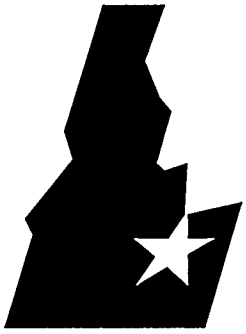


Idaho
National
Engineering
Laboratory

INEL-95/0118

October, 1995

An Investigation of Newton-Krylov Algorithms for Solving Incompressible and Low Mach Number Compressible Fluid Flow and Heat Transfer Problems Using Finite Volume Discretization

P. R. McHugh

Lockheed
Idaho Technologies Company

INEL-95/0118

An Investigation of Newton-Krylov Algorithms for Solving Incompressible and Low Mach Number Compressible Fluid Flow and Heat Transfer Problems Using Finite Volume Discretization

Paul R. McHugh

Published October 1995

Idaho National Engineering Laboratory
Lockheed Martin Idaho Technologies
Idaho Falls, Idaho 83415

Supported by the
U.S. Department of Energy
through DOE Idaho Operations Office
Contract DE-AC07-94ID13223

AN INVESTIGATION OF NEWTON-KRYLOV ALGORITHMS FOR SOLVING INCOMPRESSIBLE AND LOW MACH NUMBER COMPRESSIBLE FLUID FLOW AND HEAT TRANSFER PROBLEMS USING FINITE VOLUME DISCRETIZATION

A Dissertation

Presented in Partial Fulfillment of the Requirements for the

Degreee of Doctor of Philosophy

with a

Major in Mechanical Engineering

in the

College of Graduate Studies

University of Idaho

by

Paul R. McHugh

April 1995

Major Professor: E. Clark Lemmon, Ph.D.

AUTHORIZATION TO SUBMIT

DISSERTATION

This dissertation of Paul R. McHugh, submitted for the degree of Doctor of Philosophy with a major in Mechanical Engineering and titled "An Investigation of Newton-Krylov Algorithms for Solving Incompressible and Low Mach Number Compressible Fluid Flow and Heat Transfer Problems Using Finite Volume Discretization," has been reviewed in final form, as indicated by the signatures and dates given below. Permission is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor E Clark Lemmon Date 4-24-95
E. Clark Lemmon

Committee Members Dana Knoll Date 4/19/95
Dana A. Knoll

Clayton S Miller Date 4/20/95
Clayton S. Miller

Earl A Marwil Date 4/19/95
Earl S. Marwil

Department Administrator E Clark Lemmon Date 4-24-95
E. Clark Lemmon

Discipline's College Dean D Woodall Date 4-24-95
David M. Woodall

Final Approval and Acceptance by the College of Graduate Studies

Jeanne M Shreeve Date 5/18/95
Jean'ne M. Shreeve

ABSTRACT

Fully coupled, Newton-Krylov algorithms are investigated for solving strongly coupled, nonlinear systems of partial differential equations arising in the field of computational fluid dynamics. Primitive variable forms of the steady incompressible and compressible Navier-Stokes and energy equations that describe the flow of a laminar Newtonian fluid in two-dimensions are specifically considered. Numerical solutions are obtained by first integrating over discrete finite volumes that compose the computational mesh. The resulting system of nonlinear algebraic equations are linearized using Newton's method. Preconditioned Krylov subspace based iterative algorithms then solve these linear systems on each Newton iteration. Selected Krylov algorithms include the Arnoldi-based Generalized Minimal RESidual (GMRES) algorithm, and the Lanczos-based Conjugate Gradients Squared (CGS), Bi-CGSTAB, and Transpose-Free Quasi-Minimal Residual (TFQMR) algorithms. Both Incomplete Lower-Upper (ILU) factorization and domain-based additive and multiplicative Schwarz preconditioning strategies are studied. Numerical techniques such as mesh sequencing, adaptive damping, pseudo-transient relaxation, and parameter continuation are used to improve the solution efficiency, while algorithm implementation is simplified using a numerical Jacobian evaluation.

The capabilities of standard Newton-Krylov algorithms are demonstrated via solutions to both incompressible and compressible flow problems. Incompressible flow problems include natural convection in an enclosed cavity, and mixed/forced convection past a backward facing step. Additionally, matrix-free Newton-Krylov implementations are constructed by approximating the Jacobian-vector products appearing in the Krylov algorithms with finite difference approximations. Performance of the matrix-free implementation is found to depend upon problem size, problem nonlinearity, and Krylov algorithm selection. Higher order accurate solutions for mixed convection flow are obtained

using the third order cubic upwind interpolation scheme (CUI) convection scheme with a defect correction procedure and several CPU performance enhancement techniques. Solution efficiency for high Reynolds number forced convection flow is investigated using a discrete pressure equation formulation, alternative cell ordering strategies, and pseudo-transient relaxation. Solutions to the compressible flow problem, consisting of low Mach number subsonic flow past a backstep, found ILU preconditioners to be strongly sensitive to cell ordering and less effective than domain based preconditioners at low Mach numbers.

ACKNOWLEDGEMENTS

I express my sincere thanks to Dr. Dana A. Knoll, my dissertation advisor, for his time, inspiration, and advice during the course of this dissertation. His encouragement and assistance has been a continuous source of motivation and energy during each phase of my graduate study.

I would also like to acknowledge Dr. E. Clark Lemmon, Dr. Clayton S. Miller, and Dr. Earl S. Marwil for their work as members of my graduate committee. I thank them for the considerable time and effort they expended in guiding and assisting me through the University of Idaho graduate program.

Several co-workers and collaborators at the Idaho National Engineering Laboratory (INEL) also deserve recognition. First of all, I would like to express my appreciation to Dr. John D. Ramshaw, who has truly been a mentor for me during my tenure at the INEL. I thank him for his encouragement and support, which helped motivate me to pursue this doctoral degree. I thank Dr. Richard W. Johnson for his assistance with higher-order accurate finite volume discretization schemes. Additionally, the assistance provided by Mr. Vincent A. Mousseau and Mr. Paul G. Jacobs with regard to domain-based preconditioning schemes is greatly appreciated. I thank Dr. C. H. Chang for many helpful discussions and suggestions regarding this research. I also extend my appreciation to Dr. Glen A. Mortensen for volunteering to read the initial draft of this dissertation, and for the many valuable suggestions made afterwards. I also like to thank Dr. Mark J. Oliver for helpful early discussions regarding Krylov iterative algorithms and preconditioning. Additionally, the support and encouragement of Dr. Rod W. Douglass, Dr. Paul E. Murray, Mr. Glen A. Hansen, and Dr. Hoa D. Nugyen is appreciated.

I would also like to acknowledge the support of the INEL Long Term Research Initiative in Computational Mechanics (LTRI-CM). This project, lead initially by Dr. John

D. Ramshaw and currently by Dr. Rod W. Douglass, provided me the opportunity to pursue this research. Additionally, I thank the University of Idaho and the INEL for the opportunity to pursue this degree while residing in Idaho Falls.

Finally, I would like to thank my family for their never-ending moral support and encouragement. This thanks is especially extended to my wife, Carrie, for her abundant patience and understanding during the course of this study.

This dissertation is dedicated to my parents, Paul J. and Margaret A. McHugh

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

A	General system matrix employing sparse nonzero diagonal storage scheme
AM	General preconditioning matrix employing sparse nonzero diagonal storage scheme
aa	Perturbation constant
b	General right hand side vector
bb	Perturbation constant
C_{f_w}	Wall friction coefficient
$\tilde{c}_p$	Specific heat capacity at constant pressure
$\tilde{c}_v$	Specific heat capacity at constant volume
D	Diagonal matrix
D_H	Hydraulic diameter
d	Jacobian is evaluated every *d* Newton steps
F	Discrete governing equations vector
E	Error matrix
f	Individual discrete governing equation
$\tilde{\mathbf{g}}$	Gravitational acceleration vector
Gr	Grashof number = $\frac{\tilde{\beta}\tilde{g}\Delta\tilde{T}\tilde{L}^3}{\tilde{\nu}^2}$
H	Upper Hessenberg matrix
INUM	Finite volume cell number
INUMD	Finite volume cell number of dependency
IEQ	Equation number
IEQD	Equation number of dependency
i	Coordinate index for *x*-direction or general index depending upon context

$\hat{\mathbf{i}}$	Unit vector in x-direction
idiag	Vector array containing offsets with respect to the main diagonal for stored non zero diagonals
J	Jacobian matrix
Q^u	Flux of u-momentum.
j	Coordinate index for y-direction or general index depending upon context
$\hat{\mathbf{j}}$	Unit vector in x-direction
k	Krylov subspace dimension, thermal conductivity, ILU fill-in level, or general index depending upon context
K_k	Denotes Krylov subspace of dimension k.
L	Lower diagonal matrix
L	Characteristic length
M	Mach number
m	Inner iteration counter or general index depending upon context
N	Dimension of linear system
NC	Number of finite volume cells
NEQ	Number of governing equations
NF	Number of function evaluations per computational cell
$\hat{\mathbf{n}}$	Unit normal vector
n	Newton iteration counter
nx	Number of cells in x-direction
ny	Number of cells in y-direction
Nu	Nusselt number
P	Preconditioning matrix
p	Dimensionless pressure or general index depending upon context

Pec	Peclet number = $Re\,Pr = \frac{\tilde{V}\tilde{L}}{\tilde{\alpha}}$
Pr	Prandtl number = $\frac{\tilde{\nu}}{\tilde{\alpha}}$
$\mathbf{r}$	Inner iteration residual vector
Ra	Rayleigh number = $Gr\,Pr$
Re	Reynolds number = $\frac{\tilde{V}\tilde{L}}{\tilde{\nu}}$
$\mathbf{R}$	Restriction/prolongation matrix or error matrix depending upon context
R_n^i	Scaled inner iteration residual norm
R_n^o	Relative update for n^{th} Newton iteration
S	Surface area
S_J	Set of row and column index pairs that compose the Jacobian matrix sparsity pattern
s	Newton iteration damping parameter or general index depending upon context
T	Dimensionless temperature
$\Delta\tilde{T}$	Characteristic temperature difference
Δt	Dimensionless time step
$\mathbf{U}$	Upper diagonal matrix
u	Dimensionless velocity in x-direction
$\mathbf{V}$	Diagonal matrix whose elements are the volumes of the computational cells
V	Characteristic velocity scale or volume depending upon context
v	Dimensionless velocity in y-direction
w	Arbitrary Krylov vector
x	Cartesian coordinate variable
$\mathbf{x}$	State variable vector

Δx	Grid spacing in x-direction
$\delta \mathbf{x}$	Newton iteration update vector
Δx_j	Perturbation in the *jth* component of $\mathbf{x}$
Δy	Grid spacing in y-direction
y	Cartesian coordinate variable

Greek Symbols:

α	Newton iteration damping parameter
$\tilde{\alpha}$	Thermal diffusivity
$\tilde{\beta}$	Coefficient of thermal expansion
ε	Perturbation constant
ε^n	Tolerance parameter for Krylov iteration
γ	Ratio of specific heat capacities = $\tilde{c}_p/\tilde{c}_v$
Φ	Lagrangian interpolation polynomial for x-direction
ϕ	Orientation of x-axis with respect to horizontal
η	Time step control constant
$\tilde{\mu}$	Dynamic viscosity
$\tilde{\nu}$	Kinematic viscosity
θ	General dimensionless scalar quantity
ρ	Dimensionless density
σ	Viscous stress tensor
τ_w	Wall shear stress
ω	Relaxation parameter
Ψ	Lagrangian interpolation polynomial for y-direction

Superscripts:

i	Refers to inner iteration
n	Newton iteration number
o	Refers to outer iteration
sub	Refers to sub-domain

Subscripts:

0	Reference/Initial value
AM	Refers to preconditioning matrix
be	Break even value
bulk, b	Refer to bulk values
C	First upstream value
COL	Matrix column number
cl	Centerline value
D	Downstream value
E, e	East (right) values
INUM	Finite volume cell number
INUMD	Finite volume cell number of dependency
IEQ	Equation number
IEQD	Equation number of dependency
i	Inlet value or vector component number depending upon context
J	Refers to Jacobian matrix
k	Krylov iteration number or general index depending upon context

L	Left
max	Maximum value of a quantity
NC	Number of finite volume cells
N, n	North (top) values
n	Newton iteration number or north face depending upon context
o	Refers to outer iteration
P	Cell centered value
R	Right
ROW	Matrix row number
S, s	South values
U	Second upstream value
$wall, w$	Refer to wall values
W, w	West (left) values

Operators:

Lower case bold	Vector quantity
Upper case bold	Matrix/Tensor quantity
∇	Gradient operator
$\|\ \|$	Absolute value
$[\ ]^T$	Transpose of [ ]
$\bar{[\ ]}$	Indicates that [ ] is an averaged quantity
$\tilde{[\ ]}$	Indicates that [ ] is dimensional
$\|\| \bullet \||, \|\| \bullet \||_2$	Euclidean norm (L_2 norm)
$\|\| \bullet \||_\infty$	L_∞ norm

CHAPTER 1

INTRODUCTION

The objective of this dissertation is the development, implementation, and investigation of robust yet practical numerical algorithms for solving strongly coupled, nonlinear systems of partial differential equations that arise in the field of computational fluid dynamics. Of specific interest are the steady state, incompressible and compressible Navier-Stokes and energy equations describing the flow of a laminar, Newtonian fluid in two-dimensions [see 1 and 2]. These equations are solved numerically, in primitive variable form, by integrating them over discrete finite volumes that compose a Cartesian computational mesh, which approximates a given problem geometry. This discretization process results in a system of nonlinear algebraic equations. This study investigates fully coupled Newton-Krylov algorithms for solving these nonlinear algebraic equations. On each iteration, Newton's method [see 3] is used to first linearized the discretized system of nonlinear algebraic equations, and then a preconditioned Krylov subspace based iterative algorithm is used to solve the resulting linear system for the new solution update. The Krylov subspace based algorithms considered in this study include the Generalized Minimal RESidual (GMRES) algorithm [4], the Conjugate Gradients Squared algorithm (CGS) [5], the Bi-CGSTAB algorithm [6], and the Transpose-Free Quasi-Minimal Residual (TFQMR) algorithm [7]. These different Krylov algorithms and others are described in the Appendix.

Effective and efficient implementation of this basic process is accomplished by reducing computer memory requirements and by employing various convergence enhancement techniques to improve CPU performance. The capabilities of this implementation are demonstrated in solving the coupled equations describing fluid flow and

heat transfer in their primitive variable forms. Specifically, the problems of natural, mixed, and forced convection are studied assuming incompressible flow, while for a compressible fluid the problem of low Mach number subsonic flow is studied. The capability to handle low Mach number (low speed) regimes, in which a fluid is normally considered incompressible (Mach number less than 0.3 [8]), is important in situations such as: a low Mach number region is imbedded within a high speed flow, some thermally driven flows , and other flow situations where density variations are important, i.e. chemically reacting flow and flows with significant heat transfer.

An important contribution made by this dissertation is the documentation of useful observations and guidelines regarding the practical and efficient use of Newton-Krylov algorithms for solving challenging CFD problems of the type mentioned above. These observations and guidelines stem from the research performed during the course of this dissertation [also see 9, 10, 11, 12, 13, and 14]. For instance, research presented in this dissertation investigates many performance enhancement techniques such as: adaptive damping, mesh sequencing, pseudo-transient calculations, suitable convergence criteria, effective preconditioning and cell ordering strategies, and the use of alternative formulations such as the discrete pressure equation in the case of incompressible flow. Specifically, research contained in this dissertation was among the first studies to employ recently developed Krylov algorithms within the context of Newton-Krylov algorithms designed for solving fluid flow and heat transfer problems [12, 13]. This dissertation research represents, to the authors knowledge, the first detailed comparison of Lanczos-based and Arnoldi-based Krylov algorithms within so called "matrix-free" Newton-Krylov implementations for solving steady state incompressible fluid flow and heat transfer applications [10, 13]. Furthermore, practical "matrix-free" Newton-Krylov implementations using pseudo-transient relaxation are demonstrated and discussed herein, where the cost of periodic Jacobian and preconditioner evaluations are amortized over many pseudo-transient Newton steps (see

Reference 15 for another example of the successful application of this specific research). Additionally, research presented in this dissertation first investigated algorithm efficiency issues associated with the use of the third order accurate cubic upwind interpolation (CUI) convection discretization scheme within a Newton-Krylov algorithm. This research studied techniques such as mesh sequencing and defect correction for improving the efficiency in obtaining higher order accurate solutions [also see 11 and 14]. Finally, with regard to direct steady state (i.e., no time stepping) Newton-Krylov solutions of compressible flows, this research first demonstrated the superiority of domain-based preconditioning strategies over more conventional incomplete lower upper (ILU) type preconditioning schemes at low Mach numbers [also see 16]. The Mach numbers used in this latter study were well below those considered elsewhere using Newton-Krylov type algorithms.

Several different computational resources were employed during the course of this investigation. In general, the best computational resources available to the author at the time were employed. Consequently, initial numerical studies were performed on either a CRAY X-MP/216 computer or an IBM RISC System 6000 Model 320 workstation depending upon the CRAY availability and cost. The CRAY X-MP/216 computer had two processors, 16 megawords of main memory, an 8.5 nanosecond clock time, and a theoretical peak rate of 470 megaflops (million floating point operations per second). Note that although the CRAY computer had two processors, it was only used in a serial mode. The IBM workstation had 16 megabytes of main memory and operated at 20 MHz, with a theoretical peak rate of 40 megaflops. Latter numerical studies were performed on HP 9000/735 workstations. These workstations operated at 99 MHz with a theoretical peak rate of 200 megaflops. Additionally, a somewhat faster upgraded HP 9000/735 workstation was also employed that operated at 125 MHz with a theoretical peak rate of 250 megaflops. The memory available on these HP workstations varied from 80 megabytes up to 288 megabytes. Calculations on the CRAY computer were run in single precision, while the workstation calculations were

performed using double precision. A comparison of the performance of these different computers and others (excluding the upgraded HP 9000/735 mentioned above) using standard linear equations software (i.e., LINPACK [17]) is presented by Dongarra in Reference 18. Note that in solving the "LINPACK Benchmark" problem defined in Reference 18, the actual megaflop rates acheived by the CRAY, HP, and IBM computers listed above were 143, 41, and 9, respectively [see 18].

Section 1.1 below presents background and motivational information that discusses the relative merits of a fully coupled Newton solution algorithm compared with more conventional segregated and approximate factorization type implicit solution algorithms. Section 1.2 identifies and describes related work using Newton-Krylov solution techniques for fluid flow and heat transfer applications. An itemization of subsequent Chapters contained herein is given in Section 1.3 in order to provide information regarding dissertation organization.

1.1. BACKGROUND AND MOTIVATION

The goal of this section is to provide general background information, and to describe the motivation behind this research. As such, this goal requires a general discussion of previous research from which this work has evolved, and explanations why alternative options, including different solution algorithms and techniques, were not considered.

The motivation behind using a fully coupled, Newton iteration follows from previous research comparing this technique with other nonlinear iterative techniques. This research demonstrated that Newton's method used with direct linear solution methods (direct-Newton algorithms) can be considerably more robust than other more conventional segregated, and approximate factorization type solution techniques. These latter solution techniques have historically been popular in the finite volume and finite difference communities [see 19, 20,

21, and 22]. Some examples of the use of direct-Newton solution techniques for incompressible flow applications can be found in References 23, 24, 25, 26, 27, 28, 29, and 30. Specifically, References 25 and 29 concluded that fully coupled Newton-like solutions offer advantages over segregated solution algorithms such as the popular SIMPLE algorithm [21], although typically requiring considerably more computer memory resources. They suggested that the advantages of retaining full coupling (simultaneous solution of all variables) between the equations is especially important in strongly nonlinearly coupled problems such as natural convection at high Rayleigh numbers [25, 29], where the momentum equations are coupled to the energy equation through the buoyancy force term in the momentum equations using the Boussinesq approximation [31]. Some examples of fully coupled direct-Newton solutions for compressible flow applications are given in References 32, 33, 34, and 35. In particular, Reference 32 found advantages in the use of a direct-Newton method compared with a method using an approximate factorization of the Jacobian matrix based upon a spatial directional splitting that allowed the inverse to be computed using an alternating direction sequence of block tridiagonal solves [see 19 and 36]. It was suggested that these advantages may be especially apparent for low subsonic Mach numbers; situations where these latter algorithms sometimes encounter convergence difficulties [32]. Note that the direct-Newton references cited above all employed a finite volume or finite difference type discretization scheme. However, use of fully coupled solution techniques for solving fluid flow problems have been popular in the finite element community for some time [e.g., 37, and more recently 38 and 39]. Additionally, fully coupled Newton's method solution techniques have also been successfully applied in other disciplines such as the modeling of chemically reacting flow [40, 41, 42, 43], and edge plasma fluid modeling [44]. These disciplines typically exhibit strong nonlinear coupling between equations, and so they also benefit from a fully coupled solution approach.

The rise in popularity of these fully coupled, direct-Newton solution techniques is primarily due to rapid advances in computer technology, especially large increases in available "in core" computer memory. Direct solution methods, however, still suffer from large computer memory requirements, especially for large two-dimensional and three-dimensional problems. The primary focus of the numerical algorithm research in this dissertation is the efficient use of preconditioned Krylov subspace based iterative algorithms to solve the linear equations that arise on each Newton step; specifically, recently developed preconditioned conjugate gradient-like algorithms (see the Appendix). These algorithms enable the sparse structure of the Jacobian matrix to be exploited in order to alleviate the large computer memory requirements, while maintaining the strong convergence characteristics of Newton's method. The motivation for selecting these iterative linear solvers is discussed further in the Appendix. Note that related work using Newton-Krylov type solution algorithms is discussed in Section 1.3 below. For information and examples regarding the use of Krylov subspace algorithms within other types of solution algorithms for solving fluid flow type problems see References 45, 46, 47, 48, 49, 50, 51, 52, and 53. For information and examples regarding the use of Newton's method with other types of linear equation solvers see References 54, 55, 56, 57, and 58.

Besides memory considerations, other disadvantages in the use of a Newton type solver include the often high cost of forming and factoring the Jacobian matrix on each Newton step, and the sometimes small radius of convergence of Newton's method, which often makes convergence strongly sensitive to the initial guess. Consequently, the secondary focus of this research considers various numerical techniques to address these remaining disadvantages associated with the use of Newton's method. This dissertation combines these ideas with the goal of obtaining efficient and robust algorithms for solving the coupled equations describing both incompressible and compressible fluid flow and heat transfer.

Compensating for its disadvantages, Newton's method offers the advantage of full coupling between all variables and all equations so that all the unknowns are solved for simultaneously. This is in contrast to a segregated solution approach [e.g. 21, 22, 59, and 60] where the governing equations are solved sequentially in a decoupled and iterative fashion. Approximate factorization techniques solve for all variables simultaneously, but differ from a true fully coupled, simultaneous solution approach in that the iteration matrix is approximately factored so as to reduce memory requirements, and to facilitate matrix inversion. Examples include the Beam-Warming algorithm cited above and a similar algorithm proposed by Briley and McDonald [19, 36, 61]. Stone's Strongly Implicit Procedure (SIP) [62], which is based upon a type of Incomplete Lower-Upper (ILU) factorization, is another example of an approximate factorization technique. Newton's method makes no such approximations when solving the linear systems that arise on each Newton step. The segregated and approximate factorization type solution approaches are discussed below in order to further distinguish these approaches from a fully coupled Newton's method approach.

The segregated solution approach solves each equation [i.e. continuity (or approximate pressure equation), momentum, and energy conservation equations] sequentially, in a decoupled manner, for the variable unknowns corresponding to that equation. This means that only the variables corresponding to the equation of interest are allowed to vary, while variables corresponding to other equations are held constant. Thus, only a local reduction of the error (corresponding to the individual equation of interest) is performed, in contrast to a fully coupled or simultaneous approach where the error is reduced globally. Global error reduction in a segregated solution approach is obtained by repeating this solution sequence in an iterative manner until all equations have converged to the desired level of accuracy. Consequently, systems of equations that are strongly coupled may require many iterations and, in some instances, severe underrelaxation to achieve global error

reduction. In fact, the selection of optimal underrelaxation parameters can significantly influence the performance of a segregated solution approach [24, 25, 29, 30]. The use of a fully coupled or simultaneous approach, to a large extent, reduces or eliminates this sensitivity to parameter estimation [24, 25, 29, 30]. This feature is another significant advantage of a fully coupled solution approach.

The motivation for decoupling the different governing equations in a segregated solution technique is twofold. The first motivating factor is the reduction of computer memory requirements. This is accomplished by replacing the large global nonlinear system with a sequence of smaller nonlinear systems associated with the individual governing equations. The second motivating factor is algorithm convergence and performance. The smaller nonlinear systems are linearized in some suitable fashion, and then are often solved iteratively because of memory limitations. The performance of many of the iterative techniques used by these segregated type solvers is tied to the diagonal dominance of the linearized system [see 63]. Decoupling the individual conservation equations enables the smaller individual linear systems (corresponding to each individual equation) to be constructed so as to maintain diagonal dominance [see 21], thereby enabling the convergence of these individual linear systems. Some popular examples of the segregated solution approach include the SIMPLE algorithm [21], the ICE algorithm [22], and the method of artificial compressibility [59]. These sequential, segregated solution algorithms are very effective for many different fluid flow and heat transfer applications; although past research cited above for strongly coupled systems of equations suggests that a simultaneous solution technique may sometimes be preferable.

One general class of simultaneous solution techniques for nonlinear systems is referred to as parallel chord iteration [3]. Picard iteration and Newton iteration are members of this general class of nonlinear iterative techniques [3]. Picard iteration differs from Newton's method in that the iteration matrix represents only linear coefficients of the

nonlinear operator, whereas for Newton's method the iteration matrix is the Jacobian matrix [3]. Picard iteration often has a larger radius of convergence than Newton's method, but exhibits much slower convergence. Consequently, researchers have considered combining these two schemes to form a hybrid type of nonlinear algorithm [e.g., 24, 25, 64, and 65]. Approximate factorization techniques can be applied to either a Picard type iteration or a Newton type iteration. Note that the latter choice can also be viewed as a quasi-Newton or approximate Jacobian technique [3, 66]. The goal in an approximate factorization technique is to reduce the memory required to store the iteration matrix and/or its factorization. In two-dimensions, one possible technique is to include coupling between variables along a given spatial direction for one factor, and then include coupling between variables along the alternate spatial direction for the second factor. The product of these two factors is then used to approximate the true iteration matrix. The advantage in this technique is that the individual factors have a much simpler structure than the original matrix (i.e., often block tridiagonal) and the action of the inverse can be computed from a sequence of alternating direction solves (usually block tridiagonal), thereby avoiding the necessity of storing a full factorization of the iteration matrix. However, difficulties associated with this technique include: 1.) the factorization is only an approximation to the iteration matrix and so use of this technique will likely result in slower convergence; 2.) the factorization assumes a rectangular mesh with two independent directions, and so the technique is not easily applicable to unstructured grids. 3.) the technique is typically applied in a time marching manner so that for non time accurate solutions, the time step acts as a sort of relaxation parameter that must be tuned for optimal performance. An example of this technique is the popular Beam-Warming method [36] and the algorithm of Briley and McDonald [61]. A second approximate factorization technique is to only compute a partial or incomplete factorization of the iteration matrix. An example of this approach is Stone's SIP algorithm [62]. Once again the efficiency of this algorithm is dependent upon the accuracy of the

approximate factorization, a difficulty that a fully coupled, simultaneous solution avoids by accurately inverting the iteration matrix.

It must be pointed out, before proceeding, that the intent of the above discussion is not to criticize the segregated and approximate factorization type algorithms. Rather, the purpose is to point out reasons why a fully coupled solution approach, with no approximate factorizations, may be an attractive alternative for some strongly coupled fluid flow and heat transfer applications. The applications studied in this dissertation were selected because, based upon the information from the references cited above, they should be good candidates for a fully coupled type solution algorithm.

The reason why Newton's method was selected in this work, from the general class of simultaneous solution techniques discussed above, is due to its characteristic quadratic convergence behavior when the initial guess lies within the radius of convergence of the algorithm [see 3 and 66]. This powerful feature of Newton's method enables the error to be reduced in a quadratic fashion as the true solution is approached, thereby enabling strong convergence (often to machine round-off) during the last few iterations. This behavior is often in contrast to a segregated solution approach where the high frequency errors may be eliminated quickly, but where the long wavelength (low frequency) errors may persist for many iterations because of the equation decoupling. A fully coupled, simultaneous solution approach eliminates both high and low frequency errors equally well.

Another type of fully coupled, simultaneous solution technique is the method of steepest descent [see 25 and 66]. This technique uses equation sensitivity information contained in the Jacobian and current error information contained in the residuals to predict a new solution search direction [66]. The advantage in this approach is that the inverse of the Jacobian is not required to compute the new search direction. The disadvantage, however, is that it can be difficult to specify the scale factors that determine how far one should move along a given search direction, although several schemes have been developed for making

this determination [e.g., see 66]. These schemes greatly improve the global convergence properties of the algorithm. Unlike Newton's method, however (which also yields a descent search direction), the method of steepest descent does not converge in a quadratic manner in the local vicinity of the solution [66]. Consequently, researchers have combined steepest descent schemes with Newton's method to obtain strongly converging algorithms with improved global convergence behavior [66, 67, 68, 69, 70, and 71]. Since these hybrid algorithms are not the focus of this investigation, much simpler techniques are used to enhance the convergence of Newton's method. These simpler techniques are described in Chapter 3. Although, these simple techniques are effective for the problems considered in this dissertation, it is acknowledged that use of the hybrid algorithms cited above may further enhance the global convergence properties of the overall algorithm, and so should be included in any future investigation of this sort.

For completeness, it is instructive to mention several well known solution methods commonly used to alleviate the high cost and difficulty associated with forming and factoring the Jacobian matrix [see 3, 66, 72, and 73]. These quasi-Newton techniques include various secant methods and the modified/simplified Newton's method. The convergence of these "approximate" Newton methods is at best superlinear, but CPU efficiency features of these algorithms often compensate for their slower convergence. Although these techniques are not included in this investigation, they are commonly used to improve the CPU efficiency of a Newton-type algorithm. As such, an explanation why they are not used in this investigation is warranted.

One of the best known secant methods is Broyden's method [74]. This method enables the Jacobian inverse to be updated simply on each iteration using the Sherman-Morrison-Woodbury formula [3]. The advantage in this technique is that after the initial evaluation of the Jacobian and its inverse, they need not be explicitly computed again [66, 74]. The difficulty, however, is that the inverse cannot be assumed sparse and so problems

associated with memory requirements once again arise. MacArthur developed a modified, reduced-memory, version of Broyden's method [25], although additional memory in excess of his direct-Newton algorithm memory requirements were still needed for implementation. Memory considerations explain why this type of algorithm is not considered here. Another example of the use of a Broyden-based update is the work of Edwards and McRae [75], in which this type of quasi-Newton technique is used to accelerate an approximate factorization type algorithm [Symmetric Line Gauss-Seidel (SLGS)]. Although the memory requirements for this algorithm is considerable less than a direct-Newton algorithm, the memory needed grows with the number of quasi-Newton steps. Additionally, the algorithm is still based upon an approximate factorization type algorithm. Thus, the difficulties discussed above regarding approximate factorization type algorithms also apply to this algorithm.

The second solution method mentioned above is the modified/simplified form of Newton's method. This method consists of freezing the Jacobian and its factorization for two or more iterations in order to reduce CPU costs [e.g., see 3, 23, 26, 32, 33, 40, 41]. However, when using a conjugate gradient-like algorithm to solve the linear systems that arise on each Newton step, no savings with regard to these linear systems are realized by this technique, thereby lessening its benefits. For this reason, this method is not studied in this dissertation. Although, other techniques for simplifying the Jacobian evaluation and reducing the CPU cost and memory requirements associated with its use are employed. These techniques are described in Chapter 3.

1.2. RELATED WORK

The purpose of this section is to identify and briefly describe recent related work using fully coupled Newton-Krylov methods for solving fluid flow and heat transfer problems. In order to more efficiently accomplish this purpose, incompressible flow and

compressible flow applications are discussed separately in the next two subsections. Although finite volume discretization is employed in this work, research using other types of discretization schemes such as finite elements and spectral methods are also included in this literature review. The references cited below demonstrate that Newton-Krylov solution techniques in computational fluid dynamics are currently a very active research topic, with many of the references having been published recently.

1.2.1. Incompressible Flow

Applications of Newton-Krylov techniques for finite volume/difference solutions of incompressible flow have recently been investigated by researchers at the University of Waterloo in Waterloo, Ontario, Canada [65, 76, and 77], and by Gropp and Keyes [78]. These investigations, however, did not include heat transfer effects. Gropp and Keyes used a stream function-vorticity formulation with a Newton-GMRES algorithm for investigating domain decomposition and defect correction (a type of approximate Jacobian technique described in Chapter 3) ideas in solving flow past a backward facing step [78]. Other Lanczos based Krylov algorithms were not included in this interesting study. Chin, D'Azedevedo, Forsyth and Tang solved the primitive variable form of the incompressible flow equations for several cavity and channel type flow problems [76]. This work investigated several Newton-Krylov algorithms using different Krylov subspace solvers (CGS, Bi-CGSTAB, and Orthomin) [76] with Incomplete Lower-Upper (ILU) factorization type preconditioning. Recall that the Appendix describes these different Krylov subspace solvers. Different cell ordering, time stepping, and defect correction type strategies were considered with regard to algorithm performance. Although the GMRES algorithm was not included in this initial study, GMRES was compared with the Bi-CGSTAB algorithm in a subsequent study [77]. This paper found that for the test problems and implementations

employed, the Bi-CGSTAB algorithm performed better [77]. A third study at the University of Waterloo compared and contrasted a Picard-type iteration, a full Newton iteration, and a hybrid combination of these two schemes using the Bi-CGSTAB algorithm [65]. This study found advantages in the use of the hybrid scheme for the incompressible flow problems considered.

Examples of finite element solutions of incompressible fluid flow using Newton-Krylov solution algorithms are given in References 64, 67, 68, 79, and 80. In addition, Einset and Jensen considered nearly Boussinesq flow in which compressibility effects were neglected except in the thermal expansion of the gas [81]. Although the overall Newton-Krylov algorithm structure is independent of the particular discretization scheme employed, actual implementation and performance of the various Newton-Krylov algorithms will likely be different for different discretization schemes. Comparison of performance data is therefore only strictly valid if the same discretization schemes are employed. However, much useful qualitative information can be gleaned from this finite element research. Carey et al. [68], Einset and Jensen [81], and Howard et al [64] compared iterative solution techniques with a direct frontal solution method [82]. In general, they found that the various Krylov iterative techniques considered were preferable to the use of the frontal method provided the frontal 'width' was large enough (approximately 500 in Reference 81) and effective preconditioning was employed. Among these references, Curfman [80], Howard et al. [64], and Einset and Jensen [81] included the effects of heat transfer, but recall that the model of Einset and Jensen was not truly incompressible. Curfman considered many recent Krylov solution algorithms (CGS, Bi-CGSTAB, Bi-CGSTAB2, TFQMR, and GMRES) in an interesting and useful study that employed a subsidiary pressure equation in place of the incompressible continuity equation [80]. Curfman's work also studied several Newton-like algorithms, which were found to converge more rapidly than a segregated type solution algorithm in solving several different nonlinear fluid flow problems [80]. Howard et al. [64]

investigated both two- and three-dimensional free convection flow using both Newton and Picard iteration. Several different Krylov algorithms were considered (CGS, GCR, and BCG), although several of the more recently developed algorithms were not yet available. Brown and Saad, although not including heat transfer, investigated the use of matrix-free type implementations of Newton-Krylov solution techniques by replacing matrix-vector products with finite difference projections [67]. This study solved a single fourth order scalar equation for the stream function and employed Arnoldi-based Krylov techniques (GMRES, IOM) [67].

An example using spectral methods to solve incompressible fluid flow using a pressure equation formulation is given in Reference 83. This study did not include heat transfer effects or some of the more recently developed Lanczos-based Krylov algorithms.

This dissertation builds upon previous work cited above in investigating Newton-Krylov solution techniques for solving incompressible fluid flow and heat transfer using finite volume discretization. An important difference in this dissertation is the inclusion of heat transfer effects, especially in the context of natural and mixed convection type problem. Additionally, this dissertation includes recently developed Krylov algorithms within an inexact Newton iteration that have not been included in many of the references above. Furthermore, this dissertation investigates matrix-free type implementations of Newton-Krylov methods for solving the primitive variables form of the incompressible Navier-Stokes and energy equations using both Arnoldi-based and Lanczos-based Krylov algorithms. Successful implementations of this sort for incompressible flow have previously been considered by Brown and Saad [67]. However, they did not consider any Lanczos based techniques, and they solved a single fourth order equation for the stream function using finite element discretization [67]. Matrix-free implementations, if successful, offer many significant advantages including both memory savings and CPU efficiency improvements. These potential advantages are discussed further in Section 3.2.4, and applications of this

implementation are demonstrated in Chapter 4. The use of higher-order differencing within the context of a defect correction scheme is also considered in this dissertation. Other researchers cited above have employed various defect correction schemes [76 and 78], but they have not used the third order accurate Cubic Upwind Interpolation convection scheme employed in this study (see Chapter 3). Finally, this document enables the overall Newton-Krylov algorithm efficiency to be studied at a level of detail that is typically not allowed in many journal or conference papers. This feature enables more detailed consideration of the various performance enhancement techniques that are described in Chapter 3.

1.2.2. Compressible Flow

Finite volume solutions of viscous, compressible flow using Newton-Krylov solution techniques have been presented in References 84, 85, 86, 87, 88, 89, and 90. Venkatakrishnan used an implicit time marching technique for transonic and subsonic (Mach number of approximately 0.5) airfoil calculations [84]. This implicit technique reduced to Newton's method as the time step approached infinity provided no approximations were made in the analytically derived Jacobian. However, Venkatakrishnan did simplify certain computationally intensive Jacobian terms and so a full Newton's method was not used. Included in this work were an explicit turbulence model, a study of both ILU and block diagonal preconditioners (ILU tended to perform better), a comparison of GMRES and Chebychev linear solvers (GMRES seemed to perform better), and various performance enhancement techniques. This work was latter extended to unstructured grids by Venkatakrishnan and Mavriplis [85], where again ILU preconditioning was favored over other matrix-splitting type preconditioners. This study computed transonic flow past an airfoil, and include low subsonic flow (Mach number approximately equal to 0.2) [85]. An implicit time marching technique similar to the one described above was also employed by

Ajmani and Liou [86], again using the GMRES algorithm. This investigation compared this solution technique with other more conventional solvers in CFD codes. Several different preconditioner options were studied. The more successful preconditioners included line Gauss-Seidel (LGS), Lower-Upper Symmetric Successive Over Relaxation (LUSSOR), and a block ILU preconditioner. Both transonic and hypersonic test problems were considered. This work was extended by Ajmani, Ng, and Liou [87] to low subsonic flow past a backward facing step (inlet Mach number of approximately 0.1). Much higher values of the Courant number were considered during the implicit transient calculations in this work than the previous study. It was concluded that the preconditioned GMRES algorithm was more efficient than the more standard Line Gauss-Seidel Relaxation algorithm (LGSR) [87]. This investigation was continued by Ajmani, Liou, and Dyson [88] in an investigation of parallel implementations on distributed memory machines. Additionally, supersonic calculations using Newton's method with the CGS algorithm have been conducted by Orkwis [89, 90]. These articles studied ILU preconditioning and the effects of using an approximate Jacobian on algorithm convergence (quadratic convergence was lost). Investigations using Newton-Krylov solution techniques for inviscid compressible flow calculations can be found in References 91, 92, and 93. Additionally, related articles for reacting flow type calculations are found in References 94 and 95. Although these studies do not solve the same physical problems of interest in this work, the features of their solution algorithms are of interest.

Finite element solutions of viscous compressible fluid flow using Newton-Krylov solution algorithms have been investigated recently by several Canadian researchers [see 96, 97, 98, 99, and 100]. Dutto [96] studied various node ordering strategies and their effect on several nonlinear Krylov algorithms (GMRES, Bi-CGSTAB, and CGS) in solving two-dimensional, transonic and supersonic, external flow problems. The nonlinear Krylov algorithms were derived by using the matrix-free approximation described in Chapter 3 to approximate matrix-vector products appearing withing the Krylov iteration, and by restricting

the number of allowed Krylov iterations per Newton step so that the state vector and Jacobian are typically updated before the linear system is solved. This nonlinear Krylov iteration is in contrast to a linear Krylov iteration where the linear systems are solved to the desired tolerance before updating the state vector and Jacobian. Block ILU and block diagonal preconditioners were considered. A time accurate marching technique was used to obtain both transient and steady state solutions. It was found that the nonlinear CGS algorithms did not perform well compared to the other nonlinear Krylov algorithms. This observation was also noted within the context of a matrix-free, inexact Newton iteration in References 10 and 13, the results of which are given in Chapter 4. This work of Dutto was extended in Reference 99 to consider parallel aspects of block diagonal preconditioning using various partitioning strategies. Three-dimensional, internal flow problems have been investigated in References 97, 98, and 100 using a time accurate marching technique to obtain steady state solutions. A hybrid viscosity unloading scheme was used to enable the use of larger time steps. Dutto's work employed a simplified energy equation (representing constant enthalpy), from which the density was obtained in a lagged manner after a Newton-Krylov solve for the flow velocities and pressure. A turbulence model was also included for some calculations in a lagged manner. An important focus of this work was parallel implementations of the different portions of the calculation. A matrix partitioning strategy based upon matrix sparsity and not problem geometry (i.e., domain decomposition) was used to enable parallel implementation on shared memory workstations. This work has recently been extended in Reference 38 to include domain-based Schwarz methods (see Chapter 3) and calculations using direct linear solution techniques.

This dissertation adds to the previously cited research by specifically considering direct steady state (no time stepping) Newton-Krylov solution techniques for two-dimensional, low Mach number flow problems. The computational difficulties that make low Mach number compressible flow problems computationally challenging are addressed

further in Section 4.2.1. These difficulties can cause problems for approximate factorization type algorithms at low flow Mach numbers; and, while direct-Newton solution techniques have been effective for these flow situations [32], they have not been studied in detail using Newton-Krylov solution algorithms without use of time stepping strategies and/or approximate Jacobians. Point ILU preconditioning and domain-based preconditioning using Schwarz methods are compared and contrasted for these types of flow conditions. Additionally, the performance of Newton-Krylov solvers using both Arnoldi-based and Lanczos-based Krylov algorithms are investigated, along with the various performance enhancement techniques described in Chapter 3.

1.3. DOCUMENT ORGANIZATION

This document is organized into five main chapters including this introductory chapter. Chapter 2 describes the nonlinear partial differential equations describing both incompressible and compressible fluid flow and heat transfer that are solved in this dissertation. The finite volume discretization scheme that is used to convert these systems of nonlinear partial differential equations to systems of nonlinear algebraic equations is also presented. The model problems used to investigate the performance of the different numerical techniques are also included in this Chapter. This latter discussion includes a description of problem geometry, important flow and heat transfer parameters, and boundary conditions.

The different numerical techniques employed in this dissertation are described in Chapter 3. This chapter first describes the general Newton's method solution algorithm, then proceeds by describing several performance enhancement techniques used to improve the performance and/or convergence of the Newton iteration. Next, the chapter describes the use of Krylov subspace based iterative algorithms to solve the linear systems that arise on each

Newton step, giving rise to what are referred to as Newton-Krylov algorithms. Since an important advantage of iterative techniques for solving linear systems (compared with direct solvers) is the ability to exploit the sparseness of the system matrix, the sparse matrix storage scheme employed in this dissertation is described. Next, a discussion of the preconditioning strategies used to improve the performance of the Krylov algorithms is given. The last topic in this chapter is the use of finite difference projections with inexact Newton methods to yield matrix-free algorithm implementations. This terminology refers to the use of finite difference projection techniques to approximate the matrix-vector products that are needed by the Krylov algorithms. Since these finite difference approximations do not require forming and storing the Jacobian matrix, a matrix-free implementation is obtained whereby the Jacobian is typically only needed to generate an effective preconditioner.

The results of the numerical investigations are contained in Chapter 4. These results serve to test and investigate different algorithm features in solving several benchmark, incompressible fluid flow and heat transfer problems and one low Mach number compressible flow model problem. These results are summarized in Chapter 5 and important conclusions and observations are given. These main chapters are followed by an appendix designed to provide an general overview and description of Krylov subspace based iterative solution techniques for systems of linear equations. Finally, reference citations are accumulated in the final section.

CHAPTER 2

DESCRIPTION OF GOVERNING EQUATIONS AND MODEL PROBLEMS

This chapter describes the nonlinear partial differential equations that represent incompressible and compressible fluid flow with heat transfer. A discussion of the finite volume discretization scheme, which integrates these continuous partial differential equations over discrete finite volumes, is also included in this Chapter. These discrete finite volumes compose the computational mesh, which in turn approximates the true problem geometry. The result of this discretization process is a system of nonlinear algebraic equations designed to approximate the mathematical character of the original set of continuous partial differential equations. The solution of these algebraic systems of equations is the topic of Chapter 3. The model problems that are solved using these techniques are described below with respect to important flow parameters, geometry, and boundary conditions.

2.1. INCOMPRESSIBLE FLUID FLOW AND HEAT TRANSFER

This section describes the equations used to model steady state, incompressible (low speed) flow of a laminar, Newtonian fluid. The transport of mass, momentum, and energy is considered; while assuming that the fluid density is constant everywhere except in the buoyancy term that appears in the momentum equations. The latter restriction allows thermal effects to influence momentum transport through small, temperature-induced density variations. This coupling must be included for low speed flow whenever gravitational forces are comparable to inertia and viscous forces [1]. The small density perturbations are modeled using the Boussinesq approximation [see 31]. The conservation equations that

describe this coupled incompressible fluid flow and heat transfer system are described in Section 2.1.1 below. Section 2.1.2 presents a description of the various techniques used to discretize these equations. Finally, three incompressible flow model problems are described in Sections 2.1.3 through 2.1.5.

2.1.1. Governing Equations

The dimensionless governing equations for two-dimensional, incompressible, constant property fluid flow and heat transfer are presented below in conservative form using Cartesian coordinates.

Continuity:

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

Momentum:

$$\frac{\partial u^2}{\partial x}+\frac{\partial uv}{\partial y}=-\frac{\partial p}{\partial x}+\frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)+\frac{Gr}{Re^2}T\sin\phi \tag{2}$$

$$\frac{\partial uv}{\partial x}+\frac{\partial v^2}{\partial y}=-\frac{\partial p}{\partial y}+\frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right)+\frac{Gr}{Re^2}T\cos\phi \tag{3}$$

Energy:

$$\frac{\partial uT}{\partial x}+\frac{\partial vT}{\partial y}=\frac{1}{Re\cdot Pr}\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right) \tag{4}$$

The flow variables u, v, p, and T represent the dimensionless x-direction velocity, y-direction velocity, pressure, and temperature, respectively. The angle, ϕ, defines the orientation of the x-axis with respect to horizontal. This convention then assumes that the gravity vector is pointed in the negative vertical direction. Re is the Reynolds number, Gr is the Grashof number, and Pr is the Prandtl number. Note that the Rayleigh number, Ra, another common heat transfer parameter, is defined by $Ra = Gr\,Pr$. The Reynolds number reflects the importance of inertia forces relative to viscous forces, while the Grashof number reflects the importance of buoyancy forces relative to viscous forces. The Prandtl number represents the ratio of momentum diffusivity to thermal diffusivity. The product of Reynolds number and Prandtl number defines yet another important dimensionless heat transfer parameter, the Peclet number, Pec. The parameter, $\frac{Gr}{Re^2}$, appearing in the buoyancy term in the momentum equations indicates the relative importance of gravitational forces relative to inertia forces. In other words, it provides a measure of the importance of buoyancy or free-convection effects relative to forced convection effects. If this parameter is at least of order unity then free convection effects are important and this term should be retained. This situation is often referred to as *mixed* convection. However, if this term is very small then the problem is dominated by *forced* convection, and this term may be negligible. When there is no forced convection, the flow is entirely driven by temperature differences. This situation is referred to as *natural* convection. In this case, the reference velocity is frequently set so that the Reynolds number, appearing in Equations (2) through (4), is identically one.

