorithm. A further side-effect of decoupling is that the number of iterations typically grow with mesh refinement. The FC algorithm exhibits no such tendency.
5. We have found that for most 2D problems the FC algorithm is more efficient in CPU time and that for almost all 3D problems the PP algorithm is more efficient.

While the above segregated solution algorithms are substantially more efficient than fully coupled solution algorithms for 3D flow problems, their storage and CPU time requirements still grow rapidly with increasing mesh size. This is due to the use of the direct Gaussian elimination type solvers in these algorithms. For example for moderately sized 3D turbulent (k–ε model) flow problems involving of the order of 40 000 nodes, the PP algorithm using a direct solver can require a gigabyte of disk space and many hours of CPU time on a Cray XMP computer.

Recall that the purpose of the first phase of our project was to design segregated solution algorithms and to test their integrity and validity. With this step successfully completed the next obvious step was to replace the direct solvers with iterative solvers. The results of this second phase of the project are reported in the following sections.

3. ITERATIVE SOLUTION OF THE LINEAR EQUATION SYSTEMS

3.1. Introduction

In the segregated solution algorithms described in the previous section, two types of linear equation systems are encountered; the pressure type equations (the SCPEs) and the advection–diffusion type equations arising from the conservation equations of momentum and energy. The coefficient matrices of the SCPEs (i.e. the SCPMs) are symmetric positive-semidefinite and the coefficient matrices of the advection– diffusion type equations are non-symmetric positive real.

As noted in Section 2, the above equation systems were initially solved using direct solvers. The great advantage of direct solution is that for non-singular coefficient matrices a solution to the equation system is always obtained after a finite number of arithmetic operations. The disadvantage of direct solution is that for large-scale simulations the computer resources needed (CPU, I/O, memory and disk storage) become prohibitively large.

Given a system of N linear equations $Ax=b$ (where A is an $(N \times N)$ non-singular coefficient matrix), iterative solvers begin with an initial guess x_0 and compute a sequence of approximate solutions $\{x_n\}$ that hopefully converge towards the exact solution. The accuracy of the solution depends on the number of iterations performed. Iterative solvers have the advantage over direct solvers of preserving the sparsity of the coefficient matrix. This feature is particularly attractive in a finite element context where the coefficient matrices are characterized by a large number of zero entries. Another advantage of iterative solvers is that for most implementations only the product of the coefficient matrix and a vector is required rather than the coefficient matrix itself. This obviates the need to form the coefficient matrix directly which results in reduced storage requirements. These savings increase dramatically as the size and bandwidth of the coefficient matrix increases.

The amount of effort involved in solving a linear equation system using an iterative solver depends on the convergence rate and the desired accuracy. A significant drawback of iterative solvers is the possibility of slow or irregular convergence, or divergence of the iterates. The convergence rate of an iterative solver depends on the properties of the coefficient matrix A. These properties are themselves characterized by the spectrum of the coefficient matrix (the spectrum of A is the set of all N eigen values of A denoted by; $\lambda(A)=\{\lambda_1, \ldots, \lambda_N\}$) and the precise manner in which all the eigenvalues comprising the spectrum are distributed in the complex plane. It is well known that positive-definiteness, diagonal dominance and symmetry are properties of the coefficient matrix which lead to better convergence behaviour in iterative solvers. As these properties are lost, convergence behaviour deteriorates and it becomes necessary to adopt more sophisticated (and invariably more expensive) iterative solvers. The rate of convergence can often be dramatically increased by the technique of preconditioning. More information on iterative solvers can be found in Reference 26.

In the present study a number of iterative solvers were implemented to optionally replace the direct Gaussian elimination solvers in the above segregated algorithms. In the following sections these solvers are introduced.

3.2. The symmetric linear equation system solvers

In this section the two iterative solvers which were used to solve the pressure type equation systems are detailed. Recall that the coefficient matrices in these equations are of the SCPM type which are symmetric and positive-semidefinite. The last characteristic (i.e. semidefiniteness) results from the fact that many SCPMs contain singular modes or zero energy modes which manifest themselves in terms of mostly spurious pressure patterns which while polluting the pressure solution are transparent to the velocity field; for a thorough discussion of pressure modes and their origins see Reference 27. Unfortunately this characteristic may introduce a destabilizing influence on an otherwise well-behaved matrix.

For the simplest case of a symmetric and positive-definite coefficient matrix there is a large body of theory and there are several very successful iterative solvers (see Reference 9). The most widely used and the most successful of these is the conjugate gradient (CG) solver, which is related to the Lanczos method for tridiagonalizing a matrix. The CG solver can in fact be viewed as a direct solver since in the absence of round-off error it gives the exact solution in N steps, where N is the number of equations (or unknowns). Alternatively, it may be viewed as being an approximate solver which produces an approximate solution in far fewer steps. An attractive feature of the solver is that it does not incorporate any tuning parameters and yet has the very desirable attributes of optimality and monotonicity.