Values used for the important dimensionless flow parameters, and the parameters used to nondimensionalize the equations, are given in the sections describing each of the incompressible flow model problems. Note that the momentum equations are coupled to the temperature field via the buoyancy force terms arising from the Boussinesq approximation

[see 31]. This approximation assumes that density is constant everywhere except in the buoyancy force terms of the momentum equation. In this term, density is assumed to vary with temperature according to a first order Taylor series approximation about the reference density, $\tilde{\rho}_0$ [see 1, 31]. This approximation can be expressed as follows,

$$\tilde{\rho} = \tilde{\rho}_0 + \Delta\tilde{\rho} \approx \tilde{\rho}_0 + \left(\frac{\partial\tilde{\rho}}{\partial\tilde{T}}\right)_{\tilde{p}} \Delta\tilde{T} \approx \tilde{\rho}_0\left(1 - \tilde{\beta}\Delta\tilde{T}\right), \tag{5}$$

where $\tilde{\beta}$ is the coefficient of thermal expansion defined in this case by,

$$\tilde{\beta} = -\frac{1}{\tilde{\rho}_0}\left(\frac{\partial\tilde{\rho}}{\partial\tilde{T}}\right)_{\tilde{p}}. \tag{6}$$

Use of this approximation leads to the buoyancy force terms presented in Equations (2) and (3).

2.1.2. Discretization of Governing Equations

Discretized forms of the governing equations are solved on a two-dimensional, staggered mesh with thermodynamic variables located at cell centers and velocities located at cell faces, analogous to the MAC solution procedure [see 19]. The computational mesh consists of rectangular cells of length Δx_i in the x-direction and Δy_j in the y-direction, where i and j refer are the cell indices in those directions, respectively. Note that Δx_i is allowed to vary with i, but not with j. Similarly Δy_j is allowed to vary with j, but not with i. This variation allows some flexibility in constructing non-uniform meshes, although still rectangular and structured. The rectangular mesh is assumed to contain nx cells in the x-

direction and *ny* cells in the *y*-direction. A schematic of a typical finite volume cell is shown in Figure 1.

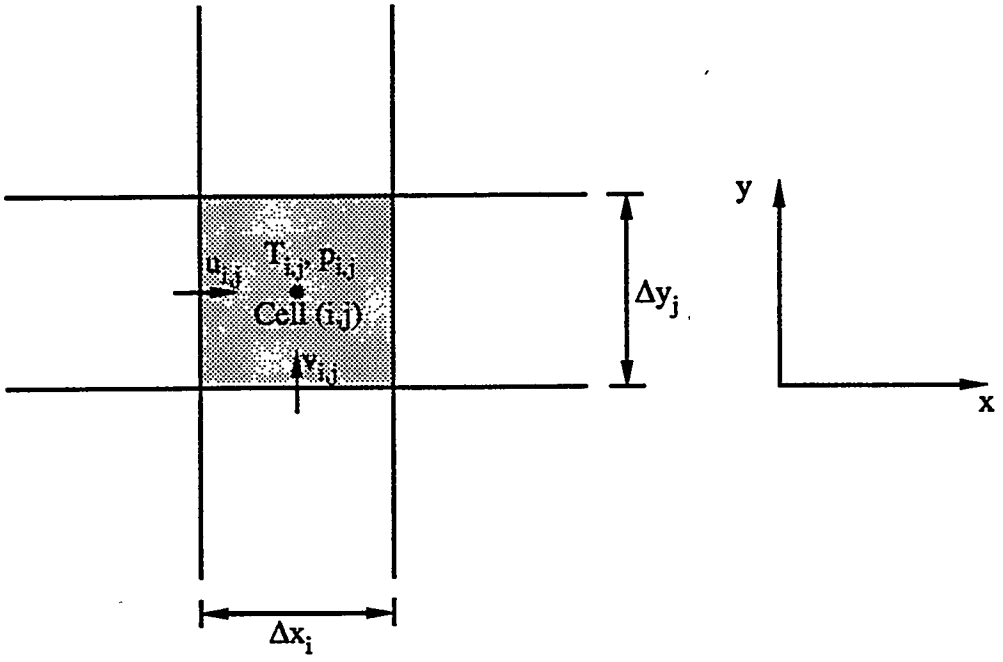


Figure 1. Schematic of typical finite volume cell and placement of cell variables.

The governing equations are spatially discretized using the control volume or finite volume approach, whereby the conservation equations are integrated (volume averaged) over the volume of the appropriate computational cell [see 21, 101, 102]. The appropriate computational cell is determined by the equation under consideration. In the case of the continuity and energy equations, the appropriate cell is the typical computational cell shown in Figure 1. However, in the case of the momentum equations, it is convenient to define two additional computational cells, called *momentum cells*. Thus, the *u*-momentum equation is

integrated over the *u*-momentum cell shown in Figure 2, while the *v*-momentum equation is integrated over the *v*-momentum equation shown in Figure 3.

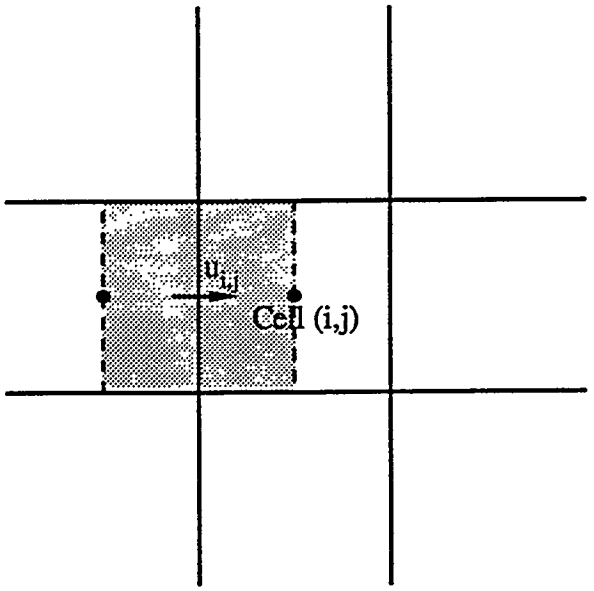


Figure 2. Schematic illustration of typical *u*-momentum cell.

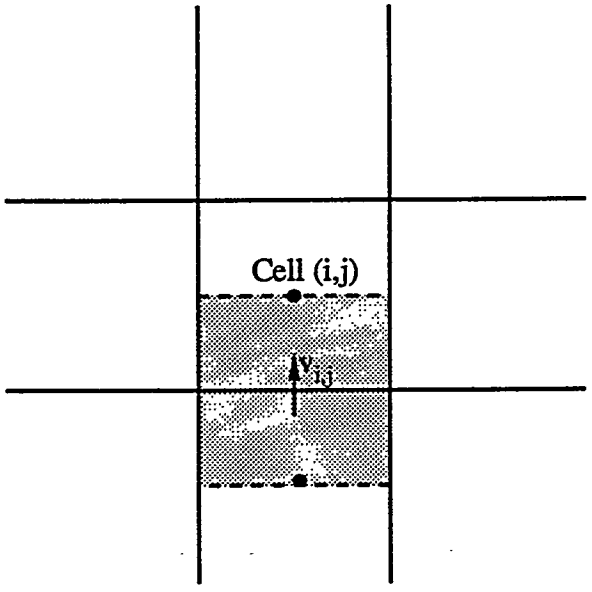


Figure 3. Schematic illustration of typical *v*-momentum cell.

The basic idea of this discretization scheme is the use of the divergence theorem to convert volume integrals of terms of divergence form to surface integrals over cell faces. The divergence theorem is expressed by [see 103],

$$\iiint_V (\nabla \cdot \mathbf{v}) dV = \iint_S \mathbf{v} \cdot \hat{\mathbf{n}} dS, \tag{7}$$

where $\mathbf{v}$ is a continuous vector defined within the finite volume, V represents the cell volume, S denotes the cell surface, and $\hat{\mathbf{n}}$ is the outward unit normal to S. The surface integral in Equation (7) is then approximated by summing the integrand over all of the finite volume faces. In two dimensions, this process is simplified because the components of the vector are zero in the third dimension, and the face areas can be computed assuming a unit depth in the third dimension. This procedure leads to conservative difference equations for mass, momentum, and energy. Several specific spatial discretization schemes are described below.

2.1.2.1. Standard Discretization

The standard finite volume discretization scheme used for the incompressible flow equations is based upon the power-law formulation of Patankar [21]. Consequently, much of the notation in Reference [21] is used here to describe this discretization scheme.

In vector form, the continuity equation expressed in Equation (1), can be written in the following divergence form,

$$\nabla \cdot \mathbf{u} = 0. \tag{8}$$

This equation is then integrated over the finite volume shown in Figure 4, where the shaded region indicates the volume over which the integration is performed. Note that in this figure and subsequent figures for the momentum cells, the notation used to define the locations of variables [i.e., north (n, N), south (s, S), east (e, E), and west (w, W)] are defined with respect to the volume (i.e., shaded region) of interest. Application of the divergence theorem to this volume integral then yields,

$$\iiint_V (\nabla \cdot \mathbf{u}) dV = \iint_S \mathbf{u} \cdot \hat{\mathbf{n}} dS \equiv u_e \Delta y_j + v_n \Delta x_i - u_w \Delta y_j - v_s \Delta x_i = (u_e - u_w)\Delta y_j + (v_n - v_s)\Delta x_i. \quad (9)$$

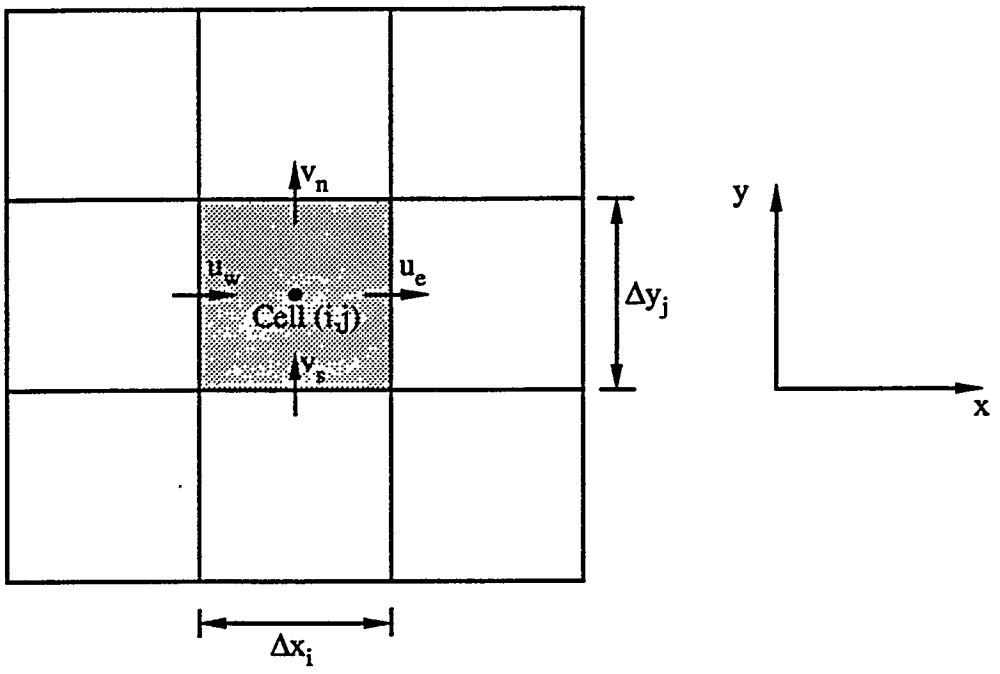


Figure 4. Finite volume stencil used to discretize incompressible flow continuity equation.

A discrete algebraic equation of the form expressed by Equation (9) arises for each computational cell in the mesh. Rows associated with Equation (9) in the resulting fully

coupled matrix system are typically aligned with the pressure variable even though pressure does not explicitly appear in this equation. Under certain conditions, this alignment choice can cause difficulties. These difficulties are addressed in Chapter 4. One remedy for difficulties of this sort is to solve a discrete Poisson type pressure equation in lieu of Equation (9). The discrete pressure equation formulation is described in Section 2.1.2.2 below.

The u-momentum equation can be treated in a similar fashion. First, Equation (2) can be expressed in vector form as,

$$\nabla \cdot \mathbf{Q}^u + \nabla \cdot \left(p\hat{\mathbf{i}}\right) - \frac{Gr}{Re^2} T \sin\phi = 0, \tag{10}$$

where

$$\mathbf{Q}^u = \mathbf{u}u - \frac{1}{Re}\nabla u. \tag{11}$$

Note that the first two terms in Equation (10) are of divergence form, and so the divergence theorem may be used to convert volume integrals of these quantities to surface integrals. The last term, however, is not of this form and so must be integrated over the finite volume. Equation (10) is integrated over the shaded region shown in Figure 5. Recall that this shaded region is the u-momentum cell. The finite volume stencil indicated in Figure 5 is used to evaluate necessary quantities either at the cell faces or at the cell center. The result of this integral approximation is given by,

$$\begin{aligned}\left(Q_e^u - Q_w^u\right)\Delta y_j + \left(Q_n^u - Q_s^u\right)\left(\frac{\Delta x_{i-1} + \Delta x_i}{2}\right) + \left(p_e - p_w\right)\Delta y_j \\ - \frac{Gr}{Re^2} T_{ave} \sin\phi \left(\frac{\Delta x_{i-1} + \Delta x_i}{2}\right)\Delta y_j = 0,\end{aligned} \tag{12}$$

where

$$T_{ave} = \frac{\Delta x_{i-1} T_e + \Delta x_i T_w}{\Delta x_{i-1} + \Delta x_i} \tag{13}$$

and

$$Q_e^u = \left(\mathbf{Q}^u \cdot \hat{\mathbf{i}}\right)_e,\ Q_w^u = \left(-\mathbf{Q}^u \cdot \hat{\mathbf{i}}\right)_w,\ Q_n^u = \left(\mathbf{Q}^u \cdot \hat{\mathbf{j}}\right)_n,\ Q_s^u = \left(-\mathbf{Q}^u \cdot \hat{\mathbf{j}}\right)_s. \tag{14}$$

The subscripts in Equations (12) through (14) indicate where the subscripted quantity is evaluated according to the notation used in Figure 5.

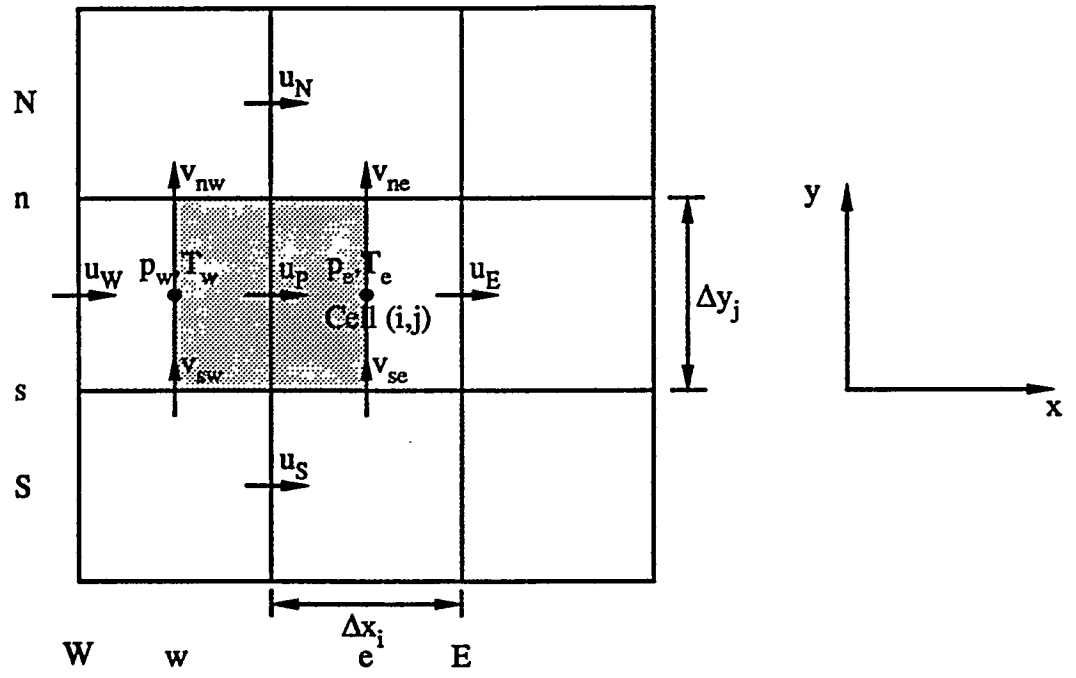


Figure 5. Finite volume stencil used to discretize the incompressible flow *u*-momentum equation.

Following the formulation outlined in Reference 21, the next step in the discretization process is to integrate the continuity equation over the u-momentum cell, multiply this discrete equation by u_P, and then subtract the result from Equation (12). This step results in the following expression,

$$\begin{aligned}&\left[\left(Q_e^u - u_e u_P\right) - \left(Q_w^u - u_w u_P\right)\right]\Delta y_j + \left[\left(Q_n^u - v_n u_P\right) - \left(Q_s^u - v_s u_P\right)\right]\left(\frac{\Delta x_{i-1} + \Delta x_i}{2}\right) \\ &+ (p_e - p_w)\Delta y_j - \frac{Gr}{Re^2} T_{ave} \sin\phi \left(\frac{\Delta x_{i-1} + \Delta x_i}{2}\right)\Delta y_j = 0,\end{aligned} \tag{15}$$

where u_e, u_w, v_n, and v_s are the convecting velocities, which are located on the u-momentum cell faces at the same x-location as u_p. These velocities are defined by,

$$\begin{aligned}u_e &= \frac{u_E + u_P}{2}, \quad u_w = \frac{u_W + u_P}{2}, \\ v_n &= \frac{\Delta x_i v_{nw} + \Delta x_{i-1} v_{ne}}{\Delta x_i + \Delta x_{i-1}}, \text{ and } \quad v_s = \frac{\Delta x_i v_{sw} + \Delta x_{i-1} v_{se}}{\Delta x_i + \Delta x_{i-1}}.\end{aligned} \tag{16}$$

Note that v_n and v_s are computed using simple linear interpolation. At this point in the discretization process, the Q^u terms in Equation (15) have not yet been approximated. Since these terms include convection, they must be evaluated in such a way so as to maintain numerical stability. Patankar [21] suggests that these terms be evaluated as follows:

$$\begin{aligned}\left(Q_e^u - u_e u_P\right)\Delta y_j &= (u_P - u_E)\left[\frac{1}{Re\Delta x_i} A\left(\left|Re\Delta x_i u_e\right|\right) + Max(-u_e, 0)\right]\Delta y_j, \\ \left(Q_n^u - v_n u_P\right)\left(\frac{\Delta x_{i-1} + \Delta x_i}{2}\right) &= (u_P - u_N)\left[\frac{1}{Re\Delta y_n} A\left(\left|Re\Delta y_n v_n\right|\right) + Max(-v_n, 0)\right]\left(\frac{\Delta x_{i-1} + \Delta x_i}{2}\right), \\ \left(Q_w^u - u_w u_P\right)\Delta y_j &= (u_W - u_P)\left[\frac{1}{Re\Delta x_i} A\left(\left|Re\Delta x_{i-1} u_w\right|\right) + Max(u_w, 0)\right]\Delta y_j, \text{ and} \\ \left(Q_s^u - v_s u_P\right)\left(\frac{\Delta x_{i-1} + \Delta x_i}{2}\right) &= (u_S - u_P)\left[\frac{1}{Re\Delta y_s} A\left(\left|Re\Delta y_s v_s\right|\right) + Max(v_s, 0)\right]\left(\frac{\Delta x_{i-1} + \Delta x_i}{2}\right),\end{aligned} \tag{17}$$

where

$$\Delta y_n = \frac{\Delta y_j + \Delta y_{j+1}}{2} \text{ and } \Delta y_s = \frac{\Delta y_j + \Delta y_{j-1}}{2}. \tag{18}$$

The function $A(|P|)$ appearing in Equation (17) specifies how the convection and diffusion terms are handled. $A(|P|) = 1$ corresponds to pure upwinding [see 19] for the convection terms and centered differencing for the diffusion terms. This option renders the approximation to the convective terms first order accurate, while the diffusion term approximation is second order accurate. $A(|P|) = 1 - \frac{|P|}{2}$ corresponds to central differencing for both terms, which is second order accurate but often leads to numerical instability. The power-law scheme [21] is obtained from,

$$A(|P|) = Max\left\{0, (1-.1|P|)^5\right\}. \tag{19}$$

The accuracy of this scheme depends upon the magnitude of the cell Reynolds number, $|P|$. For $|P| > 10$, the power-law scheme reduces to pure upwinding with diffusion set to zero, while for $|P| < 10$ the power-law scheme is more accurate than pure upwinding. Central differencing arises only for $|P| < 2$.

The final discretized form of the u-momentum equation can be expressed compactly as follows [see 21]:

$$\begin{aligned}&(u_P - u_E)a_E + (u_P - u_W)a_W + (u_P - u_N)a_N + (u_P - u_S)a_S \\ &+(p_e - p_w)\Delta y_j - \frac{Gr}{Re^2} T_{ave} \sin\phi \left(\frac{\Delta x_{i-1} + \Delta x_i}{2}\right)\Delta y_j = 0,\end{aligned} \tag{20}$$

where

$$
\begin{aligned}
a_E &= \left[\frac{1}{Re\Delta x_i} A\left(\left|Re\Delta x_i u_e\right|\right) + Max\left(-u_e, 0\right)\right]\Delta y_j, \\
a_W &= \left[\frac{1}{Re\Delta x_i} A\left(\left|Re\Delta x_{i-1} u_w\right|\right) + Max\left(u_w, 0\right)\right]\Delta y_j, \\
a_N &= \left[\frac{1}{Re\Delta y_n} A\left(\left|Re\Delta y_n v_n\right|\right) + Max\left(-v_n, 0\right)\right]\left(\frac{\Delta x_{i-1} + \Delta x_i}{2}\right), \text{ and} \\
a_S &= \left[\frac{1}{Re\Delta y_s} A\left(\left|Re\Delta y_s v_s\right|\right) + Max\left(v_s, 0\right)\right]\left(\frac{\Delta x_{i-1} + \Delta x_i}{2}\right).
\end{aligned}
\tag{21}
$$

Thus, rows associated with Equation (20) in the fully coupled matrix system are aligned with the u-velocity variable (u_P) appearing in each computational cell. The general form of the discrete equation given by Equation (20) is derived for the remaining equations below by simply replacing u with the variable of interest (i.e., either v or T), redefining the coefficients expressed in Equation (21), and then replacing the last two terms in Equation (20) by the appropriate source terms, if any.

Before proceeding, however, it is convenient to express Equation (20) in a different form that will be useful in the discussion of the discrete pressure equation formulation in Section 2.1.2.2. This form is obtained by solving Equation (20) for u_P to give,

$$
\begin{aligned}
u_P &= \frac{a_E u_E + a_W u_W + a_N u_N + a_S u_S}{a_P} - \left(p_e - p_w\right)\frac{\Delta y_j}{a_P} + \frac{Gr}{Re^2} T_{ave} \sin\phi \left(\frac{\Delta x_{i-1} + \Delta x_i}{2a_P}\right)\Delta y_j \\
&= \hat{u}_P - \left(p_e - p_w\right)\frac{\Delta y_j}{a_P},
\end{aligned}
\tag{22}
$$

where $\hat{u}_P$ is referred to by Patankar [21] as a *pseudo-velocity*, given by

$$
\hat{u}_P = \frac{a_E u_E + a_W u_W + a_N u_N + a_S u_S}{a_P} - \frac{Gr}{Re^2} T_{ave} \sin\phi \left(\frac{\Delta x_{i-1} + \Delta x_i}{2a_P}\right)\Delta y_j \tag{23}
$$

and

$$a_P = a_E + a_W + a_N + a_S. \tag{24}$$

The purpose for introducing these equations will be more apparent in Section 2.1.2.2.

The v-momentum equation can be discretized for each computational cell in a fashion analogous to the procedure used to discretize the u-momentum equation. The integration is performed over the v-momentum cell (the shaded region shown in Figure 6).

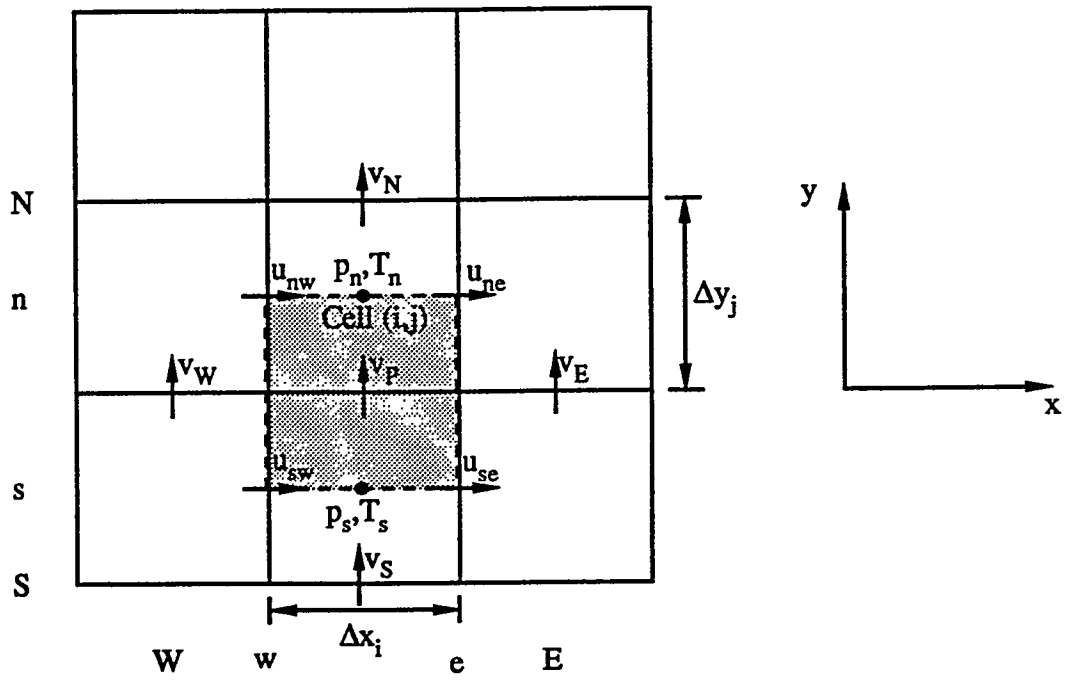


Figure 6. Finite volume stencil used to discretize the incompressible flow v-momentum equation.

The final discretization equation for the finite volume cell shown in Figure 6 is given below:

$$\begin{aligned}&(v_P - v_E)a_E + (v_P - v_W)a_W + (v_P - v_N)a_N + (v_P - v_S)a_S \\ &+(p_n - p_s)\Delta x_i - \frac{Gr}{Re^2}T_{ave}\cos\phi\left(\frac{\Delta y_{j-1} + \Delta y_j}{2}\right)\Delta x_i = 0,\end{aligned} \tag{25}$$

where

$$\begin{aligned}a_E &= \left[\frac{1}{Re\Delta x_e}A\left(\left|Re\Delta x_e u_e\right|\right) + Max(-u_e, 0)\right]\left(\frac{\Delta y_{j-1} + \Delta y_j}{2}\right), \\ a_W &= \left[\frac{1}{Re\Delta x_w}A\left(\left|Re\Delta x_w u_w\right|\right) + Max(u_w, 0)\right]\left(\frac{\Delta y_{j-1} + \Delta y_j}{2}\right), \\ a_N &= \left[\frac{1}{Re\Delta y_j}A\left(\left|Re\Delta y_j v_n\right|\right) + Max(-v_n, 0)\right]\Delta x_i, \text{ and} \\ a_S &= \left[\frac{1}{Re\Delta y_{j-1}}A\left(\left|Re\Delta y_{j-1} v_s\right|\right) + Max(v_s, 0)\right]\Delta x_i;\end{aligned} \tag{26}$$

and

$$\begin{aligned}T_{ave} &= \frac{\Delta y_{j-1}T_n + \Delta y_j T_s}{\Delta y_{j-1} + \Delta y_j}, \\ u_e = \frac{\Delta y_j u_{se} + \Delta y_{j-1}u_{ne}}{\Delta y_j + \Delta y_{j-1}},\ u_w &= \frac{\Delta y_j u_{sw} + \Delta y_{j-1}u_{nw}}{\Delta y_j + \Delta y_{j-1}} \\ v_n = \frac{v_N + v_P}{2},\ v_s &= \frac{v_S + v_P}{2} \\ \Delta x_e = \frac{\Delta x_i + \Delta x_{i+1}}{2},\ \text{and}\ \Delta x_w &= \frac{\Delta x_i + \Delta x_{i-1}}{2}.\end{aligned} \tag{27}$$

Rows associated with Equation (25) in the fully coupled matrix system are aligned with the v-velocity variable (v_P) appearing in each computational cell.

Again, it is convenient at this point to express Equation (25) in a different form that will be useful in the discussion of the discrete pressure equation formulation in Section 2.1.2.2. This form is obtained by solving Equation (25) for v_P to give,

$$\begin{aligned} v_P &= \frac{a_E v_E + a_W v_W + a_N v_N + a_S v_S}{a_P} - (p_n - p_s)\frac{\Delta x_i}{a_P} + \frac{Gr}{Re^2} T_{ave} \cos\phi \left(\frac{\Delta y_{j-1} + \Delta y_j}{2a_P} \right) \Delta x_i \\ &= \hat{v}_P - (p_n - p_s)\frac{\Delta x_i}{a_P}, \end{aligned} \tag{28}$$

where $\hat{v}_P$ is another *pseudo-velocity*, given by

$$\hat{v}_P = \frac{a_E v_E + a_W v_W + a_N v_N + a_S v_S}{a_P} - \frac{Gr}{Re^2} T_{ave} \cos\phi \left(\frac{\Delta y_{j-1} + \Delta y_j}{2a_P} \right) \Delta x_i, \tag{29}$$

and

$$a_P = a_E + a_W + a_N + a_S. \tag{30}$$

Again, the purpose for introducing these equations will be more apparent in Section 2.1.2.2.

The energy equation is discretized over a computational cell by integrating this equation over the shaded region in Figure 7. The resulting discretization equation for the finite volume defined in Figure 7 can be written as,

$$(T_P - T_E)a_E + (T_P - T_W)a_W + (T_P - T_N)a_N + (T_P - T_S)a_S = 0, \tag{31}$$

where

$$
\begin{aligned}
a_E &= \left[\frac{1}{RePr\Delta x_e} A\left(\left|RePr\Delta x_e u_e\right|\right) + Max\left(-u_e, 0\right)\right]\Delta y_j, \\
a_W &= \left[\frac{1}{RePr\Delta x_w} A\left(\left|RePr\Delta x_w u_w\right|\right) + Max\left(u_w, 0\right)\right]\Delta y_j, \\
a_N &= \left[\frac{1}{RePr\Delta y_n} A\left(\left|RePr\Delta y_n v_n\right|\right) + Max\left(-v_n, 0\right)\right]\Delta x_i, \text{ and} \\
a_S &= \left[\frac{1}{RePr\Delta y_s} A\left(\left|RePr\Delta y_s v_s\right|\right) + Max\left(v_s, 0\right)\right]\Delta x_i;
\end{aligned}
\tag{32}
$$

and

$$
\begin{aligned}
\Delta x_e &= \frac{\Delta x_i + \Delta x_{i+1}}{2}, \quad \Delta x_w = \frac{\Delta x_i + \Delta x_{i-1}}{2} \\
\Delta y_n &= \frac{\Delta y_j + \Delta y_{j+1}}{2}, \text{ and } \quad \Delta y_s = \frac{\Delta y_j + \Delta y_{j-1}}{2}.
\end{aligned}
\tag{33}
$$

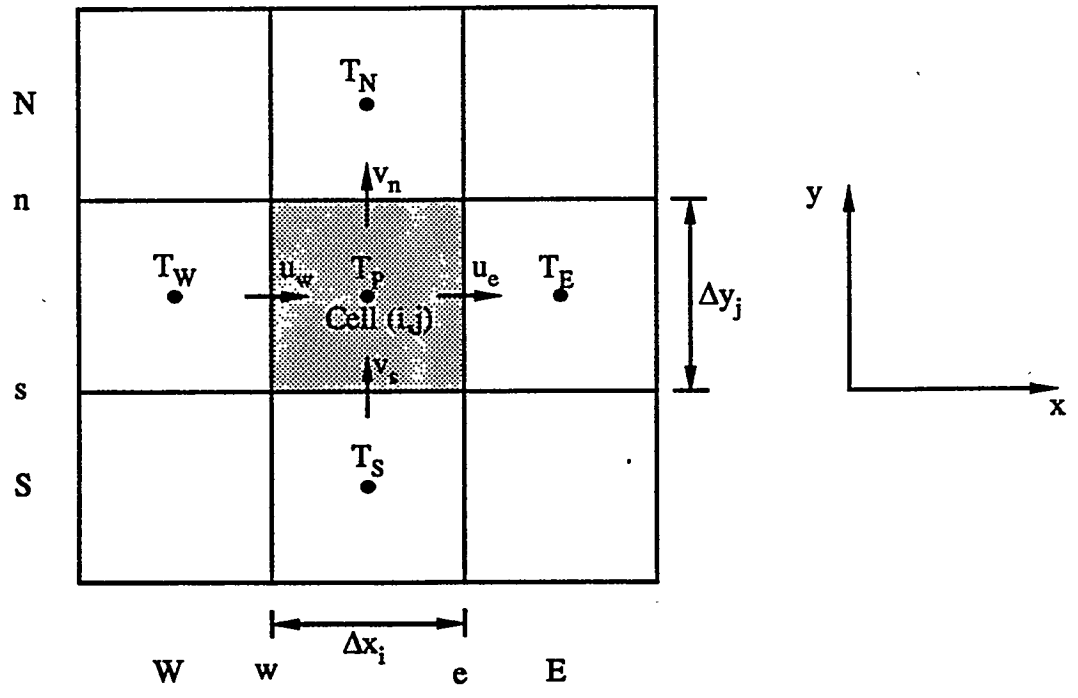


Figure 7. Finite volume stencil used to discretize the incompressible flow energy equation.

Rows associated with Equation (31) in the fully coupled matrix system are aligned with the temperature variable (T_P) appearing in each computational cell.

2.1.2.2. Discrete Pressure Equation Formulation

The discrete continuity equation given by Equation (9) yields, for each computational cell, a row in the fully coupled matrix system that is aligned with the pressure variable in that cell. In this manner, pressure is calculated so as to ensure a divergence-free velocity field. The difficulty in this approach, however, is that pressure does not explicitly appear in the continuity equation. This difficulty has led to the development of many different numerical schemes that enable the calculation of pressure from the continuity equation [for examples see 19, 20, 21, 22, 80, and 104]. In this work, a fully coupled Newton solution technique is selected, and so the problem is manifested in the appearance of zeros on the main diagonal of the Jacobian matrix. Oftentimes, this problem is not a serious concern using the solution techniques described in Chapter 3. However, some circumstances do arise when this problem must be addressed. One possible remedy in situations of this sort is to solve a Poisson type pressure equation for pressure in lieu of the continuity equation [see 104]. This equation is derived by combining the momentum equations and the continuity equation to yield a new equation for pressure.

In this work, a discrete pressure equation is formed from the discrete continuity and momentum equations. This approach is in contrast to the derivation of a continuous partial differential equation for pressure and then discretizing this equation. The advantage in the former approach is that the discrete pressure equation is guaranteed to be consistent with the discrete forms of the continuity and momentum equations, and imposition of boundary condition is simplified because the boundary conditions that are imposed on the discrete continuity and momentum equations can also be applied to the discrete pressure equation.

See Gresho [104] for a more complete discussion regarding the issues associated with using a pressure equation formulation for incompressible flow calculations.

The discrete pressure equation used in this investigation is derived following the procedure outlined in Reference 21. The required finite volume stencil for this discretization over a single cell is shown in Figure 8. The starting point in this derivation is the discrete continuity equation given in Equation (9). The velocities, u_e, u_w, v_n, and v_s are then replaced with expressions obtained from the discrete momentum equations of the form given by Equation (22) and Equation (28). These expressions can be written as:

$$\begin{aligned} u_e &= \hat{u}_e + (p_P - p_E)\frac{\Delta y_j}{a_e}, \\ u_w &= \hat{u}_w - (p_P - p_W)\frac{\Delta y_j}{a_w}, \\ v_n &= \hat{v}_n + (p_P - p_N)\frac{\Delta x_i}{a_n}, \text{ and} \\ v_s &= \hat{v}_s - (p_P - p_S)\frac{\Delta x_i}{a_s}. \end{aligned} \tag{34}$$

The pseudo-velocities, $\hat{u}_e$, $\hat{u}_w$, $\hat{v}_n$, and $\hat{v}_s$ are obtained from expressions similar to Equations (23) and (29), except defined with respect to the finite volume stencil shown in Figure 8. Similarly, the coefficients, a_e, a_w, a_n, and a_s appearing in Equation (34) are analogous with Equations (24) and (30), except defined with respect to the finite volume stencil shown in Figure 8. Substitution of these expressions into the discrete continuity equation then gives,

$$\begin{aligned} (p_P - p_E)\frac{\Delta y_j^2}{a_e} + (p_P - p_W)\frac{\Delta y_j^2}{a_w} + (p_P - p_N)\frac{\Delta x_i^2}{a_n} + (p_P - p_S)\frac{\Delta x_i^2}{a_s} \\ + (\hat{u}_e - \hat{u}_w)\Delta y_j + (\hat{v}_n - \hat{v}_s)\Delta x_i = 0. \end{aligned} \tag{35}$$

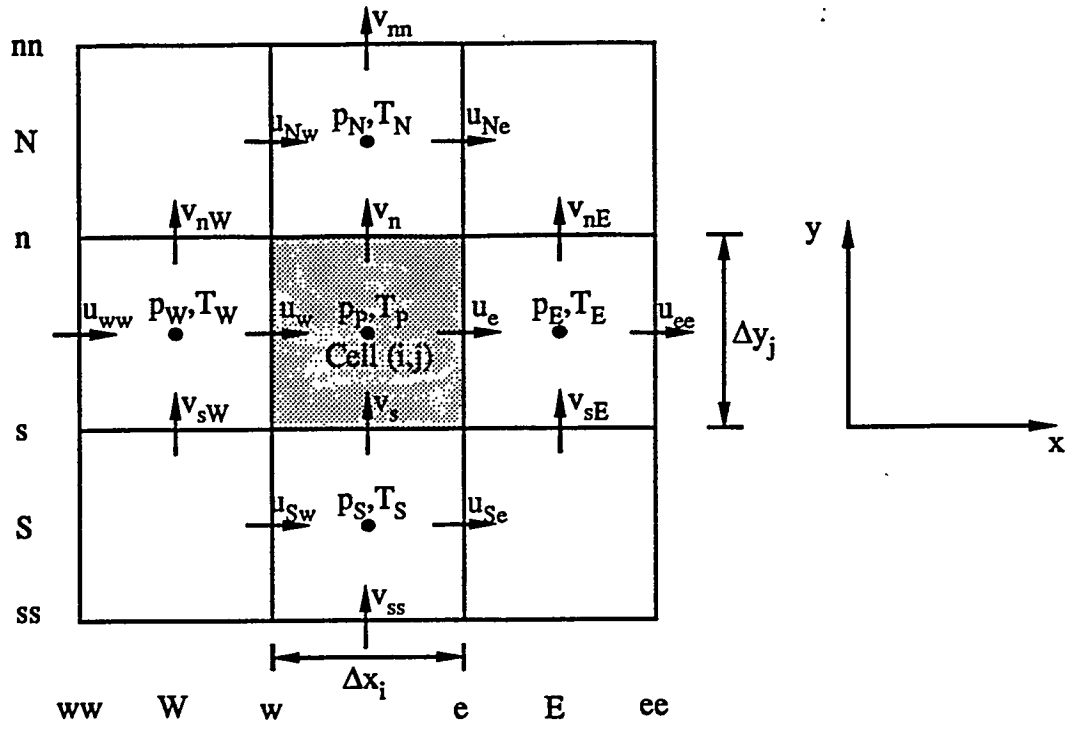


Figure 8. Finite volume stencil used in the discrete pressure equation formulation

This discrete pressure equation is considerably more complex than the original continuity equation, but it accomplishes the goal of deriving a discrete equation that contains pressure. The added complexity follows from the use of the expressions in Equation (34). These expressions represent the solution of the discrete u-momentum equation for u_e in cells $(i+1,j)$ and for u_w in cell (i,j), while the discrete v-momentum equation is solved for v_n in computational cell $(i,j+1)$ and for v_s in cell (i,j). Consequently, the finite volume stencil shown in Figure 8 contains significantly more variables than are required for the discretization of the other conservation equations. The advantages and disadvantages of the discrete pressure equation formulation described here will be investigated in Chapter 4.

2.1.2.3. Higher-Order Discrete Approximations

Recall that the power-law discretization scheme [21] that was described in Section 2.1.2.1 reduces to pure upwinding for values of the cell Reynolds number greater than ten. Unfortunately, upwinding is only a first order accurate spatial differencing scheme, which can be too inaccurate in some instances. Additionally, improving spatial accuracy via grid refinement may be impractical due to memory restrictions or in some cases due to convergence difficulties. An alternative is to improve the accuracy of the spatial differencing scheme used to approximate the convective terms, while maintaining numerical stability. Several articles that investigate the use of higher order approximations for these convective terms are cited in References 105, 106, 107, and 108. This alternative, however, typically requires enlarging the finite volume stencil used to approximate quantities on cell faces.

The power-law convection-diffusion discretization scheme provided an approximation to terms containing both convection and diffusion. These terms, analogous to Equation (11), can be expressed in one-dimension in the following manner,

$$Q_x^\theta = u\theta - \frac{1}{\Gamma}\frac{\partial \theta}{\partial x}, \tag{36}$$

where it is assumed that the quantity θ is being convected by the velocity u, and is diffusing according to the coefficient $1/\Gamma$. Finite volume discretization typically requires evaluation of terms of this form at cell faces. The use of higher order convection schemes for this evaluation often requires treatment of convection and diffusion separately. In this work, diffusion terms are approximated with conventional second order accurate central differences. For example, consider the one-dimensional finite volume stencil shown in Figure 9. For simplicity, this figure assumes a uniform grid, for which the mesh spacing

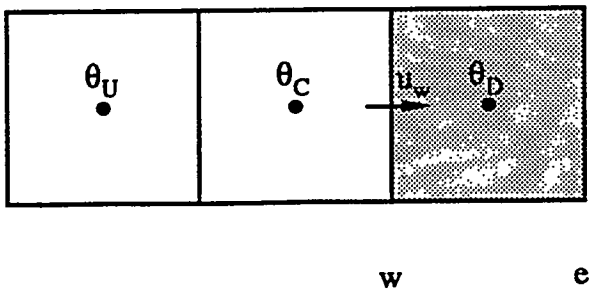


Figure 9. Schematic of one-dimensional finite volume stencil used for higher order approximations to convective terms.

between nodes and between cell faces are equal to Δx. In this figure the notation of References 105 and 106 have been adopted, where the subscript 'U' denotes the second node upstream of the cell face, the subscript 'C' denotes the first node upstream of the cell face, and the subscript 'D' denotes the node downstream from the cell face. The directions 'upstream' and 'downstream' are determined by the direction of the velocity u_w, which always points downstream. Using this notation, the derivative corresponding to diffusion on the west face (w) is approximated by,

$$\frac{\partial \theta}{\partial x} \cong \frac{\theta_D - \theta_C}{\Delta x}. \tag{37}$$

The convection term $(u\theta)_w$ on the west face can then be approximated separately using different combinations of the three node values for θ to obtain more accurate approximations. Note that for pure upwinding [see 19 and 20] this term is computed from,

$$(u\theta)_w \cong u_w \theta_C. \tag{38}$$

Use of upwinding to evaluate these face quantities results in a first order approximation to the convective derivative, which is approximated by,

$$\frac{\partial u\theta}{\partial x} \cong \frac{(u\theta)_e - (u\theta)_w}{\Delta x}. \tag{39}$$

A second order accurate approximation can be obtained using the QUadratic Interpolation for Convective Kinematics (QUICK) scheme [see 106, 108]. This scheme computes the face value from,

$$(u\theta)_w \cong u_w\left(\frac{3}{8}\theta_D + \frac{6}{8}\theta_C - \frac{1}{8}\theta_U\right). \tag{40}$$

Another second order accurate scheme would be obtained by simply averaging the first upwind node value and the downstream node value. This choice corresponds to central differencing, which is more susceptible to numerically instability at higher values of the cell Reynolds number than the QUICK scheme. A third order accurate approximation to the convective derivative can be obtained using the Cubic Upwind Interpolation scheme (CUI) [see 105 and 106]. This scheme computes the face value from,

$$(u\theta)_w \cong u_w\left(\frac{2}{6}\theta_D + \frac{5}{6}\theta_C - \frac{1}{6}\theta_U\right). \tag{41}$$

This investigation considers the use Equation (41) with conventional central differencing for the diffusion terms. Note that although use of Equation (41) results in a third order accurate approximation for the convective derivatives, the diffusive terms are still only second order accurate. Additionally, near boundaries where two upstream interpolation points are not available, simple averaging of the upstream and downstream values is used to compute the

face value. Consequently, the overall scheme is second order accurate. Examples of the use of this discretization scheme will be presented in Chapter 4.

Note that the formulae presented above assume the specified direction of the velocity. If this velocity were reversed then the 'upstream' and 'downstream' directions would also reverse. This simply requires redefining the two upstream nodes and the one downstream node. One significant effect in the use of a more accurate approximation for the convective terms, is that the finite volume stencil is necessarily enlarged because of the addition of the second upstream node. The consequences of this larger stencil mean a more dense Jacobian matrix, and possibly one that is less diagonally dominant (i.e., a more ill-conditioned matrix). The latter consequence can affect the performance solution algorithm described in the next Chapter. These issues will be discussed further in Chapter 4.

2.1.3. Natural Convection in an Enclosed Cavity Model Problem

This section describes the problem of natural convection in an enclosed cavity. This problem is a standard benchmark problem frequently used for testing different numerical techniques. Accurate solutions to this problem have been presented by de Vahl Davis [109] for different values of the Rayleigh number.

The model problem geometry and boundary conditions are shown in the schematic of Figure 10. The gravity vector is denoted by **g**, with a magnitude given by $\tilde{g}$. No-slip (zero velocity) boundary conditions are enforced on all walls. This condition renders the geometry closed in the sense that no flow into or out of the box is allowed. The temperatures are fixed on the side walls with the right side wall set at a higher temperature, while the upper and lower walls are assumed adiabatic (insulated).

The characteristic parameters used to nondimensionalize the governing equations are identified below [see 1], where a tilde indicates a dimensional quantity:

$$u=\frac{\tilde{u}}{\tilde{V}},\ v=\frac{\tilde{v}}{\tilde{V}},\ p=\frac{\tilde{p}+\tilde{\rho}_0\tilde{g}\tilde{y}-\tilde{p}_0}{\tilde{\rho}_0\tilde{V}^2},\ T=\frac{\tilde{T}-\tilde{T}_0}{\tilde{T}_H-\tilde{T}_0},\ x=\frac{\tilde{x}}{\tilde{L}},\ y=\frac{\tilde{y}}{\tilde{L}}. \tag{42}$$

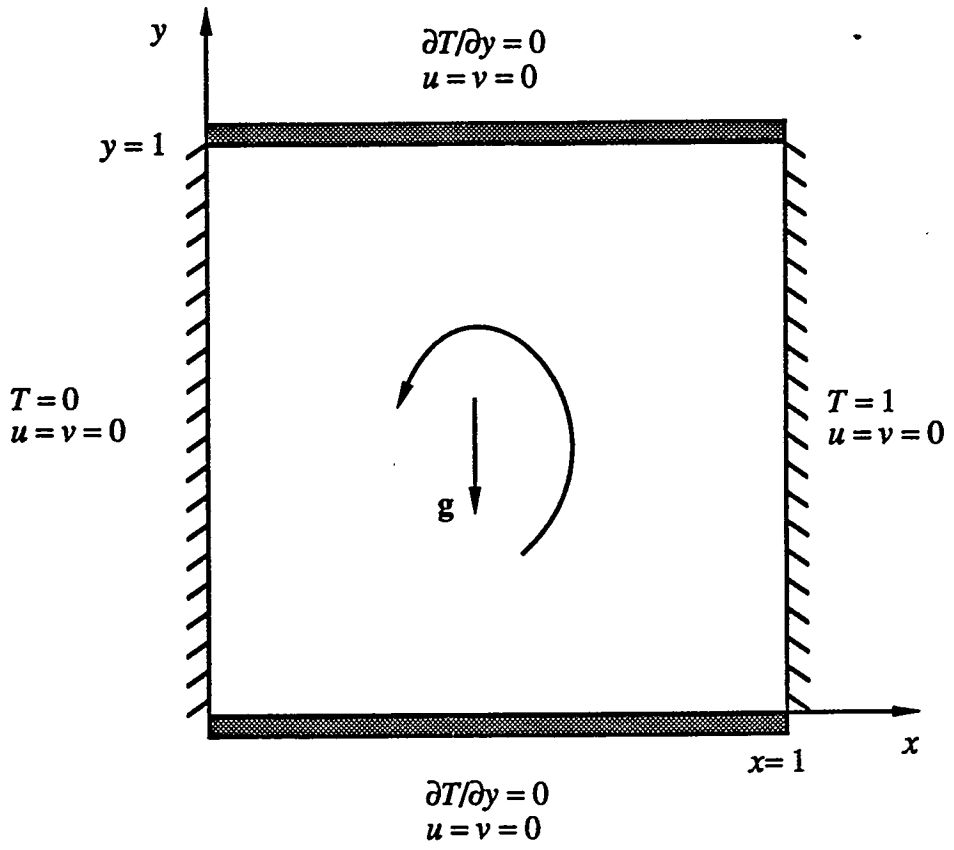


Figure 10. Geometry for natural convection model problem.

The subscript '0' indicates a reference quantity defined with respect to the cool left wall (x=0), while the subscript 'H' refers to the hot wall (x=1). Note that for incompressible flow, pressure appears only in gradient form. This means that the absolute magnitude of pressure is not really important, only changes in its magnitude from one spatial location to another. This allows the dimensionless pressure to be defined so as to include the hydrostatic term, $\tilde{\rho}_0\tilde{g}\tilde{y}$. The velocity scale, $\tilde{V}$, used in Equation (42) is defined so as to make the Reynolds

number identically one, i.e., $\tilde{V} = \tilde{\upsilon}_0/\tilde{L}$, where $\tilde{\upsilon}_0$ is the kinematic viscosity of the fluid. The length scale, $\tilde{L}$, is defined based upon the height (or width) of the square cavity.

The important parameters used in Equation (1) through Equation (4) for this problem are defined by,

$$\phi = 0,\ Re \equiv 1,\ Pr = 0.71,\ Gr = \frac{\tilde{\beta}\tilde{g}\tilde{L}^3\left(\tilde{T}_H - \tilde{T}_0\right)}{\tilde{V}_0^2},\ Ra = Gr \cdot Pr, \tag{43}$$

where $\tilde{\beta}$ is the coefficient of thermal expansion. The value of the Prandtl number listed in Equation (43) is characteristic of air. Specification of the Rayleigh number, Ra, is used to define the Grashof number, Gr.

2.1.4. Mixed Convection, Backward Facing Step Model Problem

This section describes a model problem that represents steady, two-dimensional, mixed convection flow in a channel with a backward facing step. A schematic illustration of this model problem is given in Figure 11. This problem was defined by the *ASME K-12 Aerospace Heat Transfer Committee* as a benchmark for code validation and assessment [see 11, 110]. Geometrically, this problem is distinguished from the natural convection problem by inflow and outflow sections, the non-rectangular geometry, and aspect ratios (x:y) of 5:1 upstream from the step and 15:1 downstream from the step.

The inlet dimensionless velocity at $x = -5$ for $1 \le y \le 2$ is specified by a parabolic profile defined by,

$$u(y) = \frac{3}{2}\left[1 - (3 - 2y)^2\right]. \tag{44}$$

This velocity profile at the inlet acts to impose a pressure gradient, which forces the fluid through the expansion channel and then out the boundary at the other end. Note that the sudden expansion caused by the step causes the flow to separate, thus creating a local recirculating flow pattern in the region just downstream from the step. The inlet temperature is fixed at the cold value of $\tilde{T}_0$ so that the dimensionless temperature there is zero. Walls are assumed to be no-slip (zero velocity) boundaries. The temperature of the wall attached to and downstream of the step is set at the hot temperature $\tilde{T}_H$, so that the dimensionless temperature there is one. Upstream of the step this wall is assumed adiabatic. The wall opposite the step is set to the cold temperature so that the dimensionless temperature along its length is zero. The step face wall is assumed adiabatic. Along the outflow boundary, the following fully developed, zero gradient boundary conditions are enforced,

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial x} = \frac{\partial T}{\partial x} = 0, \tag{45}$$

and the outflow pressure is set to zero.

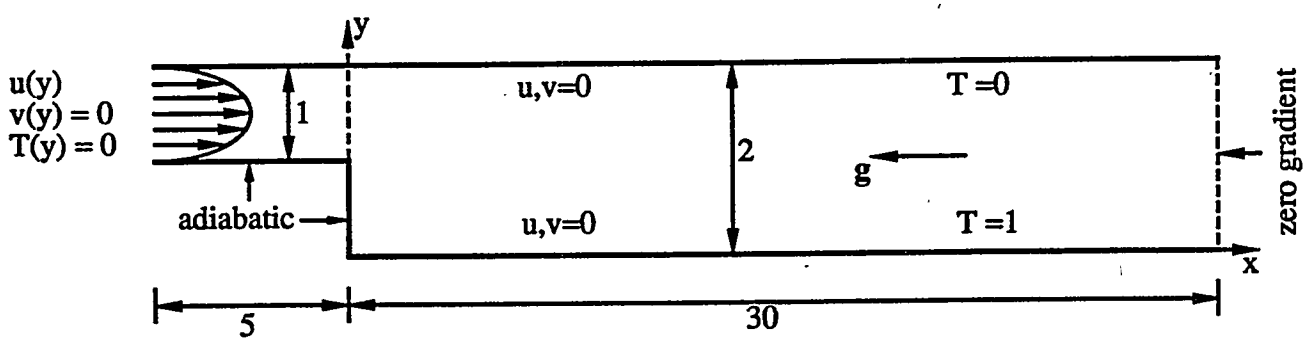


Figure 11. Schematic of mixed convection, backward facing step model problem.

The characteristic values used in the nondimensionalization of the governing equations are the same as those listed in Equation (43), except the velocity scale is defined as

the bulk average inlet velocity, the length scale is defined to be the step height, and $\phi = 90$. The values assumed for the nondimensional parameters are $Re \equiv 100$, $Pr = 0.7$, and either $Gr = 0$ or $Gr = 1000$. The former option for the Grashof number specifies a forced convection flow, while the latter option enables the effects of buoyancy to be determined.

In addition to the dimensionless flow field and temperature field, two additional solution parameters are of particular interest to a design engineer. The first is the dimensionless heat transfer coefficient along the hot and cold walls, namely the Nusselt number, Nu, defined by

$$Nu = \frac{\tilde{h}\tilde{L}}{\tilde{k}}, \tag{46}$$

where $\tilde{h}$ is the heat transfer coefficient and $\tilde{k}$ is the thermal conductivity [see 2, 111]. It is assumed that $\tilde{h}$ is defined by,

$$\tilde{h} = \frac{\tilde{q}''}{\tilde{T}_H - \tilde{T}_0}, \tag{47}$$

where $\tilde{q}''$ is the heat flux in the y-direction. Thus, along the hot wall $\tilde{q}''$ represents the heat flux into the channel, while on the cold wall it represents the heat flux out of the channel. Similarly, $\tilde{k}$ is defined in terms of the heat flux by,

$$\tilde{k} = -\frac{\tilde{q}''}{\left(\partial\tilde{T}/\partial\tilde{y}\right)_{wall}}. \tag{48}$$

Substituting for $\tilde{h}$ and $\tilde{k}$ using the above expressions and using the definitions in Equation (42), Equation (46) can be written as,

$$Nu = -\left(\frac{\partial T}{\partial y}\right)_{wall}, \tag{49}$$

which defines the Nusselt number in terms of the dimensionless wall temperature gradient.

A second important solution parameter of interest is the wall skin friction coefficient defined by,

$$C_{f_w} = \frac{\tilde{\tau}_w}{\frac{1}{2}\tilde{\rho}\tilde{V}^2}. \tag{50}$$

The wall shear stress, $\tilde{\tau}_w$, is given by,

$$\tilde{\tau}_w = \begin{cases} \tilde{\mu}\dfrac{\partial \tilde{u}}{\partial \tilde{y}}, & Hot\ \ wall\ \ (\tilde{y}=0) \\ -\tilde{\mu}\dfrac{\partial \tilde{u}}{\partial \tilde{y}}, & Cold\ \ wall\ \ (\tilde{y}=2\tilde{L}) \end{cases}, \tag{51}$$

where $\tilde{\mu}$ is the dynamic viscosity. The density dependence in Equation (50) can be eliminated by multiplying by the Reynolds number ($Re = \frac{\tilde{\rho}\tilde{V}\tilde{L}}{\tilde{\mu}}$). The quantity $C_{f_w}Re$ can then be expressed as

$$C_{f_w}Re = \begin{cases} 2\dfrac{\partial u}{\partial y}, & Hot\ \ wall\ \ (y=0) \\ -2\dfrac{\partial u}{\partial y}, & Cold\ \ wall\ \ (y=2) \end{cases}, \tag{52}$$

which defines the skin friction coefficient in terms of the dimensionless principal velocity gradient at the wall. By convention, the quantity $C_{f_w}Re$ is positive if the principle (along the

channel) velocity (u) near the wall is positive and negative if the principle velocity near the wall is negative (i.e., reversed flow).

2.1.5. Forced Convection, Backward Facing Step Model Problem

This section describes a forced convection, backward facing step model problem. This problem was also defined by *ASME K-12 Aerospace Heat Transfer Committee* as a benchmark problem for code validation and assessment [see 9, 112]. The hydrodynamic problem is the same as that solved by Gartling [113]. This backward facing step model problem differs from the one described in Section 2.1.4 with respect to geometry, boundary conditions, and assumed dimensionless parameter values. A schematic illustration of this model problem is given in Figure 12.

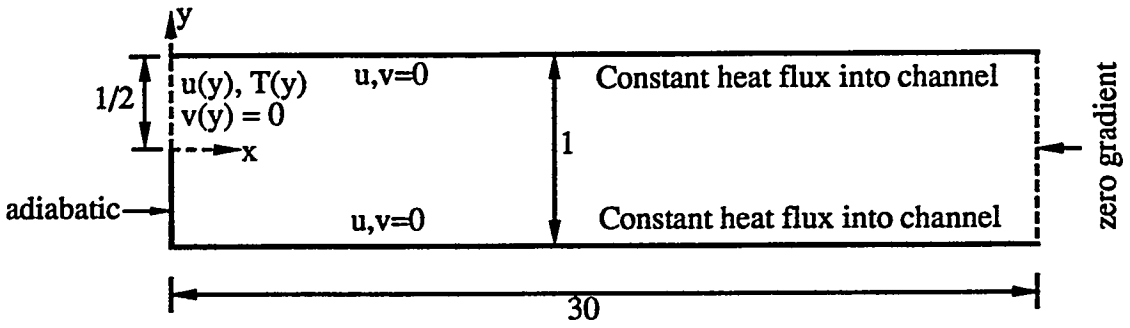


Figure 12. Schematic of forced convection, backward facing step model problem.

The characteristic values used in the nondimensionalization of the governing equations are listed below:

$$u=\frac{\tilde{u}}{\tilde{V}},\ v=\frac{\tilde{v}}{\tilde{V}},\ p=\frac{\tilde{p}-\tilde{p}_0}{\tilde{\rho}_0\tilde{V}^2},\ T=\frac{\tilde{T}-\tilde{T}_{wall}}{\tilde{T}_{cl}-\tilde{T}_{wall}},\ x=\frac{\tilde{x}}{\tilde{L}},\ y=\frac{\tilde{y}}{\tilde{L}}. \tag{53}$$

In this case, the velocity scale is defined as the bulk average inlet velocity, while the length scale is defined to be the channel width. Dimensionless temperature is defined with respect to the inlet centerline ('cl') and wall ('wall') values, respectively, since there are no fixed hot and cold wall temperatures for this problem. There is no hydrostatic term in the definition of the dimensionless pressure since gravitational effects are ignored in this model problem. The values assumed for the nondimensional parameters are $Re \equiv 800$, $Pr = 0.7$, and $Gr = 0$.

The inlet dimensionless velocity and temperature profiles at $x = 0$ and $0 \le y \le 1/2$ are assumed to be fully developed and are specified as,

$$u(y)=\frac{3}{2}(4y)(2-4y), \tag{54}$$

$$T(y)=1-\left[1-(1-4y)^2\right]\left[1-\frac{1}{5}(1-4y)^2\right]. \tag{55}$$

All fixed walls are no-slip (zero velocity) boundaries. Both the upper and lower walls are heated, with constant heat flux given by,

$$\tilde{q}''=\frac{32}{5}\frac{\tilde{k}}{\tilde{L}}(T_{wall}-T_{cl}), \tag{56}$$

which is the heat flux (into channel) that yields the dimensionless temperature profile expressed by Equation (55). This heat flux corresponds to dimensionless wall temperature gradients specified as,

$$\frac{\partial T}{\partial y} = \begin{cases} +\frac{32}{5}, & y = 0.5 \\ -\frac{32}{5}, & y = -0.5 \end{cases} . \tag{57}$$

Note that the inlet temperature profile and upper and lower wall temperature boundary conditions are consistent in that the fluid is assumed heated both upstream and downstream of the step. The wall associated with the step is assumed to be well insulated (i.e., zero heat flux). Along the outflow boundary, the following fully developed, zero gradient boundary conditions are enforced,

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial x} = \frac{\partial T}{\partial x} = 0. \tag{58}$$

In this problem the heat flux is specified, while the wall temperatures (T_w) are unknown. Consequently, there is no fixed temperature difference, upon which the local wall heat transfer coefficient can be based, as was done in Equation (47). Thus, it is necessary to introduce a quantity called the bulk fluid temperature [see 2, 111], which is defined at each x-location by,

$$T_{bulk} = \int_{-0.5}^{0.5} uT dy \Big/ \int_{-0.5}^{0.5} u dy. \tag{59}$$

This enables the local heat transfer coefficient to be defined as,

$$\tilde{h} = \frac{\tilde{q}''}{\tilde{T}_w - \tilde{T}_{bulk}}. \tag{60}$$

The Nusselt number for this model problem is defined based upon the hydraulic diameter ($\tilde{D}_H$) and not the step height [see 2, 111] so that,

$$Nu = \frac{\tilde{h}\tilde{D}_H}{\tilde{k}} = \frac{\tilde{h}(2\tilde{L})}{\tilde{k}}. \tag{61}$$

Thus, these expressions give rise to a Nusselt number, defined in terms of the dimensionless variables, that can be expressed as,

$$Nu = \begin{cases} +\dfrac{2}{T_w - T_{bulk}} \dfrac{\partial T}{\partial y} = \dfrac{64}{5} \dfrac{1}{T_w - T_{bulk}}, & y = +0.5 \\ -\dfrac{2}{T_w - T_{bulk}} \dfrac{\partial T}{\partial y} = \dfrac{64}{5} \dfrac{1}{T_w - T_{bulk}}, & y = -0.5 \end{cases} \tag{62}$$

The Nusselt number given in Equation (62) is a function of the wall and bulk temperatures at each x-location. The skin friction coefficient for this problem can be defined in a manner similar to that described for the mixed convection, backward facing step problem defined in Section 2.1.4.

2.2. COMPRESSIBLE FLUID FLOW AND HEAT TRANSFER

This section describes the partial differential equations that describe two-dimensional, laminar, compressible fluid flow and heat transfer. These equations also represent the transport of mass, momentum, and energy; but unlike the incompressible flow case, density is assumed variable everywhere. Thus, there are five unknowns in this system including density, two components of velocity, pressure, and temperature (energy). Since only four transport equations exist, an additional equation is required for closure. This additional

equation arises from a thermodynamic equation of state, which relates the three thermodynamic variables.

The compressible flow equations solved in this work are presented in Section 2.2.1. The finite volume discretization scheme used to convert this continuous set of partial differential equations into a system of discrete algebraic equations is described in Section 2.2.2. Finally, the compressible flow model problem used to investigate the numerical solution techniques of Chapter 3 is described in Section 2.2.3.

2.2.1. Governing Equations

The partial differential equations, representing constant property laminar flow of a compressible fluid, are presented below in dimensionless form:

Continuity:

$$\frac{\partial \rho u}{\partial x}+\frac{\partial \rho v}{\partial y}=0 \tag{63}$$

Momentum:

$$\frac{\partial \rho u^2}{\partial x}+\frac{\partial \rho uv}{\partial y}=-\frac{\partial p}{\partial x}+\frac{1}{Re}\frac{\partial}{\partial x}\left[2\frac{\partial u}{\partial x}-\frac{2}{3}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)\right]+\frac{1}{Re}\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right) \tag{64}$$

$$\frac{\partial \rho uv}{\partial x}+\frac{\partial \rho v^2}{\partial y}=-\frac{\partial p}{\partial y}+\frac{1}{Re}\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)+\frac{1}{Re}\frac{\partial}{\partial y}\left[2\frac{\partial v}{\partial y}-\frac{2}{3}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)\right] \tag{65}$$

Energy:

$$\frac{\partial \rho u T}{\partial x}+\frac{\partial \rho v T}{\partial y}=\frac{\gamma}{Pec}\left[\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right]-\gamma(\gamma-1)M_i^2 p\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right) \tag{66}$$

Equation of state:

$$p=\frac{\rho T}{\gamma M_i^2} \tag{67}$$

where ρ is now the fluid density and constant transport properties have been assumed along with Stokes condition [1]. Additionally, heat generation, viscous dissipation, and the effects of buoyancy forces have been neglected. Note that because this investigation is concerned with low Mach number compressible flow, the effects of viscous dissipation should be negligible. Additionally, the model problem described in Section 2.2.3 does not include any sources of energy and only adiabatic temperature boundary conditions. Consequently, only slight changes in temperature occur, which makes the assumptions of constant properties and negligible buoyancy effects reasonable. The Peclet number, Pec, appearing in Equation (66) is defined as the product of the Reynolds number and the Prandtl number, i.e., $Pec = Re \cdot Pr$. M_i is the reference Mach number based upon a reference velocity and temperature, and γ is the ratio of the specific heat capacities. Note that the thermodynamic equation of state in Equation (67) assumes a perfect gas. In this work, the state equation is used to eliminate pressure from the conservation equations so that the four unknown variables are u, v, ρ, and T.

The characteristic parameters used to nondimensionalize the governing equations are identified below [see 1], where once again a tilde indicates a dimensional quantity:

$$u=\frac{\tilde{u}}{\tilde{V}},\ v=\frac{\tilde{v}}{\tilde{V}},\ p=\frac{\tilde{p}}{\tilde{\rho}_0\tilde{V}^2},\ T=\frac{\tilde{T}}{\tilde{T}_0},\ \rho=\frac{\tilde{\rho}}{\tilde{\rho}_0},\ x=\frac{\tilde{x}}{\tilde{L}},\ y=\frac{\tilde{y}}{\tilde{L}}. \tag{68}$$

The subscript '0' indicates a reference thermodynamic quantity. The specific velocity and length scales, and reference conditions for the model problem of interest are defined in Section 2.2.3.

2.2.2. Discretization of Governing Equations

The governing equations presented in the previous section are discretized on a finite volume mesh similar to that described in Section 2.1.2. The only difference between the incompressible flow mesh described previously and the compressible flow mesh used here is that the pressure variable, p, in each computational cell is replaced with the density variable, ρ. This replacement is required because the equation of state is used to eliminate pressure dependencies.

The development of the discretized compressible flow equations proceeds in the same manner as for the incompressible flow case. The continuity and energy equations are integrated over the standard finite volume cell, while the components of the momentum equation are integrated over the appropriate momentum cell. Once again, the basic feature of the discretization process is the use of the divergence theorem to convert volume integrals of divergence form to surface integrals, which are then be approximated by summing quantities evaluated at cell faces. One important difference in the discretization of the compressible flow equations is that pure upwinding [see 19, 20] is used to evaluate the convective terms instead of the power-law convection-diffusion formulation that was employed in the discretization of the incompressible flow equations. The use of this technique is illustrated

below in the development of the discretized forms of the continuity, momentum, and energy equations.

The continuity equation can be written in vector form as,

$$\nabla \cdot (\rho \mathbf{u}) = 0. \tag{69}$$

This equation is then integrated over the finite volume stencil shown in Figure 13, where the variables identified in this figure are used to evaluate the necessary integral quantities. This integration process yields,

$$\iiint_V \nabla \cdot (\rho \mathbf{u}) dV = \iint_S (\rho \mathbf{u}) \cdot \hat{\mathbf{n}} dS \cong \left[(\rho u)_e - (\rho u)_w\right] \Delta y_j + \left[(\rho v)_n - (\rho v)_s\right] \Delta x_i. \tag{70}$$

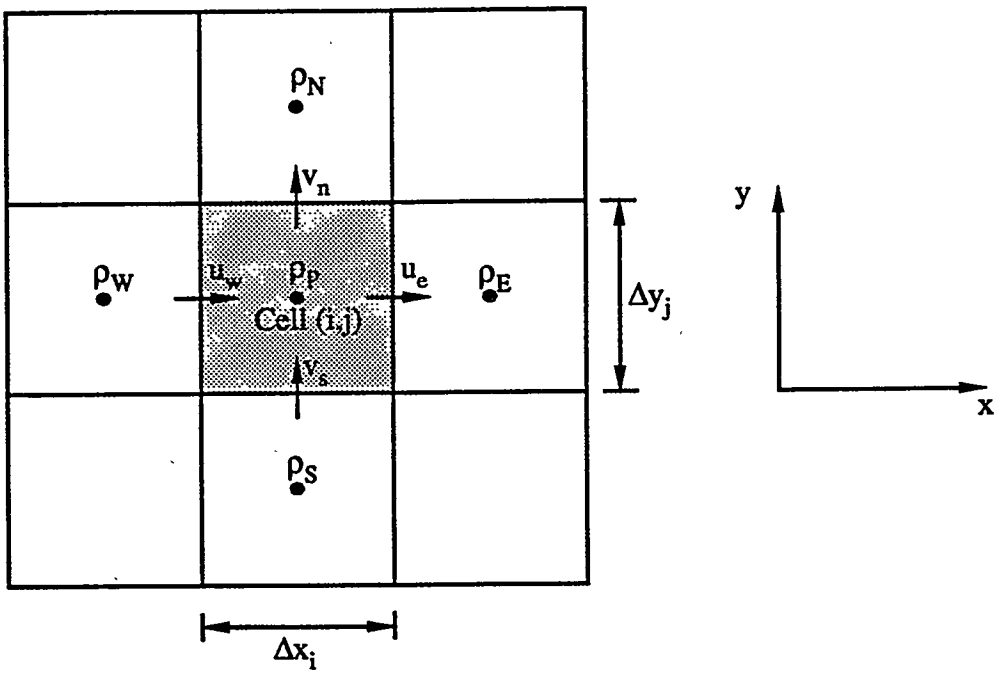


Figure 13. Finite volume stencil used to discretize compressible flow continuity equation.

The quantities in parenthesis, which must be evaluated at cell faces, are computed using upwinded values for the density. These quantities can be computed as follows:

$$\begin{aligned}
(\rho u)_e &= \rho_P Max(u_e,0) - \rho_E Max(-u_e,0), \\
(\rho u)_w &= -\rho_P Max(-u_w,0) + \rho_W Max(u_w,0), \\
(\rho v)_n &= \rho_P Max(v_n,0) - \rho_N Max(-v_n,0), \text{ and} \\
(\rho v)_s &= -\rho_P Max(-v_s,0) + \rho_S Max(v_s,0).
\end{aligned} \tag{71}$$

Similarly, the u-momentum equation can be written in the following divergence form,

$$\nabla \cdot (\rho \mathbf{u} u) + \nabla \cdot \left(p\hat{\mathbf{i}}\right) + \frac{1}{Re}\nabla \cdot (\sigma_x) = 0, \tag{72}$$

where

$$\sigma_x = \bar{\sigma} \cdot \hat{\mathbf{i}} = \left(\nabla u + \frac{\partial \mathbf{u}}{\partial x}\right) - \frac{2}{3}\nabla \cdot \mathbf{u}, \tag{73}$$

and $\bar{\sigma}$ represents the viscous stress tensor. This form of the u-momentum is then integrated over the shaded finite volume shown in Figure 14. Application of the divergence theorem to this integral results in three surface integrals corresponding to the three terms in Equation (72). These surface integrals can be approximated numerically by converting the integrals to summations over the cell faces. This process yields the following discretization equation,

$$\begin{aligned}
&\left\{[(\rho u)u]_e - [(\rho u)u]_w\right\}\Delta y_j + \left\{[(\rho v)u]_n - [(\rho v)u]_s\right\}\left(\frac{\Delta x_{i-1} + \Delta x_i}{2}\right) + (p_e - p_w)\Delta y_j \\
&-\frac{1}{Re}\left\{\left[\left(\sigma_x \cdot \hat{\mathbf{i}}\right)_e - \left(\sigma_x \cdot \hat{\mathbf{i}}\right)_w\right]\Delta y_j + \left[\left(\sigma_x \cdot \hat{\mathbf{j}}\right)_n - \left(\sigma_x \cdot \hat{\mathbf{j}}\right)_s\right]\left(\frac{\Delta x_{i-1} + \Delta x_i}{2}\right)\right\} = 0.
\end{aligned} \tag{74}$$

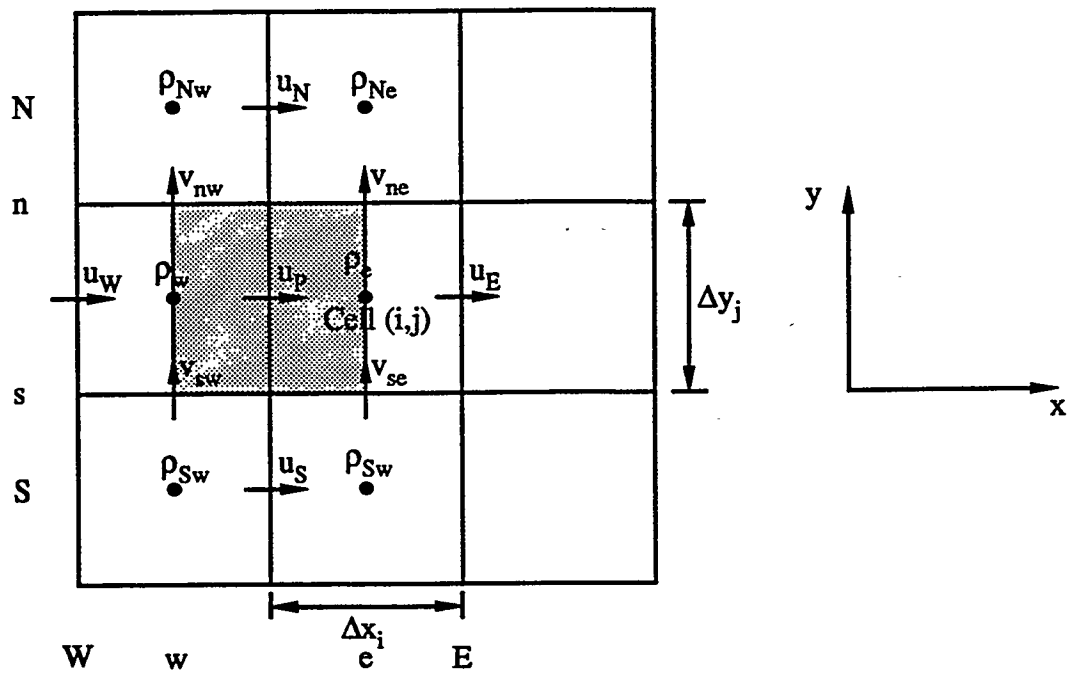


Figure 14. Finite volume stencil used to discretize compressible flow *u*-momentum equation.

The first two terms in brackets in Equation (74) contain the convection terms. The terms in parenthesis, within the brackets, are interpreted as the convecting quantities, while the terms they multiply are interpreted as convected quantities. In this implementation, the convected quantities are upwinded, so that the convection terms are computed from:

$$\begin{aligned}
\left[(\rho u)u\right]_e &= u_P Max\left\{(\rho u)_e, 0\right\} - u_E Max\left\{-(\rho u)_e, 0\right\}, \\
\left[(\rho u)u\right]_w &= -u_P Max\left\{-(\rho u)_w, 0\right\} + u_W Max\left\{(\rho u)_w, 0\right\}, \\
\left[(\rho v)u\right]_n &= u_P Max\left\{(\rho v)_n, 0\right\} - u_N Max\left\{-(\rho v)_n, 0\right\}, \text{ and} \\
\left[(\rho v)u\right]_s &= -u_P Max\left\{-(\rho v)_s, 0\right\} + u_S Max\left\{(\rho v)_s, 0\right\};
\end{aligned} \tag{75}$$

where

$$
\begin{aligned}
(\rho u)_e &= \rho_e\left(\frac{u_E + u_P}{2}\right), \\
(\rho u)_w &= \rho_w\left(\frac{u_W + u_P}{2}\right), \\
(\rho v)_n &= \left(\frac{\rho_P \Delta y_{j+1} + \rho_N \Delta y_j}{\Delta y_j + \Delta y_{j+1}}\right)\left(\frac{v_{ne}\Delta x_{i-1} + v_{nw}\Delta x_i}{\Delta x_i + \Delta x_{i-1}}\right), \text{ and} \\
(\rho v)_s &= \left(\frac{\rho_P \Delta y_{j-1} + \rho_S \Delta y_j}{\Delta y_j + \Delta y_{j-1}}\right)\left(\frac{v_{se}\Delta x_{i-1} + v_{sw}\Delta x_i}{\Delta x_i + \Delta x_{i-1}}\right),
\end{aligned}
\tag{76}
$$

and

$$
\begin{aligned}
\rho_N &= \left(\frac{\rho_{Ne}\Delta x_{i-1} + \rho_{Nw}\Delta x_i}{\Delta x_i + \Delta x_{i-1}}\right), \\
\rho_P &= \left(\frac{\rho_e \Delta x_{i-1} + \rho_w \Delta x_i}{\Delta x_i + \Delta x_{i-1}}\right), \text{ and} \\
\rho_S &= \left(\frac{\rho_{Se}\Delta x_{i-1} + \rho_{Sw}\Delta x_i}{\Delta x_i + \Delta x_{i-1}}\right).
\end{aligned}
\tag{77}
$$

The pressure terms in Equation (74) are evaluated using the equation of state as follows,

$$
p_e = \frac{\rho_e T_e}{\gamma M_i^2} \text{ and } p_w = \frac{\rho_w T_w}{\gamma M_i^2}. \tag{78}
$$

The remaining viscous terms are computed from the following expressions:

$$
\begin{aligned}
\left(\sigma_x \cdot \hat{\mathbf{i}}\right)_e &= \left(2\frac{\partial u}{\partial x}\right)_e - \frac{2}{3}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)_e, \\
\left(\sigma_x \cdot \hat{\mathbf{i}}\right)_w &= \left(2\frac{\partial u}{\partial x}\right)_w - \frac{2}{3}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)_w, \\
\left(\sigma_x \cdot \hat{\mathbf{j}}\right)_n &= \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)_n, \text{ and } \left(\sigma_x \cdot \hat{\mathbf{j}}\right)_s = \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)_s,
\end{aligned}
\tag{79}
$$

where the derivative terms appearing in Equation (79) can all be computed on the appropriate cell face using conventional central differencing with the variables appearing in Figure 14.

The procedure for discretizing the *v*-momentum equation is analogous to the procedure for the *u*-momentum equation. The vector form of the v-momentum equation can be expressed as,

$$\nabla\cdot(\rho\mathbf{u}v)+\nabla\cdot\left(p\hat{\mathbf{j}}\right)+\frac{1}{Re}\nabla\cdot\left(\sigma_y\right)=0, \tag{80}$$

where

$$\sigma_y=\bar{\sigma}\cdot\hat{\mathbf{j}}=\left(\nabla v+\frac{\partial\mathbf{u}}{\partial y}\right)-\frac{2}{3}\nabla\cdot\mathbf{u}, \tag{81}$$

and $\bar{\sigma}$ again represents the viscous stress tensor. This form of the *v*-momentum is then integrated over the shaded finite volume shown in Figure 15. Application of the divergence theorem to this integral results in three surface integrals corresponding to the three terms of divergence form in Equation (80). These surface integrals are approximated by summations over the cell faces. This process yields the following discretization equation,

$$\begin{aligned}&\left\{\left[(\rho u)v\right]_e-\left[(\rho u)v\right]_w\right\}\left(\frac{\Delta y_{j-1}+\Delta y_j}{2}\right)+\left\{\left[(\rho v)v\right]_n-\left[(\rho v)v\right]_s\right\}\Delta x_i+(p_n-p_s)\Delta x_i\\&-\frac{1}{Re}\left\{\left[\left(\sigma_y\cdot\hat{\mathbf{i}}\right)_e-\left(\sigma_y\cdot\hat{\mathbf{i}}\right)_w\right]\left(\frac{\Delta y_{j-1}+\Delta y_j}{2}\right)+\left[\left(\sigma_y\cdot\hat{\mathbf{j}}\right)_n-\left(\sigma_y\cdot\hat{\mathbf{j}}\right)_s\right]\Delta x_i\right\}=0.\end{aligned} \tag{82}$$

The convected quantities are upwinded as was done for the *u*-momentum equation, so that the convection terms are computed from:

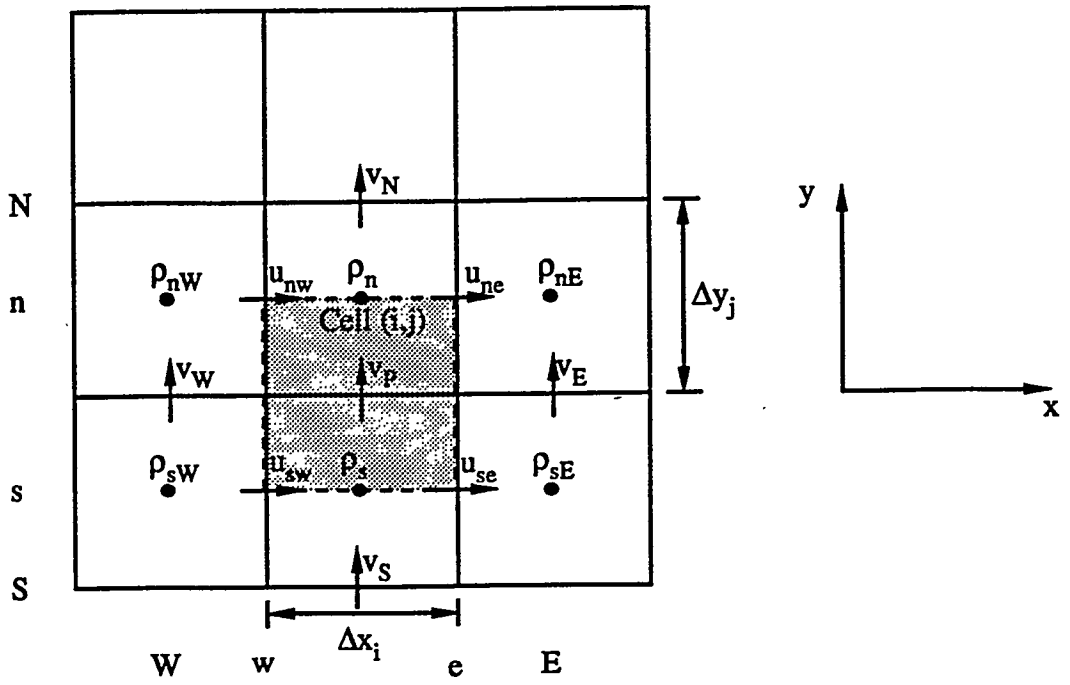


Figure 15. Finite volume stencil used to discretize compressible flow v-momentum equation.

$$
\begin{aligned}
\left[(\rho u)v\right]_e &= v_P Max\left\{(\rho u)_e, 0\right\} - v_E Max\left\{-(\rho u)_e, 0\right\}, \\
\left[(\rho u)v\right]_w &= -v_P Max\left\{-(\rho u)_w, 0\right\} + v_W Max\left\{(\rho u)_w, 0\right\}, \\
\left[(\rho v)v\right]_n &= v_P Max\left\{(\rho v)_n, 0\right\} - v_N Max\left\{-(\rho v)_n, 0\right\}, \text{ and} \\
\left[(\rho v)v\right]_s &= -v_P Max\left\{-(\rho v)_s, 0\right\} + v_S Max\left\{(\rho v)_s, 0\right\};
\end{aligned}
\tag{83}
$$

where

$$
\begin{aligned}
(\rho u)_e &= \left(\frac{\rho_P \Delta x_{i+1} + \rho_E \Delta x_i}{\Delta x_i + \Delta x_{i+1}}\right)\left(\frac{u_{ne}\Delta y_{j-1} + u_{se}\Delta y_j}{\Delta y_j + \Delta y_{j-1}}\right), \\
(\rho u)_w &= \left(\frac{\rho_P \Delta x_{i-1} + \rho_W \Delta x_i}{\Delta x_i + \Delta x_{i-1}}\right)\left(\frac{u_{nw}\Delta y_{j-1} + u_{sw}\Delta y_j}{\Delta y_j + \Delta y_{j-1}}\right), \\
(\rho v)_n &= \rho_n\left(\frac{v_N + v_P}{2}\right), \text{ and } (\rho v)_s = \rho_c\left(\frac{v_S + v_P}{2}\right);
\end{aligned}
\tag{84}
$$

and

$$\rho_E = \left(\frac{\rho_{nE}\Delta y_{j-1} + \rho_{sE}\Delta y_j}{\Delta y_j + \Delta y_{j-1}} \right),$$
$$\rho_P = \left(\frac{\rho_n \Delta y_{j-1} + \rho_s \Delta y_j}{\Delta y_j + \Delta y_{j-1}} \right), \text{ and} \tag{85}$$
$$\rho_W = \left(\frac{\rho_{nW}\Delta y_{j-1} + \rho_{sW}\Delta y_j}{\Delta y_j + \Delta y_{j-1}} \right).$$

The pressure terms in Equation (82) are evaluated using the equation of state as follows,

$$p_n = \frac{\rho_n T_n}{\gamma M_i^2} \text{ and } p_s = \frac{\rho_s T_s}{\gamma M_i^2}. \tag{86}$$

The remaining viscous terms are computed from the following expressions:

$$\left(\sigma_y \cdot \hat{\mathbf{i}}\right)_e = \left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)_e, \left(\sigma_y \cdot \hat{\mathbf{i}}\right)_w = \left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)_w,$$
$$\left(\sigma_y \cdot \hat{\mathbf{j}}\right)_n = \left(2\frac{\partial v}{\partial y}\right)_n - \frac{2}{3}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)_n, \text{ and} \tag{87}$$
$$\left(\sigma_y \cdot \hat{\mathbf{j}}\right)_s = \left(2\frac{\partial v}{\partial y}\right)_s - \frac{2}{3}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)_s,$$

with central differencing being used to approximate the necessary derivatives.

The energy equation can be written as,

$$\nabla \cdot (\rho \mathbf{u} T) - \frac{\gamma}{Pec}\nabla \cdot (\nabla T) + \gamma(\gamma - 1)M_i^2 p(\nabla \cdot \mathbf{u}) = 0. \tag{88}$$

Note that the first two terms of Equation (88) are of divergence form, while the last is not. Thus, the divergence theorem may be applied to the first two terms, while the last term represents a volumetric source that must be integrated over the entire volume. Integration of Equation (88) over the shaded region shown in Figure 16 then yields the following discrete equation,

$$\begin{aligned}&\left\{\left[u(\rho T)\right]_e-\left[u(\rho T)\right]_w\right\}\Delta y_j+\left\{\left[v(\rho T)\right]_n-\left[v(\rho T)\right]_s\right\}\Delta x_i\\&-\frac{\gamma}{Pec}\left\{\left[\left(\frac{\partial T}{\partial x}\right)_e-\left(\frac{\partial T}{\partial x}\right)_w\right]\Delta y_j+\left[\left(\frac{\partial T}{\partial y}\right)_n-\left(\frac{\partial T}{\partial y}\right)_s\right]\Delta x_i\right\}\\&+\gamma(\gamma-1)M_i^2 p_P\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)_P\Delta x_i\Delta y_j=0.\end{aligned}\tag{89}$$

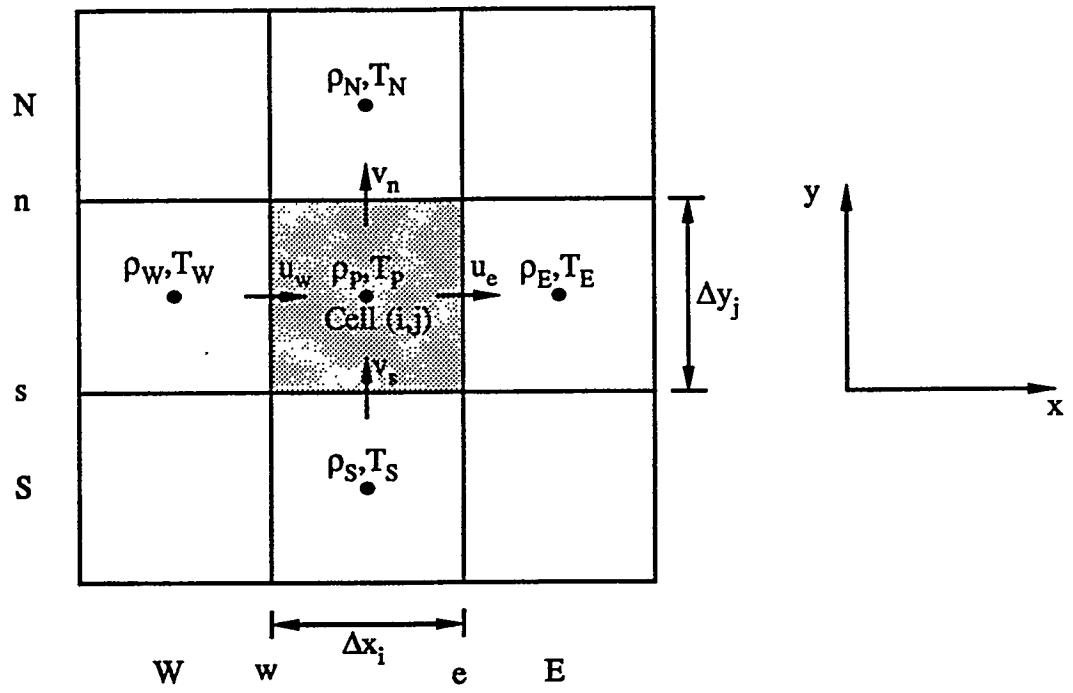


Figure 16. Finite volume stencil used to discretize compressible flow energy equation.

The convected quantities (ρT) are again upwinded, so that the convection terms are computed from:

$$\begin{aligned}
\left[u(\rho T)\right]_e &= (\rho T)_P Max\{u_e,0\} - (\rho T)_E Max\{-u_e,0\},\\
\left[u(\rho T)\right]_w &= -(\rho T)_P Max\{-u_w,0\} + (\rho T)_W Max\{u_w,0\},\\
\left[v(\rho T)\right]_n &= (\rho T)_P Max\{v_n,0\} - (\rho T)_N Max\{-v_n,0\}, \text{ and}\\
\left[v(\rho T)\right]_s &= -(\rho T)_P Max\{-v_s,0\} + (\rho T)_S Max\{v_s,0\}.
\end{aligned} \tag{90}$$

The pressure in Equation (89) is evaluated using the equation of state,

$$p_P = \frac{\rho_P T_P}{\gamma M_i^2}, \tag{91}$$

while central differencing is used to approximate the necessary velocity and temperature derivatives appearing in Equation (89).

2.2.3. Backward Facing Step Model Problem

The model problem described here is compressible flow past a backward facing step. The model problem geometry is shown in Figure 17. The velocity scale used in the nondimensionalization in Equation (31) is defined as the uniform inlet velocity, while the length scale is defined as the step height. Note that all rigid walls are assumed to have zero velocity and are adiabatic. The Mach number that appears in Equation (67) and Equation (30) is defined with respect to the inlet velocity and temperature conditions,

$$M_i = \frac{\tilde{V}}{\sqrt{\gamma \tilde{R} \tilde{T}_0}}, \tag{92}$$

where $\tilde{R}$ is the gas constant of the fluid. At the inlet, the principle (along the channel) velocity and temperature are assumed uniform, while the transverse velocity (across the channel) and the density gradient are set to zero. At the outlet, the pressure is fixed at its reference value, which is determined by evaluating Equation (30) at the reference density and temperature, i.e.,

$$p = \left(\frac{\rho T}{\gamma M_i^2} \right)_0 = \frac{1}{\gamma M_i^2}, \tag{93}$$

while the following fully developed, zero gradient conditions are enforced,

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial x} = \frac{\partial T}{\partial x} = 0. \tag{94}$$

Note that density at the outlet is defined via the equation of state in Equation (67), which relates density to the pressure and temperature at the outlet.

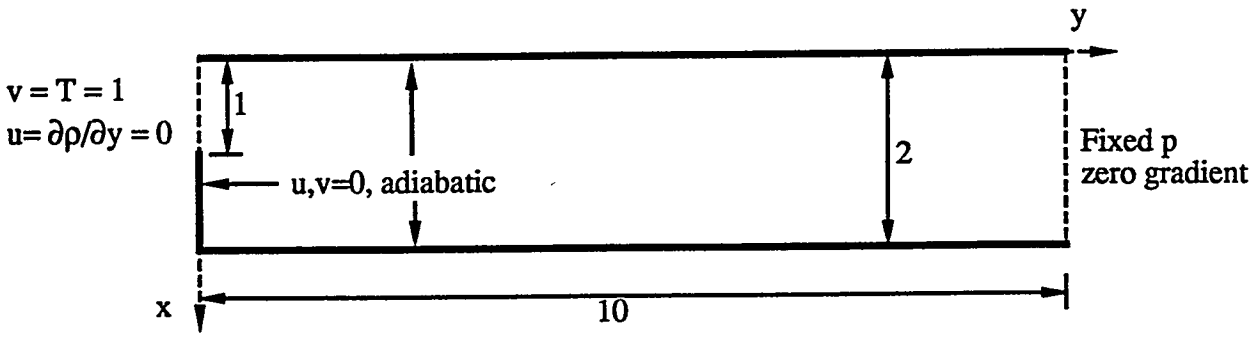


Figure 17. Schematic of compressible, backward facing step model problem.