Another solver that is very effective for symmetric positive definite matrices is the conjugate residual (CR) solver (see References 26 and 28). This solver is closely related to the CG solver and differs mainly in the definition of the functional used to minimize the error in the inner iterations. Like the CG solver, the CR solver possesses the properties of optimality and monotonicity without the introduction of any tuning parameters.

Both of these solvers have been employed in this study to solve the SCPEs appearing in the segregated algorithms.

3.2.1. The CG solver. Consider the system of N linear equations $Ax=b$, where A is a symmetric matrix. The basic algorithmic steps of the CG solver are as follows.

Given an initial guess x_0, obtain a sequence $\{x_n\}$ of approximations to the solution x from:
choose x_0,
set $r_0=b-Ax_0$,
set $p_0=r_0$,

for $i=0,1,2,\ldots$ until convergence do:

$$a_i=(r_i, r_i)/(p_i, Ap_i),$$
$$x_{i+1}=x_i+a_ip_i,$$
$$r_{i+1}=r_i-a_iAp_i,$$
$$b_i=(r_{i+1}, r_{i+1})/(r_i, r_i),$$
$$p_{i+1}=r_{i+1}+b_ip_i.$$

The computational effort per inner iteration involves $5N$ multiplications plus one matrix–vector product. The minimum storage requirement is a very modest four vectors of length N.

3.2.2. The CR solver. The basic algorithmic steps of the CR solver are closely related to that of the CG solver. An efficient implementation of the CR solver with respect to the number of operations per inner iteration is:

choose x_0,
set $r_0=b-Ax_0$,
set $p_0=r_0$,
for $i=0,1,2,\ldots$ until convergence do:

$$a_i=(r_i, Ar_i)/(Ap_i, Ap_i),$$
$$x_{i+1}=x_i+a_ip_i,$$
$$r_{i+1}=r_i-a_iAp_i,$$
$$b_i=(r_{i+1}, Ar_{i+1})/(r_i, Ar_i),$$
$$p_{i+1}=r_{i+1}+b_ip_i,$$
$$Ap_{i+1}=Ar_{i+1}+b_iAp_i.$$

The computational effort is slightly more than the CG solver. This involves $6N$ multiplications and one matrix–vector product. Five vectors of length N need to be stored as a minimum.

3.3. The non-symmetric linear equation system solvers

The diagonal dominance and asymmetry characteristics of the coefficient matrices of advection–diffusion type equation systems depends on the grid Reynolds and/or Peclet numbers. At very low grid Reynolds/Peclet numbers, these coefficient matrices are dominated by the positive-definite symmetric diffusion operators. However, as advection in the computational mesh becomes important, the anti-symmetric advection operators become dominant and not only lead to a non-symmetric coefficient matrix but also diminish the diagonal dominance.

As the iterative solution of equation systems with non-symmetric coefficient matrices is more difficult, there is in general less theory available and it is more challenging to provide rigorous proof or guarantee of convergence. This is particularly true for the more difficult case of equation systems with non-positive real matrices. However, as a result of intense research activity in recent years, increasingly more sophisticated and robust iterative solvers for the above equation systems have become available. In the present study we have employed two such iterative solvers for the solution of the advection–diffusion type linear equation systems which appear in the segregated algorithms. These are the conjugate gradient squared (CGS) and generalized minimum residual (GMRES) iterative solvers, which are briefly described below.

3.3.1. The CGS solver. The CGS solver is a conjugate gradient-like solver which has been derived from the biconjugate gradient method (BCG) (see Reference 29 and 30).

The algorithmic steps of the CGS solver are as follows:

choose x_0,
set $r_0 = b - Ax_0$,
set $g_0 = u_0 = r_0^* = r_0$,
for $i = 0, 1, 2, \ldots$ until convergence do:

$$\begin{aligned} p_{i+1} &= u_i - a_i A g_i, \\ x_{i+1} &= x_i + a_i(u_i + p_{i+1}), \\ r_{i+1} &= r_i - a_i A(u_i + p_{i+1}), \\ u_{i+1} &= r_{i+1} + b_i p_{i+1}, \\ g_{i+1} &= u_{i+1} + b_i(b_i g_i + p_{i+1}), \\ Ap_{i+1} &= Ar_{i+1} + b_i Ap_i, \end{aligned}$$

where

$$\begin{aligned} a_i &= (r_0^*, r_i)/(r_0^*, Ag_i), \\ b_i &= (r_0^*, r_{i+1})/(r_0^*, r_i). \end{aligned}$$

The CGS algorithm can be implemented with six vectors of length N.