CHAPTER 3

NUMERICAL SOLUTION ALGORITHM

The finite volume discretization of the governing equations described in the previous chapter results in a system of nonlinear discrete algebraic equations that must be solved for the primitive variable unknowns. This chapter describes the use of a fully coupled solution algorithm to solve these nonlinear systems. The foundation of the nonlinear solution algorithm is the use of Newton's method to linearize the nonlinear algebraic equations. The resulting linear system of equations are in turn solved on each Newton step using preconditioned Krylov subspace based iterative methods. Use of iterative techniques with Newton's method is often referred to as inexact Newton's method because the linear systems arising on each step are typically not solved exactly on each iteration. Additionally, several algorithm enhancements are used to improve overall algorithm robustness and to simplify algorithm implementation. The robustness enhancements techniques include adaptive damping of the Newton updates, mesh sequencing to enlarge the radius of convergence, pseudo-transient relaxation, and simple continuation techniques. The large, complex Jacobian matrices that arise on each Newton step are evaluated numerically in order to simplify implementation. These features of the solution algorithm are discussed in more detail in the subsections that follow.

3.1. NEWTON'S METHOD AND ALGORITHM PERFORMANCE ENHANCEMENT TECHNIQUES

Newton's method is a powerful technique for solving systems of nonlinear equations of the form,

$$\mathbf{F}(\mathbf{x}) = \left[f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_N(\mathbf{x})\right]^T = 0, \tag{95}$$

where N is the dimension of the system (number of unknowns) and the state variable vector, $\mathbf{x}$, can be expressed as,

$$\mathbf{x} = \left[x_1, x_2, \ldots, x_N\right]^T. \tag{96}$$

Newton's method is derived from a first order Taylor's series expansion of $\mathbf{F}(\mathbf{x}^{n+1})$ about the approximate solution $\mathbf{x}^n$, where n is the Newton iteration number. This approximation can be expressed as,

$$\mathbf{F}(\mathbf{x}^{n+1}) \approx \mathbf{F}(\mathbf{x}^n) + \frac{\partial \mathbf{F}}{\partial \mathbf{x}^n}\delta\mathbf{x}^n \approx \mathbf{F}(\mathbf{x}^n) + \mathbf{J}^n\delta\mathbf{x}^n. \tag{97}$$

The goal of Newton's method is to pick the solution update, $\delta\mathbf{x}^n$, in order to drive $\mathbf{F}(\mathbf{x}^{n+1})$ to zero. Thus, application of Newton's method requires the solution of the linear system given by,

$$\mathbf{J}^n\delta\mathbf{x}^n = -\mathbf{F}(\mathbf{x}^n), \tag{98}$$

where the elements of the Jacobian matrix, $\mathbf{J}^n$, are defined by,

$$J^n_{ROW,COL} = \frac{\partial f^n_{ROW}}{\partial x^n_{COL}}, \tag{99}$$

and the new solution approximation is obtained from,

$$\mathbf{x}^{n+1} = \mathbf{x}^n + s \cdot \delta\mathbf{x}^n. \tag{100}$$

The scalar, s, which lies between zero and one, is used to damp the update. The damping strategy is discussed further in Section 3.1.2. This iteration is continued until the norm of $\delta\mathbf{x}^n$ and the norm of $\mathbf{F}(\mathbf{x}^n)$ are below some suitable tolerance level. Note that this Newton iteration is often referred to in this work as the outer iteration, while the Krylov iteration (described in Section 3.2.1) used to solve Equation (98) is referred to as the inner iteration. The convergence criteria for the outer Newton iteration is based upon a relative update defined by

$$R^o_n = \underset{all\ m}{Max}\left[\frac{|\delta x^n_m|}{Max\{|x^n_m|,1\}}\right], \tag{101}$$

where the superscript on R^o_n refers to the outer Newton iteration and the subscript indicates the dependence on the Newton iteration. Convergence is then assumed when

$$R^o_n < 1\text{x}10^{-6} \quad and \quad \|\mathbf{F}(\mathbf{x}^n)\|_\infty < 1\text{x}10^{-6}. \tag{102}$$

The first criteria requires six digits of accuracy when the magnitude of the state variable is greater than one, and six decimal places of accuracy are required when the magnitude of the

state variable is less than one. The second criteria simply requires the maximum steady state residual to be less than $1x10^{-6}$.

The state variable vector is defined in terms of the variables of the problem. For incompressible flow the primitive variables are u, v, p, and T. For compressible flow, p is replaced by ρ. A natural ordering of these variables is assumed, which means that the primitive variables are numbered sequentially within a finite volume before moving to the next finite volume. In this manner, the state variable is defined for incompressible flow as,

$$\mathbf{x} = \left[u_1, v_1, p_1, T_1, \ldots, u_{INUM}, v_{INUM}, p_{INUM}, T_{INUM}, \ldots, u_{NC}, v_{NC}, p_{NC}, T_{NC}\right]^T, \tag{103}$$

while for compressible flow it becomes,

$$\mathbf{x} = \left[u_1, v_1, \rho_1, T_1, \ldots, u_{INUM}, v_{INUM}, \rho_{INUM}, T_{INUM}, \ldots, u_{NC}, v_{NC}, \rho_{NC}, T_{NC}\right]^T, \tag{104}$$

where the subscript now refers to the finite volume cell number (*INUM*) and *NC* is the total number of finite volume cells. Note that the total number of unknowns, N, equals *NC* times the number of variables/equations per cell, *NEQ*, which in this case is four. The cell number, *INUM*, and the equation number, *IEQ*, of interest then determine the row number of the Jacobian used in Equation (99), i.e.,

$$ROW = IEQ + (INUM - 1) \cdot NEQ. \tag{105}$$

The column number of the Jacobian element depends on the state variable dependency of interest. For example, assume the dependency is with respect to variable number *IEQD* in cell *INUMD*. The column number in Equation (99) is then determined from the offset or distance between the equation number (*IEQ*) in the current cell (*INUM*) [f^n_{ROW} in Equation

(99)] and the variable number (*IEQD*) in the dependent cell (*INUMD*) [x^n_{COL} in Equation (99)], i.e.,

$$COL = ROW + (IEQD - IEQ) + (INUMD - INUM) \cdot NEQ. \tag{106}$$

The state variable vector can also be expressed in terms of the grid indices if one knows how the finite volume cells are ordered, i.e., the grid dependence of *INUM*. Assume a finite volume cell within the grid is defined by an (i, j) location, where i is the cell number in the x-direction and j is the cell number in the y-direction. Assume for simplicity that row-ordering is used to number the finite volume cells. This ordering starts numbering with cell (1,1) and finishes numbering with cell (nx, ny) with i the fastest running index and nx and ny denoting the maximum number of cells in the x-direction and y-direction, respectively, i.e., $INUM(i, j) = i + (j-1) \cdot NEQ$ (see Section 3.2.3.2). These assumptions enable the state vector to be expressed in terms of the grid indices as follows,

$$\mathbf{x} = \left[u_{1,1}, v_{1,1}, p_{1,1}, T_{1,1}, u_{2,1}, v_{2,1}, p_{2,1}, T_{2,1}, \ldots, u_{nx,ny}, v_{nx,ny}, p_{nx,ny}, T_{nx,ny}\right]^T \tag{107}$$

for incompressible flow, and

$$\mathbf{x} = \left[u_{1,1}, v_{1,1}, \rho_{1,1}, T_{1,1}, u_{2,1}, v_{2,1}, \rho_{2,1}, T_{2,1}, \ldots, u_{nx,ny}, v_{nx,ny}, \rho_{nx,ny}, T_{nx,ny}\right]^T \tag{108}$$

for compressible flow.

The vector, $\mathbf{F(x)}$, represents the nonlinear, discrete algebraic equations resulting from the finite volume discretization of either the incompressible or the compressible flow governing equations. Each component of $\mathbf{F(x)}$ is aligned with a component of the state

vector, **x**. In this work, the x-component of the momentum equation, which can be denoted as $f^{u-momentum}$, is aligned with u; while the y-component of the momentum equation, $f^{v-momentum}$, is aligned with v. For incompressible flow, the continuity equation, $f^{continuity}$, is aligned with pressure, p, while for compressible flow this equation is aligned with density, ρ. Note that if the discrete pressure equation is solved in the case of incompressible flow, then this equation replaces $f^{continuity}$. Finally, the energy equation, f^{energy}, is aligned with temperature. For example, the state variable component, $u_{i,j}$, is aligned with the row in the Jacobian matrix associated with the x-component of the momentum equation discretized over the finite volume cell (i,j), i.e., $f_{i,j}^{u-momentum}$. With this notation the vector $\mathbf{F}(\mathbf{x})$ can be expressed as,

$$\mathbf{F}(\mathbf{x}) = \left[f_{1,1}^{u-momentum}, f_{1,1}^{v-momentum}, f_{1,1}^{continuity}, f_{1,1}^{energy}, \ldots, f_{nx,ny}^{u-momentum}, f_{nx,ny}^{v-momentum}, f_{nx,ny}^{continuity}, f_{nx,ny}^{energy} \right]^T, \quad (109)$$

corresponding to the state variable vector defined by Equation (107) or Equation (108). This choice for the alignment of equations and variables on a structured grid results in a Jacobian matrix with a sparse banded structure that can be easily exploited using non-zero diagonal type storage schemes and sparse iterative linear equation solvers.

The alignment of discrete equations and variables described above is a natural choice, but note that other alignments are possible. This possibility arises because the equations and variables are solved in a fully coupled fashion. For example, Vanka and Leaf [29] investigated aligning the incompressible continuity equation with the y-component of velocity, v, and the y-component of momentum with pressure, p. This choice eliminated the need for pivoting in the direct solve of Equation (98). The disadvantage in this approach, however, is that the banded structure of the matrix is altered and enforcement of boundary

conditions can be more difficult. For these reasons, the simpler alignment choice described above is employed in this study.

The advantages of using fully coupled Newton's method include the ability to solve the equations in a fully coupled fashion, and the characteristic quadratic convergence exhibited by Newton's method [see 66]. The latter convergence feature distinguishes Newton's method from other nonlinear solution techniques and makes it a very powerful method. This convergence property implies that the error from one iteration to the next is reduced quadratically as the true solution is approached, resulting in very rapid convergence.

The disadvantages in using Newton's method include [see 66]: the memory/CPU cost and difficulty associated with forming the Jacobian matrices on each Newton step, the relatively small radius of convergence of the method, and the possibility of very poorly conditioned Jacobians arising on a given Newton step that will make solution of Equation (98) very difficult. The quasi-Newton techniques described in the following subsections are attempts to address and circumvent some of these disadvantages. The high memory cost disadvantage is addressed in Section 3.2.

3.1.1. Numerical Jacobian Evaluation

The elements of the Jacobian in Equation (99) are evaluated numerically using finite difference approximations,

$$J_{ROW,COL} = \frac{\partial f_{ROW}}{\partial x_{COL}} = \frac{f_{ROW}(x_1, x_2, \ldots, x_{COL} + \Delta x_{COL}, \ldots, x_n) - f_{ROW}(x_1, x_2, \ldots, x_n)}{\Delta x_{COL}}, \qquad (110)$$

where

$$\Delta x_{COL} = aa \cdot x_{COL} + bb, \tag{111}$$

and *aa* and *bb* are small perturbation constants that are on the order of the square root of computer round-off [23, 32, 40, 66, 80, 114].

The advantages of using numerically evaluated Jacobians are simplicity and flexibility. Analytic Jacobian evaluations are feasible for simple systems and can be computed either by hand or by using symbolic manipulation packages. However, these techniques become cumbersome in the case of complicated physical systems or in situations where the physics of the problem or the discretization scheme may be frequently modified.

An algorithm is used in this study that maintains the flexibility of a standard numerical Jacobian, but requires a minimum number of function evaluations [23]. The energy equation is selected here to demonstrate the application of this numerical Jacobian algorithm. In this case, derivatives of f^{energy} with respect to its state variable dependencies are of interest. Consider the typical finite volume stencil shown in Figure 18 centered about cell (i,j). Shown in this figure are the state variable dependencies for f^{energy}. This equation can be expressed in terms its nine state variable dependencies as,

$$f_{i,j}^{energy} = f_{i,j}^{energy}\left(T_{i,j-1}, T_{i-1,j}, u_{i,j}, v_{i,j}, T_{i,j}, u_{i+1,j}, T_{i+1,j}, v_{i,j+1}, T_{i,j+1}\right). \tag{112}$$

In this example, $IEQ = 4$, $NEQ = 4$, and the cell number is determined from the function, $INUM(i,j)$. Consequently, derivatives of Equation (112) with respect to the indicated variable dependencies compute Jacobian elements on the row number given by,

$$ROW = 4 + (INUM(i,j) - 1) \cdot 4 \tag{113}$$

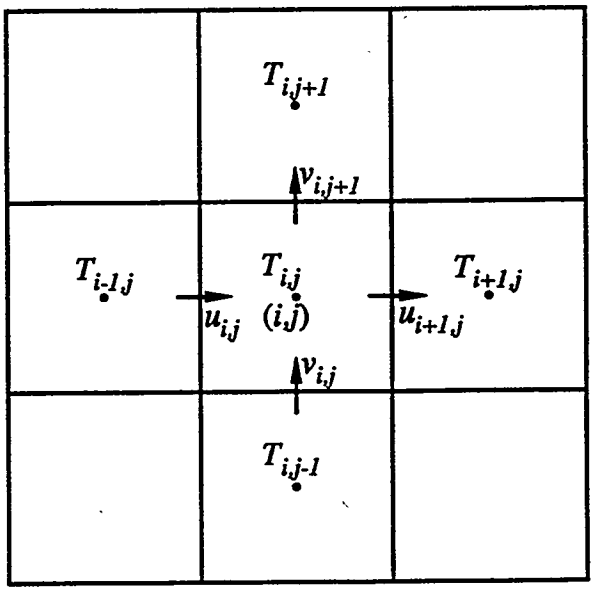


Figure 18. Finite volume stencil for energy equation centered about grid cell (i,j).

from Equation (105), while the column number is dependent upon the state variable. For example, consider the second to last state variable dependency, $v_{i,j+1}$. In this case, $IEQD = 2$ and $INUMD = INUM(i, j+1)$. The column number is determined from Equation (106) as,

$$COL = ROW - 2 + [INUM(i, j+1) - INUM(i, j)] \cdot 4. \tag{114}$$

Thus, the derivative of $f_{i,j}^{energy}$ with respect to $v_{i,j+1}$ computes $J_{ROW,COL}$ as,

$$J_{ROW,COL} = \frac{f_{i,j}^{energy}\left(T_{i,j-1}, T_{i-1,j}, u_{i,j}, v_{i,j}, T_{i,j}, u_{i+1,j}, T_{i+1,j}, v_{i,j+1} + \Delta v_{i,j+1}, T_{i,j+1}\right) - f_{i,j}^{energy}}{\Delta v_{i,j+1}}, \tag{115}$$

where $\Delta v_{i,j+1} = aa \cdot v_{i,j+1} + bb$ and the row and column numbers are given by Equation (113) and Equation (114), respectively. Derivatives with respect to the remaining state variable dependencies are computed in an analogous fashion. Computation of the remaining three rows of the Jacobian for cell (i,j) corresponding to $f_{i,j}^{u-momentum}$, $f_{i,j}^{v-momentum}$, and $f_{i,j}^{continuity}$ are identical to the process described for $f_{i,j}^{energy}$. However, the finite volume stencil or state variable dependencies are, of course, different for each equation.

3.1.2. Adaptive Damping Strategy

The damping strategy used to scale the update in Equation (100) is designed to prevent the calculation of negative thermodynamic variables, and to scale large variable updates when the solution is far from the true solution [40, 44]. The damping strategy restricts changes in the thermodynamic variables of interest to be less than a specified percentage, α, where $0 < \alpha < 1$. Note that more sophisticated scaling options are available [see 25, 66, 70]. However, this simple scaling choice seems to work well in practice and it requires little additional computational cost. Damping is especially important when a good initial guess for the true solution is not available. In these situations, large updates may arise that could lead to divergence. Damping is then effective in scaling the updates and preventing divergence. Typically, the thermodynamic variables are not allowed to change by more than say 20-25% on a given Newton step (i.e., $\alpha = 0.2 - 0.25$).

In the case of incompressible flow, this damping constant is based upon temperature only. This choice follows from the fact that pressure in incompressible flow is really only determined up to an additive constant. Pressure only appears in the governing equations in the form of a gradient, which means that derivatives of pressure are the important quantities

not the magnitude of pressure itself. Consequently, pressure was neglected from the definition of s. Therefore, the damping parameter is determined from,

$$s = Min\left\{1, \underset{all\ i,j}{Min}\left(\frac{\alpha T_{i,j}}{|\Delta T_{i,j}|}\right)\right\}, \tag{116}$$

when damping is activated, otherwise $s = 1$.

In the case of compressible flow, the scale factor is determined from the following expression that uses both temperature and density,

$$s = Min\left\{1, \underset{all\ i,j}{Min}\left(\frac{\alpha \rho_{i,j}}{|\Delta \rho_{i,j}|}, \frac{\alpha T_{i,j}}{|\Delta T_{i,j}|}\right)\right\}, \tag{117}$$

when damping is employed, otherwise $s = 1$.

The scaling strategy reflected by Equation (116) and Equation (117) is adaptive in the sense that as the true solution is approached and the variable updates become small, s will be set equal to one.

3.1.3. Mesh Sequencing

Mesh sequencing is one technique for improving the overall convergence of Newton's method [see 23, 33]. The technique consists of initially solving the problem on a coarse mesh and then interpolating through a series of pre-defined meshes, solving the problem on each mesh, until the desired mesh refinement is obtained. The radius of convergence of the Newton algorithm typically decreases as the number of unknowns increases [3, 66] making a good initial guess very important on the finer grids. The goal of mesh sequencing is then to

produce an initial guess on the final grid that lies within the radius of convergence. The advantage in this approach is that the coarse grid iterations are typically much less expensive than the fine grid calculations. Consequently, the savings produced by a better initial guess on the fine grid often far outweigh the cost of the coarse grid computations. Difficulties arise, however, if the initial grid is too coarse to resolve the important features of the flow. In those situations, there may be little benefit in using the interpolated coarse grid solution as an initial guess.

In this implementation, the Lagrange form of the interpolating polynomials [see 115] are used to interpolate a coarse grid solution up to a finer grid. In one dimension these interpolating polynomials are defined by,

$$\Phi_{m,i}(x)=\prod_{\substack{k=1,\\ k\neq i}}^{m}\frac{(x-x_k)}{(x_i-x_k)}, \tag{118}$$

where $(m-1)$ is the order of the interpolant, $\Phi_{m,i}(x)$, passing through the m interpolation points given by x_i, $i=1,...,m$. Note that within this context, x refers to a spatial dimension and is not in any way associated with the state variable. One advantage of the Lagrange form of the interpolating polynomials is that the interpolation points need not be evenly spaced, enabling application to non-uniform grids. Additionally, the polynomials exhibit the following property,

$$\Phi_{m,i}(x_k)=\begin{cases}0, & k\neq i\\ 1, & k=i\end{cases}, \tag{119}$$

which forces the interpolant to assume the nodal values of the function when evaluated at the node points. Thus, the one-dimensional interpolation of a general function, u, takes the form,

$$u(x)=\sum_{i=1}^{m} u_i \Phi_{m,i}(x), \tag{120}$$

where u_i represents the value of the function at x_i.

Two dimensional interpolation is handled by interpolating in the two dimensions independently. Thus, one can define a second interpolant for the y-direction denoted as $\Psi_{m,j}(y)$, which passes through the m interpolation points y_j, $j=1$ *to* m, i.e.,

$$\Psi_{m,j}(y)=\prod_{\substack{k=1,\\ k\neq j}}^{m} \frac{(y-y_k)}{(y_j-y_k)}, \tag{121}$$

with

$$\Psi_{m,j}(y_k)=\begin{cases}0, & k\neq j\\ 1, & k=j\end{cases}. \tag{122}$$

Interpolation of the two-dimensional function, u, to the point $(\bar{x},\bar{y})$ then uses the (mxm) interpolation points (x_i,y_j), where $i=1,...,m$ and $j=1,...,m$. Here, this interpolation is accomplished using two steps. First, u is interpolated to the desired y-location, $\bar{y}$, at each of the x_i locations, i.e.,

$$u(x_i,\bar{y})=\sum_{j=1}^{m} u_{i,j}\Psi_{m,j}(\bar{y}),\ i=1,...,m. \tag{123}$$

Next, u is interpolated to the point $(\bar{x},\bar{y})$ using the m interpolation points computed in step one, i.e., $(x_i,\bar{y})$ where $i=1,...,m$. The final step can be written as

$$u(\bar{x},\bar{y}) = \sum_{i=1}^{m} u(x_i,\bar{y}) \cdot \Phi_{m,i}(\bar{x}). \tag{124}$$

These two steps are used to interpolate each of the dependent variables (i.e., u, v, p, T or u, v, ρ, T) from a coarse grid solution onto a finer grid.

Typically, a series of two or more grids are employed, where the finer mesh is generated from the previous coarse grid by dividing each computational cell into four new cells. However, since the Lagrangian interpolation process described above is sufficiently general, the finer grid need not be dependent upon the coarse mesh.

3.1.4. Pseudo-Transient Relaxation

An alternative method for improving the initial guess supplied to Newton's method is the use of pseudo-transient relaxation [see 34, 76, and 116]. This technique can be implemented using both fixed time steps and variable time steps, controlled via an expression similar to that used by the switched evolution relaxation algorithm (SER) [116]. This technique involves adding artificial transient terms to the main diagonal of the Jacobian matrix, specifically terms corresponding to time derivatives of the principal variables. Consequently, when this technique is used, Equation (98) is replaced by,

$$\left(\frac{\mathbf{V}}{\Delta t^n} + \mathbf{J}^n\right)\delta\mathbf{x}^n = -\mathbf{F}(\mathbf{x}^n). \tag{125}$$

where $\mathbf{V}$ is a diagonal matrix whose entries are the volumes of the computational cell corresponding to that equation row. Δt^n must be chosen sufficiently small to ensure

convergence yet large enough to obtain efficient steady state calculations. Within the SER algorithm, Δt^n is chosen adaptively as follows,

$$\Delta t^n = \eta \frac{\|\mathbf{F}(\mathbf{x}^o)\|_\infty \Delta t^0}{\|\mathbf{F}(\mathbf{x}^{n-1})\|_\infty}, \tag{126}$$

where the vector $\mathbf{F}(\mathbf{x})$ represents the residuals of the steady state equations. The initial time step, Δt^0 is a user specified starting time step, usually selected near the explicit convective and/or diffusive stability limits. The time step is further controlled by the scaling parameter, η, which typically is taken to be *10*. Equation (126) forces Δt^n to be small when the transient is far from steady state, but allows it to increase rapidly as steady state is approached. This pseudo-transient technique can be combined with mesh sequencing or it can be used independently, thereby possibly avoiding computations on the coarse grids.

3.1.5. Parameter Continuation

A technique that is useful for obtaining difficult solutions is parameter continuation. Oftentimes solutions are very sensitive to certain flow parameters such as Reynolds number, Grashof number, Mach number, etc.... Thus, it may be more efficient to gradually vary these parameters in moving from one converged solution to another converged solution. Thus, several intermediate solutions may be obtained before the reaching the final desired solution. This technique can also be used with the aforementioned convergence enhancement techniques.

3.1.6. Defect Correction

The accuracy of a numerical solution is always important. Consequently, much attention has been devoted to the construction of higher-order accurate finite volume discretization schemes [for example see 14 and 107]. Use of these schemes is warranted in situations where high accuracy is desired or required and the physical model is well understood (i.e., governing equations, transport coefficients, etc...). In some instances, a lower order discretization scheme introduces too much artificial diffusion, thereby smearing and corrupting the true solution. On the other hand, the use of higher-order schemes is relatively expensive, both in terms of memory requirements and CPU time, due to larger finite volume stencils and adverse effects on convergence. Convergence is typically affected because the higher order approximations lessen the diagonal dominance of the iteration matrix, which in this study is the Jacobian matrix, thereby making the matrix more ill-conditioned. This effect can make convergence more difficult by reducing the radius of convergence of the Newton algorithm, and it can also make solution of the linear systems given in Equation (98) much more difficult. The latter effect is especially important if an iterative technique is used to solve these linear systems. Thus, if the numerical analyst is primarily interested in qualitative effects, has only limited computer resources, or if significant uncertainties exist with respect to the physical model, a lower order approximation may suffice.

However, situations do arise where higher-order accurate solutions are required, but computer memory limitations or convergence difficulties require a different computational approach. One procedure for maintaining higher order accurate solutions in these situations is based upon a simple defect correction method [11, 76, 77, 78, 107, 117, 118]. The basic idea is to evaluate the coefficient (or iteration) matrix with a lower order discretization, while including the higher-order corrections in the right hand side (source terms). Of particular

interest in this study is the application of the defect correction technique within the fully coupled Newton algorithm. In this case, a converged solution can be obtained using a lower order differencing scheme in forming both the Jacobian matrix and the residuals in Equation (98). This solution can then be corrected by restarting the algorithm using a higher order scheme in the evaluation of the residuals in Equation (98) [14]. Note that this replacement is made only in the right hand side of Equation (98), and not in the Jacobian. Since the right hand side of Equation (98) determines the accuracy of the final numerical solution, driving these residuals to zero guarantees the solution will be of higher-order accuracy. The drawback with this technique, however, is that the characteristic quadratic convergence behavior of Newton's method depends upon the correctness of the Jacobian matrix with respect to the discrete governing equations. Therefore, the price to be paid for increased accuracy when using the defect correction technique is a loss of quadratic convergence. In spite of the degraded convergence behavior, defect correction is often very useful in maintaining higher order accurate solutions.

3.2. NEWTON-KRYLOV METHODS

The main drawback of direct-Newton methods is the large memory required to store the factored Jacobian matrix. This drawback has been countered with advances in Krylov subspace based iterative solution algorithms that enable the sparseness of the Jacobian to be efficiently exploited. Specifically, the development of efficient conjugate gradient-like algorithms for the solution of nonsymmetric, non-positive definite linear systems [119, 120] has enabled the implementation of "in memory", multidimensional, fully coupled Newton's method solutions for the Navier-Stokes and energy equations. Since the use of an iterative solver does not require the exact solution of the linear systems on each Newton step, the tolerance of the linear equation solve can be relaxed when far from the true solution, and

tightened as the true solution is approached. This feature is commonly referred to as an "inexact" Newton iteration [121, 122]. The coupling of Newton's method with a Krylov subspace based iterative technique gives rise to what is termed a Newton-Krylov algorithm, which in this study is assumed to be a specific algorithm from the more general class of "inexact" Newton Methods. Note that Krylov subspace based iterative techniques are described below and in more detail in the Appendix.

For completeness, it is noted that "out of memory" matrix solvers, such as the frontal method can also be used to handle large Jacobian matrices that exceed available memory [81, 82]. Only a limited number of matrix entries (contributing to the active 'frontal matrix') must be stored in memory yet partial and full pivoting is possible in the frontal matrix. Einset and Jensen found that despite the advantages of the frontal method, there is a break-even point in front width above which iterative solutions become more efficient [81]. Their preconditioned iterative method outperformed the frontal method in their tests when the frontal width exceeded approximately 500. These results as well as those of other researchers [49] have encouraged the work in this dissertation to focus on the performance of "in memory" Krylov subspace based iterative algorithms.

Besides memory considerations, an advantage in coupling an iterative linear equation solver with Newton's method is that the linear system can be solved less accurately during the initial Newton iterations when far from the true solution, and more accurately as the true solution is approached. This is in contrast to the use of a direct solver, which requires the same amount of work whether one is close to the true solution or not. In this study this behavior is implemented via an inner iteration convergence criteria similar to that proposed by Averick and Ortega [121] and Dembo [122]. Specifically, the inner iteration is assumed converged when,

$$R_i^n = \frac{\|\mathbf{J}^n \delta\mathbf{x}^n + \mathbf{F}(\mathbf{x}^n)\|_2}{\|\mathbf{F}(\mathbf{x}^n)\|_2} < \varepsilon^n, \tag{127}$$

where the subscript on the convergence parameter, R_i^n, denotes the inner iteration, and the superscript, in all cases, indicates the dependence on the Newton iteration. Note that Equation (127) represents the ratio of the L_2-norm of the residual of the linear system expressed by Equation (98) and the L_2-norm of the residual of the outer Newton iteration. Thus, when the outer Newton residual is large, which normally occurs during the initial stages of the calculation, the inner iteration convergence criteria is less restrictive. In contrast, as the Newton residual norm becomes small, the inner iteration convergence criteria becomes more restrictive, which enables superlinear convergence during the later stages of the calculation. The selection of the best value of the tolerance, ε^n, is a highly empirical process. In Chapter 4, two options for setting ε^n are investigated. The first is to set ε^n to a constant value and the second is to let ε^n vary on each Newton iteration in a prescribed manner [67].

3.2.1. Krylov Subspace Based Iterative Methods

Equation (98) requires solution of large linear systems of dimension N on each Newton step. Direct solution techniques often become impractical for large linear systems because of high memory and CPU cost. An alternative in these situations is the use of iterative techniques that can exploit the sparse structure of the Jacobian matrix. Krylov subspace based methods are powerful iterative techniques for solving these types of linear systems. These methods compute new approximations to the solution, denoted as $\delta\mathbf{x}_k^n$, from the affine (translated) subspace defined by

$$\delta \mathbf{x}_0^n + K_k(\mathbf{r}_0, \mathbf{J}), \tag{128}$$

where the Krylov subspace of dimension k is defined by

$$K_k(\mathbf{r}_0, \mathbf{J}) \equiv span(\mathbf{r}_0, \mathbf{J}\mathbf{r}_0, \mathbf{J}^2\mathbf{r}_0, \ldots, \mathbf{J}^{k-1}\mathbf{r}_0), \tag{129}$$

and $\mathbf{r}_0$ is the initial residual of the linear system determined from the initial solution guess, $\delta \mathbf{x}_0^n$, i.e., $\mathbf{r}_0 = \left(-\mathbf{F}(\mathbf{x}^n)\right) - \mathbf{J}\delta \mathbf{x}_0^n$. Recall that the superscript, n, refers to the Newton iteration number, whereas the subscript (k or 0) refers to the inner iteration number.

There are some excellent references discussing Krylov subspace based methods. Some of the more recent discussions are given in References 71, 80, 119, 123, 124, 125, and 126. Some earlier, but still extremely valuable, review articles are found in References 127, 128, 129, 130, and 131. A very interesting annotated, historical bibliography of the conjugate gradient and Lanczos methods is presented in Reference 132. Information from these sources and others are compiled in the overview of Krylov subspace based methods presented in the Appendix. Consequently, only a brief discussion of these methods is presented here. Listings of the various algorithms that are used in this work are contained in the Appendix. Additionally, the Appendix also explains the different perspectives from which these algorithms are derived.

The classical conjugate gradient (CCG) method of Hestenes and Stiefel [133] is probably the best known Krylov subspace based method. Interestingly enough, this algorithm was originally derived as a direct method. Its full potential as an iterative technique was not realized until Reid revived it in 1971 [134], and latter Concus in 1976 [135]. However, the idea of using the CCG algorithm as a direct method illustrates an important property of many Krylov subspace based algorithms not shared by other iterative

techniques, namely a finite termination property. Thus, with exact precision mathematics the CCG method is guaranteed to converge within N iterations, but satisfactory convergence is likely for much less than N iterations. The CCG algorithm also does not require iteration parameter estimation to improve performance, unlike successive over relaxation (SOR), many alternating direction implicit (ADI) schemes, and Chebychev iteration. Also, the CCG algorithm typically converges more rapidly than typical matrix-splitting iterative schemes such as Jacobi and Gauss-Seidel iteration. CCG is optimal in the sense that the residual norm is minimized on each iteration and that new search directions are computed with economical vector recurrences so that work and storage requirements per iteration are small. In fact, these latter two properties define a "true" conjugate gradient method. However, the difficulty associated with the CCG algorithm is that it is applicable only to symmetric, positive definite matrices. As a result, considerable research has been devoted to generalizations of the conjugate gradient method to more general systems.

The two main options for generalization of the CCG ideas to nonsymmetric linear systems are the following:

1. Application of the CCG algorithm to the normal equations.
2. Development of conjugate gradient-like algorithms.

The normal equations option can be applied in two different ways. First, is the application of the CCG algorithm to systems of the form, $\mathbf{J}^T\mathbf{J}\delta\mathbf{x} = -\mathbf{J}^T\mathbf{F}(\mathbf{x})$, which results in what is referred to as the CGNR algorithm [133]. The capital 'N' refers to the normal equations and the capital 'R' indicates that the residual norm is minimized over the Krylov subspace. Secondly, one can apply the CCG algorithm to systems of the form, $\mathbf{J}\mathbf{J}^T\mathbf{y} = -\mathbf{F}(\mathbf{x})$, where $\delta\mathbf{x} = \mathbf{J}^T\mathbf{y}$. This latter choice is referred to as the CGNE algorithm [136]. In this case the capital 'E' indicates that the norm of the error is minimized over the

Krylov subspace. The disadvantage of the normal equation approach is that the condition number of the new system is squared, which can lead to very slow convergence in some instances. Additionally, working with the matrix transpose is often undesirable because it makes sparse storage and parallel/vector implementations more difficult, and because it prohibits use of finite difference projection techniques to approximate matrix-vector products within inexact Newton iterations [67], as discussed in Section 3.2.4. These reasons have recently made the use of conjugate gradient-like algorithms a more attractive option.

Conjugate gradient-like algorithms are derived by relaxing either or both of the properties that define a "true" conjugate gradient method, namely optimality and economical vector recurrences. Some of the more popular and more recent transpose-free conjugate gradient-like algorithms that are used in this work include: the generalized minimal residual algorithm (GMRES) [4], the conjugate gradient squared algorithm (CGS) [5], the Bi-CGSTAB algorithm [6], and the transpose-free quasi-minimal residual algorithm (TFQMR) [7]. Note that the first algorithm is based upon the Arnoldi process [137], while the other algorithms are based upon the nonsymmetric Lanczos process [138].

The Arnoldi-based GMRES algorithm [4] was derived so as to maintain optimality, but at the expense of economical vector recurrences. Consequently, the work and storage requirements of GMRES increase with the iteration count. Therefore, practical implementations frequently require use of the restarted version, GMRES(m), where m is the maximum dimension of the Krylov subspace. The restarted algorithm is then only optimal within a cycle, and so frequent restarts can lead to slow convergence or even algorithm stall.

In contrast, the Lanczos-based algorithms were derived so as to maintain economical recurrences, but at the expense of optimality. Additionally, the nonsymmetric Lanczos process itself is susceptible to breakdowns, making algorithms derived based upon this process also susceptible to breakdown. CGS was the first transpose-free algorithm of this type developed [5]. It was derived by squaring the polynomial relations of the bi-conjugate

gradients (BCG) algorithm [138, 139]. CGS can exhibit very rapid convergence compared to BCG, but its convergence is marred by sometimes very wild oscillations, which under certain conditions can lead to inaccurate solutions [6]. This difficulty led to the development of both Bi-CGSTAB [6], which uses local steepest descent steps, and TFQMR [7], which uses the quasi-minimization idea, to obtain more smoothly convergent CGS-like solutions.

In this investigation each of the transpose-free conjugate gradient-like algorithms mentioned above are coupled with Newton's method to yield a particular Newton-Krylov algorithm. These different Newton-Krylov algorithms will be referred to as Newton-CGS, Newton-BCGSTAB, Newton-TFQMR, and Newton-GMRES(k), respectively. Additionally, if a direct solution technique such as Gaussian elimination is used to solve Equation (98), then this algorithm is referred to as direct-Newton.

3.2.2. Sparse Matrix Storage Scheme

The previous section indicated that an important advantage in the use of Krylov subspace based iterative methods is the ability to exploit the sparse structure of the Jacobian matrix. Chapter 2 indicated that the Jacobian matrix possesses a banded structure. This structure results from the natural ordering of variables ('uvpT' or 'uvρT'), and the finite volume discretization stencil employed. A simple scheme for exploiting this non-zero sparsity pattern is to only store matrix bands or diagonals that contain non-zero matrix components. Thus, matrix bands or diagonals with only zero components are not stored. This storage scheme can be implemented using an array to store the non-zero matrix diagonals [see 140]. Let this array be denoted as $A(N, ndiag)$, where the leading dimension is the total number of unknowns, defined as $NC \cdot NEQ$, and the second dimension is the total number of non-zero diagonals. Note that the total number of non-zero diagonals is a function of the number of equations and variables, NEQ, the ordering of the variables and equations

within a finite volume cell, the ordering of the finite volume cells, and the coupling between the variables and equations within the finite volume stencil. Also needed is an integer logic vector to store the offsets of these non-zero diagonals with respect to the main diagonal, denoted here by **idiag**(*ndiag*). The main diagonal is assigned the index *mdiag*, and **idiag**(*mdiag*) = 0. By choice, all indices less than *mdiag* correspond to sub-diagonals (i.e., **idiag**(m) < 0, $m < mdiag$), while all indices greater than *mdiag* correspond to super-diagonals (i.e., **idiag**(m) > 0, $m > mdiag$). In this manner, the entries in the logic vector are arranged in ascending order for $1 \le m \le ndiag$. This integer logic vector is computed in a preprocessing step. In converting the Jacobian matrix to the sparse storage scheme, row indices are preserved while column indices are related to the *ROW* and non-zero diagonal indices by,

$$COL = ROW + \mathbf{idiag}(m). \tag{130}$$

Thus, the elements of the sparse Jacobian array are related to the elements of the original Jacobian matrix by the algorithm expressed in Equation (131).

```
      Do 10 ROW = 1,N
      Do 20 m = 1,ndiag
        COL = ROW + idiag(m)
        A(ROW,m) = J_ROW,COL                    (131)
   20 Continue
   10 Continue
```

Tests must be included inside the loops of Equation (131) to ensure that $1 \le COL \le N$. If this condition is not satisfied then $\mathbf{A}(ROW,m) = 0$. Also note that the original Jacobian matrix is never stored. As the elements of the Jacobian are computed they are immediately stored in the sparse Jacobian array.

The Jacobian matrix is needed within the Newton-Krylov solution algorithm to generate a preconditioning matrix and to compute Jacobian-vector products. The effect of this sparse matrix storage scheme with respect to preconditioning is discussed in the next section, while Jacobian-vector products can be computed in a straightforward manner using an algorithm of the form shown in Equation (132) [see 140].

$$
\begin{array}{ll}
 & \textit{Do } 10 \;\; m = 1, ndiag \\
 & \quad ILIM1 = Max(1, 1 - \mathbf{idiag}(m)) \\
 & \quad ILIM2 = Min(N, N - \mathbf{idiag}(m)) \\
 & \quad \textit{Do } 20 \;\; IROW = ILIM1, ILIM2 \\
 & \qquad JCOL = IROW + \mathbf{idiag}(m) \\
 & \qquad \mathbf{y}(IROW) = \mathbf{y}(IROW) + \mathbf{A}(IROW, m) * \mathbf{b}(JCOL) \\
20 & \quad \textit{Continue} \\
10 & \textit{Continue}
\end{array}
\tag{132}
$$

One disadvantage in using a sparse storage scheme of this type is that some zero entries are still necessarily stored. These zero entries arise from two different sources. The first source corresponds to *ROW* and **idiag**(m) combinations for which the condition, $1 \le COL \le N$, is not satisfied. This situation arises because all diagonals in the original Jacobian matrix, except the main diagonal, have fewer than N entries, but the sparse storage array assumes that all have N entries. The second source follows from the fact that on each row of the Jacobian, the same number of entries are stored, with this number being equal to the total number of non-zero diagonals, *ndiag*. Since this number must account for all the possible coupling between all variables and all equations, it is a conservative value that represents the maximum possible number of non-zero entries that could occur on a given row of the Jacobian. Although some rows of the Jacobian possess fewer non-zero entries than *ndiag*, the non-zero diagonal storage scheme used here still requires storage of *ndiag* entries. More compact storage schemes such as compressed row storage (CRS), compressed column storage (CCS), and others [see 140] do exist that would eliminate this unnecessary storage of

zero elements. However, this additional, unnecessary storage is typically very small compared to the large storage reduction obtained using the diagonal storage scheme. Additionally, these other storage schemes are necessarily more general and somewhat more complicated than the simple diagonal storage scheme considered here. Consequently, consideration of other more efficient yet more complicated storage schemes was not deemed important for this investigation.

3.2.3. Preconditioning

An important issue with regard to the efficient implementation of conjugate gradient-like algorithms is preconditioning. Preconditioning is a classical way to improve the performance of these iterative techniques. The equivalent right preconditioned form of Equation (98) is expressed by,

$$\mathbf{J}^n\mathbf{P}_R^{-1}\mathbf{P}_R\delta\mathbf{x}^n = -\mathbf{F}(\mathbf{x}^n). \tag{133}$$

The goal of preconditioning, as its name implies, is to improve the condition number of the new equivalent system. In the $L_2 - norm$, the condition number is defined as the ratio of the largest eigenvalue to the smallest eigenvalue [126]. Thus, it is hoped that the matrix, $\mathbf{J}^n\mathbf{P}_R^{-1}$, has a lower condition number than $\mathbf{J}^n$. In order for this to occur, the preconditioning matrix must reasonably approximate $\mathbf{J}^n$. In fact, the ideal preconditioner is the matrix itself, since then $\mathbf{J}^n\mathbf{P}_R^{-1} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. The eigenvalues of $\mathbf{J}^n\mathbf{P}_R^{-1}$ then equal one and so the condition number of this new system matrix is also one. Implementation of right preconditioning within a Krylov iterative algorithm, typically consists of replacing $\mathbf{J}^n$ with $\mathbf{J}^n\mathbf{P}_R^{-1}$ and $\delta\mathbf{x}^n$ with $\mathbf{P}_R\delta\mathbf{x}^n$. Thus, it is also very important that systems of the form $\mathbf{P}_R\mathbf{w} = \mathbf{z}$, where $\mathbf{w}$ and $\mathbf{z}$ are general vectors, can be inverted easily, since these types of

systems arise frequently within the Krylov iteration. Note that the norm of the residual of the right preconditioned linear system is given by,

$$\|\mathbf{r}\| = \left\|\left(-\mathbf{F}(\mathbf{x}^n)\right) - \mathbf{J}^n\mathbf{P}_R^{-1}\mathbf{P}_R\delta\mathbf{x}^n\right\| = \left\|\left(-\mathbf{F}(\mathbf{x}^n)\right) - \mathbf{J}^n\delta\mathbf{x}^n\right\|, \tag{134}$$

which is identical to the residual norm obtained without preconditioning.

The equivalent left preconditioned version of Equation (98) can be written as,

$$\mathbf{P}_L^{-1}\mathbf{J}^n\delta\mathbf{x}^n = -\mathbf{P}_L^{-1}\mathbf{F}(\mathbf{x}^n). \tag{135}$$

Implementation of left preconditioning within a Krylov iterative algorithm, typically consists of replacing $\mathbf{J}^n$ with $\mathbf{P}_L^{-1}\mathbf{J}^n$ and the right hand side vector with $-\mathbf{P}_L^{-1}\mathbf{F}(\mathbf{x}^n)$. The norm of the residual of the left preconditioned linear system is given by,

$$\|\mathbf{r}\| = \left\|\mathbf{P}_L^{-1}\left(-\mathbf{F}(\mathbf{x}^n) - \mathbf{J}^n\delta\mathbf{x}^n\right)\right\|, \tag{136}$$

which is now dependent upon the inverse of the left preconditioning matrix. Since most Krylov iterative algorithms generate a residual vector, and base convergence upon the norm of this vector, it is important to realize that this convergence measure is influenced by the left preconditioning matrix. This issue does not arise when right preconditioning is employed. Note that both right and left preconditioning can be used simultaneously via,

$$\mathbf{P}_L^{-1}\mathbf{J}^n\mathbf{P}_R^{-1}\mathbf{P}_R\delta\mathbf{x}^n = -\mathbf{P}_L^{-1}\mathbf{F}(\mathbf{x}^n), \tag{137}$$

although this option is not employed in this study.

Some classical preconditioning choices are based upon simple matrix-splitting ideas. These ideas stem from splitting the Jacobian matrix into separate lower diagonal, main diagonal, and upper diagonal parts, which when summed together equal the original matrix, i.e.,

$$\mathbf{J} = \mathbf{L} + \mathbf{D} + \mathbf{U}. \tag{138}$$

Recall that a Jacobi iteration [63] for solving $\mathbf{J}\delta\mathbf{x} = \mathbf{b}$ $[\mathbf{b} = -\mathbf{F}(\mathbf{x})]$, where the superscripts have been dropped for clarity, can be expressed as,

$$\mathbf{D}\delta\mathbf{x}_{k+1} = \mathbf{b} - (\mathbf{L} + \mathbf{U})\delta\mathbf{x}_k. \tag{139}$$

Jacobi type preconditioning is then based upon this simple Jacobi iteration scheme and therefore assumes that the preconditioner, $\mathbf{P}$ (which could be applied from the left or the right), is given by the Jacobi iteration matrix, namely $\mathbf{P} = \mathbf{D}$. Note that since $\mathbf{P}$ is simply a diagonal, matrix systems of the form $\mathbf{P}\mathbf{w} = \mathbf{z}$ are easily inverted. However, the effectiveness of Jacobi preconditioning, like Jacobi iteration, is strongly tied to the diagonal dominance of $\mathbf{J}$. For completeness, additional preconditioner choices derived from other matrix splitting based iterative techniques [see 63, 141] include:

1. Gauss-Seidel (GS) based preconditioning, $\mathbf{P} = (\mathbf{L} + \mathbf{D})$.
2. Symmetric Gauss-Seidel (SGS) based preconditioning, $\mathbf{P} = (\mathbf{L} + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U})$.
3. Successive-Over-Relaxation (SOR) based preconditioning, $\mathbf{P} = \left[\mathbf{L} + (1/\omega)\mathbf{D}\right]$, where ω is the SOR-based weighting parameter.
4. Symmetric Successive-Over-Relaxation (SSOR) based preconditioning, $\mathbf{P} = \dfrac{\omega}{2-\omega}\left(\mathbf{L} + \dfrac{1}{\omega}\mathbf{D}\right)\mathbf{D}^{-1}\left(\dfrac{1}{\omega}\mathbf{D} + \mathbf{U}\right)$.

Note that in each of the above examples, systems of the form $\mathbf{Pw} = \mathbf{z}$ can be solved simply using forward elimination, backward substitution, and simple inversion of a diagonal matrix. Although, these preconditioner options are simple in form and inexpensive with respect to memory and CPU cost, their effectiveness (like Jacobi preconditioning) is tied to the diagonal dominance of **J**. Consequently, these types of preconditioners are not generally useful for the problems considered here, unless used in conjunction with a pseudo-transient calculation, which may guarantee the diagonal dominance of the matrix, $\left(\frac{\mathbf{V}}{\Delta t^n} + \mathbf{J}^n\right)$.

Another technique to improve the effectiveness of these types of preconditioners is to use block implementations [142] instead of the point implementations described above. In a block implementation, the matrix is split into a block lower diagonal matrix, a block diagonal matrix, and a block upper diagonal matrix. Thus, the elements of **L**, **D**, and **U** are no longer single 'point' numbers, but rather sub-matrices. Block implementations of Jacobi and Gauss-Seidel type preconditioning based upon domain decomposition are described in Section 3.2.3.3 below.

There are also other types of preconditioners besides those based upon matrix-spitting ideas. These include approximate factorization type preconditioning [143] and polynomial based preconditioning [see 144, 145, 146, 147]. Note that polynomial based preconditioners were not investigated in this study. However, those of the approximate factorization type were considered, specifically the Incomplete Lower-Upper (ILU) factorization family of preconditioners, which are described in Section 3.2.3.1. Since many of these preconditioners exhibit sensitivity to cell ordering, several simple ordering schemes are described in Section 3.2.3.2.