3.3.2. The GMRES solver. The GMRES solver is a sophisticated iterative solver which cannot break down even for indefinite matrices (see References 31 and 32). The solver is derived from the full orthogonalization method (FOM) of Saad[33, 34] and employs the Arnoldi process to construct an orthogonal basis of Krylov subspaces. The methodology behind this solver is detailed and complex; here we only give a brief description. For a detailed exposition of the solver and the implementation of its algorithmic steps, see References 35 and 36.

As in the other solvers described above, the GMRES solver generates a sequence of iterates converging to the solution of the linear equation system $Ax = b$. If x_0 is the initial guess for the true solution x, then letting $x = x_0 + z$, we have the equivalent system

$$Az = r_0,$$

where $r_0 = b - Ax_0$ is the initial residual. Letting K_m be the Krylov subspace

$$K_m \equiv \text{span } \{r_0, Ar_0, \ldots, Ar_0^{m-1}\},$$

GMRES finds an approximate solution

$$x_m = x_0 + z_m, \quad \text{with } z_m \in K_m$$

such that

$$\|b - Ax_m\|_2 = \min \|b - Ax\|_2 \ (= \min \|r_0 - Az\|_2).$$

It can be shown that minimizing the above expression is equivalent to finding a vector y_m^{GM} in R^m which minimizes the following norm

$$\|\beta e_1 - H_m^* y\|_2,$$

where m is the inner iteration index of the GMRES solver. Moreover,

$$\beta = \|r_0\|_2,$$

$$e_1 = [1, 0, 0, \ldots, 0]^T$$

and H_m^* is the so-called Hessenberg matrix which is constructed from the orthogonal basis $V_m = [v_1, v_2, \ldots, v_m]$ of the Krylov subspace K_m (the orthogonal vectors v_i are obtained from the Arnoldi process noted above).

Once y_m^{GM} is obtained, the approximate GMRES solution x_m^{GM} is given by

$$x_m^{GM} = x_0 + V_m y_m^{GM}.$$

GMRES is guaranteed to converge to the exact solution of $Ax = b$ in at most N steps using exact arithmetic for any non-singular $(N \times N)$ matrix A. However, a practical difficulty with GMRES is that the number of vectors that need to be stored increases with the inner iterations and the number of multiplications is proportional to the square of the inner iterations. Clearly the solver becomes increasingly more expensive as the number of inner iterations increases. Various strategies have been developed in the literature to limit (or cap) the potentially high cost of the GMRES solver. These range from setting a maximum upper limit on m and restarting the GMRES algorithm with the approximate solution at that point, to limiting the number of vectors that need to be made orthogonal in the Arnoldi process. Unfortunately, with these changes convergence of the solver can no longer be guaranteed and in practice stagnation (or plateauing) of convergence is sometimes observed.

3.4. Preconditioning

Recall from Section 3.1 that the rate of convergence of an iterative solver depends on the properties of the coefficient matrix A which is itself characterized by its spectrum $\lambda(A)$. It was also noted that positive-definiteness, diagonal dominance and symmetry are properties of the coefficient matrix which lead to better convergence behaviour in iterative solvers. The goal of preconditioning is to manipulate the original system of linear equations into an equivalent one that has a coefficient matrix with properties that are more amenable to iterative solution. This may not only help to accelerate the rate of convergence of an iterative solver but also help to stabilize the convergence characteristics of an otherwise stagnant or diverging solution.

The choice of a good preconditioner is absolutely crucial to the viability of an iterative solver; experience has shown that the availability of a good preconditioner is far more important than the difference between various iterative solvers. However, despite this there is little theory available to guide the design of cost-effective and efficient preconditioners for the various types of coefficient matrices encountered in CFD applications. The limited theory that is available applies to the simplest (best) case of a symmetric positive-definite coefficient matrix A. In this case it may be shown (see Reference 26) that the upper bound on the number of iterations, *iter*, needed by the CG or CR solvers to obtain a solution with a specified tolerance, *tol*, is

$$iter = \tfrac{1}{2}(\ln(2/tol)) \times (\kappa(A))^{1/2}$$

where $\kappa(A)$ is the condition number of the coefficient matrix A, which is the ratio between the absolute values of the largest and smallest eigen values of A. *iter* is therefore proportional to the square root of the condition number and could be a large number if $\kappa(A)$ is large. Thus, the goal of preconditioning in the context of linear equation systems with a symmetric positive definite coefficient matrix is to manipulate the original system of equations into an equivalent one that has a substantially lower condition number. In the case of non-symmetric and/or non-positive-

definite coefficient matrices and in the absence of firm theoretical insight, the design of preconditioners has been guided to a greater degree by intuition and experience.

Recall that the above segregated algorithms involve outer iterations owing to the intercoupled and non-linear nature of the incompressible flow equations. As a result, the various linear equation systems characterized by the generic system $Ax=b$ are solved sequentially and repeatedly during the course of a typical simulation. Moreover, relaxation of the solution is also almost invariably used to ensure a stable convergence behaviour. Iterative solvers can be made to advantageously exploit the above two characteristics of segregated algorithms. Firstly, the outer iterations produce a good initial guess for the iterative solution of $Ax=b$ which helps to reduce the number of inner iterations of the iterative linear equation solvers. Secondly, the need for relaxation of the solution provides the opportunity of manipulating (or preconditioning) the linear systems in such a manner as to make them better suited to iterative type solvers.

Preconditioning can be achieved in various ways. In the present algorithms two preconditioning strategies are employed; implicit preconditioning and conventional preconditioning. The preconditioners associated with these two strategies are applied at two distinct stages in the solution algorithms.

3.4.1. Implicit preconditioning. As noted in Remarks 2.4, implicit relaxation is used in the non-symmetric advection–diffusion type linear equation systems. This has the combined effect of increasing the diagonal dominance and reducing the condition number of the coefficient matrices. Thus implicit relaxation is in essence a form of preconditioning which is automatically (and fortuitously) applied as part of the main segregated solution algorithms at no extra cost.

It is important to note that implicit relaxation is applied to $Ax=b$ before the conventional preconditioners described in the next section are employed. Thus, implicit relaxation could alternatively be viewed as being a process through which the original linear equation systems are conditioned before conventional preconditioning is applied. This is of great importance since as a result much simpler (and inexpensive) conventional preconditioners can be used in the solution of the implicitly relaxed $Ax=b$ systems.

3.4.2. Conventional preconditioning. In the conventional preconditioning approach, the generic linear equation system $Ax=b$ is replaced by the following equivalent system of equations:

$$[P_l^{-1}AP_r^{-1}][P_r x]=P_l^{-1}b,$$

where P_r and P_l are the right and left preconditioning matrices, respectively. Three different approaches are possible:

1. left preconditioning approach; $P_r=I$,
2. right preconditioning approach; $P_l=I$,
3. split preconditioning approach; P_r and $P_l\neq I$.

In the present study both the left and split preconditioning approaches have been used.

The primary objective of conventional preconditioning is to make $P_l^{-1}AP_r^{-1}$ 'look like' the identity matrix so that it has a small condition number of order unity. For preconditioning to be effective, the faster rate of convergence must overcome the added cost of applying the preconditioning, so that the total cost of solving the linear system is lower. Note that as an extreme (and impractical) example one could use $P_l^{-1}=A^{-1}$ and $P_r^{-1}=I$; thus, when applied in conjunction with an iterative solver, conventional preconditioning can be viewed as being a technique which spans the gap between a direct and an iterative solution of $Ax=b$. Experience shows that in order to be cost-effective, the preconditioning matrix should be sparse, easy to compute and inexpensive

to invert. In the present study two types of conventional preconditioning matrices have been used; diagonal and SSOR type preconditioning matrices.

Diagonal preconditioning. Preconditioning with diagonal matrices is the simplest possible form of preconditioning. Since P is a diagonal matrix operations with P are inexpensive and fast. It has been shown that for symmetric positive-definite systems the best (or nearly the best choice) is $P=\text{diag}(A)$. The low storage and small extra CPU cost per inner iteration of this scheme are its virtues; the large number of iterations that may still be required is its drawback.

In our algorithms diagonal preconditioning of the left type (i.e. $P_l=\text{diag}(A)$, $P_r=I$) can be optionally used in the iterative solution of both symmetric and non-symmetric equation systems.

SSOR preconditioning. SSOR preconditioning is one approach from a much wider class of methods which are based on incomplete factorizations of the coefficient matrix A. Noting that in the context of classical SSOR iterative solvers A is assumed to have the following decomposition:

$$A=L+D+U,$$

where L and U are lower and upper matrices, respectively , and $D=$ diag(A). The classical SSOR matrix Q is then

$$Q=\frac{1}{2-\omega}(L+D/\omega)(D/\omega)^{-1}(\mathrm{D}/\omega+U),$$

where ω is the associated SSOR relaxation/acceleration factor.

In the present study the inverse of Q with $\omega=1$ is employed as the preconditioning matrix, viz.

$$Q^{-1}=(U+D)^{-1}D(L+D)^{-1}.$$

We use the split approach to apply the SSOR preconditioning. Lack of space precludes a detailed exposition of the various steps involved in the implementation. The major component of cost associated with applying SSOR preconditioning is of the order of one matrix–vector multiplication which, while non-trivial, is still relatively inexpensive.