3.2.3.1. Incomplete Lower-Upper Factorization (ILU) Preconditioning

A very popular class of preconditioners are those of the incomplete lower-upper (ILU) factorization type [143]. The basic idea is to express the Jacobian matrix as an approximate factorization of upper and lower matrices, i.e.,

$$\mathbf{J} = \mathbf{LU} + \mathbf{E}, \tag{140}$$

where $\mathbf{E}$ is an error or remainder matrix associated with the approximate factorization. The preconditioning matrix, $\mathbf{P}$ (which can be applied from the left or right), is then set equal to the approximate factorization of $\mathbf{J}$, namely $\mathbf{LU}$. As long as the errors associated with this approximate factorization are small, which means that the terms in $\mathbf{E}$ are small, then $\mathbf{P}$ satisfies the requirements of an effective preconditioner outlined previously. Systems of the form $\mathbf{Pw} = \mathbf{z}$ are then solved using a simple two-step process, consisting first of a forward solve and then a backward solve. This preconditioner class followed from the idea of truncated direct elimination [148]. Note that the popular Strongly Implicit (SIP) iterative scheme of Stone [62] can be viewed as a form of truncated direct elimination method. In fact, within this context, $\mathbf{P}$ can be used to compute an initial guess for the Krylov iterative solve [i.e., by setting $\delta\mathbf{x}^0 = -\mathbf{P}^{-1}\mathbf{F}(\mathbf{x}^0)$]. However, Meijerink and van der Vorst [143] were the first to apply this technique within the context of matrix preconditioning.

Specifically, in this work ILU preconditioners with various amounts of fill-in are considered based on a modified version of the level of fill-in idea of Watts [148]. This idea assumes that all original terms in the sparsity pattern are set to *level-0*. In this work, the original sparsity pattern consists of all original non-zero diagonals of $\mathbf{J}$ that are stored in $\mathbf{A}$. Then, elimination of a *level-0* term gives rise to a *level-1* fill-in element. In general, the level of a fill-in term is determined by the sum of the level of the term being eliminated and the

level of the term used to perform the elimination. This procedure can be continued up to say *level-m*, resulting in a preconditioner denoted by ILU(m). Thus, the ILU(0) preconditioner assumes the same non-zero sparsity pattern as the Jacobian matrix, while preconditioners of a higher level contain additional elements outside the original sparsity pattern of the Jacobian.

Another way to view the determination of the different levels of fill-in is described by Langtangen [46]. In this description, the original sparsity pattern is determined by the set of non-zero matrix entries denoted by $S_J^0 = \{(ROW,COL):\ J_{ROW,COL} \neq 0\}$, where the subscript indicates that the sparsity pattern is with respect to the Jacobian matrix and the superscript indicates *level-0* fill-in. Using this notation the sparsity pattern for *level-m* fill-in can be determined from *level-(m-1)* as follows. First, assume that the sparsity pattern for level-(m-1) is given by S_J^{m-1}. This requires that the approximate ILU(m-1) factors have sparsity patterns given by S_J^{m-1}. Next form the product of these **L** and **U** factors. In general, this product will contain non-zero fill-in terms not included in S_J^{m-1}. Adding these non-zero (ROW,COL) values to S_J^{m-1} then determines S_J^m, the sparsity pattern for the *level-m* preconditioner, ILU(m) [see 46]. A general algorithm for computing ILU factorizations is given in Equation (141) [see 149].

$$
\begin{aligned}
&Do\ \ ROW = 1,N \\
&\quad Do\ \ COL = 1,N \\
&\qquad If\ \ (ROW,COL) \in S_J^m\ \ then \\
&\qquad\quad \mathbf{SUM}_{ROW,COL} = \mathbf{J}_{ROW,COL} - \sum_{s=1}^{Min(ROW,COL)-1} \mathbf{L}_{ROW,s}\mathbf{U}_{s,COL} \\
&\qquad\quad If\ \ (ROW \geq COL)\ \ \mathbf{L}_{ROW,COL} = \mathbf{SUM}_{ROW,COL} \\
&\qquad\quad If\ \ (ROW < COL)\ \ \mathbf{U}_{ROW,COL} = \mathbf{SUM}_{ROW,COL}/\mathbf{L}_{ROW,ROW} \\
&\qquad Endif \\
&\quad End\ \ Do \\
&End\ \ Do
\end{aligned}
\tag{141}
$$

In this work, the same non-zero diagonal storage scheme described in the previous section is also used to store the preconditioning matrix, denoted by $\mathbf{AM}(N,ndiagm)$, where

again the leading dimension is the total number of unknowns while the second dimension is the total number of non-zero diagonals in the preconditioning matrix ($ndiagm \geq ndiag$). Preconditioner fill-in is then derived assuming this non-zero diagonal sparsity pattern. Thus, if a preconditioner fill-in element does not lie on an original stored non-zero diagonal, then a new non-zero diagonal must be added to the sparsity pattern in order to accommodate that fill-in term. As a result, every element in the new non-zero diagonal is computed as a preconditioner fill-in term. The more compact storage schemes mentioned in the previous section could be used to avoid this additional fill-in, but in many cases the inclusion of these additional fill-in terms may actually result in a more effective preconditioner [see 89]. The new non-zero diagonal offsets are simply appended to the **idiag** logic vector, while the added non-zero diagonals are appended to the **AM** array. In this manner, the diagonals corresponding to indices between 1 and *ndiag* represent the original non-zero diagonals, while indices greater than *ndiag* represent additional diagonals resulting from fill-in terms. In terms of the process described by Langtangen [46], the *level-0* sparsity pattern for the non-zero diagonal storage scheme is defined by $S^0_{\mathbf{AM}} = \{(1{:}N,m)\text{: } Any \;\; AM(1{:}N,m) \neq 0\}$, which indicates that an entire matrix diagonal is added to the sparsity pattern if a single entry in that diagonal is non-zero. Determining the sparsity pattern for *level-m* fill-in from *level-(m-1)* is then analogous to the process described above with $S^m_{\mathbf{J}}$ replaced by $S^m_{\mathbf{AM}}$. Note that no actual multiplication of **L** and **U** is required, only simulated multiplication that generates the additional non-zero diagonals due to fill-in terms. Thus, the new offsets that must be appended to **idiag** are computed in a preprocessing step. This later feature is advantageous in the sense that the sparsity pattern is known *a priori* before the calculation begins. This feature is in contrast to threshold based ILU factorizations [see 140] where fill-in terms are accepted or rejected according to whether their magnitude is greater than or less than a specified threshold tolerance, respectively. Although possibly resulting in a more effective preconditioner, the threshold based ILU preconditioner sparsity pattern can differ from one

calculation to the next or even from one Newton iteration to the next. Consequently, this work is concerned only with level of fill-in type ILU preconditioners.

One of the difficulties in solving the primitive variable forms of the incompressible Navier-Stokes equations is that pressure does not explicitly appear in the continuity equation. Thus, if the continuity equation is aligned with pressure then a zero will appear on the main diagonal in all of the rows in the Jacobian matrix representing the continuity equation. A similar problem arises whenever a very small (nearly zero) term arises on the main diagonal in the compressible flow application. The primary concern in the latter case is that errors caused by these very small pivot values will induce errors throughout the rest of the factorization, thereby corrupting the effectiveness of the incomplete factorization as a preconditioner. This feature renders the efficiency of incomplete factorization schemes of this type dependent upon the ordering of the unknowns. Thus, the goal is to order the unknowns such that small pivot values disappear or appear only near the end of the matrix, thereby restricting the amount of induced errors that can occur.

This difficulty is often completely avoided when using LINPACK type full factorization routines [17] that use pivoting, but pivoting is typically not practical when computing incomplete lower upper (ILU) factorizations. Alternatives to pivoting in this case include adding non-zeros to the diagonal using some sort of penalty function [37, 49, 50, 51], and realigning the equations and variables to avoid zeros on the main diagonal [30]. In the case of an iterative solver using ILU preconditioning, fill-in resulting from the incomplete factorization will generate non-zero terms in most of these zero diagonal rows. However, Chin et al. [76] pointed out that for a natural ordering (i.e. 'uvpT') in the case of incompressible flow, there will be no fill-in on the continuity equation row if the finite volume lies adjacent to a corner boundary such that the bottom and left face coincide with the boundary. They further note that if there is only one such cell (as is the case for the incompressible natural convection problem) the difficulty can be removed by arbitrarily

fixing the pressure in that cell. However, problems arise when there more than one such cell exists in the computational grid, or in situations where fixing pressure in a given cell is not allowed such as for compressible flow applications or problems with inflow/outflow boundaries where pressure is already specified.

Chin et al. chose to investigate clever alternative ordering strategies to solve this problem [30]. An alternative technique, which avoids complex variable re-ordering schemes of this sort, is the use of Kershaw's method for treating unstable pivots in incomplete LU factorizations [150]. This method allows near-zero pivots to be adjusted in such a way that the incomplete factorization algorithm is kept stable and the error associated with the pivot adjustment is minimized [150]. One advantage in this approach is that very small pivots, which may cause algorithm instability, are adjusted along with the hard zero pivots. Note that the algorithm listed in Equation (141) is modified when this technique is implemented, as shown in Equation (142) on the following page. Implementation of the algorithm listed in Equation (142) requires use of the non-zero diagonal storage scheme described above. In Equation (142), this requirement is reflected in the tests on whether a given row and column pair are members of the ILU(m) sparsity pattern. The benefits of using this technique as well as the use of simple alternative ordering schemes, such as those discussed below, are investigated in Chapter 4.

3.2.3.2. Cell Ordering Strategies

This section describes some simple finite volume cell ordering schemes. The effectiveness of incomplete factorization schemes are often sensitive to the ordering of the unknowns. Consequently, it is desirable to posses the ability to vary the unknown ordering scheme so as to improve preconditioner effectiveness. The alternate cell ordering schemes described in this section offer some limited flexibility in this regard. However, the cell

•*Sweep down main diagonal*
Do $ROW = 1, N$
$\sigma_{ROW} = \mu_{ROW} = 0$
$COL = ROW$
•*Compute the* COL^{th} *column of* $\mathbf{L}$
Do $i = ROW, N$
If $(i, COL) \in S_J^m$ *then*

$$\mathbf{L}_{i,COL} = \mathbf{J}_{i,COL} - \sum_{s=1}^{Min(i,COL)-1} \mathbf{L}_{i,s}\mathbf{U}_{s,COL}$$

$$\sigma_{ROW} = Max\left\{\sigma_{ROW}, \left|\mathbf{L}_{i,COL}\right|\right\}$$

Endif
End Do
•*Compute the* ROW^{th} *row of* $\mathbf{U}$
Do $j = COL+1, N$
If $(ROW, j) \in S_J^m$ *then*

$$\mathbf{U}_{ROW,j} = \mathbf{J}_{ROW,j} - \sum_{s=1}^{Min(ROW,j)-1} \mathbf{L}_{ROW,s}\mathbf{U}_{s,j}$$

$$\mu_{ROW} = Max\left\{\mu_{ROW}, \left|U_{ROW,j}\right|\right\}$$

Endif
End Do
•*Test for unstable pivots* ($t \equiv$# *binary digits used to store mantissa on computer*)
If $(\mathbf{L}^2_{ROW,ROW} < 2^{-t}\sigma_{ROW}\mu_{ROW})$ *then*

$$\mathbf{L}_{ROW,ROW} = Sign(\mathbf{L}_{ROW,ROW}) \cdot \sqrt{2^{-t}\sigma_{ROW}\mu_{ROW}}$$

Stop if $\mathbf{L}_{ROW,ROW} = 0$
Endif
•*Finish computing row of* $\mathbf{U}$ *by dividing by* $\mathbf{L}_{ROW,ROW}$
Do $j = COL+1, N$
If $(ROW, j) \in S_J^m$ *then*

$$\mathbf{U}_{ROW,j} = \mathbf{U}_{ROW,j} \Big/ \mathbf{L}_{ROW,ROW}$$

Endif
End Do
End Do

(142)

ordering schemes described here are selected so as to maintain the banded or diagonal structure of the Jacobian matrix, so as to maintain the ability to use the sparse matrix storage scheme described in Section 3.2.2. Note that an excellent description of a variety of other cell numbering schemes is given by Duff and Meurant [151]. Their work also investigates

the effects of the various ordering schemes on the performance of preconditioned conjugate gradient algorithms.

Recall from Section 3.1 above that the cell number was denoted by *INUM*. This parameter can be defined in terms of a Cartesian grid with (i,j) indices corresponding to the physical (x,y) coordinates, where $1 \le i \le nx$ and $1 \le j \le ny$. In this case *INUM* becomes a two-dimensional array denoted by $INUM(i,j)$. Examples of the four cell ordering schemes considered here are described below in terms of a simple rectangular Cartesian grid and $INUM(i,j)$:

1.) *Row Ordering*. A schematic representation of row ordering is shown in Figure 19, while the algorithm for setting $INUM(i,j)$ is given by Equation (143).

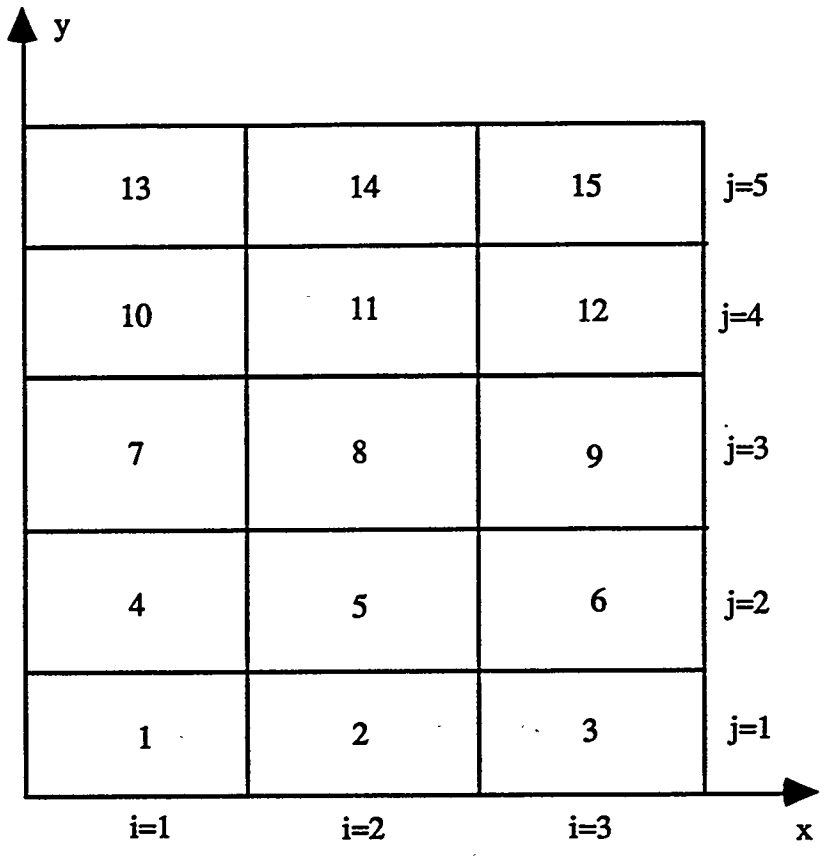


Figure 19. Schematic description of the row ordering scheme on a square Cartesian grid.

$$
\begin{aligned}
&icount = 1 \\
&Do\ \ j = 1, ny \\
&Do\ \ i = 1, nx \\
&\quad INUM(i,j) = icount \\
&\quad icount = icount + 1 \\
&End\ \ Do \\
&End\ \ Do
\end{aligned}
\tag{143}
$$

2.) *Reverse Row Ordering.* A schematic representation of reverse row ordering is shown in Figure 20, while the algorithm for setting $INUM(i,j)$ is given by Equation (144).

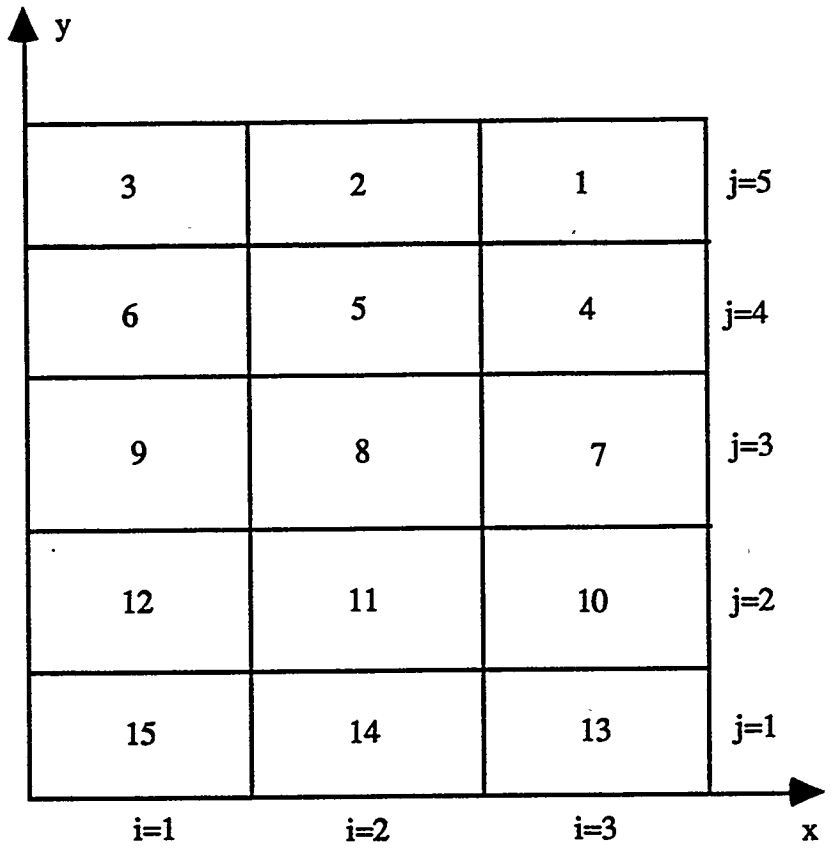


Figure 20. Schematic description of the reverse row ordering scheme on a square Cartesian grid.

$$
\begin{aligned}
&icount = 1 \\
&Do\ \ j = ny,1,-1 \\
&Do\ \ i = nx,1,-1 \\
&\quad INUM(i,j) = icount \\
&\quad icount = icount + 1 \\
&End\ \ Do \\
&End\ \ Do
\end{aligned}
\tag{144}
$$

3.) *Column Ordering*. A schematic representation of column ordering is shown in Figure 21, while the algorithm for setting $INUM(i,j)$ is given by Equation (145).

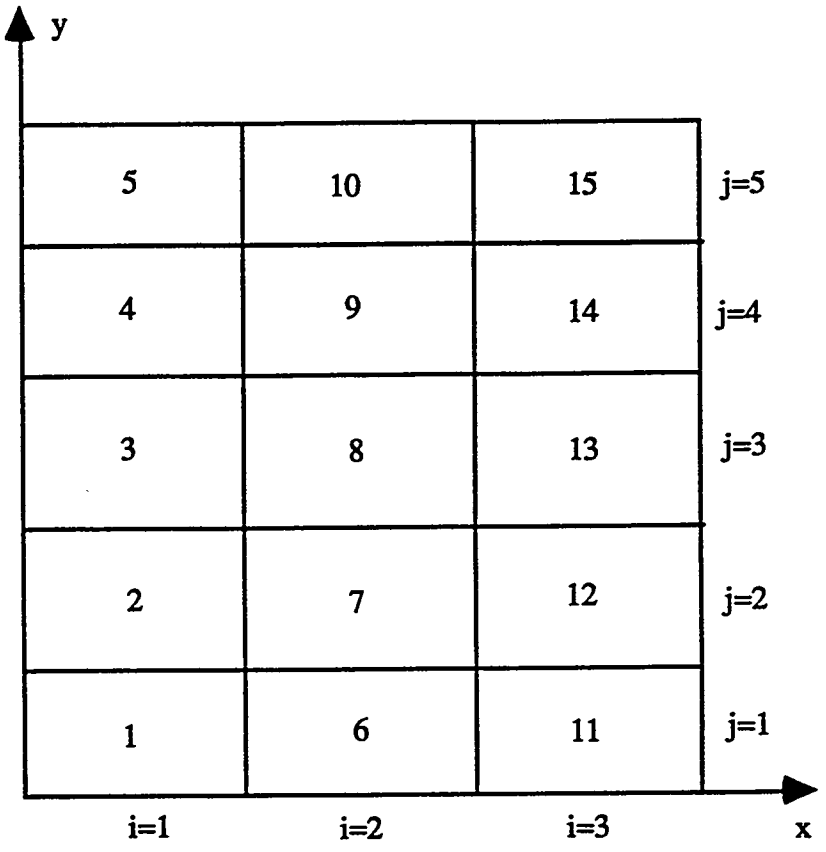


Figure 21. Schematic description of the column ordering scheme on a square Cartesian grid.

$$
\begin{aligned}
&icount = 1 \\
&Do\ \ i = 1, nx \\
&Do\ \ j = 1, ny \\
&\quad INUM(i,j) = icount \\
&\quad icount = icount + 1 \\
&End\ \ Do \\
&End\ \ Do
\end{aligned}
\tag{145}
$$

4.) *Reverse Column Ordering*. A schematic representation of reverse column ordering is shown in Figure 22, while the algorithm for setting $INUM(i,j)$ is given by Equation (146).

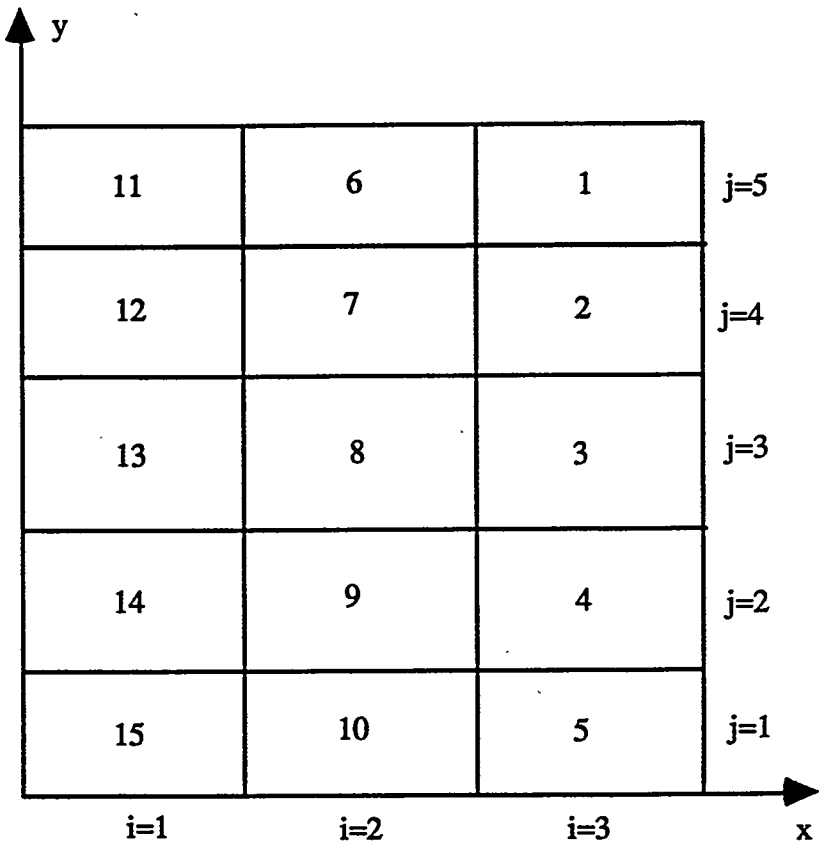


Figure 22. Schematic description of the reverse column ordering scheme on a square Cartesian grid.

$$
\begin{aligned}
&icount = 1 \\
&Do\ \ i = nx,1,-1 \\
&Do\ \ j = ny,1,-1 \\
&\quad INUM(i,j) = icount \\
&\quad icount = icount + 1 \\
&End\ \ Do \\
&End\ \ Do
\end{aligned}
\tag{146}
$$

3.2.3.3. Domain-Based Preconditioning

Recall from Section 3.2.3 that one technique for improving the effectiveness of matrix splitting type preconditioners is to use block implementations [142] instead of point implementations. In a block implementation, the matrix is split into a block lower diagonal matrix, a block diagonal matrix, and a block upper diagonal matrix. Thus, the elements of **L**, **D**, and **U** are no longer single 'point' numbers, but rather sub-matrices. Block implementations of Jacobi and Gauss-Seidel type preconditioning can be derived via domain decomposition, which gives rise to another class of preconditioners. These include preconditioners based upon both the one-level, algebraic additive and multiplicative Schwarz algorithms [see 91, 152, 153, 154, 155, 156]. Blocked preconditioners have the advantage of being more amenable to parallel implementation than the ILU type preconditioners discussed previously [see 99, 100].

Block preconditioners derived via domain decomposition rely on a physical decomposition of the computational domain to determine the matrix blocks. This idea is in contrast to the approach whereby the matrix blocks are determined via matrix partitioning after the matrix has already been formed. The use of domain decomposition to determine matrix blocks enables greater flexibility and control in lumping computational cells that are tightly coupled to one another into a single block. Domain decomposition techniques also

allow for overlapping domains so that at least some degree of coupling between domains/blocks can always be maintained.

The one-level algebraic additive and multiplicative Schwarz preconditioners used in this work are described below. Note that this description closely follows the discussion and notation of Cai and Saad [153]. The first step in the development of a domain based preconditioner is to partition the global computational domain into p sub-domains, whose union is the global domain. The goal then is to solve the global linear problem defined by Equation (98) using a preconditioner operator that incorporates solutions of the sub-domain problems.

This global preconditioner operator is developed as follows. Let $\mathbf{J}_i^{sub}$ denote the sub-block matrix portion of $\mathbf{J}$ that corresponds to the i^{th} sub-domain. This sub-domain matrix is often referred to as the restriction of $\mathbf{J}$ to the i^{th} sub-domain. If it is assumed that the sub-domain contains s state variable unknowns, then $\mathbf{J}_i^{sub}$ has dimensions of sxs. Also, for each sub-domain there exists a sub-domain identity matrix, $\mathbf{I}_i$, of dimension N; whose diagonal elements are equal to one if that state variable unknown lies within the sub-domain, but are equal to zero otherwise. This sub-domain identity matrix serves to project a global vector onto the i^{th} sub-domain, and can also be used to compute the extension of the restriction of $\mathbf{J}$ ($\mathbf{J}_i^{sub}$) to the N-dimensional space defined by the global domain. This extension is denoted by $\mathbf{J}_i$, and is computed from,

$$\mathbf{J}_i = \mathbf{I}_i \mathbf{J} \mathbf{I}_i = \mathbf{R}_i^T \mathbf{J}_i^{sub} \mathbf{R}_i, \tag{147}$$

where $\mathbf{R}_i$ is termed a restriction matrix, which has dimensions of sxN. $\mathbf{J}_i$ is often referred to as the section of $\mathbf{J}$ on the i^{th} sub-domain. Note that the sub-domain matrix, $\mathbf{J}_i^{sub}$, is obtained via,

$$\mathbf{J}_i^{sub} = \mathbf{R}_i \mathbf{J} \mathbf{R}_i^T . \tag{148}$$

The restriction matrix can be expressed in terms of the sub-domain identity matrix by the following,

$$\mathbf{I}_i = \mathbf{R}_i^T \mathbf{I}_i^{sub} \mathbf{R}_i , \tag{149}$$

where $\mathbf{I}_i^{sub}$ is simply an identity matrix with dimensions of sxs. Within the context of Equations (147) and (148), the restriction matrix can be viewed as a mapping between the i^{th} sub-domain and the global domain. Thus a (ROW,COL) entry in $\mathbf{R}_i$ is equal to one only if the sub-domain component number given by ROW is mapped to the global component number given by COL, otherwise the entry is zero. Practical implementation of these techniques subsequently requires establishing mapping arrays that convert local sub-domain variable unknown numbers into global variable unknown numbers and vice versa.

Figure 23 illustrates the partitioning of a global domain into four overlapping domains, whose union is the global domain. In this simple example, row ordering of the finite volume cells is assumed for both the global domain and within the four sub-domains. Additionally, for simplicity it is assumed that there is only one cell-centered variable unknown per cell. These assumptions enable the global state vector to be expressed as,

$$\mathbf{x} = \left[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16}\right]^T , \tag{150}$$

while the restrictions of this global state vector to the four sub-domains (expressed in terms of the global component numbers) are given by:

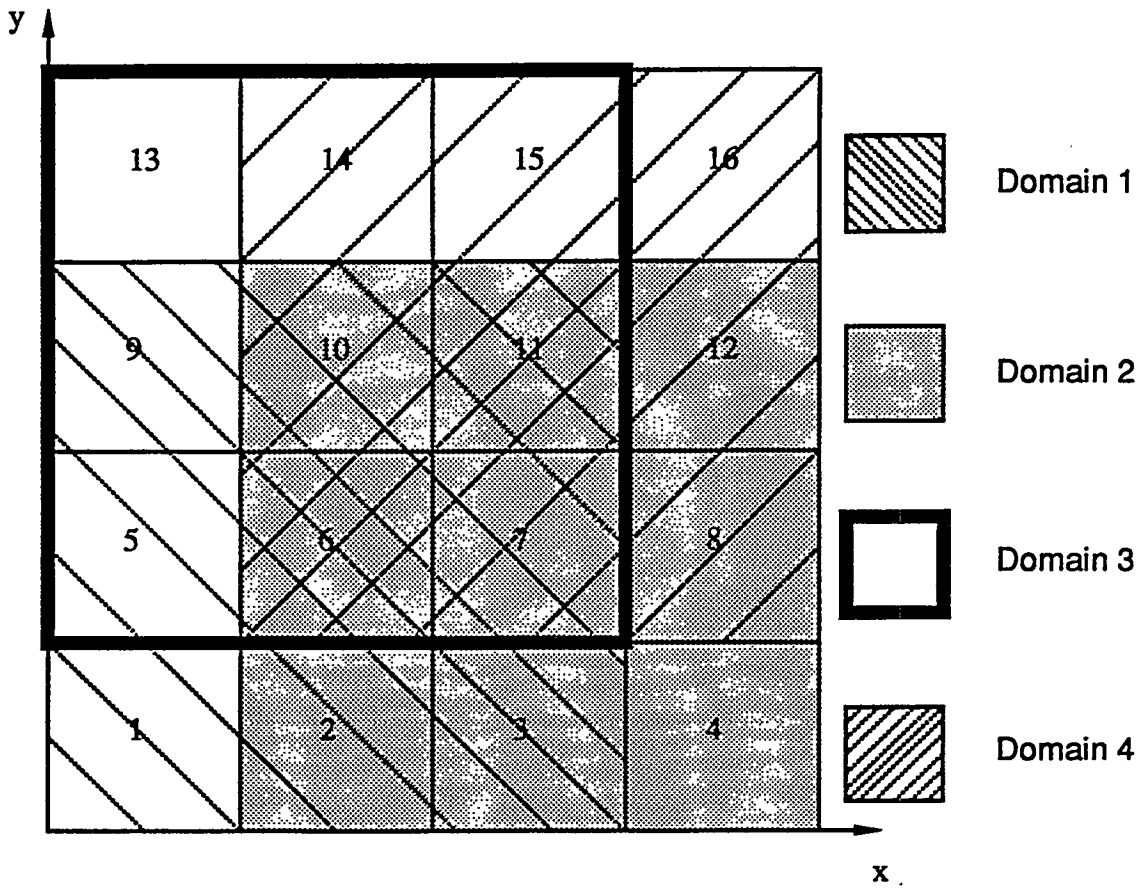


Figure 23. Schematic illustration of the partitioning of a global domain into four overlapping sub-domains.

$$\mathbf{x}_1^{sub} = \mathbf{R}_1\mathbf{x} = \left[x_1, x_2, x_3, x_5, x_6, x_7, x_9, x_{10}, x_{11}\right]^T, \tag{151}$$

$$\mathbf{x}_2^{sub} = \mathbf{R}_2\mathbf{x} = \left[x_2, x_3, x_4, x_6, x_7, x_8, x_{10}, x_{11}, x_{12}\right]^T, \tag{152}$$

$$\mathbf{x}_3^{sub} = \mathbf{R}_3\mathbf{x} = \left[x_5, x_6, x_7, x_9, x_{10}, x_{11}, x_{13}, x_{14}, x_{15}\right]^T, \text{ and} \tag{153}$$

$$\mathbf{x}_4^{sub} = \mathbf{R}_4\mathbf{x} = \left[x_6, x_7, x_8, x_{10}, x_{11}, x_{12}, x_{14}, x_{15}, x_{16}\right]^T. \tag{154}$$

The restriction matrices used in Equation (151) through Equation (154) are defined as:

$$\mathbf{R}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{155}$$

$$\mathbf{R}_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{156}$$

$$\mathbf{R}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \text{ and} \tag{157}$$

$$\mathbf{R}_4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{158}$$

The restrictions of the Jacobian, $\mathbf{J}_i^{sub}$, corresponding to the sub-domains shown in Figure 23 are computed via Equation (148), resulting in four $9x9$ sub-domain matrices.

The notation introduced thus far enables the description of two different constructions of global preconditioner operators based upon the local sub-domain solves, which can be denoted by $\left(\mathbf{J}_i^{sub}\right)^{-1}$. In this work, these sub-domain solves are computed using LINPACK banded Gaussian elimination [17]. The global preconditioner operator [in this case applied from the left as in Equation (135)] can then be derived from sub-domain contributions defined by,

$$\mathbf{J}_i^{-1} = \mathbf{R}_i^T \left(\mathbf{J}_i^{sub}\right)^{-1} \mathbf{R}_i. \tag{159}$$

The algebraic additive Schwarz algorithm is obtained by defining the left preconditioning matrix in Equation (135) as,

$$\mathbf{P}_L^{-1} = \mathbf{J}_1^{-1} + \mathbf{J}_2^{-1} + \ldots + \mathbf{J}_i^{-1} + \ldots + \mathbf{J}_p^{-1} = \sum_{i=1}^{p} \mathbf{J}_i^{-1}. \tag{160}$$

Equation (150) illustrates the parallel nature of the additive Schwarz preconditioners. Whenever $\mathbf{P}_L^{-1}$ is required to act upon a general vector, this action can be separated into p-simultaneous operations. Each of these separate actions, in turn, are computed using the sub-domain solutions, which generally represent much smaller linear systems. Note that if no domain overlap is allowed, then this algorithm can be regarded simply as a block Jacobi preconditioner.

In the case of algebraic multiplicative Schwarz preconditioning, the left preconditioned Jacobian matrix in Equation (135) takes the form,

$$\mathbf{P}_L^{-1}\mathbf{J} = \mathbf{I} - \left(\mathbf{I} - \mathbf{J}_1^{-1}\mathbf{J}\right)\left(\mathbf{I} - \mathbf{J}_2^{-1}\mathbf{J}\right)\ldots\left(\mathbf{I} - \mathbf{J}_i^{-1}\mathbf{J}\right)\ldots\left(\mathbf{I} - \mathbf{J}_p^{-1}\mathbf{J}\right) = \mathbf{I} - \prod_{i=1}^{p}\left(\mathbf{I} - \mathbf{J}_i^{-1}\mathbf{J}\right). \qquad (161)$$

The calculation of the preconditioner operation upon a general vector, i.e. $\mathbf{w} = \mathbf{P}_L^{-1}\mathbf{v}$, can be implemented as in Cai and Saad [153] by:

$$\begin{array}{l} 1.\ \ \mathbf{s} = \mathbf{J}_1^{-1}\mathbf{v} \\ 2.\ \ \textit{Do}\ \ i = 2, p \\ \qquad \mathbf{w} = \mathbf{s} + \mathbf{J}_i^{-1}(\mathbf{v} - \mathbf{J}\mathbf{s}), \\ \qquad \mathbf{s} = \mathbf{w} \\ \qquad \textit{End Do} \end{array} \qquad (162)$$

where s is just temporary vector. In the absence of overlap, the multiplicative Schwarz preconditioner can be viewed as being analogous to a block Gauss-Seidel preconditioner.

The notation used above is extremely valuable in demonstrating the process of developing a global preconditioner operator from local sub-domain solves. However, practical implementation of these preconditioner operators precludes generating all of the $\mathbf{J}_i^{-1}$ section matrices that appear in Equation (160) through Equation (162) because of prohibitive memory cost. Fortunately, the operations required of these section matrices can be shifted to sub-domain calculations. For instance, the $\mathbf{J}_i^{-1}$ appear only in operations of the form, $\mathbf{J}_i^{-1}\mathbf{u}$, where $\mathbf{u}$ is a general global vector. Using Equation (159), these operations can be written as,

$$\mathbf{J}_i^{-1}\mathbf{u} = \left[\mathbf{R}_i^T\left(\mathbf{J}_i^{sub}\right)^{-1}\mathbf{R}_i\right]\mathbf{u} = \mathbf{R}_i^T\left\{\left(\mathbf{J}_i^{sub}\right)^{-1}\left[\mathbf{R}_i\mathbf{u}\right]\right\}. \qquad (163)$$

The significance of this expression is that it implies that only the sub-domain inverses, $\left(\mathbf{J}_i^{sub}\right)^{-1}$, need be explicitly computed and stored. In fact, the terms inside the brackets represent a sub-domain solve, with the restriction matrix, $\mathbf{R}_i$, extracting the portions of the $\mathbf{u}$-vector corresponding the to the i^{th} sub-domain (i.e., the restriction of $\mathbf{u}$ onto the i^{th} sub-

domain). The outer action of the restriction matrix transpose serves to map the resulting sub-domain vector back to the global domain (the extension of the sub-domain vector to the global domain). The use of Equation (163) then enables practical implementations of the one-level, algebraic additive and multiplicative Schwarz preconditioner operators. A comparison of these preconditioners with the previously described ILU type preconditioners, with respect to parallel implementation, clearly favors these domain-based preconditioners.

3.2.4. Finite Difference/Inexact Newton Projection Methods (Matrix-Free Implementations)

The Krylov subspace based algorithms used in this investigation require the Jacobian matrix only in the form of matrix-vector products. These products may be approximated by finite difference projections of the form,

$$\mathbf{Jw} \approx \frac{\mathbf{F}(\mathbf{x}+\varepsilon\mathbf{w})-\mathbf{F}(\mathbf{x})}{\varepsilon}, \tag{164}$$

where ε is a small perturbation constant [not to be confused with the convergence tolerance, ε^n, used in Equation (127)] and $\mathbf{w}$ is a typical Krylov vector. Use of this approximation leads to so-called matrix-free inexact Newton iteration [see 67, 157, 158, and 159]. When preconditioning is employed, $\mathbf{J}$ is replaced by $\mathbf{JP}_R^{-1}$ and the finite difference approximation to $\mathbf{JP}_R^{-1}\mathbf{w}$ is computed from,

$$\mathbf{JP}_R^{-1}\mathbf{w} \approx \frac{\mathbf{F}(\mathbf{x}+\varepsilon\mathbf{P}_R^{-1}\mathbf{w})-\mathbf{F}(\mathbf{x})}{\varepsilon}. \tag{165}$$

Implementation of this approximation is accomplished by replacing matrix-times-a-vector operations within the Krylov algorithms with calls to a routine that uses Equation (164). Thus, each time this routine is called during the inner iteration the discrete governing equations vector must be evaluated with perturbed values of the state variable. Since these function evaluations can be expensive, the number of required inner iterations should be kept as low as possible. This can be accomplished by limiting the maximum number of inner iterations, using pseudo-transient calculations, and by using effective preconditioning.

The potential benefits of this approximation is significant considering it enables the action of the Jacobian without requiring explicit formation or storage of the Jacobian matrix. This feature can be extremely advantageous in problems where forming and storing the Jacobian is a very costly. However, problems such as those considered here require effective preconditioning and so the full Jacobian matrix is often still needed, at least periodically, to compute a new preconditioner. In this situation, the primary advantage of the matrix-free implementation is in reducing the total required function evaluations by amortizing the cost of forming the Jacobian and preconditioning matrices over many Newton iterations [15]. The use of this technique is demonstrated in Section 4.1.4.3 and in Reference 15.

A simple analysis can be used to predict a "break-even" average inner Krylov iteration count with respect to required function evaluations. This analysis simply counts the required number of function evaluations for n standard Newton-Krylov iterations and the required number of function evaluations for n matrix-free Newton-Krylov iterations. During the standard Newton-Krylov iteration, the numerical Jacobian, as described in Section 3.1.1, is formed on each Newton step; whereas during the matrix-free implementation, the Jacobian is formed only every d Newton steps. The matrix-free Newton-Krylov implementation also assumes an average of $\overline{m}$ inner iterations per Newton step. Equating these two function evaluation counts and solving for $\overline{m}$ gives the "break even" average inner iteration value,

$\overline{m}_{be}$. Average inner iteration counts below this value will yield lower required numbers of function evaluations for the matrix-free implementation.

As an example, consider the case of incompressible flow with heat transfer described in Section 2.1. The finite volume stencils used in the discretization of the equations for this case were shown in Figures 4 through 8. Based upon the numerical Jacobian evaluation scheme described in Section 3.1.1 and the variable dependencies shown in these figures, the following function evaluations are required in each computational cell: 14 for each of the discretized momentum equations (i.e., $f^{u-momentum}$ and $f^{v-momentum}$), 5 for the discretized continuity equations (i.e., $f^{continuity}$), and 10 for the discretized energy equation (i.e., f^{energy}). Note that the discretized pressure equation requires 27 function evaluations (i.e., $f^{pressure}$). These individual function evaluations can be summed over all the computational cells (NC) to give a total number of required function evaluations per standard Newton-Krylov step. When using the discrete continuity equation, this total is $43 \cdot NC$, while the use of the discrete pressure equation formulation gives $65 \cdot NC$. In general, this total can be written as $NF \cdot NC$, where NF denotes the number of function evaluations required in each computational cell.

In order to obtain an estimate for the required function evaluations for a single matrix-free Newton-Krylov step, assume that the GMRES algorithm is selected. This Krylov algorithm requires one Jacobian-vector product per GMRES iteration and one additional Jacobian-vector product to compute the initial GMRES residual as described in the Appendix. Each Jacobian-vector product requires an additional $NEQ \cdot NC$ function evaluations [i.e., $(\overline{m}+1)(NEQ \cdot NC)$]. This number is in addition to the $NEQ \cdot NC$ function evaluations required for the initial Newton residual evaluation. Thus, on each matrix-free Newton-Krylov step, $(\overline{m}+2) \cdot NEQ \cdot NC$ function evaluations are required, assuming the Jacobian is not evaluated. However, if the Jacobian is evaluated in order to generate a new

preconditioner, then an additional $NF \cdot NC$ function evaluations are required on that particular Newton step.

The matrix-free implementation often allows the Jacobian and preconditioner to be frozen for d Newton steps without degrading the convergence of the Newton iteration. In this manner, the cost of these additional function evaluations can be amortized over d Newton steps. In this case, $\overline{m}_{be}$ can be found from the following expression that equates the required number of function evaluations for n standard Newton-Krylov iterations with the required number of function evaluations for n matrix-free Newton-Krylov iterations:

$$n(NF \cdot NC) = n(\overline{m}_{be} + 2)(NEQ \cdot NC) + \frac{n}{d}(NF \cdot NC). \tag{166}$$

Solving for $\overline{m}_{be}$ gives,

$$\overline{m}_{be} = \left(1 - \frac{1}{d}\right)\frac{NF}{NEQ} - 2. \tag{167}$$

CPU efficiency improvements are possible with the matrix-free implementation if the average number of inner iterations ($\overline{m}$) is significantly below $\overline{m}_{be}$ and $d > 1$. Note that the expression for $\overline{m}_{be}$ in Equation (167) does not depend upon the number of computational cells, NC. Consequently, $\overline{m}_{be}$ remains constant as the grid is refined, while the required number of inner iterations often increases with NC. As a result, use of effective preconditioning and/or pseudo-transient calculations is typically required for CPU efficient matrix-free implementations, especially on finer grids. Values of $\overline{m}_{be}$, computed from Equation (167), are given in Table 1 below for various values of d using both the discretized continuity equation and the discretized pressure equation. Use of the discretized pressure equation results in higher values of $\overline{m}_{be}$ due to the larger number of variable dependencies

(i.e., stronger coupling) associated with the discretized pressure equation. In general, problems represented by a large number of strongly coupled governing equations (i.e., large values of NF/NEQ) will assume higher values of $\overline{m}_{be}$, and so more significant reductions in the number of required function evaluations are possible [e.g., see 15].

Table 1. Estimated "break even" inner iteration values.

d	$\overline{m}_{be}$	
	Discrete continuity eq. ($NF = 43$)	Discrete pressure eq. ($NF = 65$)
2	3.4	6.1
5	6.6	11.0
10	7.7	12.6
15	8.0	13.2
20	8.2	13.4
∞	10.8	16.3

Equation (164) indicates that the accuracy of the matrix-free approximation is strongly dependent upon the vector, $\mathbf{w}$. Since this vector changes within the inner Krylov iteration, the accuracy of the matrix-free approximation is subject to some uncertainty. In our matrix-free implementation, the perturbation constant, ε, is chosen as follows,

$$\varepsilon = \frac{1}{N}\sum_{i=1}^{N}\varepsilon_i, \tag{168}$$

where

$$\varepsilon_i = aa \cdot x_i + bb. \tag{169}$$

x_i is the i^{th} component of the state vector of dimension N and aa is a perturbation constant whose magnitude is on the order of the square root of computer round-off. It is noted for completeness that alternative options exist for determining ε [66, 67, 157, and 160]. The use of the matrix-free approximation requires a new definition of the inner iteration convergence criteria expressed by Equation (127). This new convergence criteria is expressed as,

$$R_i^n = \frac{\left\| \frac{\mathbf{F}(\mathbf{x}^n + \varepsilon\delta\mathbf{x}^n) - \mathbf{F}(\mathbf{x}^n)}{\varepsilon} + \mathbf{F}(\mathbf{x}^n) \right\|}{\|\mathbf{F}(\mathbf{x}^n)\|} < \varepsilon^n, \tag{170}$$

where once again the distinction between ε, which is a perturbation constant, and ε^n, which is the convergence tolerance parameter, should be noted. The use of this "matrix-free" approximation is investigated in Chapter 4.

CHAPTER 4

NUMERICAL RESULTS

The goal of this chapter is to investigate the various features of the numerical solution algorithm and to evaluate algorithm performance based upon solutions to incompressible and compressible fluid flow and heat transfer problems. Consequently, algorithm performance data and results are presented from solutions to the model problems described in Chapter 2 using the numerical techniques described in Chapter 3. The problems of incompressible and compressible flow are treated separately in order to isolate the specific issues relevant to each case. Thus, Section 4.1 considers incompressible fluid flow and heat transfer applications; specifically the natural, mixed, and forced convection model problems presented in Chapter 2. Next, Section 4.2 demonstrates the application of Newton-Krylov solution techniques for low Mach number compressible flow past a backward facing step. This latter section focuses primarily upon effective preconditioning strategies for low Mach number flow regimes.

4.1. INCOMPRESSIBLE FLOW

In this section, Newton-Krylov algorithms and finite volume discretization are used to solve the steady, incompressible Navier-Stokes and energy equations that were presented in Section 2.1.1. The problems of natural convection in an enclosed cavity (see Section 2.1.3), mixed convection flow past a backward facing step (see Section 2.1.4), and forced convection flow past a backward facing step (see Section 2.1.5) are considered. Solutions to these three different incompressible fluid flow and heat transfer problems are used to investigate different aspects of the overall numerical algorithm.

4.1.1. Important Computational Issues

One important computational issue requires specific attention when solving the primitive variable form of the steady, incompressible Navier-Stokes equations. This issue arises because for incompressible flow, pressure does not appear explicitly in the continuity equation, although it is strongly linked to this equation through its influence on the velocity components. Additionally, there is no equation of state for incompressible flow, from which pressure can be determined. Thus, the problem is determining how to best calculate pressure. If a direct linear solver with efficient pivoting is used with a simultaneous solution technique, pressure can be aligned with the continuity equation because pivoting will handle the zero main diagonal entries that arise. Since pressure does appear in the momentum equations, another option is to re-align the equations and variables so that pressure is determined from one of the momentum equations. This option was successfully employed by Vanka [30] using a direct sparse matrix package that avoided expensive pivoting operations [161]. However, Clift and Forsyth found that this option did not perform well when an iterative solver was used in place of a direct solver, and suggested that efficient cell ordering strategies be used to avoid this problem [65]. Other techniques used to solve this problem of indirect pressure linkage include use of a derived pressure equation in place of the continuity equation [21, 22, 104], stream function/vorticity formulations [19, 20, 78], penalty methods [37, 49], and artificial compressibility methods [59, 60].