In our algorithms, SSOR preconditioning can be optionally used in the iterative solution of both symmetric and non-symmetric equation systems.

3.5. Numerical tests and discussion of results

In this section selected results are presented from numerical tests that were conducted to study the performance of the PP segregated algorithm using iterative solvers — hereafter denoted by PP(I). Comparisons are made with performance data from mainly the PP algorithm using the direct Gaussian elimination type solvers, PP(D), and where possible, from the base fully coupled solution algorithm, FC. Moreover, mostly 3D test cases are considered since the speed gains of iterative solvers are typically realized for large problems.

First, the topic of relative performance of the various iterative solvers is considered. Table V contains pertinent data from a series of four runs conducted with the PP(I) algorithm with various combinations of the CG, CR, CGS and GMRES iterative solvers. The test problem, Case 5, is 3D laminar flow in a blade mixer. The mesh contains 11517 nodes and consists of 10240 8-node brick elements.

Table V. Performance data from the PP(I) algorithm using various combinations of iterative linear equation solvers

Iterative solver combination (symmetric/non-symmetric)	CG/CGS	CR/CGS	CG/GMRES	CR/GMRES
Typical no. of inner iterations for each non-symmetric equation (u, v *or* w)	22	22	30	30
Typical no. of inner iterations for each symmetric equation (p or p^s)	160	65	160	65
Average combined solution time for one pass of u, v and w (CPU seconds)	21·5	21·5	24	24
Average combined solution time for one pass of p and p^s (CPU seconds)	60	29	60	29

Remarks 3.5.1

1. In the above runs (and in the subsequent runs reported herein), for the conventional preconditioners we have employed SSOR preconditioning for the symmetric pressure-like equations and diagonal preconditioning for the non-symmetric advection–diffusion type equations. We have so far found this choice of conventional preconditioners to be the most optimal in the above segregated algorithms. Specifically, it is found that SSOR preconditioning applied to the pressure-like equation systems leads to between three and six times fewer inner iterations compared to diagonal preconditioning. The corresponding solution time for solving the pressure-like equations is between 1·5 to 3 times less. The difference between the choice of preconditioner is less critical for the non-symmetric advection–diffusion type equations. It is typically found that SSOR preconditioning leads to half as many inner iterations compared to diagonal preconditioning. However, the solution time is mostly unaffected by the choice of preconditioner.
2. The above table indicates that the best combination for iterative solvers for Case 5 is CR for the symmetric equations and CGS for the non-symmetric equations. Note that for the symmetric equations the CR solver is significantly more efficient than the CG solver. On the other hand, for the non-symmetric equations the less sophisticated CGS solver is more efficient than the GMRES solver, but not by such a large margin. This performance behaviour is representative of most other cases that have been studied so far. The superior performance of CGS solver over the GMRES solver in the context of the present segregated algorithms is somewhat fortunate since it means that less reliance needs to be placed upon the GMRES solver whose cost and storage requirements are unpredictable. Based on the above experience we are currently employing the CR/CGS combination as the default setting for the iterative solvers. This combination is used in the performance results reported in the following sections.
3. The above table also indicates that the symmetric equations require a substantially larger number of inner iterations for the convergence of the iterative solvers compared to the non-symmetric equations. As a result in the above algorithms the cost associated with solving the pressure-like variables is proportionately (and substantially) more expensive than the corresponding cost of solving the remainder of the primary flow variables. Referring to Table V it can be seen that the cost of solving one symmetric equation (say, p) is about four times more expensive than the cost of solving one non-symmetric equation (say, u). In extreme cases this cost ratio can be as much as 20.

Table VI. Relative performance data of PP(I), PP(D), and FC algorithms as a function of mesh refinement. Superscript * indicates extrapolated value from coarser meshes

No. of nodes	Solution algorithm	Matrix storage (megawords)	No. of iterations	Time per iteration (CPU seconds)
$15 \times 7 \times 15 = 1575$	PP(I)	0·035	10	17·9
	PP(D)	0·335	10	25·15
	FC	2·48	9	56
$31 \times 15 \times 31 = 14\,415$	PP(I)	0·356	22	203·9
	PP(D)	13·4	22	1263
	FC	151·3	N.A.	N.A.
$45 \times 23 \times 45 = 46\,575$	PP(I)	1·18	36	641·7
	PP(D)	96·5	36	15 969
	FC	1223	N.A.	N.A.
$61 \times 31 \times 61 = 115\,351$	PP(I)	2·98	53	1681
	PP(D)	436·5	53	130 300*
	FC	> 15000	N.A.	N.A.

4. In an attempt to reduce the cost associated with solving the pressure-like variables, implicit relaxation was introduced in the symmetric equations in the three segregated algorithms. While this significantly reduced the number of inner iterations needed for the solution of the pressure like variables, it adversely affected the convergence rate of the outer iterations of the segregated algorithms and lead to substantially higher overall solution times.

Next, the relative performance of the PP(I), PP(D), and where possible, the FC algorithms with mesh refinement is considered. Case 4 described earlier is used as the test problem for this purpose. Table VI contains the pertinent performance data from four levels of mesh refinement.

The information contained in Table VI and some additional information relating to experience we have gained in comparing the performances of the above algorithms is summarized below.

Remarks 3.5.2

1. In implementing iterative solvers three alternative approaches may be used to perform the matrix–vector product required at every inner iteration. In the first approach the above product is performed at the element level and the element matrices are recomputed every time they are needed. This approach while requiring the least amount of storage is very wasteful of CPU time since element matrices are recomputed repeatedly. In the second approach the matrix–vector product is performed at the element level but the element matrices are stored (typically on low speed storage devices) to avoid their recomputation. This approach while requiring less CPU time than the first approach requires substantially more storage. For large problems where out-of-core storage is used, additional I/O costs are also incurred. In the third approach the matrix–vector product is performed at the global level. This requires the assembled global matrix which is typically stored in compact sparse form. Depending on the amount of memory available, this matrix may be kept either in core or stored out of core. The storage requirements and the speed of this third approach are comparable to those of the second approach. In our algorithms the third approach is employed. This increases the memory requirement. As can be seen from the above table, however, this extra storage is still quite small relative to that required by the direct solvers.

2. Table VI shows that the PP(I) algorithm has the least storage and CPU requirements. The FD(C) algorithm has the most storage and CPU requirements.
3. The storage and CPU requirements of the PP(D) and FC(D) algorithms are proportional to NEQ × BW and NEQ × BW^2, respectively, where NEQ is the number of equations and BW is the bandwidth of the matrix.
4. For the meshes used in Case 4 these requirements of the PP(D) and PP(I) algorithms may be expressed in terms of the number of nodes, NODES, and the number of primary degrees of freedom, NDOF (i.e. u, v, w and p) as discussed below.
5. The storage requirements of the PP(D) and FC(D) algorithms are proportional to NODES$^{5/3}$ and NODES$^{5/3}$ × NDOF^2, respectively. The latter expression becomes more accurate with increasing mesh size.
6. The number of arithmetic operations required by the direct solver in the PP(D) and FC(D) algorithms is proportional to NODES$^{7/3}$ and NODES$^{7/3}$ × NDOF^2, respectively.
7. The timing information contained in Table VI for the PP(D) algorithm follow the NODES$^{7/3}$ trend on the finer meshes. The reasons for this are twofold. Firstly, on coarse meshes the matrix solution phase takes up a small portion of the overall CPU time, while on fine meshes most of the CPU time is taken up by the matrix solution phase. The other steps involved in the algorithm, such as element matrix formation and assembly, are linearly proportional to the number of elements present in the mesh and consequently take up proportionally less CPU time as the mesh size increases. The second reason is associated with the characteristics of the vector arithmetic performed on the Convex C220. The efficiency of the vectorization initially increases with increasing vector length (mesh refinement) but tapers off to a maximum for the long vector lengths resulting from the finer meshes. Consequently it is only at the limit of fine meshes that arithmetic vector operations become linearly scalable.
8. The storage and CPU requirements of the PP(I) algorithm increase only linearly with the number of nodes in the computational mesh. As a result, the efficiency of the PP(I) algorithm grows dramatically with mesh refinement.

Table VII contains further performance data from various other flow problems. The efficiency gains of the PP(I) algorithm with increasing problem size are once again seen to be quite dramatic.

Table VII. Performance comparison between PP(I) and PP(D) algorithms on various 3-D flow problems. Superscript * indicates estimated value obtained from extrapolated CPU time reported in Table VI

	No. of nodes	Run time ratio	Matrix storage ratio
Flow in blade mixer (Case 5) 3D laminar; $Re=60$	11 517	4·97	28·5
Flow in blade mixer (Case 5) 3D laminar; $Re=60$	33 995	6·56	42·11
Flow around 90° bend in channel 3D turbulent; $Re=40\,000$	43 648	4·76	40·6
Flow over bluff body 3D turbulent; $Re=1{\cdot}4\times10^6$	39 482	7·92	51·7
Flow in driven cavity (Case 4) 3D turbulent; $Re=100$	46 575	24·9	81·8
Flow over submarine 3D turbulent: $Re=10^7$	103 501	37	264
Flow in driven cavity (Case 4) 3D laminar; $Re=100$	115 351	77·5*	146·5

4. CONCLUSIONS AND FUTURE DIRECTIONS

In this paper we have presented finite element solution algorithms for the numerical solution of large-scale incompressible flow problems. These algorithms are based on the two techniques of segregated formulation and iterative linear equation solution.