This difficulty is treated in several different ways in this investigation. Recall from Section 3.2.3.1, that Kershaw's method [150] for treating unstable pivots in incomplete LU factorizations is one way to mitigate the problems associated with zero (or near zero) terms appearing on the main diagonal of the Jacobian matrix, which is used to derive the ILU preconditioner. Another technique is the use of alternate cell ordering strategies as suggested by Chin et al. [76] and Clift and Forsyth [65], although the cell ordering schemes considered

here are restricted to the simple cases described in Section 3.2.3.2. The third technique considered here is the use of the discrete pressure equation formulation described in Section 2.1.2.2.

Another important computational issue concerns the buoyancy force term in the momentum equations that couples these equations to the energy equation using the Boussinesq approximation [31]. This source of nonlinear coupling can cause convergence problems for some segregated solution approaches, especially at high Rayleigh numbers [25, 29]. In this work, however, full coupling is retained between these equations, which has shown to be advantageous in past work using a direct linear equation solver [25, 29]. The results below demonstrate fully coupled solutions of high Rayleigh number problems using Krylov subspace-based iterative solvers.

4.1.2. Natural Convection in an Enclosed Cavity Model Problem

The natural convection model problem solved in this section is described in Section 2.1.3. Solutions to this problem are used to evaluate several different Newton-Krylov solvers with respect to computer memory requirements, standard algorithm performance, and performance using the matrix-free implementation described in Section 3.2.4. Note that much of the numerical results presented in this subsection can also be found in References 10, 13, and 12.

In the case of the natural convection problem, one 'problem' cell exist (lower left corner assuming row ordering), where a zero appears on the main diagonal of the Jacobian matrix that cannot be filled in during the generation of the ILU preconditioner (recall the discussion in Section 3.2.3.1 and above). This difficulty is overcome here by simply fixing the pressure to a constant value in that cell, which is justified for this model problem and the incompressible flow assumption [104].

The convergence criteria for the outer Newton iteration for all test cases shown here are given in Equation (102). The inner iteration convergence criteria defined by Equation (127) is used with a limit on the maximum number of inner iterations. In the numerical experiments presented here, this upper limit was set equal to two-hundred. Situations where this upper limit is encountered frequently are presented, which demonstrate that the selected iterative algorithm should not display erratic convergence behavior such as that exhibited by BCG [138, 139] and CGS [5]. Various techniques for setting the tolerance in Equation (127) are considered below.

4.1.2.1. Computer Memory Considerations

Because the inexact Newton algorithms considered here use conjugate gradient-like algorithms to solve Equation (98), significant reductions in computer memory requirements are possible. Table 2 demonstrates this memory advantage for the natural convection problem, comparing the memory required for a preconditioned Newton-Krylov algorithm with LINPACK banded Gaussian elimination [17] (direct-Newton). The direct-Newton data represents the memory required to store the factored Jacobian matrix, while the Newton-Krylov data represents the memory required to store the sparse Jacobian matrix and the ILU preconditioner. Four different levels of fill-in are considered for the ILU(k) preconditioner, where k denotes the level of fill-in. Table 2 shows that the potential memory advantage increases with grid refinement. For the 60x60 grid, the memory required for the direct solve is roughly an order of magnitude larger than that required for the iterative solve. For the coarsest grid listed, the direct solve memory requirement is a factor of two larger than the iterative solve with ILU(3) preconditioning and a factor of five larger with ILU(0) preconditioning. Note that direct-Newton computations were run on a CRAY X-MP/216

with 16 megawords of memory. Thus, the 60x60 grid was the finest allowable grid for calculations with the direct solver.

Table 2. Comparison of algorithm memory requirements using direct vs. iterative linear equation solvers (in megawords).

Grid	Direct solver	Iterative Solver			
		ILU(0)	ILU(1)	ILU(2)	ILU(3)
15x15	0.174	0.0342	0.0414	0.0558	0.077
30x30	1.343	0.1368	0.166	0.2232	0.31
60x60	10.56	0.547	0.662	0.8928	1.24
120x120	83.69	2.189	2.65	3.57	4.95

This data clearly demonstrates the computer memory advantage associated with the use of the Krylov iterative techniques. Recall, however, that "out of core" matrix solvers, such as the frontal method can also be used to handle large Jacobian matrices that exceed available "in core" memory. Only a limited number of matrix entries (contributing to the active 'frontal matrix') must be stored in core memory yet partial and full pivoting is possible in the frontal matrix [82]. Einset and Jensen found that despite the advantages of the frontal method, there is a break-even point in front width above which iterative solutions become more efficient [81]. Their preconditioned iterative method outperformed the frontal method in their tests when the frontal width exceeded approximately five-hundred. These results as well as those of other researchers [49] have encouraged the focus on "in core" solvers in this investigation.

4.1.2.2. Standard Newton-Krylov Algorithm Performance

In this section the performance of inexact Newton's method is benchmarked against a direct-Newton iteration using LINPACK banded Gaussian elimination. The performance of the inexact Newton's method using ILU preconditioned TFQMR is studied with respect to the inexact Newton convergence parameter (ε^n), the level of fill-in used in the ILU(k) preconditioner, and the use of mesh sequencing. In addition, the performance of the algorithm using TFQMR is compared with the performance obtained using the CGS algorithm. The TFQMR algorithm was selected for the first part of this investigation because it is a recently developed algorithm that has not yet received much attention for these types of applications. It is compared against the CGS algorithm because TFQMR was developed in an attempt to obtain more smoothly convergent CGS-like solutions. Thus, it is instructive to determine how successful TFQMR is in this regard. Note that the TFQMR algorithm provides an upper bound for the residual norm that was not used in this study (see the Appendix). Use of this upper bound could make the TFQMR algorithm less expensive because the calculation of the residual norm could be postponed until this upper bound was small enough. The calculations presented in this subsection were obtained using a CRAY X-MP/216 computer.

Performance data using LINPACK banded Gaussian elimination is presented in Table 3 for $Ra = 10^4$. The required number of Newton iterations (n) as well as the required CPU time are listed for three different mesh sizes. The last row represents data using mesh sequencing. In this case the required number of iterations on each grid is listed in the second column. Table 3 shows that the required CPU time increases significantly as the grid dimensions are doubled in both directions. Note also that the use of mesh sequencing reduced the required CPU time by roughly 34%. Mesh sequencing enables this savings by providing a better initial guess on the finest grid. This results in fewer iterations when the

CPU cost per iteration is high. For this problem, the CPU cost of a single iteration on the finest grid was equivalent to approximately seventy-five iterations on the coarsest grid. This behavior is consistent with previous results [23] where CPU savings of approximately 45% were observed. Differences between these results and those of Reference 23 are due to the use of different convergence criteria.

Table 3. Performance data using LINPACK banded Gaussian elimination for $Ra = 10^4$.

Grid	Newton Iterations (n)	CPU Time (seconds)
15x15	7	3.52
30x30	7	30.72
60x60	7	262.82
15x15 > 30x30 > 60x60	7, 4, 4	172.9

The efficiency of an inexact Newton iteration is closely tied to the proper selection of ε^n. If ε^n is chosen too small, needless extra work will be performed when the Newton iteration is not within the radius of convergence of the algorithm. Conversely, if ε^n is chosen too large, the convergence of the Newton iteration will be slow. Here, we investigate two options for setting ε^n. The first is to set ε^n to a constant value and the second is to let ε^n vary on each Newton iteration using an expression similar to that proposed by Reference 157. Table 4 demonstrates the effect of varying ε^n on algorithm performance. The results presented in Table 4 were obtained for $Ra = 10^4$ on a 60x60 grid starting from a flat initial guess ($u=v=0$, $T=0.5$) using ILU(2) preconditioned TFQMR to solve Equation (98). The expression used for ε^n, the required number of Newton iterations (n), the average TFQMR iterations per Newton iteration ($\bar{m}$), and the total CPU time are given. The results suggest

that $\varepsilon^n < 10^{-3}$ is too restrictive during the initial Newton iterations. Conversely, $\varepsilon^n > 10^{-1}$ was not sufficiently restrictive when the Newton iteration was close to the true solution, resulting in a larger number of required Newton iterations. The best overall results were obtained using $\varepsilon^n = (1/2)^{\text{Min}\{n,10\}}$. For this selection, ε^n initially assumes the value of 1/2 and is reduced with the Newton iteration number until it reaches a minimum value on the order of 10^{-3}. Thus, ε^n becomes more restrictive as the true solution is approached. The effect of varying ε^n on the algorithm convergence behavior is shown in Figure 24. Observe that $\varepsilon^n = 0.5$ results in slow convergence, while reduced ε^n values yield much faster (superlinear) convergence. Another important observation is that $\varepsilon^n = (1/2)^{\text{Min}\{n,10\}}$, although large initially, still produces very favorable convergence behavior. Similar performance was observed for coarser grids and other levels of ILU fill-in.

Table 4. Effect of varying ε^n on algorithm performance (60x60 grid, flat initial guess).

ε^n	n	$\bar{m}$	CPU Time (s)
10^{-4}	7	62	568.8
10^{-3}	8	51	536.6
10^{-2}	9	38	474.4
10^{-1}	11	25	408.76
$\frac{1}{2}$	24	12	545.9
$\left(\frac{1}{2}\right)^{\text{Min}\{n,10\}}$	9	32	403.8
$\left(\frac{1}{4}\right)^{\text{Min}\{n,10\}}$	8	40	441.7

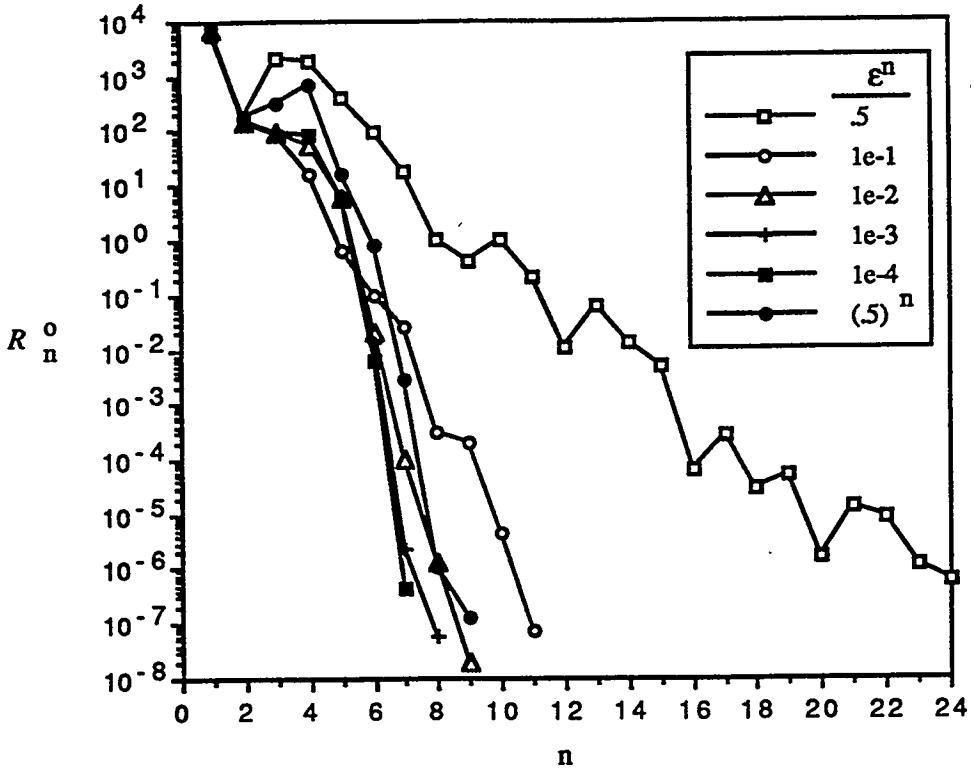


Figure 24. Effect of ε^n on algorithm convergence.

Analogous results obtained using mesh sequencing are shown in Table 5, where a 60x60 grid solution for $Ra = 10^4$ was obtained using a 15x15 and 30x30 grid sequence. In this case, values for n and $\bar{m}$ are presented for each grid. A comparison of Tables 4 and 5 again demonstrates the benefits of mesh sequencing. For example, using $\varepsilon^n = 10^{-2}$, mesh sequencing enabled a CPU time savings of approximately 42% (i.e., 474.4 sec. versus 278.8 sec.). Table 5 also suggests that when using mesh sequencing a more restrictive convergence criteria is needed during the initial Newton iterations in order to take advantage of the

improved initial guesses on the finer grids. The adaptive convergence selections did not work as well with mesh sequencing. In fact, a constant value of $\varepsilon^n = 10^{-2}$ yielded the best overall results. Potential savings using $\varepsilon^n = (1/2)^{\text{Min}\{n,10\}}$ on the initial grid is not warranted because the CPU cost of the initial grid solution is typically a small fraction of the cost of the total calculation.

Table 5. Effect of varying ε^n on algorithm performance when using mesh sequencing.

ε^n	n	$\bar{m}$	CPU Time (s)
10^{-4}	7,4,4	10,18,50	321.7
10^{-3}	7,4,4	8,16,45	275.4
10^{-2}	8,5,5	6,12,34	272.8
10^{-1}	9,7,8	3,7,20	293.9
$\left(\frac{1}{2}\right)^{\text{Min}\{n,10\}}$	7,7,8	5,7,31	390.5
$\left(\frac{1}{4}\right)^{\text{Min}\{n,10\}}$	7,6,6	6,12,33	321.7

Based on the results presented in Tables 4 and 5, it appears that the selection of $\varepsilon^n = 10^{-2}$ is advisable when a good initial guess is available. However, when a good initial guess is not available, the adaptive convergence criteria of $\varepsilon^n = (1/2)^{\text{Min}\{n,10\}}$ appears the best overall selection.

Effective preconditioning is essential in improving the performance of the TFQMR algorithm, as well as the other Krylov algorithms considered in this dissertation. In this section, the effectiveness of ILU-type preconditioning is investigated. One measure of an effective preconditioner is how well it approximates the system matrix. For an incomplete LU factorization, allowing more fill-in will most likely improve this approximation. The drawback, however, is higher CPU and memory storage cost. This suggests an optimal level

of fill-in that balances CPU time and memory considerations against preconditioner effectiveness. Table 6 demonstrates the effect of different levels of fill-in (k) versus grid size for $Ra = 10^4$. Listed for each grid are the required number of Newton iterations (n), the average TFQMR iterations per Newton iteration ($\bar{m}$), and the total CPU time in seconds (t). The solution on each grid was obtained from a flat initial guess ($u=v=p=0, T=0.5$). Although the use of $k>0$ on the 15x15 grid reduced the average TFQMR iterations, the total CPU time actually increased. For the 30x30 grid, a small reduction in CPU time was observed using ILU(1) and ILU(2). On the 60x60 grid the benefits of $k>0$ became more significant. ILU(2) preconditioning reduced the average TFQMR iterations by a factor of four and the total CPU time by approximately 30% compared with ILU(0) preconditioning. Note that the CPU performance of ILU(3) was poor on the coarser grids and was not as efficient as ILU(2) on the 60x60 grid. For this problem, use of ILU(2) preconditioning provides a good compromise between CPU time and memory considerations and preconditioner effectiveness.

Table 6. Effect of higher levels of ILU fill-in on algorithm performance (a flat initial guess was used on each grid).

	15x15 Grid			30x30 Grid			60x60 Grid		
k	n	$\bar{m}$	t	n	$\bar{m}$	t	n	$\bar{m}$	t
0	8	11	3.96	9	37	47.51	9	121	563.6*
1	8	5	4.95	9	16	46.98	8	60	521.6
2	7	5	6.23	8	11	40.54	9	32	403.8
3	9	6	14.63	8	13	70.76	9	26	487.3

* indicates the TFQMR iteration limit of 200 was encountered

It is instructive to benchmark the performance of the inexact Newton algorithm using the TFQMR algorithm against the use of the CGS algorithm. In addition, the performance of the inexact Newton iteration is compared with a direct-Newton iteration using LINPACK banded Gaussian elimination [17]. Table 7 compares the CPU performance of these different algorithms versus grid size for $Ra = 10^4$. The inexact Newton algorithms use ILU(0) on the coarsest grid and ILU(2) on the two finer grids. $\varepsilon^n = (1/2)^{\text{Min}\{n,10\}}$ is used as the inner iteration convergence criteria. The last two columns of Table 7 present ratios of the required CPU time using the iterative solvers to the required CPU time using the direct solver. Thus, R_{TFQMR} represents the ratio of total CPU time using the Newton-TFQMR algorithms to the total CPU time using the direct-Newton algorithm. Similarly, R_{CGS} represents the ratio of total CPU time using the Newton-CGS algorithm to the total CPU time using the direct-Newton algorithm. The results indicate that the Newton-CGS algorithm was more efficient than both the direct-Newton and Newton-TFQMR algorithms. The Newton-TFQMR algorithm was less efficient than the direct Newton iteration, but still competitive. For both inexact Newton algorithms, forward-backward solve operations associated with the ILU preconditioning dominated the CPU time. Our implementation of the CGS algorithm requires three of these operations per iteration, while the TFQMR algorithm requires five such operations. In addition, the TFQMR algorithm performs three more vector additions per iteration than the CGS algorithm. These differences roughly account for the increased CPU times observed for the TFQMR algorithm since the iteration counts for the two algorithms were similar. As noted previously, the TFQMR algorithm could be made more efficient by making use of the available upper bound for the residual norm.

The reduced CPU efficiency of TFQMR compared with CGS is compensated with smoother convergence behavior. CGS displays rather erratic convergence behavior as is discussed in the Appendix. The TFQMR algorithm attempts to smooth the convergence characteristics of CGS. Smooth convergence is important within the context of an inexact

Table 7. Comparison of CPU performance of two inexact Newton algorithms and a direct Newton algorithm (flat initial guess on each grid).

	CPU Time (s)				
Grid	Direct-Newton	Newton-TFQMR	Newton-CGS	R_{TFQMR}	R_{CGS}
15x15	3.52	3.96	2.25	1.125	0.64
30x30	30.72	40.54	28.3	1.32	0.92
60x60	262.82	403.76	221.54	1.54	0.84

Newton iteration because the inner iteration may be terminated before convergence (i.e., after reaching the maximum iteration limit of two-hundred). In this situation, it is important for the inner iteration to at least return a reasonable update that may eventually enable the Newton iteration to converge. Because of the erratic CGS convergence behavior, the return of an extremely poor Newton update in this situation is possible. In fact, if ILU(0) preconditioning is used to solve a higher Ra number problem ($Ra = 10^5$) on a 60x60 grid with $\varepsilon^n = 10^{-3}$, the Newton-CGS algorithm fails after only the second Newton iteration, while the Newton-TFQMR algorithm converges after 13 Newton iterations. The cause of the Newton-CGS failure is the erratic convergence behavior of the CGS algorithm. On the second Newton iteration both the CGS and the TFQMR algorithms encountered this upper limit. While the erratic convergence behavior of CGS returned a very poor approximate solution to Equation (98), the TFQMR algorithm returned an acceptable solution that allowed the eventual convergence of the algorithm. This behavior is illustrated in Figures 25 and 26, which show the convergence behavior of CGS and TFQMR on the first two Newton steps. Figure 25 shows that both algorithms converged to the desired tolerance on the first Newton step, although the convergence of the CGS algorithm is very erratic. Note how the TFQMR algorithm successfully controls and smoothes the erratic CGS convergence behavior.

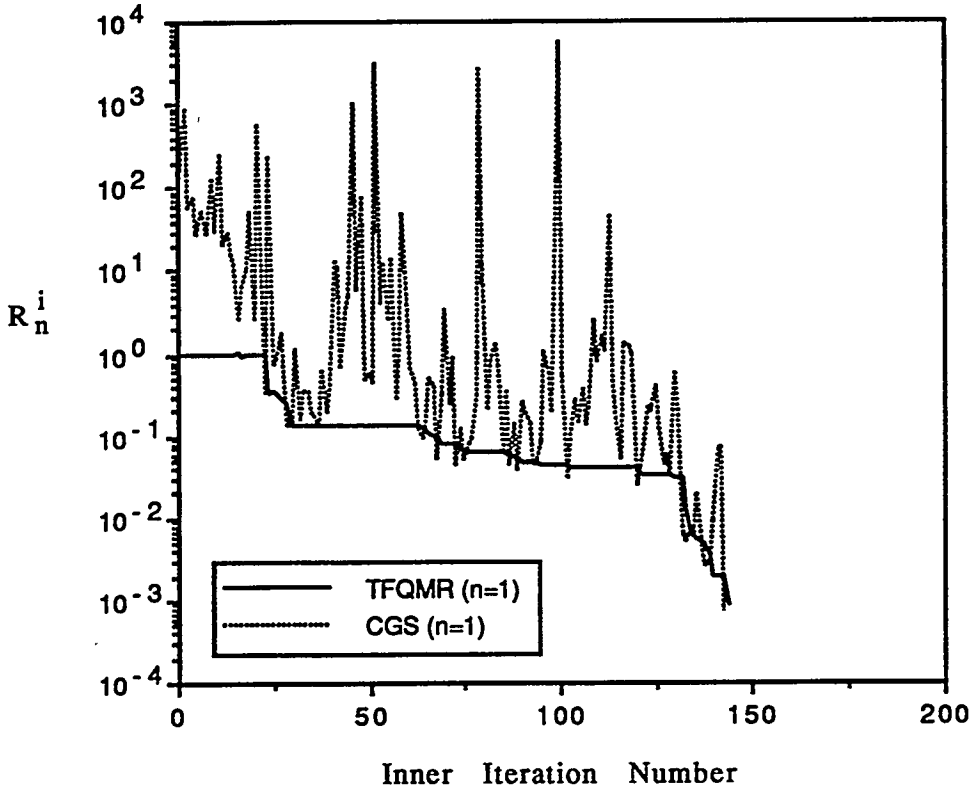


Figure 25. Convergence comparison between TFQMR and CGS on first Newton iteration.

Figure 26 shows that during the second Newton iteration neither algorithm converged to the desired tolerance after two-hundred iterations. The CGS algorithm terminated with R_n^i approximately equal to forty-five. This means that the residual norm of the linear system [Equation (98)] was forty-five times larger than if the Newton update was assumed zero. This poor update resulted in the failure of the algorithm on the subsequent Newton iteration. The TFQMR algorithm, on the other hand, terminated with R_n^i approximately equal to 0.043,

which represented an acceptable approximate solution to Equation (98). This behavior might be avoided in some instances by using better preconditioning to improve the convergence behavior of the iterative algorithms. However, this option may not always be feasible due to memory limitations or to the lack of a better known preconditioner. For this reason, the advantage of robustness associated with the use of TFQMR may often be more important than the CPU efficiency advantage obtained with CGS.

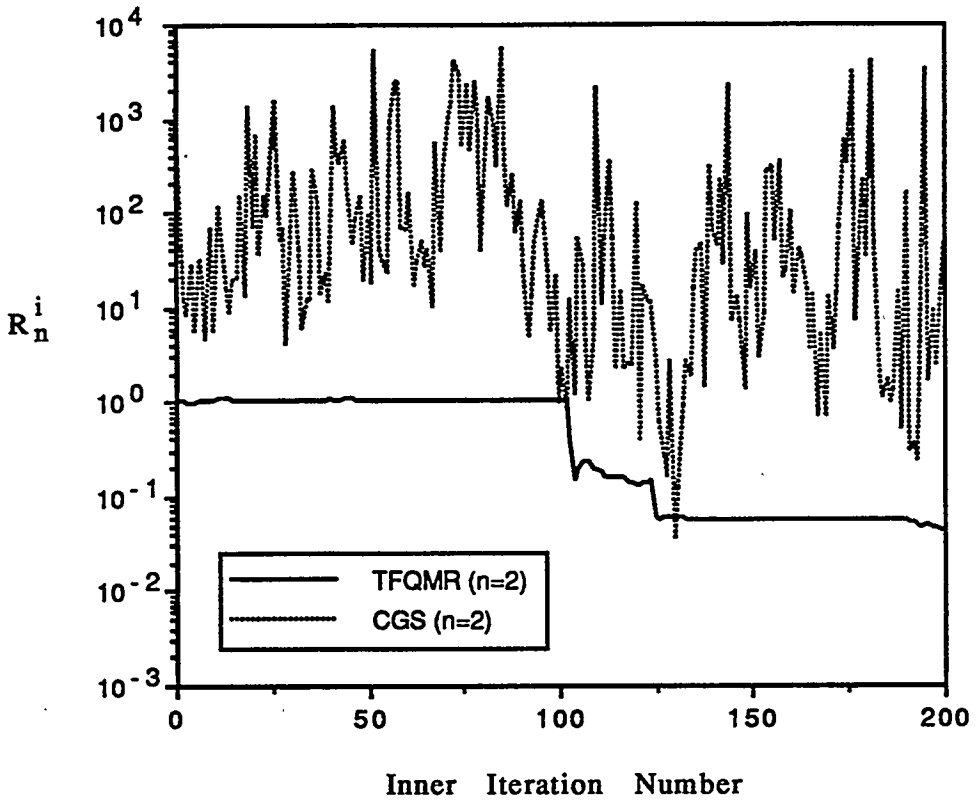


Figure 26. Convergence comparison between TFQMR and CGS on second Newton iteration.

4.1.2.3. Matrix-Free Newton-Krylov Algorithm Performance

The Krylov methods considered in this work require the Jacobian matrix only in matrix-vector products of the form $\mathbf{J}w$. This feature becomes very important within the context of an inexact Newton iteration because these products may be approximated by finite differences using Equation (164). The existence of this approximation is significant because it suggest the possibility of matrix-free implementations of Newton's method, thereby circumventing the main drawback associated with its use, namely high computer memory cost. The performance of this matrix-free implementation compared with the standard implementation is the focus of this subsection. The standard implementation computes matrix-vector products in the normal manner and uses Equation (127) to determine when the inner iteration has converged. In contrast, the matrix-free implementation computes matrix-vector products using Equation (165) since right ILU(0) preconditioning is employed. Convergence of the inner iteration for the matrix-free implementation is determined from Equation (170). Based on the results of the previous section, the inner iteration convergence tolerance is set from $\varepsilon^n = (1/2)^{\text{Min}\{n,10\}}$ when starting from a flat initial guess, and from $\varepsilon^n = 10^{-2}$ when starting from a reasonably good initial guess (i.e., a the interpolated solution from a coarser grid).

The advantages and disadvantages of the matrix-free implementation are studied with respect to performance and robustness. The computer memory advantages of the matrix-free implementation are obvious. Although in many practical applications formation of the Jacobian, or parts thereof, are still needed in order to generate an effective preconditioner. The potential performance advantage lies in reducing the CPU cost of forming and using the Jacobian matrix, without inhibiting or degrading convergence. Performance is studied using both Lanczos-based iterative solvers (CGS, TFQMR, and Bi-CGSTAB) and an Arnoldi-based iterative solver [GMRES(20)]. Note that in applications with a large number of

equations, where the cost of forming the Jacobian matrix is a significant fraction of the total CPU time, the potential advantages of the matrix-free implementation are very appealing [e.g., see 15]. Performance data is obtained for the steady state solution of the natural convection test problem with $Ra = 10^4$ and $Pr = 0.71$. All computations were run on an IBM RISC/6000 model 320 workstation. Calculations were initiated from a flat initial guess (i.e., $u=v=p=0, T=0.5$).

Tables 8 through 11 present performance data for solutions of the natural convection problem using both standard and matrix-free implementations on three different sized grids of increasing refinement. This data includes the required number of Newton iterations (n), the average number of inner iterations per Newton iteration ($\bar{m}$), the required CPU time, and the number of times the maximum inner iteration limit (m_{max}) was encountered. The maximum inner iteration limit was set equal to the square root of the number of unknowns, a selection that was empirically found to perform well with the matrix-free implementation. Note that the matrix-free implementation used here is not practical in the sense that a new ILU(0) preconditioner was formed each Newton iteration, which in turn requires the formation of the Jacobian matrix. A more practical implementation might use the same preconditioner for several Newton iterations (i.e., see Reference 15 and Section 4.1.4.3), or a less expensive preconditioner that does not require forming the complete Jacobian matrix. The former option is especially attractive when pseudo-transient calculations (see Section 3.1.4) are needed to ensure convergence [for an example see Reference 15 and Section 4.1.4.3]. However, the main purpose of this study is to investigate the effects of the matrix-free approximation. With this goal in mind, a more practical implementation was not absolutely necessary, and consequently ILU(0) was selected as the only preconditioner. Thus, CPU times should not be used as a basis for comparing the two implementations, but rather as a basis for comparing the performance of the different iterative solvers. Convergence behavior is used as a basis of comparison for the standard and matrix-free

implementations. Recall that the TFQMR algorithm provides an upper bound for the residual norm that was not used in this study. Use of this upper bound could make the TFQMR algorithm less expensive because the calculation of the residual norm could be postponed until this upper bound was small enough. In this implementation, however, the residual norm was computed on each iteration in order to provide a more accurate convergence check as well as to provide information regarding convergence history.

Table 8. Comparison of standard and matrix-free implementations on a 10x10 grid (m_{max} = 20).

	Standard Implementation				Matrix-Free Implementation			
Iterative Solver	n	$\bar{m}$	CPU Time (sec)	m_{max} hits	n	$\bar{m}$	CPU Time (sec)	m_{max} hits
CGS	7	7	2.3	0	9	9	6.4	2
TFQMR	8	7	2.9	0	8	12	7.2	3
Bi-CGSTAB	8	7	2.5	0	7	8	4.4	1
GMRES(20)	8	8	2.6	0	8	8	3.3	0

Table 9. Comparison of standard and matrix-free implementations on a 20x20 grid (m_{max} = 40).

	Standard Implementation				Matrix-Free Implementation			
Iterative Solver	n	$\bar{m}$	CPU Time (sec)	m_{max} hits	n	$\bar{m}$	CPU Time (sec)	m_{max} hits
CGS	8	17	16.7	0	-	-	-	-
TFQMR	8	21	24.4	0	10	34	85.4	6
Bi-CGSTAB	8	18	17.3	0	10	27	60.3	4
GMRES(20)	9	25	16.3	3	9	25	26.3	3

Table 10. Comparison of standard and matrix-free implementations on a 40x40 grid (m_{max} = 80).

	Standard Implementation				Matrix-Free Implementation			
Iterative Solver	n	$\bar{m}$	CPU Time (sec)	m_{max} hits	n	$\bar{m}$	CPU Time (sec)	m_{max} hits
CGS	9	55	188.2	3	-	-	-	-
TFQMR	10	61	305.7	4	58	79	4313.4	57
Bi-CGSTAB	13	69	322.0	8	-	-	-	-
GMRES(20)	19	74	320.0	16	21	74	631.2	18

Table 11. Comparison of standard and matrix-free implementations on a 40x40 grid using a 10x10, 20x20, and 40x40 mesh sequence (m_{max} = 20, 40, and 80, respectively).

	Standard Implementation				Matrix-Free Implementation			
Iterative Solver	n	$\bar{m}$	CPU Time (sec)	m_{max} hits	n	$\bar{m}$	CPU Time (sec)	m_{max} hits
CGS	5	62	131.7	0	-	-	-	-
TFQMR	5	70	195.1	3	29	80	2249.9	29
Bi-CGSTAB	7	71	196.6	4	28	79	1841.1	26
GMRES(20)	15	80	290.9	15	16	80	538.7	16

Table 8 presents performance data for a coarse 10x10 grid solution for each of the selected Krylov algorithms. Corresponding convergence plots are shown for both the standard and the matrix-free implementations in Figures 27 and 28, respectively. These figures plot the maximum relative Newton update, R_n^o, as a function of the Newton iteration count. The performance of the different iterative solvers is similar for both the standard implementation and the matrix-free implementation for this coarse grid.

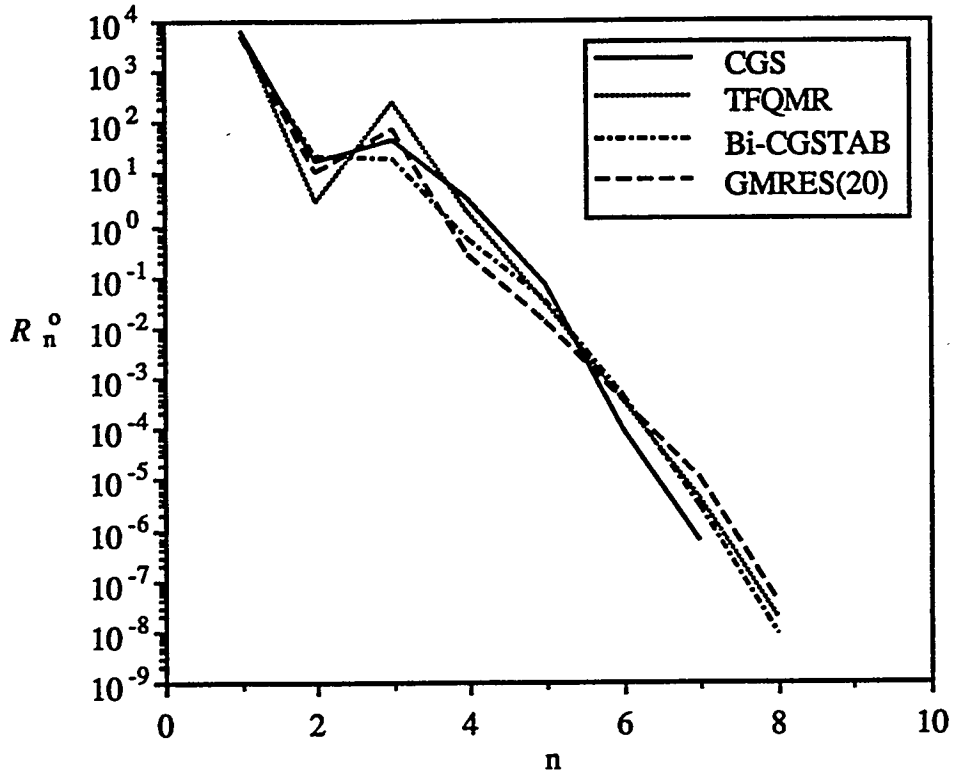


Figure 27. Standard inexact Newton iteration convergence behavior (10x10 grid).

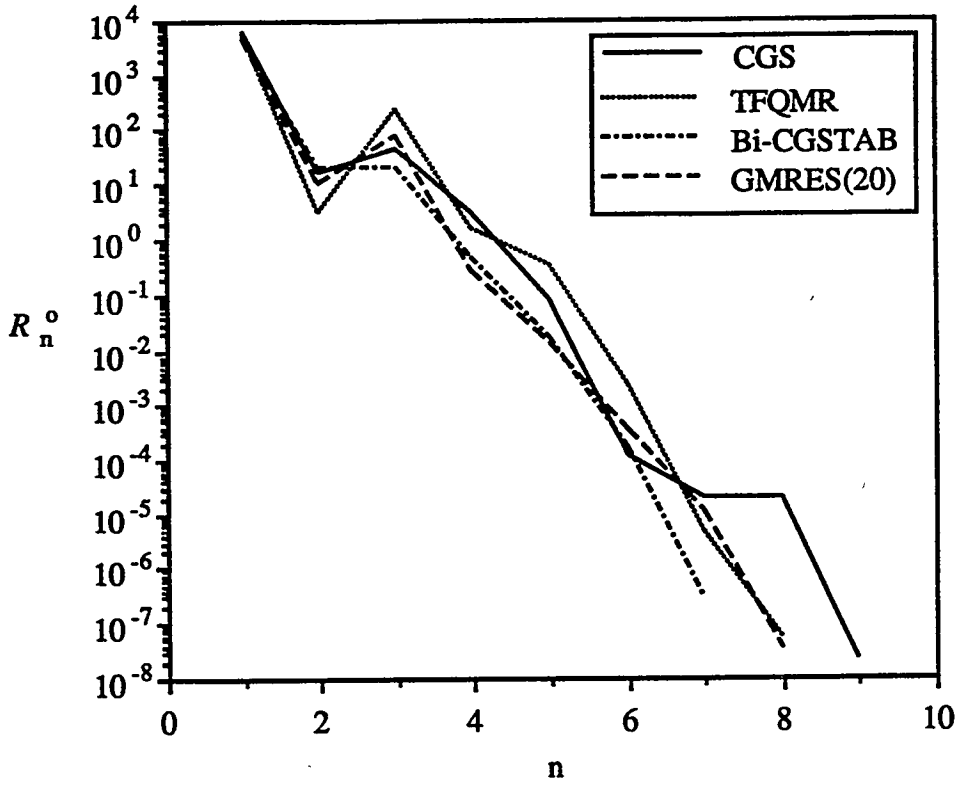


Figure 28. Matrix-free inexact Newton iteration convergence behavior (10x10 grid).

Tables 9 and 10 investigate the effect of grid refinement. Use of the Lanczos based iterative algorithms with the matrix-free approximation led to a marked degradation in performance as the grid was refined; while the Arnoldi based method (GMRES) performed similarly for both implementations. In fact, for the 40x40 grid (Table 10) no solutions were obtained using the matrix-free approximation with CGS or Bi-CGSTAB. The use of mesh sequencing in Table 11 led to a solution using Bi-CGSTAB, but still did not enable a solution using CGS.

Convergence plots for the 40x40 grid solutions in Table 11 are shown in Figures 29 and 30. These figures further illustrate the degradation in performance for the matrix-free implementation with the Lanczos-based algorithms. Note from Figure 29, the relatively slow convergence obtained using GMRES(20) for the standard implementation. This behavior follows from the choice for the dimension of our Krylov subspace ($k=20$). Twenty iterations was not sufficient to satisfy the inner iteration convergence criteria. This necessitated periodic algorithm restarts, which in turn slowed the convergence of the GMRES algorithm. This observation is evidenced by the large number of m_{max} hits encountered in Tables 10 and 11, and the convergence flattening trend shown in Figure 31 for the GMRES(20) curve. Figure 31 is a plot of R_n^i versus the inner iteration number for the first Newton iteration on the 40x40 grid corresponding to the standard implementation solutions in Table 11. Note that the first iteration was chosen because then each of the iterative algorithms are roughly solving the same linear system. GMRES(20) convergence is excellent during the first twenty iterations, but thereafter begins to flatten or stall as more periodic algorithm restarts are needed. Increasing the dimension of the Krylov subspace would improve performance, but it would also further increase algorithm memory requirements, which were already approximately twice that of the Lanczos-based algorithms.

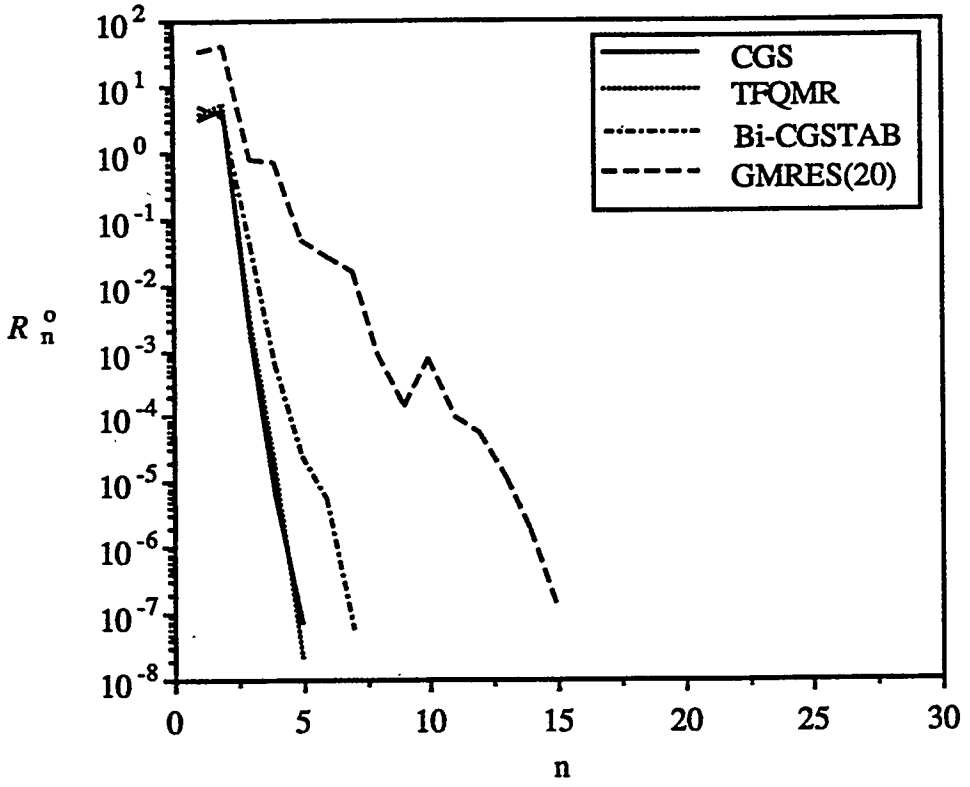


Figure 29. Standard inexact Newton iteration convergence behavior (40x40 grid).

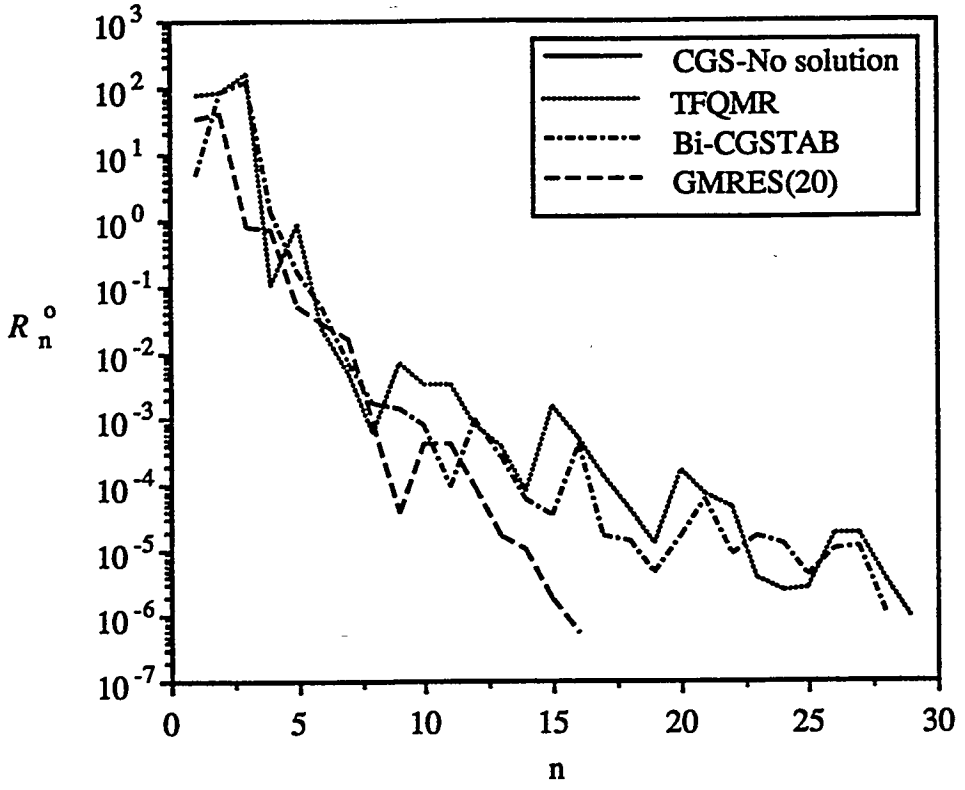


Figure 30. Matrix-free inexact Newton iteration convergence behavior (40x40 grid).

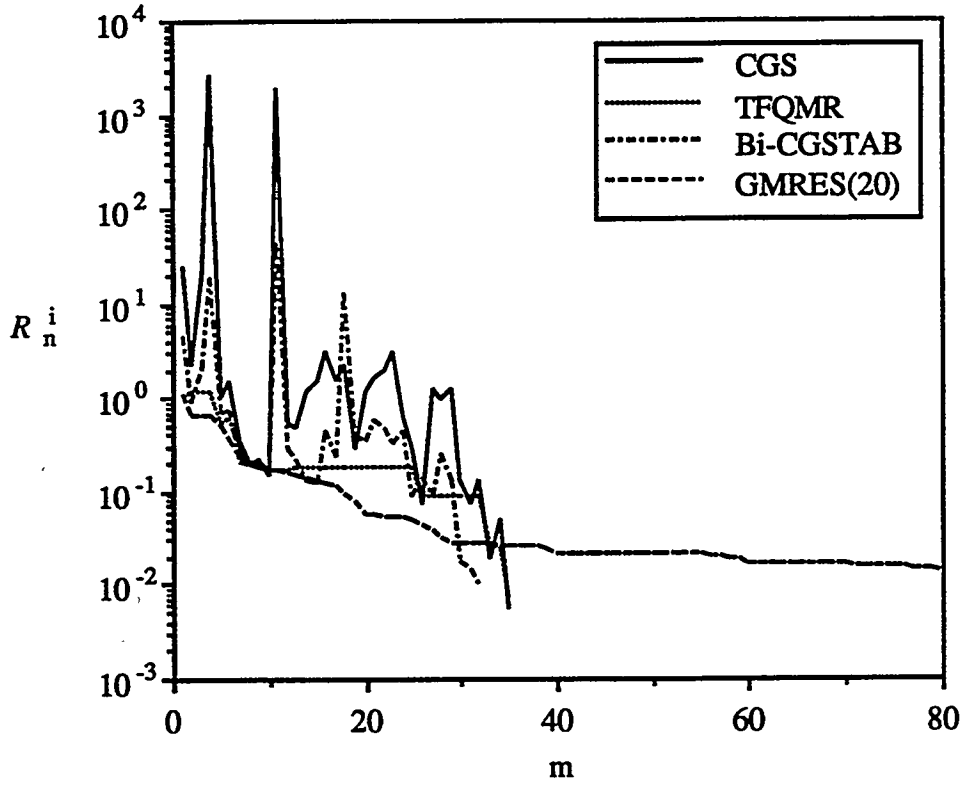


Figure 31. Inner iteration convergence on the first standard Newton iteration (40x40 grid).

Several additional observations can be gleaned from Figure 31. Notice again the rather erratic convergence behavior of the CGS algorithm that was shown previously. In spite of this behavior, the tabulated data shows that when the CGS algorithm converged it was more CPU efficient than the other algorithms. Figure 31 shows that for this problem, TFQMR is more successful than Bi-CGSTAB at controlling the erratic CGS convergence behavior, but at a noticeable higher CPU cost as can be seen in the tabulated results. The TFQMR convergence curve tends to temporarily stall or flatten out when CGS is displaying very erratic convergence behavior. The Bi-CGSTAB convergence curve, although more controlled than the CGS curve, still exhibits some erratic convergence behavior.

These observations may lend some insight into the relatively poor performance of matrix-free implementation when the Lanczos based methods are used. This performance is illustrated in Figure 30, which once again corresponds to the solutions presented in Table 11. Recall that no solution was obtained using the CGS algorithm with the matrix-free approximation. In the case of GMRES(20), replacing the standard implementation with the matrix-free approximation resulted in nearly identical convergence behavior. This suggests that Equation (165) yielded acceptable approximations for the needed matrix-vector products. In contrast, the convergence behavior using TFQMR and Bi-CGSTAB degraded appreciably when the standard implementation was replaced with the matrix-free approximation. In an attempt to relate this behavior to the observations cited above, consider the convergence behavior of the iterative solvers as shown in Figure 32 for the first Newton iteration. The erratic convergence behavior of CGS coupled with the use of Equation (165) results in very poor approximations for the needed matrix-vector products. The CGS algorithm could not recover from the erratic jumps and eventually returned a very bad Newton update that led to divergence. Note that this behavior has also been recently observed in Reference 96. Once again, the TFQMR algorithm is observed to stall out when the CGS iteration is behaving badly. During this Newton iteration, TFQMR stalls with a value of R_n^i near one. This

behavior, although resulting in poor convergence, does not cause divergence of the algorithm. During this first Newton iteration it is fortuitous that Bi-CGSTAB converged, because during later iterations it most often encountered the m_{max} limit as shown in Table 11. In addition, note that a solution could not be obtained using the matrix-free approximation with Bi-CGSTAB on the 40x40 grid starting from a flat initial guess. In that case, behavior similar to that of CGS over several Newton iterations led to divergence.