Three different segregated solution algorithms were first presented and discussed. Using the direct Gaussian elimination type linear equation solvers, the integrity and relative performance of these algorithms were demonstrated via numerical tests. It was shown that the PP algorithm is the most efficient (in terms of CPU time) of the segregated algorithms. This was closely followed by the PU algorithm with the PC algorithm being the slowest by a substantial margin. The performance of the PP algorithm using direct solvers was also compared with a fully coupled solution algorithm. It was shown that while the PP algorithm requires substantially less storage compared to the fully coupled algorithm, it runs faster only for 3D problems.

Next, the four iterative linear equation solvers used to replace the direct Gaussian type linear equation solvers in the segregated algorithms were presented and discussed together with the preconditioning techniques used. Through numerical tests it was shown that CR solver with SSOR preconditioning is the most optimal choice for solving the symmetric pressure type equation systems and that the CGS solver with diagonal preconditioning is the optimal choice for the non-symmetric advection–diffusion type equations systems. The crucial importance of the implicit relaxation employed in the solution algorithms on the efficiency of the iterative solvers was also discussed. This increases the diagonal dominance of the non-symmetric equations and in so doing makes them significantly more amenable to iterative solvers.

Through additional numerical tests it was shown that the combined strategy of segregated formulation and iterative linear equation solution brings about significant improvement in algorithm performance which becomes increasingly more dramatic as the size and physical complexity of the problem increases.

In our future work we intend to improve further the efficiency and speed of the above segregated algorithms. This may be done by a combination of using more advanced preconditioners for the symmetric pressure-like equations and more efficient element matrix formation routines. At the global algorithmic level the possibility of replacing the Picard scheme with the Newton–Raphson-based schemes will also be investigated.

ACKNOWLEDGEMENTS

The study reported herein was partially supported by a grant from the Chrysler Corporation. The work has proceeded in close consultation with Dr. Richard Sun from the Corporation's Aerodynamics Department.

REFERENCES

1. A. J. Chorin, 'Numerical solution of the Navier–Stokes equations', *Math. Comput*, **22**, 745 (1968).
2. K. J. Arrow, L. Hurwicz and H. Uzawa, *Studies in Non-linear Programming*, Stanford University Press, Stanford, 1958.
3. F. H. Harlow and J. E. Welsch 'Numerical calculation of time-dependent viscous incompressible flows of fluids with free surface', *Phys. Fluids*, **8**, 2182–2189 (1965).
4. C. W. Hirt, A. A. Amsden and J. L. Cook, 'An arbitrary Lagrangian–Eulerian computing method for all speeds', *J. Comput. Phys.*, **14**, 227–253 (1974).
5. S. V. Patankar and D. B. Spalding, 'A calculation procedure for heat, mass and momentum transfer in three-dimensional flows', *Int. J. Heat Mass Transfer*, **15**, 1787 (1972).
6. B. Kashiwa, 'A generalized MAC method for incompressible flow problems', *Report LA-10853-MS*, Los Alamos National Laboratory, 1986.
7. S. V. Patankar, *Numerical Heat Transfer and Fluid Flow*, Hemisphere, Washington, DC, 1980.

8. S. V. Patankar, 'Calculation procedure for two-dimensional elliptic situations', *Numer. Heat Transfer*, **4**, 409–425 (1981).
9. M. S. Engelman and I. Hasbani, 'Matrix-free solution algorithms in a finite element context', *Technical Report 88-1*, Fluid Dynamics International, Evanston, IL, 1988.
10. M. S. Engelman, V. Haroutunian and I. Hasbani, 'A proposed finite element algorithm for incompressible fluid flows', in T. J. Chung and G. R. Karr (eds), *Finite Element Analysis in Fluids, Proc. 7th Int. Conf. on Finite Element Methods in Flow Problems*, UAH Press, Huntsville, Alabama, 1989, pp. 790–803.
11. M. S. Engelman, *FIDAP Theoretical Manual*, Version 6.0, Fluid Dynamics International (FDI), 1991.
12. P. M. Gresho, S. T. Chan, R. L. Lee and C. D. Upson, 'A modified finite element method for solving the time-dependent incompressible Navier–Stokes equations. Part 1: theory', *Int. j. numer. methods fluids*, **4**, 557 (1984).
13. P. M. Gresho and R. L. Sani, 'On pressure boundary conditions for the incompressible Navier–Stokes equations', *Int. j. numer. methods fluids*, **7**, 1111–1145 (1987).
14. A. C. Benim and W. Zinser, 'A segregated formulation of Navier–Stokes equations with finite elements', *Comput. Methods Appl. Mech. Eng.*, **57**, 223–237 (1986).
15. Y. M. Kim and T. J. Chung, 'Finite-element analysis of turbulent diffusion flames, *AIAA J.*, **27**, 330–339 (1989).
16. J. G. Rice and R. J. Schnipke, 'An equal-order velocity–pressure formulation that does not exhibit spurious pressure modes, *Comput. Methods Appl. Mech. Eng.*, **58**, 135–149 (1986).
17. C. Nonino and S. Del Giudice, 'An improved procedure for finite element analysis of two-dimensional elliptic flows', in C. Taylor *et al.* (eds), *Numerical Methods in Laminar and Turbulent Flow*, Pineridge, Swansea, 1985, pp. 597–608.
18. C. T. Shaw, 'Using a segregated finite element scheme to solve the incompressible Navier–Stokes equations', *Int. j. numer. methods fluids*, **12**, 81–92 (1991).
19. J. Cahouet and J. P. Chabard, 'Some fast 3D finite element solvers for the generalized stokes problem', *Int. j. numer. methods fluids*, **8**, 869–895 (1988).
20. M. Fortin and A. Fortin, 'A generalization of Uzawa's algorithm for the solution of the Navier-Stokes equations', *Commun. Appl. Numer. methods*, **1**, 205–208 (1985).
21. M. Fortin and S. Boivin, 'Iterative stabilization of the bilinear velocity-constant pressure element', *Int. j. numer. methods fluids*, **10**, 125–140 (1990).
22. J. P. Chabard, G. Pot, B. Metivet and B. Thomas, 'A numerical solution of the Navier-Stokes equations for industrial applications', Electricité de France, *EDF Report, HE-41/89.10.*, 1989.
23. M. Fortin and R. Glowinski, *Augmented Lagrangian Methods*, North-Holland, Amsterdam, 1983.
24. B. E. Launder and D. B. Spalding, 'The numerical computation of turbulent flows', *Comput. Methods Appl. Mech. Eng.*, **3**, 269 (1974).
25. V. Haroutunian and M. S. Engelman, 'On modeling wall-bound turbulent flows using specialized near-wall finite elements and the standard k–ε turbulence model', in *Advances in Numerical Simulation of Turbulent Flows*, FED-Vol. 117, ASME, New York, 1991.
26. H. C. Elman, 'Iterative methods for large, sparse, nonsymmetric systems of linear equations', *Ph.D. Thesis*, Dept. of Computer Science, Yale University, New Haven, 1982.
27. R. L. Sani, P. M. Gresho, R. L. Lee and D. F. Griffiths, 'The cause and cure(?) of the spurious pressures generated by certain FEM solutions of the incompressible Navier–Stokes equations', *Int. j. numer. methods fluids*, **1**, 17, 171 (1981).
28. R. Chandra, 'Conjugate gradient methods for partial differential equations', *Ph.D. Thesis*, Dept. of Computer Science, Yale University, New Haven, 1978.
29. R. Fletcher, 'Conjugate gradient methods for indefinite systems', in G. A. Watson (ed.), *Proc. Dundee Biannual Conf. on Numer. Analysis*, Springer, New York, 1975; see also, Lecture Notes in Mathematics, Vol.506, Springer, New York, 1976.
30. P. Sonneveld, P. Wesseling and P. M. de Zeeuw, 'Multigrid and conjugate gradient methods as convergence acceleration techniques, in D. J. Paddon and H. Holstein (eds), Multigrid Methods for Integral and Differential Equations, Clarendon Press, Oxford, 1985, pp. 117–167.
31. Y. Saad and M. H. Schultz, 'GMRES: a general minimal residual algorithm for solving non-symmetric linear systems', *Research Report 254*, Dept. of Computer Science, Yale University, New Haven, 1983.
32. Y. Saad and M. H. Shultz, 'Conjugate gradient like algorithms for solving nonsymmetric linear systems of equations', *Research Report 283*, Dept. of Computer Science, Yale University New Haven, 1983; see also, *Math. Comput.*, **44**, 417–424 (1985).
33. Y. Saad, 'Krylov subspace methods for solving large unsymmetric linear systems', *Math. Comput.* **37**, 105 (1981).
34. Y. Saad, 'Practical use of some Krylov subspace methods for solving indefinite and unsymmetric linear systems', *Research Report 214*, Dept. of Computer Science, Yale University, New Haven, 1982.
35. P. N. Brown and A. C. Hindmarsh, 'Reduced storage matrix methods in stiff ODE systems, *UCRL Report* 95088, Revision 1, Lawrence Livermore National Laboratory, 1987.
36. Y. Saad, 'GMRES: a generalized minimal residual algorithm', *SIAM J. Sci. Stat. Comput.*, **7**, 856–869 (1986).