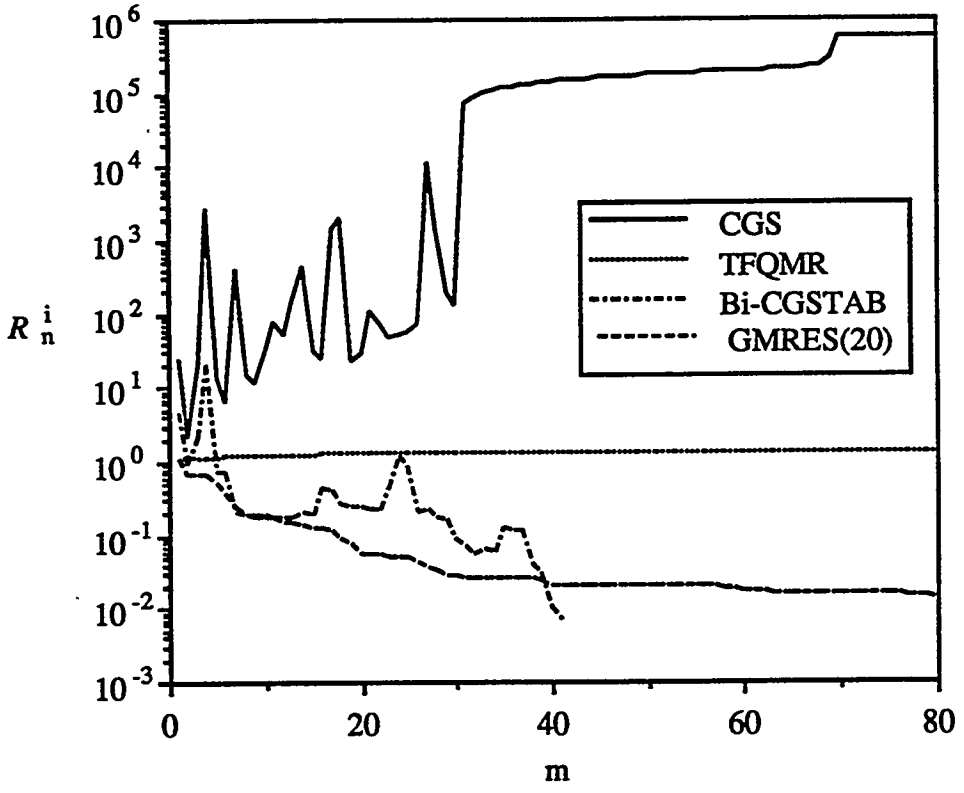


Figure 32. Inner iteration convergence on the first matrix-free Newton iteration (40x40 grid).

Recall that the accuracy of the matrix-free approximation in Equation (165) is dependent upon the Krylov vector, w. In the case of the Lanczos based algorithms, the characteristics of this vector may vary wildly as evidenced by the sometimes erratic CGS convergence behavior. In the case of GMRES, however, an orthonormal basis is constructed for the Krylov subspace so that only normalized ($L_2 - norm$) vectors appear in matrix-vector products. Presumably, this is one feature that enables Equation (165) to generate acceptable approximations for the required matrix-vector products needed within the GMRES algorithm.

4.1.2.4. Solutions

The solutions to the natural convection problem for $Ra = 10^4$ are compared with the benchmark solution of de vahl Davis [109] in Table 12. Note that the velocity data from Reference 109 were multiplied by the factor Pr^{-1} to account for the different choices in scaling. Additionally, the positions were adjusted to account for the reversed circulation direction in Reference 109. Table 12 presents the maximum horizontal velocity component (u) and its corresponding y-location along the line $x=0.5$, and the maximum vertical velocity component (v) and its x-location along the line $y=0.5$. Data from four different grids of increasing refinement are compared with the benchmark solution of Reference 109. Table 12 shows the improved agreement with the benchmark solution as the grid is refined. Excellent agreement between the benchmark solution and the 120x120 grid solution is obtained. Based upon the results of Section 4.1.2.2, ILU(2) preconditioning was selected with an inner iteration convergence tolerance of $\varepsilon^n = 1x10^{-2}$. Note that the solutions were run using the standard Newton-Krylov implementation. This calculation employed a four mesh sequence (uniform 15x15, 30x30, 60x60 and finally 120x120 mesh), and required 6 Newton iterations with an average of 114 TFQMR iterations per Newton step on the finest 120x120 grid. Note that these average inner iteration counts are quite high compared to earlier results. Based

upon previous results, one would expect that efficient use the GMRES algorithm in this situation would require either a very large specified Krylov subspace dimension, or a reduction of the average inner iteration counts via better preconditioning and/or pseudo-transient relaxation. The required Newton iterations on the intermediate grids were 8, 6, and 5, respectively (coarse grid to fine grid). The corresponding average TFQMR iterations on the three intermediate grids were 6, 12, and 35, respectively. The average TFQMR data for the four grid sequence illustrates a difficulty often associated with ILU type preconditioning, namely the poor scaling (measured by inner iteration counts) as the problem size is increased. The total required CPU time was 3464.1 seconds on an HP Model 735 workstation.

Table 12. Comparison with benchmark solution of de vahl Davis [109] for $Ra = 10^4$.

		Current Solution			
	Benchmark Solution	15x15	30x30	60x60	120x120
u_{max}	22.786	22.663	22.738	22.784	22.788
y	0.177	0.167	0.183	0.175	0.179
v_{max}	27.630	27.599	27.657	27.594	27.640
x	0.881	0.900	0.883	0.875	0.879

Data for $Ra = 10^5$ and $Ra = 10^6$ are presented in Tables 13 and 14, respectively (note that $Ra = 10^6$ data was the highest Rayleigh number data presented by Reference 109). Both of these calculation employed the same four grid sequence used for the $Ra = 10^4$ calculation. The difference between the 120x120 grid data and the benchmark data in these tables is less than 1% for each of the different Rayleigh numbers. On the final 120x120 grid, the $Ra = 10^5$

solution required 6 Newton iterations and an average of 112 TFQMR iterations per Newton step. The intermediate grid Newton iteration counts were 8, 6, and 5, respectively; while the corresponding average TFQMR iteration counts were 6, 12, and 35, respectively. The total CPU time for the entire calculation was 3438.1 seconds. In order to obtain the $Ra = 10^6$ solution, the adaptive damping strategy described in Section 3.1.2 was needed with α in Equation (116) set to 0.25. The $Ra = 10^6$ solution required 6 Newton iterations and an average of 88 TFQMR iterations per Newton step on the 120x120 grid. The intermediate grid Newton iteration counts were 17, 10, and 8, respectively; while the corresponding average TFQMR iteration counts were 11, 15, and 23, respectively. A total of 2937.0 CPU seconds were required for this calculation. The higher Rayleigh number solutions presented in Tables 13 and 14 demonstrate the robustness of the overall solution algorithm.

Table 13. Comparison with benchmark solution of de vahl Davis [109] for $Ra = 10^5$.

		Current Solution			
	Benchmark Solution	15x15	30x30	60x60	120x120
u_{max}	48.916	49.038	49.564	49.103	48.978
y	0.145	0.167	0.150	0.142	0.146
v_{max}	96.606	96.633	95.157	96.196	96.571
x	0.934	0.967	0.950	0.942	0.937

Table 14. Comparison with benchmark solution of de vahl Davis [109] for $Ra = 10^6$.

		Current Solution			
	Benchmark Solution	15x15	30x30	60x60	120x120
u_{max}	91.028	98.139	94.101	92.635	91.622
y	0.150	0.100	0.117	0.142	0.146
v_{max}	308.958	319.892	290.839	309.300	311.421
x	0.962	0.967	0.983	0.958	0.962

For completeness, note that the higher Rayleigh number solutions could have also been obtained from lower Rayleigh number solutions using parameter continuation (see Section 3.1.5). For example, restarting the 120x120 grid, $Ra = 10^5$ solution with the Rayleigh number increased to 10^6 required 12 Newton iterations with an average of 75 TFQMR iterations per Newton step. The required CPU time for this calculation was 4425.1 seconds. Thus, for this specific case, the use of mesh sequencing proved to be somewhat more efficient. Note additionally, that higher order discretization schemes can often be employed to obtain accurate solution resolution using coarser meshes than a lower order discretization scheme would require [see 14]. Use of one such scheme with a defect correction technique is the topic of the following section. Also, the use of non-uniform grids with cells clustered in regions where the solution is rapidly changing, is another technique for improving accuracy for a fixed number of computational cells. An example of the use of a nonuniform grid is given in Section 4.1.3.2.

4.1.3. Mixed Convection, Backward Facing Step Model Problem

The mixed convection, backward facing step model problem solved in this section was described previously in Section 2.1.4. In contrast to the previous natural (free) convection problem, this problem is characterized by a combination of both forced and free convection. The relative importance of these two effects are determined by the parameter, Gr/Re^2, appearing in Equations (2) and (3). For this benchmark problem, the Grashof number (Gr) is 1000, while the Reynolds number (Re) is 100, making $Gr/\mathrm{Re}^2 = 0.1$. However, the case of only forced convection (i.e., $Gr = 0$) is also considered to help identify the effects of buoyancy. Additionally, several calculations are run with $\mathrm{Re} = 200$.

Higher order accurate solutions to this problem are obtained using the defect correction technique described in Section 3.1.6. The use of higher order discretization schemes often enable a certain level of accuracy to be obtained on a mesh that is coarser than a lower order discretization scheme would require for the same level of accuracy. This feature helps to alleviate computer memory difficulties by enabling more accurate solutions on coarser grids. The intent of this section is to demonstrate the use of a higher order discretization scheme within the context of a fully coupled Newton-Krylov solution algorithm, and to evaluate its effect on algorithm performance. Note that much of the numerical results presented in this subsection can also be found in References 11 and 14.

In the case of this backward facing step problem, pressure cannot be fixed to a constant value in the interior of the computational domain because it is already specified on the outflow boundary. Consequently, the remedy used in the case of the natural convection problem to avoid the 'problem' cell cannot be used in this case. Alternatively, Kershaw's method for treating unstable pivots and the use of an alternative cell ordering scheme (see Section 4.1.4.1) are relied upon to remedy this difficulty. Recall that the former technique was described in Section 3.2.3.1. Although this technique was activated in the case of the

natural convection problem discussed above, the fixing pressure in the 'problem' cell made pivot adjustments unnecessary in all but a few cases. The cell ordering scheme employed is illustrated in Figure 33 below. Note that this figure is not intended to accurately portray the grid, but rather only to describe the cell ordering scheme employed. Note that cells numbered 37, 38, 41, and 42 lie inside the step and would be assumed inactive if the grid represented by Figure 33 were actually employed.

The convergence criteria for the outer Newton iteration for all test cases considered here are given in Equation (102). The inner iteration convergence criteria defined by Equation (127) is used with a limit on the maximum number of inner iterations. A value of 500 was used when investigating various numerical techniques in the next section, while a value of 200 was used when computing the solutions in Section 4.1.3.2. Note that ILU(0) preconditioning was employed unless otherwise specified.

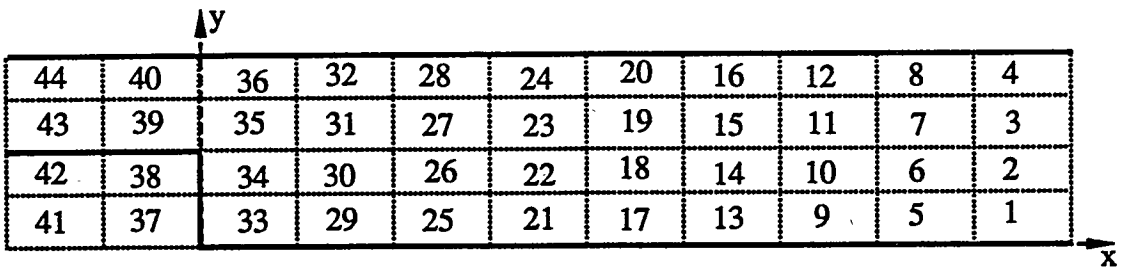


Figure 33. Schematic of the reverse row ordering scheme used with the mixed convection model problem.

4.1.3.1. Defect Correction Technique

The procedure used here to maintain overall second order accurate solutions is based upon a simple defect correction method (see Section 2.1.2.3). Often, a converged solution is initially obtained using a lower order discretization scheme, such as upwind differencing [see 19] or the simple power law convection-diffusion differencing scheme of Patankar [21]. The low order scheme is used in forming both the Jacobian matrix and the residuals in Equation (98). This low order accurate solution can then be improved by restarting the algorithm using a higher order discretization scheme for the calculation of the residuals in Equation (98). Note that this replacement is made only in the right hand side of Equation (98), not in the Jacobian. Alternatively, the defect correction technique can also be applied without using an initial low order accurate solution as an initial guess. The defect correction technique, although sometimes resulting in slow convergence during the final stages of the calculation, is very useful in maintaining overall second order accurate solutions. In this work, an implementation of the third order accurate CUI convection discretization scheme [see Section 2.1.2.3] is used along with conventional central differences for the diffusion terms. This implementation results in an algorithm that overall yields second order accurate solutions.

Before proceeding with solutions to the mixed convection, backward facing step model problem, it is instructive to review the natural convection results of Johnson et al. [14]. These results can be summarized by the following observations, obtained for values of the Rayleigh number ranging from 10^3 to 10^6 [see 14]:

1. Defect correction is a useful technique for obtaining higher order accurate solutions, but the CPU expense can be considerably higher than the cost for a lower order solution using a conventional Newton iteration. Consequently, use of this technique is not warranted on fine grids where the solutions produced by both low and high order discretization schemes are comparable. Rather, its benefit lies

in improving the accuracy of the solution on a grid, for which the low order discretization scheme cannot achieve a comparable level of accuracy.

2. For highly nonlinear problems (i.e., high Rayleigh numbers), improved efficiency was obtained if the defect correction calculation was restarted from a lower order solution on the final desired grid. However, for lower Rayleigh numbers mesh sequencing with defect correction used on each mesh proved efficient.
4. Defect correction computations of higher order solutions were less expensive than a conventional Newton iteration where the higher order discretization scheme was used in both the residual and the Jacobian evaluation. This result was attributed to several factors. First, the higher order discretization scheme made the Jacobian and residual evaluations more expensive. Second, ILU(0) preconditioner was less effective when derived from the less diagonally dominant Jacobian, which was evaluated with the higher order discretization scheme. And finally, a more ill-conditioned Jacobian matrix resulted from the use of the higher order discretization scheme, especially at higher Rayleigh numbers, making convergence more difficult.
5. The defect corrected Newton-TFQMR algorithm solved a 40x40 grid, natural convection problem with $Ra = 10^4$, five times faster than an analogous implementation using the SIMPLE algorithm.

These observations help identify the important issues regarding the use of the defect correction technique to obtain higher order accurate solutions.

As with the third observation above for the natural convection problem, the use of a conventional Newton iteration to obtain higher order accurate solutions for this benchmark solution did not prove effective. Note that another option, not considered here, is the use of a lower order scheme to generate the preconditioner, while still using the higher order scheme in the Jacobian and residual evaluation [see 78]. One difficulty in this approach is that the Jacobian must be evaluated twice, once for the preconditioner and once using the higher order discretization scheme. However, there is the possibility that this difficulty could be avoided by using the matrix-free implementation discussed previously. Although these

techniques deserve further study, the higher order accurate solutions presented in this section rely upon the defect correction techniques discussed in Section 3.1.6.

Several different techniques for improving the efficiency of computing higher order accurate solutions are investigated in this section for $Gr = 1000$ and two Reynolds numbers (100 and 200). The solution of the mixed convection test problem on a 140x40 [(x-cells) by (y-cells)] uniform grid is selected to test these various numerical techniques. Efficiency is measured with respect to the total required CPU time. In order to provide a reference point for evaluating efficiency, calculations using upwind differencing (see Section 2.1.2) were obtained starting from a flat initial guess (u=1, v=p=0, T=0.5) on the 40x140 grid. The $\mathrm{Re} = 100$ solution required 8 Newton iterations, an average of 126 TFQMR iterations per Newton step, and a total required CPU time of 2744.8 seconds on an IBM RISC/6000 workstation. The $\mathrm{Re} = 200$ solution required 9 Newton iterations, an average of 128 TFQMR iterations per Newton step, and a total required CPU time of 3110.3 seconds. These calculations serve as reference points for determining the additional CPU cost required to obtain a higher order accurate solution.

Starting the defect correction technique from a flat initial guess on the 140x40 grid did not result in a converged solution for either Reynolds number considered. Consequently, the upwind solution was used as the initial guess for the defect correction technique. The improved initial guess supplied by the initial upwind solution enabled convergence for both Reynolds number calculations. The restarted $\mathrm{Re} = 100$ solution required 21 Newton iterations, an average of 93 TFQMR iterations per Newton step, and a required CPU time of 6222.4 seconds. Thus, the total CPU time necessary to compute the higher order solution is the original upwind solution time plus the restarted calculation time, or 8967.2 seconds. This total time is 3.3 times the reference upwind solution time. The restarted $\mathrm{Re} = 200$ solution required 27 Newton iterations, an average of 108 TFQMR iterations per Newton step, and a total required CPU time of 8747.4 seconds. Thus the total time for the higher order accurate

solution was 11,587.7 seconds, or 3.8 times the first order accurate, upwind solution time. Although, additional CPU cost was incurred, a second order accurate solution was obtained.

Mesh sequencing was subsequently employed in an effort to improve the efficiency of these initial higher order accurate solutions. The grids employed were uniform 35x10, 70x20, and 140x40 meshes. Several different mesh sequencing options were investigated, these included:

1. Starting the mesh sequencing calculation from a flat initial guess using the defect correction technique.
2. Starting the calculation using the upwind differencing scheme and a conventional Newton-Krylov iteration to obtain an initial upwind solution on the coarse 35x10 mesh. Next, mesh sequencing was initialized from this upwind solution using the defect correction technique.
3. Starting the mesh sequencing calculation using upwind differencing and a conventional Newton-TFQMR iteration to obtain an upwind solution on the final 40x140 grid. Next, the defect correction scheme is started using this 40x140 upwind solution as an initial guess.

Table 15 presents performance data for the $Re = 100$ test run. Included in this table are the required number of iterations and average inner iteration counts per Newton step for each grid during both the conventional Newton-Krylov part of the calculation using upwinding (UDS) and the defect correction part of the calculation using the CUI scheme (CUI). Note that dashed lines indicate that the entry was not part of the calculation for that option. Additionally, CPU time associated with each part of the calculation and the total CPU time are also listed. The numbers listed in the last column of the table present the ratios of the total CPU time to the reference upwind solution CPU time identified above. The results for this test run indicate that switching to the defect correction technique after obtaining an initial 35x10 grid upwind solution proved to be the most efficient option.

Table 15. Performance data using several defect correction calculation options with mesh sequencing ($Re = 100$).

Option		35x10 grid		70x20 grid		140x40 grid		CPU (sec)	Total CPU (sec)	Ratio
		n	$\overline{m}$	n	$\overline{m}$	n	$\overline{m}$			
1	UDS	---	---	---	---	---	---	---		
	CUI	42	26	42	39	16	81	6704.6	6704.6	2.44
2	UDS	6	22	---	---	---	---	47.5		
	CUI	36	27	42	38	16	82	6603.0	6650.5	2.42
3	UDS	6	22	5	57	4	130	1723.1		
	CUI	---	---	---	---	21	99	6448.0	8171.1	2.98

The performance data obtained for the $Re = 200$ test runs are shown in Table 16. This case is different than the lower Reynolds number case in that no converged CUI solutions were obtained on the coarse 35x10 grid. The CUI solutions on the 35x10 grid failed to converge even after 500 Newton iterations. Even though convergence was not obtained on these grids, the unconverged solutions were still interpolated up to the next level grid, where convergence was eventually obtained, although with considerable effort. Thus, the best results for this Reynolds number were obtained by using the upwind solution on the 140x40 grid as an initial guess.

Table 16. Performance data using several defect correction calculation options with mesh sequencing ($Re = 200$).

Option		35x10 grid		70x20 grid		140x40 grid		CPU (sec)	Total CPU (sec)	Ratio
		n	$\overline{m}$	n	$\overline{m}$	n	$\overline{m}$			
1	UDS	---	---	---	---	---	---	---		
	CUI	500*	35	206	57	25	105	24280	24280	7.81
2	UDS	7	24	---	---	---	---	56.4		
	CUI	500*	34	182	58	25	106	23020	23077	7.4
3	UDS	7	24	6	39	5	124	2014.1		
	CUI	---	---	---	---	27	106	8649.1	10663	3.4

*Solution did not converge within the allowed 500 Newton iterations on this grid, but this unconverged solution was still interpolated up to next level grid.

The results for these two different Reynolds numbers indicate that the optimal grid size, to switch over to the defect correction technique, is likely problem dependent. However, the results in Table 16 demonstrate that if the switch is made on too coarse a grid, the overall efficiency is severely degraded. Recall that similar observations were made by Johnson et al [14] in solving the natural convection model problem. Thus, a lower risk option may be to delay the switch to the defect correction technique until an initial low order accurate solution is obtained on the fine grid. At that point, the analyst can determine if additional accuracy is deemed necessary; and, if so, the defect correction technique can be initiated to improve solution accuracy. This is the solution approach adopted below.

4.1.3.2. Solutions

This section presents solutions obtained for this mixed convection, backward facing step test problem. The values assumed for the nondimensional parameters are once again given by $Re=100, Pe=70$, and $Gr=1000$. Solutions to this mixed problem as well as a solution for $Gr=0$ (no buoyancy forces) are presented, along with memory and CPU requirements. The solutions presented here differ from those discussed above in that these solutions employed a nonuniform grid (described below), a reversed ordering scheme (i.e., numbering from the inlet first instead of the outlet), and a low order scheme based upon the power-law convection-diffusion discretization scheme instead of pure upwinding for the convective terms. Additionally, these runs were made using a version of the pilot code with Jacobian evaluation routines that were more CPU efficient (but also more problem specific) than those used to obtain the results presented above. Consequently, the CPU times presented below (which are smaller) should not be compared with those presented above.

Lin et al. [162] demonstrated that solutions to problems of this type are strongly dependent upon the value of the buoyancy parameter, Gr / Re^2. Based upon the results of

Lin et al. and the value of this parameter for the benchmark problem (0.1), one would expect the solution to be characterized by a primary recirculation zone behind the step and a secondary recirculation zone that occurs at the step corner. In order to resolve these recirculation zones, we employed a non-uniform grid. The mesh spacing in the y-direction assumes a cosine-type distribution. The x-direction spacing is controlled such that the finest mesh region occurs two step heights downstream of the step. Away from this region the mesh spacing increases linearly towards both the flow inlet and outlet. The schematic of the mesh for a coarse 35x10 grid is shown in Figure 34. The results presented in this section were obtained using a 140x40 nonuniform grid. Note that no noticeable changes in the presented figures were observed when the number of grid cells were doubled in both dimensions. (i.e., using an 280x80 nonuniform mesh). Note that all figures associated with these solutions are accumulated at the end of this subsection.

The results for $Gr=0$ are presented first in order to better study the effects of mixed convection. Figures 35 through 37 show the principal velocity, transverse velocity, and temperature profiles at $x = 0, 3, 7, 15,$ and 30, respectively. Figure 38 presents the Nusselt number variation along both the hot and cool walls, where the Nusselt number is defined by Equation (49). Figure 39 presents the variation of the friction coefficient $C_{f_w}Re$ along both the hot and cool walls, which is defined by Equation (52). The results indicate that the flow reattaches to the hot wall at $x = 4.976$. The length of the secondary recirculation zone at the step corner was calculated to be approximately 0.1. The peak Nusselt number along the hot wall was 1.8 at $x = 5.084$. At the outlet, the hot wall Nusselt number was 0.7, while the cool wall Nusselt number was 0.3.

The mixed convection results with $Gr=1000$ are shown in figures 40 through 42, which present principal velocity, transverse velocity, and temperature profiles again at $x = 0, 3, 7, 15,$ and 30, respectively. Figure 43 presents the Nusselt number variation along both the hot and cool walls, while figure 44 presents the variation of the friction coefficient $C_{f_w}Re$

along both the hot and cool walls. In this case, the flow reattaches to the heated wall at x =*2.91*, and the length of the second recirculating zone at the step corner was approximately *0.6*. Another recirculation zone appeared along the cool wall starting at x = *3.985* and ending at x = *4.763* (length = *0.778*). The peak Nusselt number along the hot wall was *2.2* at x = *4.075*. At the outlet, the hot wall Nusselt number was *0.8*, while the cool wall Nusselt number was *0.3*.

Compared with the forced convection solution (*Gr*=0), the mixed convection results predicted a 71% reduction in the flow reattachment length, but a six fold increase in the length of the secondary recirculating zone occurring at the step corner. Additionally, a secondary recirculation zone along the cool wall appeared in the mixed convection solution, but not in the forced convection solution. Further differences included: a 10% larger outlet peak velocity located away from the channel center and near the hot wall, a 20% increase in the peak Nusselt number. Also, the mixed convection wall friction coefficient was much larger along the hot wall, but much smaller along the cool wall compared with the forced convection solution. Note that the reasonable agreement obtained between this solution and others (using various numerical schemes) can be seen in Reference 110.

The memory requirements for this problem are illustrated in Table 17. The memory required to store the Jacobian matrix and the ILU preconditioner is shown as a function of grid size and level of ILU fill-in. The table shows that even for the finest grid considered, the algorithm memory requirements are relatively modest, especially considering that it is a fully coupled solution algorithm.

Mesh sequencing was once again used to improve the efficiency of this inexact Newton-TFQMR algorithm Thus, the solutions presented above for the 140x40 grid were obtained using a 35x10, 70x20, and a 140x40 mesh sequence. The total required CPU time for the initial power-law solution was 1308.1 seconds on an IBM RISC/6000 model 320 workstation with 16 megabytes of main memory. On the final 140x40 grid, only 4 Newton

Table 17. Memory requirements of the inexact Newton algorithm versus grid size and level of ILU fill-in.

grid	Memory (MW)		
	ILU(0)	ILU(1)	ILU(2)
35x10	0.053	0.064	0.087
70x40	0.21	0.26	0.35
140x40	0.85	1.0	1.4

iterations were needed for convergence, with an average of 171 TFQMR iterations per Newton iteration using ILU(0) preconditioning. The use of ILU(1) preconditioning reduced the average TFQMR iterations to 112 but the required CPU time increased to 1412.8 seconds. This implies that ILU(0) preconditioning was sufficient for the 140x40 grid and so ILU(2) was not considered. However, for other grid sizes the benefits of increased fill-in may become significant, as was the case for the natural convection model problem. Note that the additional computer time needed to correct the initial solution was 2604.4 seconds, approximately twice the computational time needed to obtain the initial solution. This additional CPU cost was due to the 8 additional Newton iterations that were needed on the final 140x40 grid. Thus, the total required CPU time was 3912.5 seconds. Note that far fewer Newton iterations were required to correct the power-law solution on the 140x40 nonuniform grid, than were required to correct the previously considered upwind solution on the 140x40 uniform grid (8 iterations versus 21 iterations). This observation indicates that the power-law solution on the nonuniform grid was a significantly better approximation to the corresponding CUI solution than was the upwind solution.

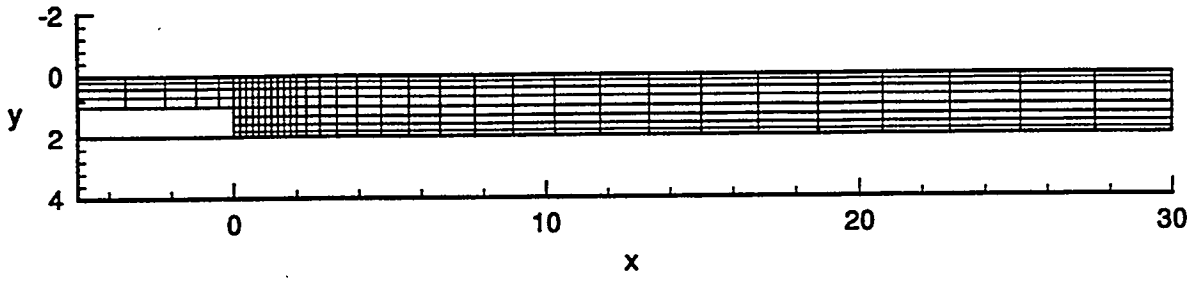


Figure 34. Schematic of coarse 35x10 nonuniform mesh.

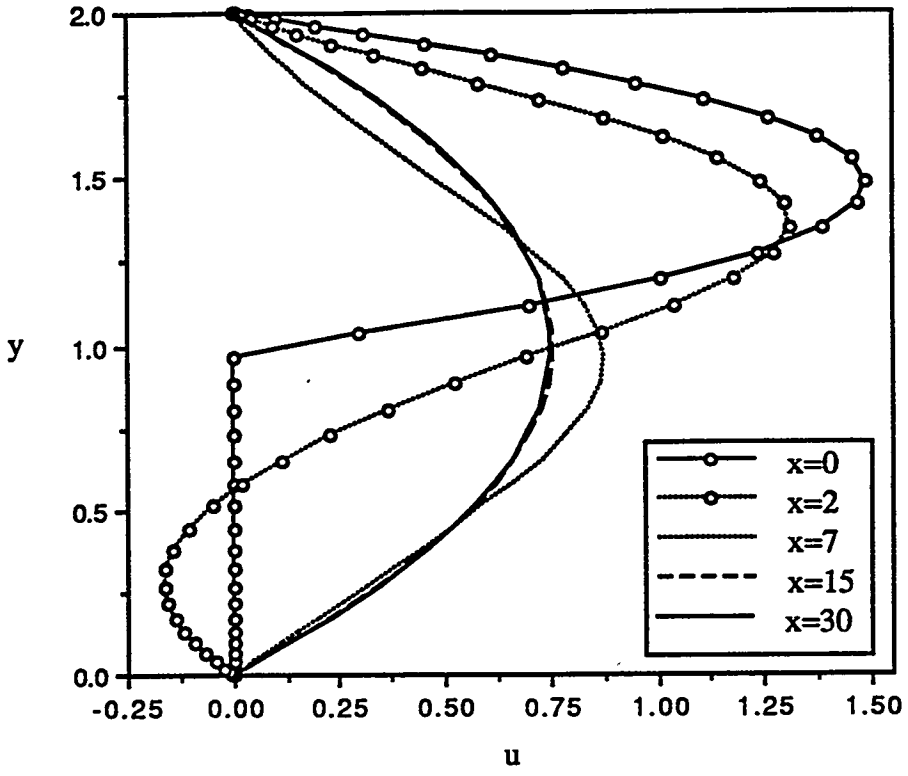


Figure 35. Steady state principal velocity profiles at $x = 0, 2, 7, 15$, and 30 ($Re = 100$, $Pec = 70$, $Gr = 0$).

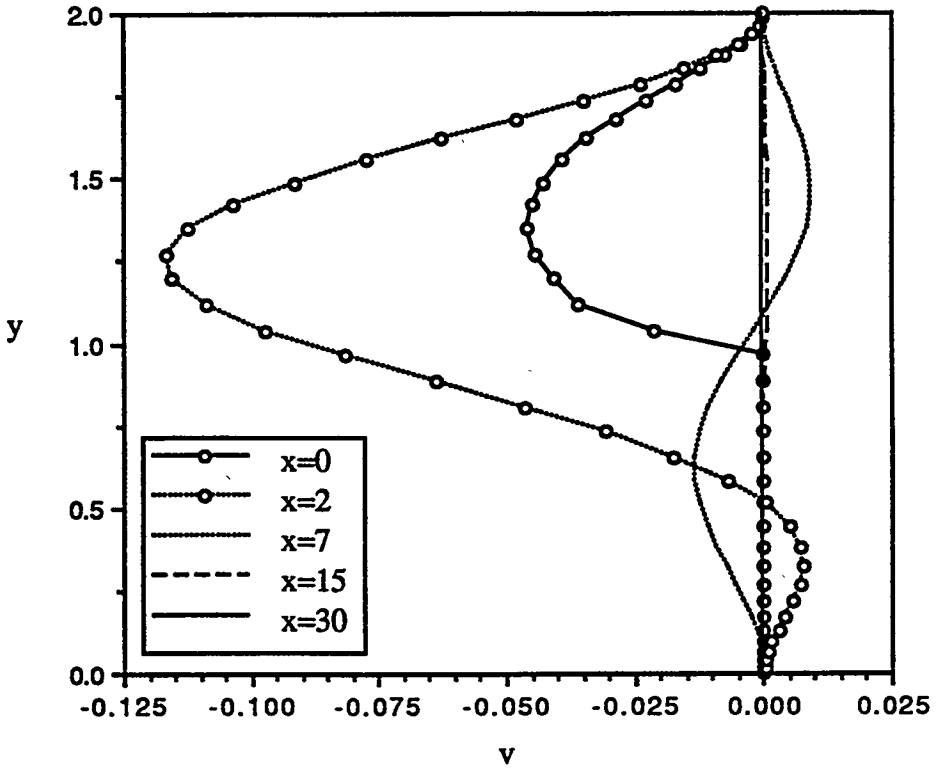


Figure 36. Steady state transverse velocity profiles at $x = 0, 2, 7, 15$, and 30 ($Re = 100$, $Pec = 70$, $Gr = 0$).

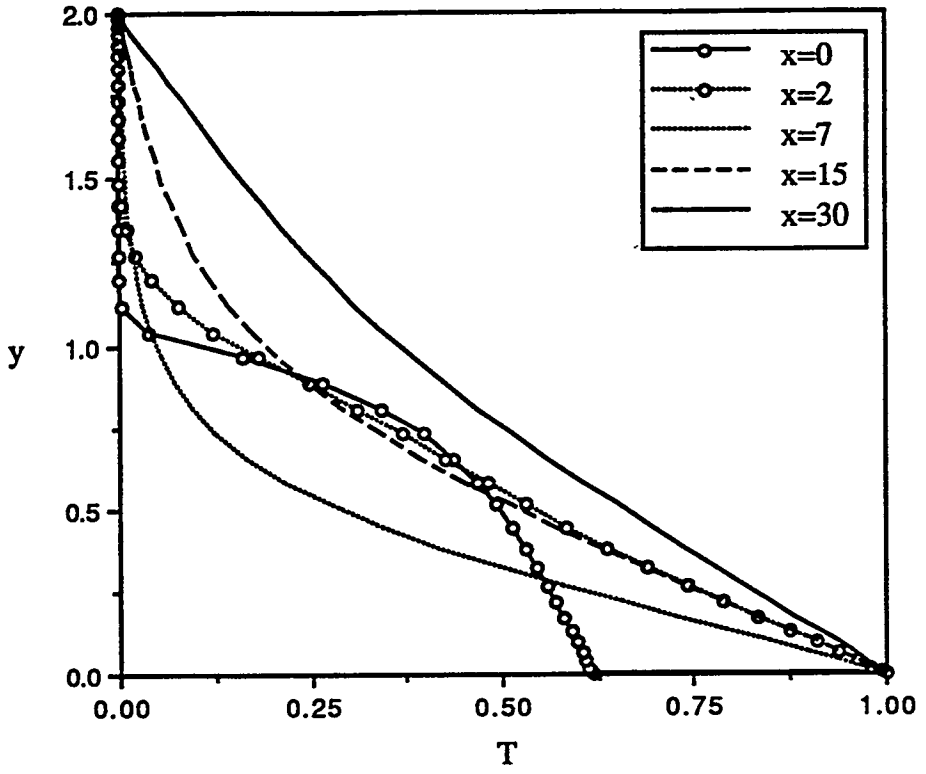


Figure 37. Steady state temperature profiles at $x = 0, 2, 7, 15$, and 30 ($Re = 100$, $Pec = 70$, $Gr = 0$).

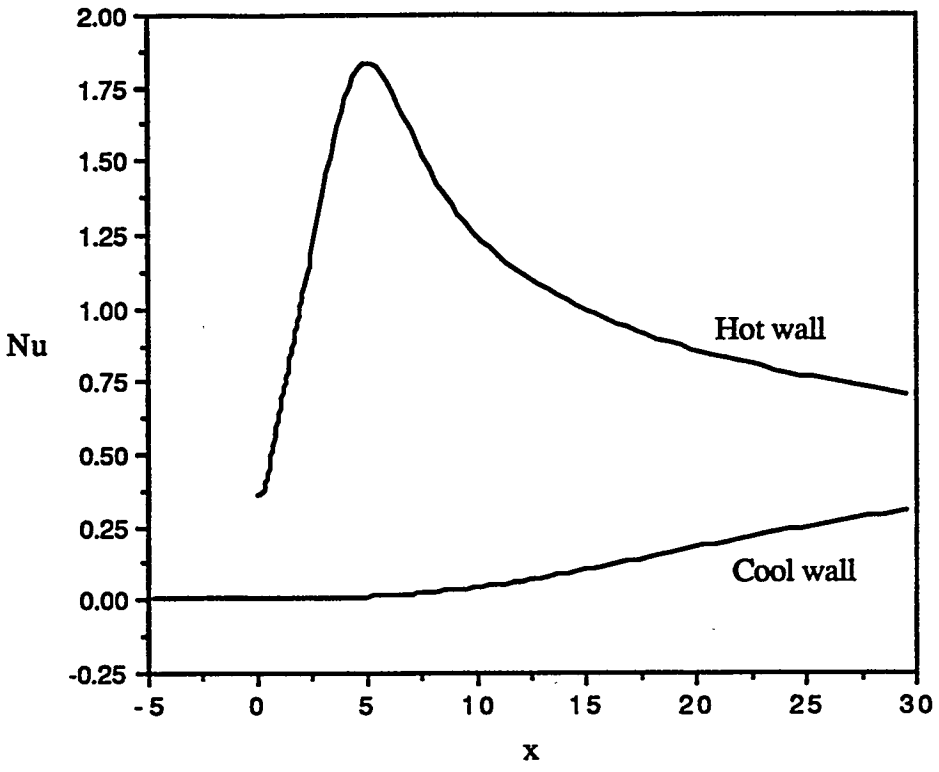


Figure 38. Steady state Nusselt number variation along both hot and cold walls ($Re = 100$, $Pec = 70$, $Gr = 0$).

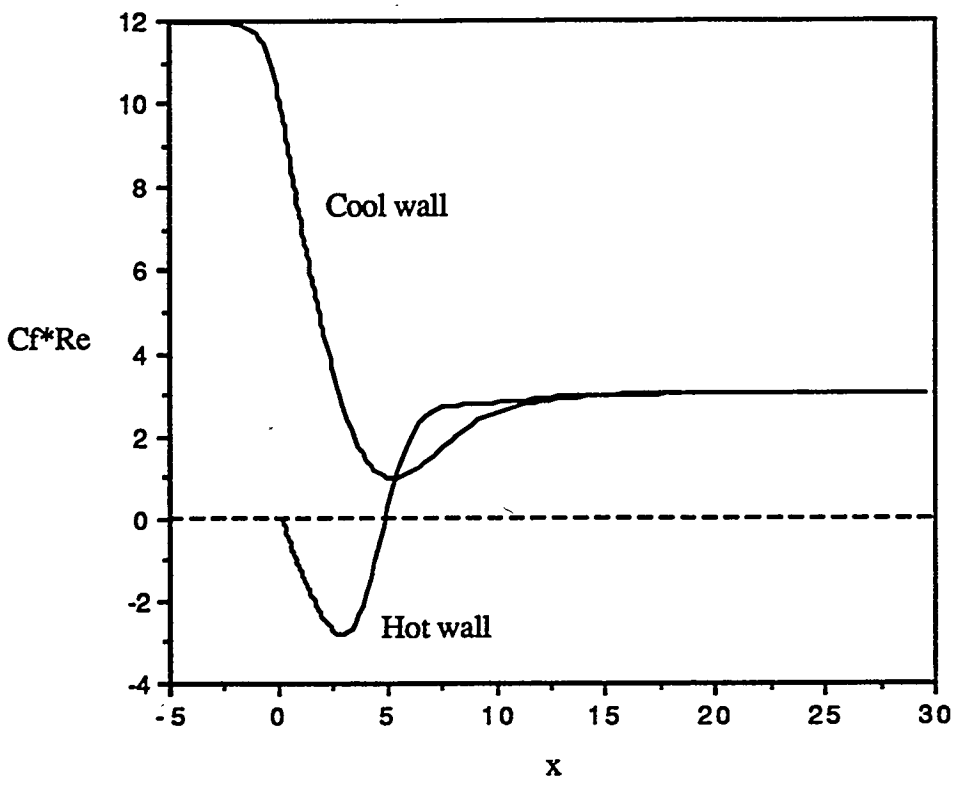


Figure 39. Steady state variation of the friction coefficient, $C_{f_w}Re$, along both hot and cold walls ($Re = 100$, $Pec = 70$, $Gr = 0$).

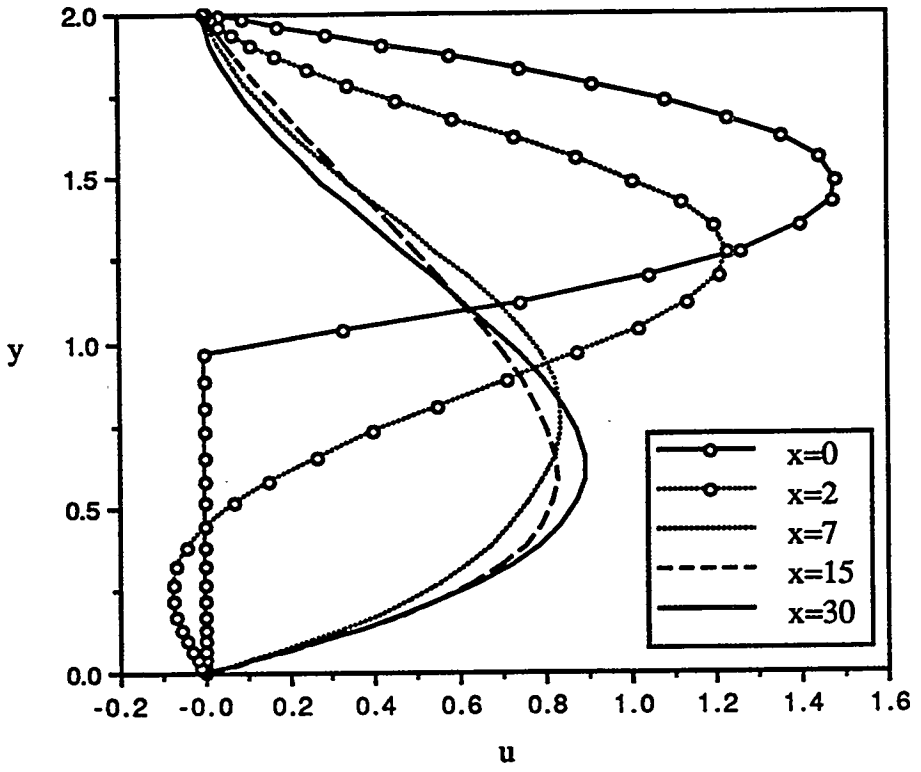


Figure 40. Steady state principal velocity profiles at $x = 0, 2, 7, 15$, and 30 ($Re = 100$, $Pec = 70$, $Gr = 1000$).

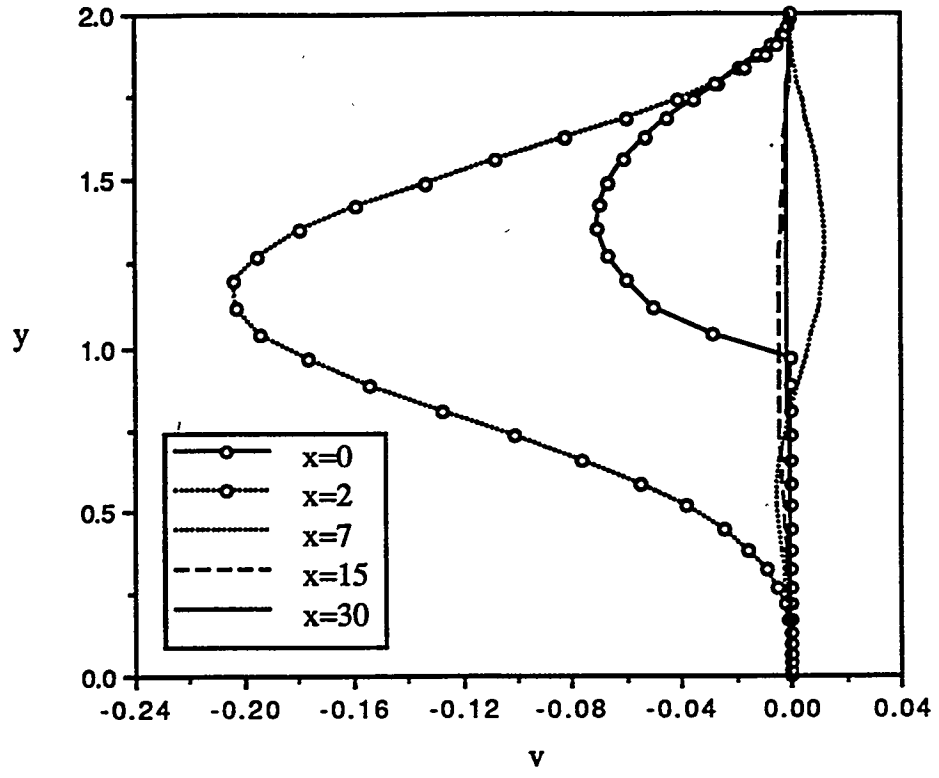


Figure 41. Steady state transverse velocity profiles at $x = 0, 2, 7, 15$, and 30 ($Re = 100$, $Pec = 70$, $Gr = 1000$).

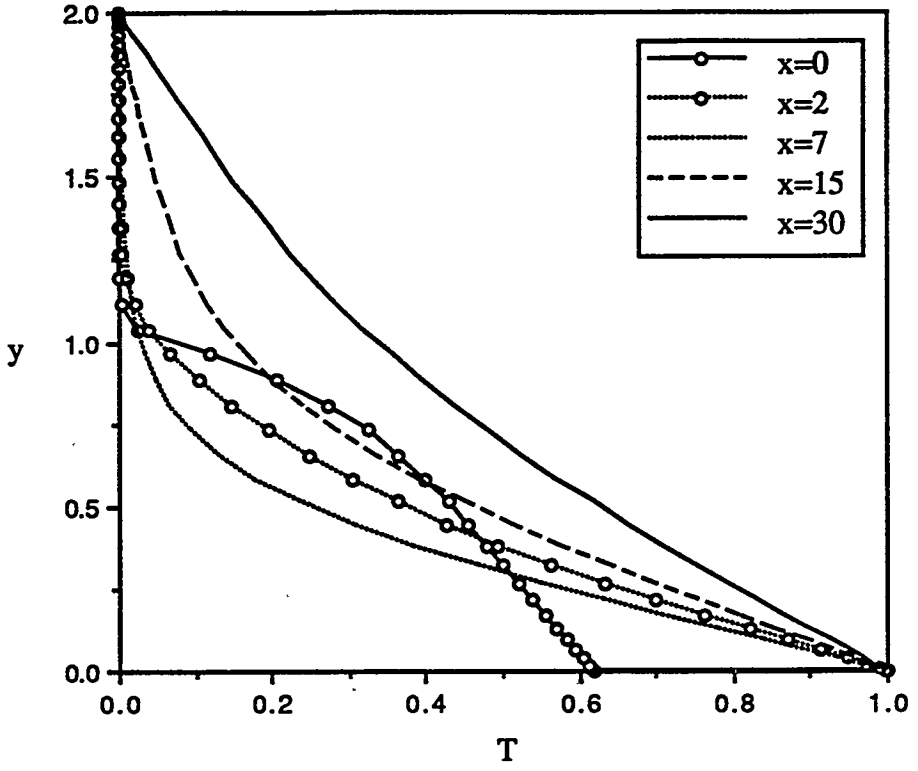


Figure 42. Steady state temperature profiles at $x = 0, 2, 7, 15$, and 30 ($Re = 100$, $Pec = 70$, $Gr = 1000$).

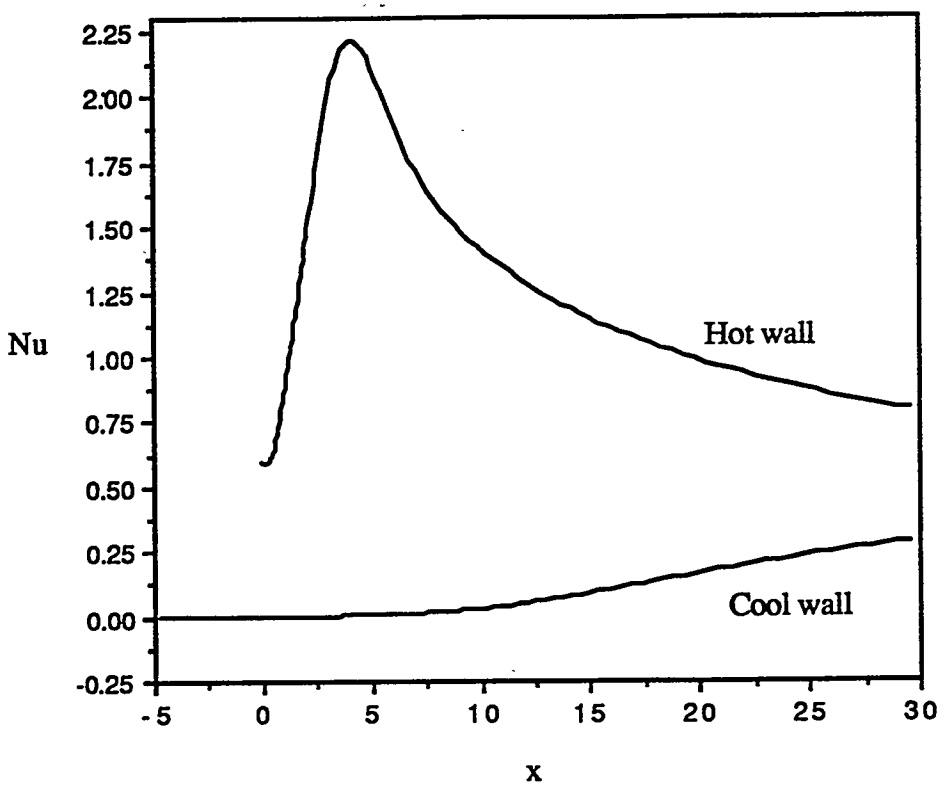


Figure 43. Steady state Nusselt number variation along both hot and cold walls ($Re = 100$, $Pec = 70$, $Gr = 1000$).

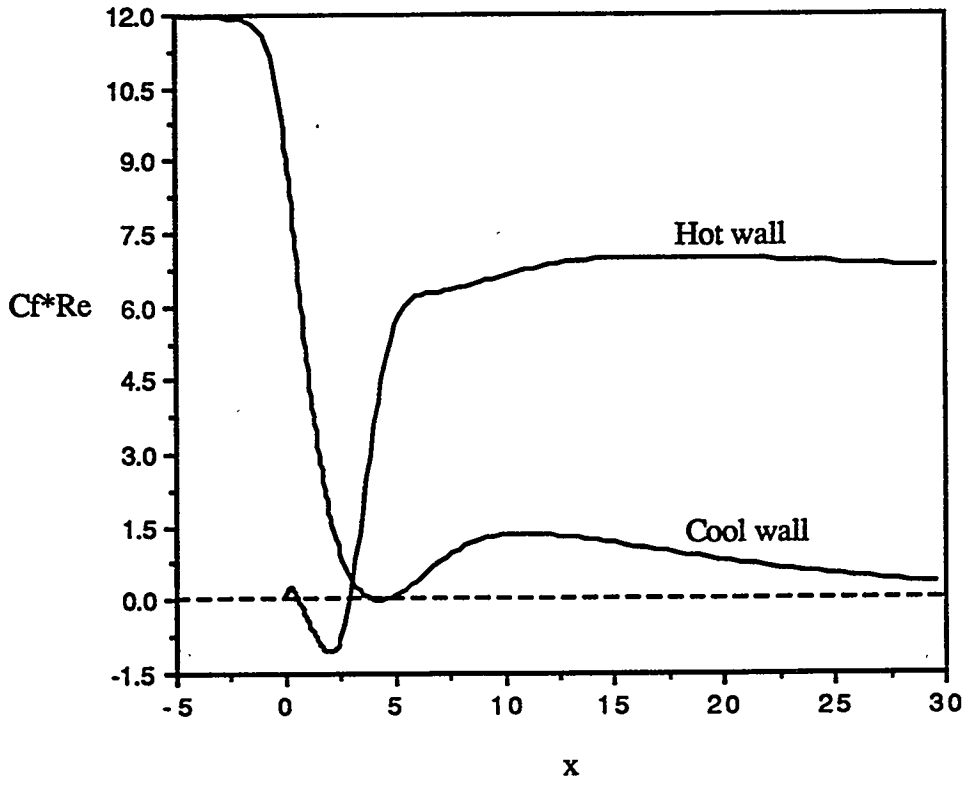


Figure 44. Steady state variation of the friction coefficient, $C_{f_w}Re$, along both hot and cold walls ($Re = 100$, $Pec = 70$, $Gr = 1000$).

4.1.4. Forced Convection, Backward Facing Step Model Problem

The forced convection, backward facing step model problem considered in this section was described previously in Section 2.1.5. In contrast to the previous backward facing step problem, this problem is characterized by a much higher Reynolds number (800). Consequently, it is reasonable to assume that the relative importance free convection effects are relatively small. Thus, for this benchmark problem, the Grashof number (Gr) is set to zero. The Prandtl number is again assumed to be 0.7 so that the Peclet number assumes a value of 560. Heat transfer boundary conditions also differ between this step problem and the previous one. Previously, isothermal upper and lower walls were assumed; whereas in this case a constant heat flux into the channel is assumed as described in Section 2.1.5.

The high aspect ratio associated with the physical geometry and the high Reynolds number combine to make this test problem especially challenging. The difficulty is manifested primarily in the preconditioned Krylov portion of the Newton-Krylov iteration, but nonlinear convergence troubles were also observed. Note that similar difficulties have also been experienced recently by Clift and Forsyth [65], with regard to the solution of the hydrodynamic portion of this problem. They attempted several numerical techniques to overcome these difficulties, including: a hybrid iteration scheme where a Picard-like iterative scheme is coupled with a full Newton iteration, several cell ordering strategies, and a use of what can be referred to as a pre-eliminated ILU(k) preconditioning with relatively high levels of fill-in (4-5). The required number of nonlinear iterations and inner iterations were found to be strongly dependent upon both grid size and Reynolds number [65]. Additionally, Gartling [113], who originally published solutions to the hydrodynamic portion of this problem, used parameter continuation to enable a solution at a Reynolds number of 800.

The problem considered here is further complicated by the addition of heat transfer effects. The numerical techniques employed in this section include: study of the

combination of the different cell ordering strategies discussed in Section 3.2.3.2 and different levels of fill-in for the ILU preconditioner used in this work. Additionally, the effects of using the discrete pressure equation, described in Section 2.1.2.2, are also considered. Note that the discrete pressure equation formulation employed here may, in some sense, yield a preconditioner that is similar to the one produced by the pre-eliminated ILU preconditioner of Reference 65. Additionally, the benefits of mesh sequencing (see Section 3.1.2), adaptive damping (see Section 3.1.3), and pseudo-transient relaxation (see Section 3.1.4) are also considered. Note that the calculations performed in this section were run on HP Model 735 workstations.

4.1.4.1. Cell Ordering Effects on ILU(k) Preconditioner Effectiveness

In this section, different cell ordering strategies are investigated with the goal of improving the effectiveness of the ILU(k) preconditioners for this test problem. The four different cell ordering strategies described in Section 3.2.3.2 are considered. However, these ordering schemes are defined somewhat differently for this model problem geometry. The different cell ordering schemes applied to this model problem are presented in Figures 45 through 48.

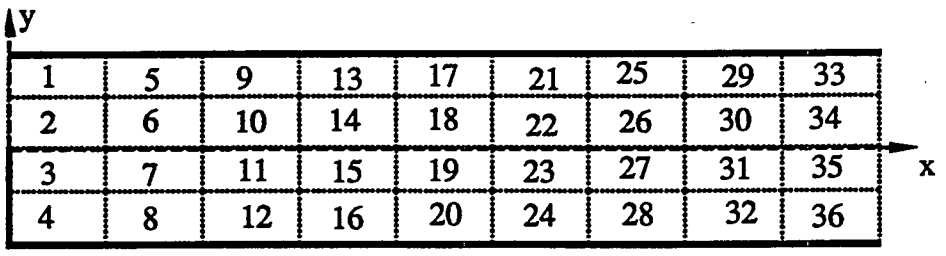


Figure 45. Schematic of the row ordering scheme as applied to the forced convection model problem.

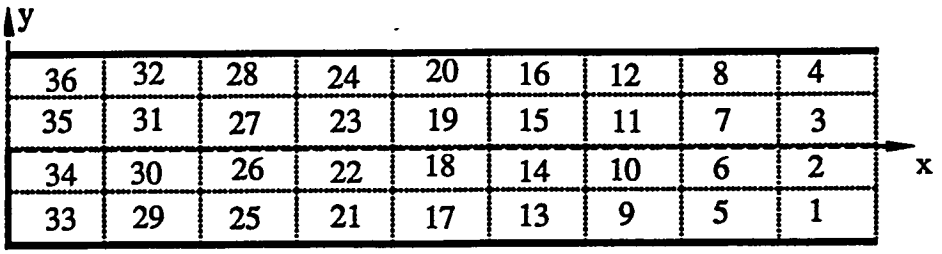


Figure 46. Schematic of the reverse row ordering scheme as applied to the forced convection model problem.

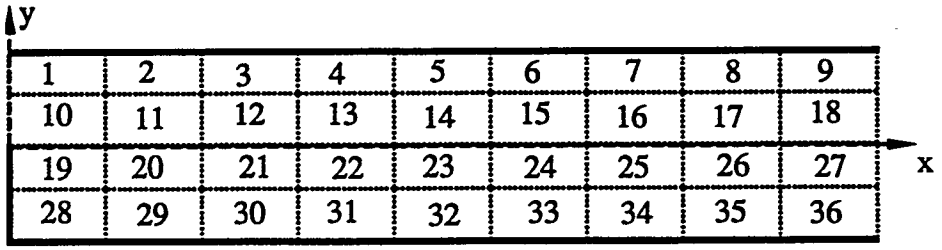


Figure 47. Schematic of the column ordering scheme as applied to the forced convection model problem.

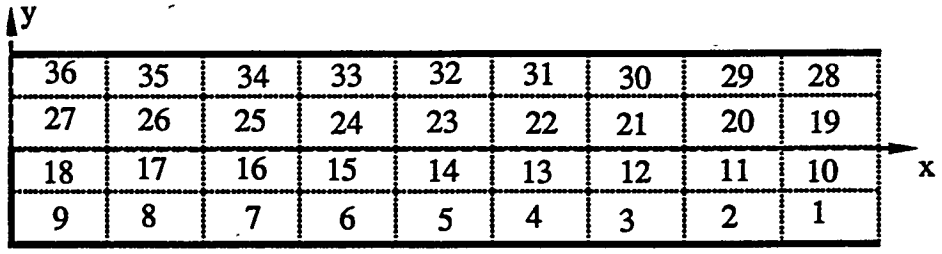


Figure 48. Schematic of the reverse column ordering scheme as applied to the forced convection model problem.

As with the previous backward facing step problem, pressure cannot be fixed to a constant value in the interior of the computational domain because it is already specified on the outflow boundary. Thus, if the row or column ordering schemes are adopted for this problem, zero elements appear on the main diagonal that are not filled in during the incomplete factorization process. Consequently, techniques such as Kershaw's method for treating unstable pivots (see Section 3.2.3.1), the use of alternative cell ordering schemes (see Section 3.2.3.2), and the discrete pressure equation formulation (see 2.1.2.2) must be used to remedy this difficulty. The use of the first two options is the topic of this section.

Table 18 shows performance data obtained for solutions to this forced convection model problem using various cell ordering strategies and four different fill-in levels with ILU preconditioning. The power-law discretization scheme was employed. The number of required nonzero diagonals for storing the preconditioner is indicated in parenthesis for each of the cell ordering strategies. A three mesh sequence is used to solve this problem. This sequence consists of a 30x8 grid, a 60x16 grid, and a 120x32 grid. The mesh spacing in the x-direction and the y-direction are both constant, but assume different values. Shown in this table for each grid solution are the total required number of Newton iterations (n), the average number of required TFQMR iterations per Newton iteration ($\overline{m}$), the number of times the inner iteration limit of 200 was encountered ($m_{\max}$ *hits*), whether or not Kershaw's method for treating unstable pivots was necessary, and the total required CPU time for the three mesh sequence. The symbol 'NS' implies that no solution was obtained within the allowed 50 Newton iterations, while the symbol 'DIV' implies that the algorithm diverged. A dashed line indicates the solution was not computed due to failure to converge on a previous grid. Algorithm divergence was often due to the appearance of a non-adjustable pivot in the ILU preconditioner following one or more encounters with the upper limit for the inner iterations. Note that the code terminates on this condition before a divide-by-zero can occur.

The results in Table 18 demonstrate that column ordering was a poor choice, while reverse row and reverse column ordering performed favorably when convergence was achieved. The reverse row and reverse column orderings were also successful in eliminating pivots that needed adjusting by Kershaw's method. Additionally, a marked reduction in inner iteration counts was observed when the level of ILU fill-in was increased from 0 to 1. However, the benefits of higher fill-in levels appeared to diminish beyond level 1, while the memory requirements continued to grow. Thus, the coarse grid information in Table 18 suggests that ILU(1) preconditioning is a good compromise between effectiveness and memory cost, especially if used with either reverse row or reverse column ordering.

Note from Table 18, that only two solutions were obtained for the 120x32 grid. This observation illustrates the computational challenge associated with this test problem. It is felt that the convergence difficulties on the 120x32 grid were caused by a combination of problems with both the Newton iteration and the Krylov iteration. The Newton iteration on this grid yielded some especially difficult linear systems that the preconditioned Krylov algorithms could not solve within the allowed 200 iterations. In most cases this behavior either led to very slow convergence or eventually caused algorithm divergence. In only two cases was the algorithm able to recover and eventually converge within 50 Newton iterations. It should be noted, however, that those two solutions did encounter the inner iteration limit during the course of the Newton iteration and so convergence may have been fortuitous. One possible cause of this undesirable behavior is that the secondary recirculation zone that exists along the upper wall is not yet resolved on the two coarse grids and so does not appear in these solutions. However, the 120x32 grid is fine enough to predict the appearance of this secondary recirculation zone. Consequently, the solutions yielded by the two coarser grids do not represent a good initial guess with respect to predicting the existence of this secondary recirculation zone. Thus, the quality of the initial guess is one likely reason for the lack of performance on the 120x32 grid.

Table 18. Effect of ordering and level of ILU fill-in on preconditioner effectiveness in solving the forced convection model problem.

ILU (k)	Order	30x8 Grid				60x16 Grid				120x32 Grid				CPU Time (sec)
		n	$\overline{m}$	m_{max} *hits*	Piv. ?	n	$\overline{m}$	m_{max} *hits*	Piv. ?	n	$\overline{m}$	m_{max} *hits*	Piv. ?	
0	row (19)	8	39	0	Yes	9	165	2	Yes	DIV	DIV	DIV	Yes	---
	rev. row (25)	8	24	0	No	9	74	0	No	16	170	2	No	2595
	col. (21)	8	20	0	Yes	9	98	0	Yes	15	198	13	Yes	2417
	rev. col. (23)	DIV	DIV	DIV	No	---	---	---	---	---	---	---	---	---
1	row (27)	8	30	0	Yes	9	133	0	Yes	NS	NS	NS	Yes	---
	rev. row (45)	9	9	0	No	9	27	0	No	DIV	DIV	DIV	No	---
	col. (33)	DIV	DIV	DIV	Yes	---	---	---	---	---	---	---	---	---
	rev. col. (39)	8	9	0	No	9	22	0	No	DIV	DIV	DIV	No	---
2	row (43)	9	22	0	Yes	9	85	0	Yes	NS	NS	NS	Yes	---
	rev. row (73)	8	0	0	No	9	26	0	No	DIV	DIV	DIV	No	---
	col. (53)	NS	NS	NS	Yes	---	---	---	---	---	---	---	---	---
	rev. col. (63)	8	5	0	No	9	19	0	No	DIV	DIV	DIV	No	---
3	row (67)	8	0	0	Yes	11	147	4	Yes	NS	NS	NS	Yes	---
	rev. row (115)	8	0	0	No	9	12	0	No	DIV	DIV	DIV	No	---
	col. (83)	DIV	DIV	DIV	Yes	---	---	---	---	---	---	---	---	---
	rev. col. (99)	8	4	0	No	9	44	0	No	DIV	DIV	DIV	No	---

Table 19. Effect of ordering and level of ILU fill-in on preconditioner effectiveness in solving the forced convection model problem with adaptive damping.

ILU (k)	Order	30x8 Grid				60x16 Grid				120x32 Grid				CPU Time (sec)
		n	$\overline{m}$	m_{max} *hits*	Piv. ?	n	$\overline{m}$	m_{max} *hits*	Piv. ?	n	$\overline{m}$	m_{max} *hits*	Piv. ?	
0	row (19)	34	40	0	Yes	11	165	5	Yes	NS	NS	NS	Yes	---
	rev. row (25)	34	26	0	No	11	74	0	No	20	170	4	No	3260
	col. (21)	34	21	0	Yes	11	98	0	Yes	22	200	22	Yes	3581
	rev. col. (23)	NS	NS	NS	No	---	---	---	---	---	---	---	---	---
1	row (27)	34	26	0	Yes	11	133	0	Yes	NS	NS	NS	Yes	---
	rev. row (45)	34	11	0	No	10	27	0	No	20	66	0	No	2343
	col. (33)	NS	NS	NS	Yes	---	---	---	---	---	---	---	---	---
	rev. col. (39)	34	9	0	No	10	22	0	No	20	57	0	No	1763
2	row (43)	34	20	0	Yes	11	99	0	Yes	NS	NS	NS	Yes	---
	rev. row (73)	34	0	0	No	10	28	0	No	20	59	0	No	3656
	col. (53)	34	NS	NS	Yes	---	---	---	---	---	---	---	---	---
	rev. col. (63)	NS	6	0	No	10	22	0	No	19	32	0	No	1980
3	row (67)	34	0	0	Yes	13	174	9	Yes	NS	NS	NS	Yes	---
	rev. row (115)	34	0	0	No	10	15	0	No	20	36	0	No	4495
	col. (83)	NS	NS	NS	Yes	---	---	---	---	---	---	---	---	---
	rev. col. (99)	34	4	0	No	10	48	0	No	19	98	4	No	7696

These observations motivated the use of other techniques to modify the Newton iteration on this grid. The first of these techniques was use of an adaptively damped Newton iteration. Table 19 presents analogous performance data for the same runs shown in Table 18, except that the damping strategy of Section 3.1.2 is activated. Damping enabled many more solutions on the 120x32 grid than were previously possible. Note especially that all of the solutions, using either reverse row or reverse column orderings with higher levels of ILU fill-in (k>0), converged within the allowed 50 iterations. Also note that all of the instances of algorithm divergence observed in Table 18 were avoided with the use of damping. The best combination of ordering and preconditioning in Table 19 was the use of reverse column ordering with ILU(1) preconditioning. This calculation resulted in a 27% speed-up compared with the best undamped calculation in Table 18.

In some cases, however, severe damping caused slow convergence. An alternative that appeared to remedy this difficulty was the switch from the power-law discretization scheme to the upwind differencing scheme in both the evaluation of the Jacobian and the residuals (right hand side). This switch (still using mesh sequencing) enabled the number of required Newton iterations on the 120x32 grid to be reduced from 20 to 10, for the case of reverse column ordering and ILU(1) preconditioning. The average inner TFQMR iterations was reduced from 57 to 47. The corresponding CPU time was reduced by a factor of 2.2 to 798.4 seconds. In order to further investigate the effects of grid scaling upon algorithm performance, this 120x32 upwind solution was then interpolated up to a 240x32 grid. This finer grid restart calculation required 8 Newton iterations, an average of 119 TFQMR iterations per Newton iteration, and 3217.5 CPU seconds. In comparing this solution with that of Gartling [113], it was deemed that additional grid refinement was necessary in order to obtain quantitative agreement. Finer grid solutions (960x64 grid) are presented in Section 4.1.4.4.

4.1.4.2. Discrete Pressure Equation Formulation

The discrete pressure equation formulation described in Section 2.1.2.2 is another remedy for avoiding zero elements on the main diagonal of the Jacobian matrix caused by the lack of pressure explicitly appearing in the continuity equation. Table 20 presents performance data using this discrete pressure equation formulation (using the power-law discretization scheme) with various ordering strategies and ILU(k) preconditioners. The same data presented in Tables 18 and 19 are included in Table 20 for comparison purposes. Note that use of the pressure equation proved effective in avoiding pivot adjustments during the incomplete factorization process used to generate the preconditioner. This observation was true for all the cell ordering strategies considered. The price paid for this benefit, however, was additional computer memory (as reflected in the larger numbers of nonzero diagonals requiring storage) and somewhat more complex Jacobian and residual evaluations due to the enlarged finite volume stencil associated with the discrete pressure equation. Based upon the two coarser grid solutions, it appears that ILU(1) with column ordering represents a good compromise between preconditioner effectiveness, memory requirements, and CPU cost.

Table 20. Effect of ordering and level of ILU fill-in on preconditioner effectiveness in solving the forced convection model problem using the discrete pressure equation formulation.

ILU (k)	Order	30x8 Grid				60x16 Grid				120x32 Grid				CPU Time (sec)
		n	$\overline{m}$	m_{max} *hits*	Piv. ?	n	$\overline{m}$	m_{max} *hits*	Piv. ?	n	$\overline{m}$	m_{max} *hits*	Piv. ?	
0	row (28)	11	17	0	No	10	64	0	No	DIV	DIV	DIV	No	---
	rev. row (32)	12	42	0	No	DIV	DIV	DIV	No	---	---	---	---	---
	col. (30)	12	18	0	No	10	66	0	No	DIV	DIV	DIV	No	---
	rev. col. (32)	15	131	6	No	0	163	1	No	29	198	28	No	6763
1	row (48)	12	10	0	No	10	35	0	No	DIV	DIV	DIV	No	---
	rev. row (66)	11	7	0	No	9	25	0	No	DIV	DIV	DIV	No	---
	col. (54)	11	4	0	No	10	12	0	No	DIV	DIV	DIV	No	---
	rev. col. (59)	11	7	0	No	10	25	0	No	DIV	DIV	DIV	No	---
2	row (85)	DIV	DIV	DIV	No	---	---	---	---	---	---	---	---	---
	rev. row (121)	10	0	0	No	10	19	0	No	DIV	DIV	DIV	No	---
	col. (99)	11	2	0	No	11	7	0	No	DIV	DIV	DIV	No	---
	rev. col. (108)	12	5	0	No	10	82	2	No	DIV	DIV	DIV	No	---
3	row (147)	10	0	0	No	10	72	1	No	DIV	DIV	DIV	No	---
	rev. row (213)	10	0	0	No	11	2	0	No	DIV	DIV	DIV	No	---
	col. (173)	10	1	0	No	11	4	0	No	DIV	DIV	DIV	No	---
	rev. col. (190)	12	3	0	No	12	57	2	No	DIV	DIV	DIV	No	---

A discouraging aspect of these results is the appearance of the same convergence problems on the 120x32 grid that were experienced previously. Consequently, adaptive damping was once again activated in attempt to overcome these convergence difficulties. The performance data using damping with ILU(1) preconditioning is shown in Table 21. In this case, a damped Newton iteration avoided the algorithm divergences that appeared in Table 20, but convergence was slowed to the point that no calculation converged within the allowed 50 iterations. This trend again suggested the switch to upwind differencing in place of the power-law discretization scheme. This switch proved effective again in that convergence on the 120x32 grid using mesh sequencing was enabled after just 10 Newton iterations using column ordering. The required inner TFQMR iterations per Newton step was 19, and the total required CPU time was only 760.8 seconds. This solution was again interpolated up to a 240x32 grid in order to further investigate performance on a finer mesh. This restarted calculation required 9 Newton iterations, an average of 44 inner TFQMR iterations per Newton step, and 2191.1 CPU seconds. Thus, the 240x32 grid solution obtained using the discrete pressure equation formulation was 32% faster than the previous solution obtained using the continuity equation. This improved efficiency motivated the use of the discrete pressure equation formulation, upwind discretization, ILU(1) preconditioning, and column ordering for all subsequent calculations.

Table 21. Performance data obtained using an adaptively damped Newton iteration with the discrete pressure equation formulation and ILU(1) preconditioning.

Order	30x8 Grid				60x16 Grid				120x32 Grid			
	n	$\overline{m}$	m_{max} *hits*	Piv. ?	n	$\overline{m}$	m_{max} *hits*	Piv. ?	n	$\overline{m}$	m_{max} *hits*	Piv. ?
row (48)	45	16	0	No	11	40	0	No	NS	NS	NS	No
rev. row (66)	45	10	0	No	11	31	0	No	NS	NS	NS	No
col. (54)	45	5	0	No	11	11	0	No	NS	NS	NS	No
rev. col. (59)	45	9	6	No	11	25	0	No	NS	NS	NS	No

4.1.4.3. Pseudo-Transient Calculations

The pseudo-transient calculation technique discussed in Section 3.1.4 is another technique that helps to improve the robustness of a Newton-type algorithm. This technique is especially useful if mesh sequencing and adaptive damping prove to be either ineffective or unreasonable. For example, if one has a solution on a fine grid and only wants to perturb a flow parameter in order to obtain a different solution on the same grid, mesh sequencing may not be warranted. Furthermore, mesh sequencing is a valuable tool only if the coarse grid solution is a good approximation to the fine grid solution. Additionally, a damping strategy based only on temperature changes (such as that employed here) may not prove efficient in certain instances. In these situations, a pseudo-transient calculation can be a useful tool to improve the convergence behavior of the Newton-Krylov algorithm.

Consider the 120x32 grid solution to the forced convection flow model problem of interest in this section. Recall from previous discussions that convergence difficulties were experienced on this grid unless the adaptive damping strategy was activated, even with the use of mesh sequencing. It is of interest to investigate the use the pseudo-transient calculation technique to overcome these convergence difficulties instead of the use of adaptive damping. Consequently, a pseudo-transient calculation was initiated on this 120x32 grid starting from a flat initial guess (u=1, v=p=0, T=0.5) using the discrete pressure equation formulation, upwind discretization, ILU(1) preconditioning, and column ordering. In this case the GMRES(20) algorithm was selected as the Krylov solver in order to enable consideration of efficient matrix-free implementations. The time step was initialized to a small value ($4.2\text{x}10^{-3}$) near the explicit stability limit and then allowed to vary according to Equation (126). The standard (no matrix-free) pseudo-transient calculation required 86 Newton iterations with an average of 6 GMRES(20) iterations per Newton step. The total required CPU time was 2540.1 seconds. The final time step size was on the order of $1\text{x}10^{6}$.

Note that the CPU time required for this calculation is more than 3 times larger than the previous calculation using mesh sequencing and adaptive damping. However, this calculation offered the advantage of starting from the fine grid and did not require a damped Newton iteration.

One technique that is commonly employed to improve the efficiency of a Newton iteration is the use of a simplified or modified Newton where the Jacobian is frozen for one or more iterations. The underlying idea is that the number expensive Jacobian evaluations and, in the case of a direct-Newton iteration, expensive factorizations can be reduced. In the case of a Newton-Krylov algorithm the factorization expense is relevant only with regard to the formation of the preconditioner, but the Jacobian evaluation expense still applies directly. Consequently, the idea of evaluating the Jacobian and preconditioner only every tenth pseudo-transient Newton iteration was considered (i.e., $d = 10$). This calculation required 242 pseudo-transient Newton iterations with an average of 13 inner GMRES(20) iterations on each outer iteration. The total required CPU time increased from the previous value of 2540.1 seconds to 2802.0 seconds. Thus, this technique did not prove effective because the reduced number of Jacobian and preconditioner evaluations was outweighed by the slower convergence caused by the lagged Jacobian.

A remedy for this convergence degradation, however, is the use of the matrix-free implementation. This implementation enables the effects of the true Jacobian to be captured in the Jacobian-vector products of the inner GMRES(20) iteration without explicitly forming the Jacobian. Thus, the Jacobian and preconditioner can be frozen without suffering the significant slow down in convergence observed previously. The main consequence in this approach is that the preconditioner is still lagged, and so the average inner iterations per Newton step may rise compared to the situation when the preconditioner is updated on each pseudo-transient Newton iteration (e.g., 13 versus 6). The significance of this consequence is that the number of function evaluations [Equation (95)] is now directly tied to the average

number of inner iterations because of the use of Equation (165). The goal of this approach is to keep the number of inner iterations low, while still lagging the Jacobian and preconditioner. In this manner, the cost of these evaluations are amortized over many pseudo-transient Newton iterations. The benefits of this amortization, however, will be significant only if the total number of function evaluations are reduced. As an example, the previous calculation was re-run using the matrix-free implementation. In this case, the total number of required pseudo-transient Newton iterations was 84, the average inner GMRES(20) iterations per outer iteration was 12, and the total CPU time was 2382.5 seconds. Recall from Table 1 in Section 3.2.4 that the "break even" value for the average inner iteration count ($\overline{m}_{be}$) for this case was 12.6. Since the average number of inner iterations ($\overline{m}$) was only 5% less than $\overline{m}_{be}$, only a 6% reduction in total CPU time was observed. In this case, the CPU time reduction was also partly due to the fact that two fewer pseudo-transient Newton steps were required. Note that this savings could likely be improved if additional effort was expended to determine the optimal Jacobian evaluation frequency, and to effeciently lower $\overline{m}$ through the use of more effective preconditioning or the use of smaller time steps. For completeness, note that this approach is extremely attractive in other applications where the Jacobian evaluation is the dominant portion of the overall calculation [e.g., see 15]. These applications are typically characterized by a larger system of strongly coupled governing equations (i.e., multi-species flow) that requires many function evaluations to explicitly form the Jacobian matrix. As a result, $\overline{m}_{be}$ is typically much higher than the value of 12.6 encountered for the model problem of this section. In these situations, the use of the matrix-free implementation within a pseudo-transient calculation may enable a significant reduction in the total required number of function evaluations, thereby improving CPU efficiency [e.g., see 15].

4.1.4.4. Solutions

Solutions to the forced convection model problem obtained using a 960x64 grid are presented in this section. This fine grid solution was obtained in three stages starting from the 240x32 grid solution mentioned previously. At each stage the hydrodynamic solution was compared with the results of Gartling [113] in order to determine if additional grid refinement was necessary. The discrete pressure equations formulation was employed in all cases using upwind discretization, ILU(1) preconditioning, column ordering, and adaptive damping. The steady state equations were solved directly without any pseudo-time marching. First, the 240x32 grid solution was interpolated up to a 480x32 grid. Convergence on that grid required 7 Newton iterations, an average of 103 TFQMR iterations per Newton step, and 6873.1 CPU seconds (using a slightly faster HP Model 735 workstation than previously employed). Next, this solution was interpolated up to a 960x32 grid; which required 9 Newton iterations, an average of 143 TFQMR iterations per Newton step, and 24,279 CPU seconds (6.7 hours). The final stage of the calculation refined the mesh in the transverse direction so that convergence was obtained on a 960x64 grid. This calculation required only 6 Newton iterations, but an average of 335 TFQMR iterations per Newton step (the maximum inner iteration limit was increased to 500 for this single run). The CPU time required to obtain this solution was 22.5 hours. Note that all figures associated with this latter solution (as discussed below) are accumulated at the end of this subsection.

The stream function and temperature contours for this 960x64 grid solution are shown in Figures 49 and 50, respectively. Both of these contour plots were truncated at x=10 for clarity and because this is the region where most of the interesting flow phenomena occur. The stream function contours show that the flow detaches at the step causing a primary recirculation zone to form at the step corner. The flow reattaches to the lower wall at approximately x=5.6. Note that Gartling's solution predicted this reattachment length at

x=6.1 [113] (approximately an 8% difference). This figure also shows the secondary recirculation zone that appears along the upper wall. Downstream from the end of this figure, the flow reattaches and begins to resemble a fully developed channel flow. The temperature contours in Figure 50 illustrate the high temperature regions that occur in the recirculating zone at the step corner and, to a lesser degree, the recirculating zone along the upper wall. Also visible is the lower temperature core flow surrounded by higher temperature fluid near the walls. These contours also show the rise in the temperature of the fluid as it moves downstream from the step.

The principal u-velocity profiles at x=7 and x=15 are shown in Figures 51 and 52, respectively. The secondary recirculation zone appearing along the upper wall is apparent in the velocity profiles shown in Figure 51, as is the shift in the location of the peak velocity towards the lower wall. Reasonable agreement with Gartling's solution is obtained (approximately 8.5% difference in peak velocities), especially considering that the position of the secondary recirculation zone occurs slightly further upstream in the current solution compared with Gartling's solution. Figure 52 shows that at x=15, the flow begins to resemble a fully developed profile. Again, reasonable agreement is obtained with Gartling's solution (less than 6% difference in peak velocity).

The heat transfer aspects of this solution are reflected in the Nusselt number data of Figure 53 and the temperature data of Figure 54. The Nusselt number along both the upper and lower walls are shown in Figure 53. Along the upper wall, the Nusselt number is initially high because of the higher velocities associated with the narrower inlet channel. Its value quickly drops until a minimum is reached in the region of the secondary recirculation zone along the upper wall. Further downstream, the upper wall Nusselt number approaches 8.235, the value expected for fully developed channel flow with a uniform wall heat flux [see 2 and 111]. The lower wall Nusselt number is low in the region of the primary recirculation zone at the step corner, but then reaches a peak value in the region of the secondary

recirculation zone, where the peak fluid velocity is located nearer to the lower wall. Again, further downstream the lower wall Nusselt number approaches the expected value for a fully developed channel flow with a uniform heat flux. Figure 54 shows that bulk fluid temperature increases monotonically because of the uniform heat flux into the channel. The upper wall temperature rises in the secondary recirculation zone, while the lower wall temperature is especially high at the step corner because of the primary recirculation zone.

The agreement between the current solution and the hydrodynamic solution of Gartling improved each time the grid was refined. On the 960x64 grid, the reattachment point data and peak velocities at x=7 and x=15 differed by less than 10%. This agreement is reasonable considering that Gartling used an adapted finite element mesh with a second order accurate discretization scheme and very fine meshes. However, in retrospect, it is not clear whether the additional accuracy obtained on the finest grid (compared with the coarser mesh solutions) really warranted the additional CPU cost. The use of the defect correction procedure discussed previously in order to obtain a higher order accurate solution on a coarser grid may have been a better alternative in this case. On the other hand, the 960x64 grid solution (245,760 unknowns) demonstrates the viability of Newton-Krylov solution algorithms for solving large problems.

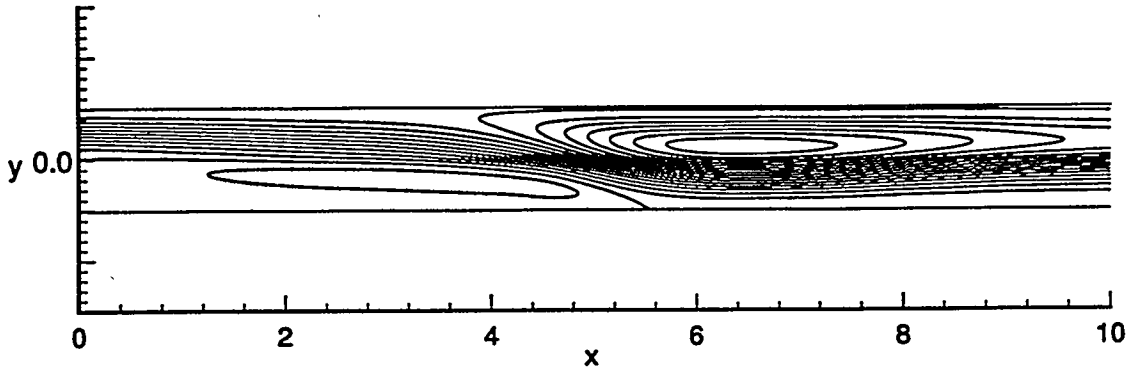


Figure 49. Stream function contours [-0.298692 (0.025) 0.201308] from the 960x64 grid solution to the forced convection model problem. The stream function is set to zero along the upper wall.

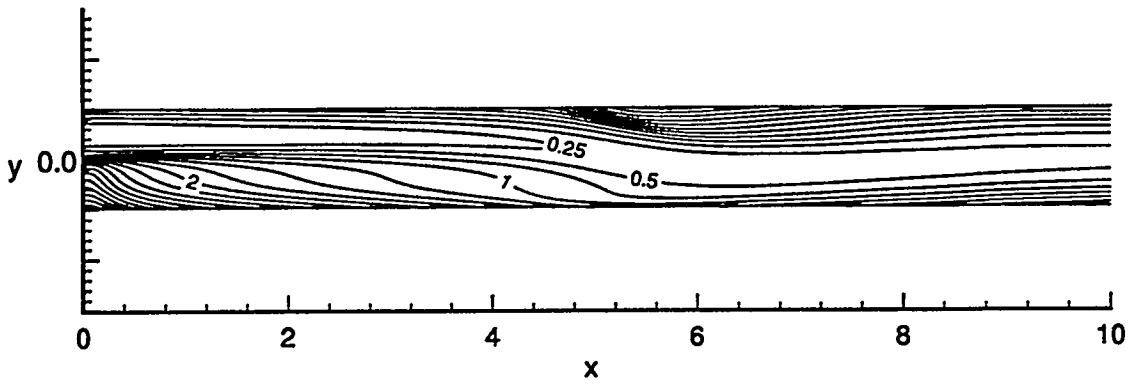


Figure 50. Temperature contours [0 (0.25) 5] from the 960x64 grid solution to the forced convection model problem.

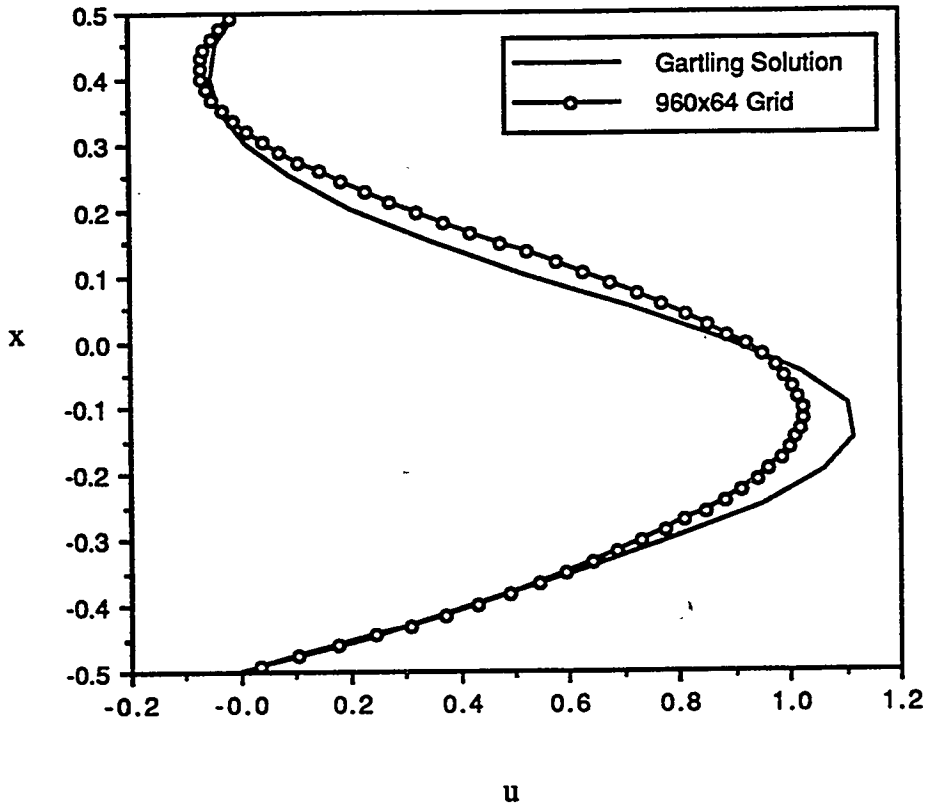


Figure 51. Principal velocity (u) profile at x=7 from 960x64 grid solution to the forced convection model problem.

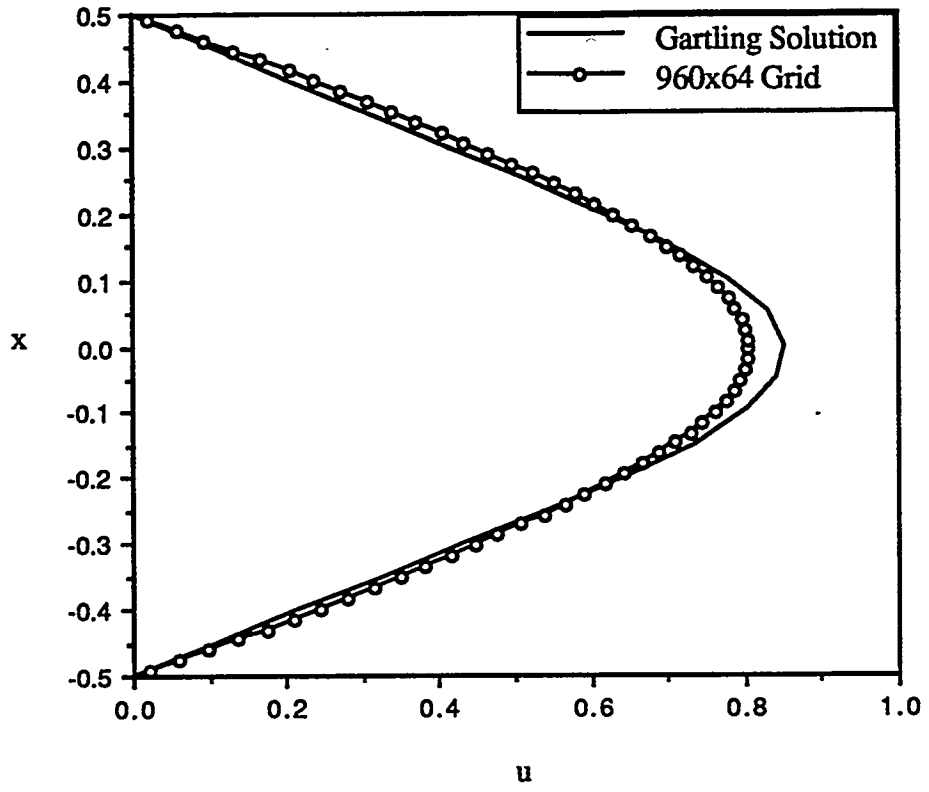


Figure 52. Principal velocity (u) profile at x=15 from 960x64 grid solution to the forced convection model problem.

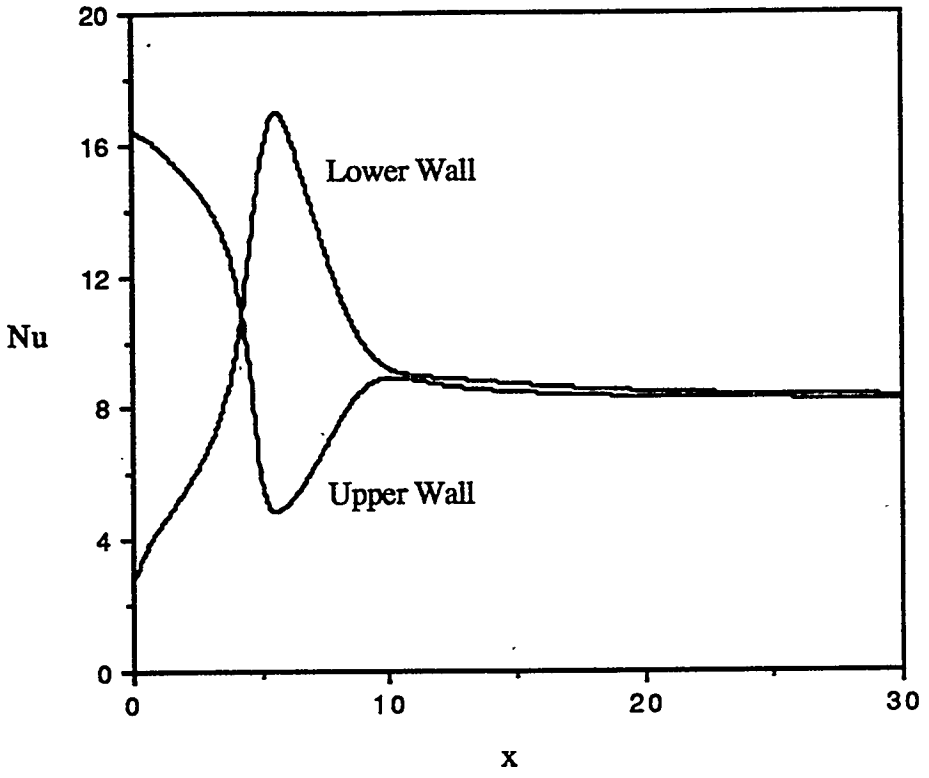


Figure 53. Axial Nusselt number variation along both the upper and lower walls in the case of the forced convection model problem.

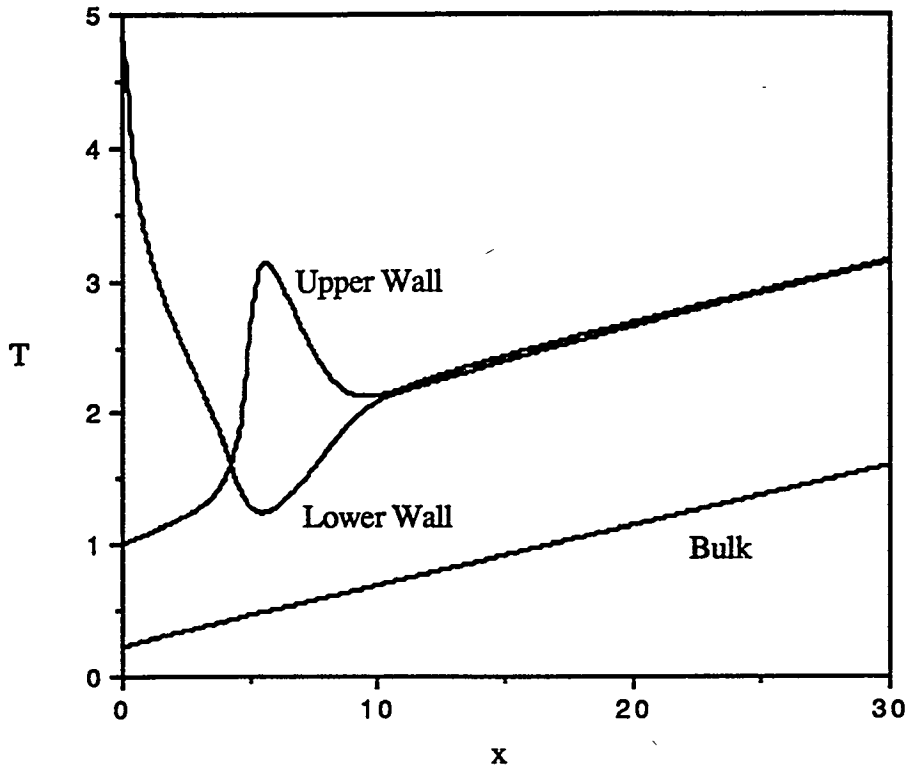


Figure 54. Axial upper and lower wall and bulk temperature variation in the case of the forced convection model problem.

4.2. COMPRESSIBLE FLOW

The objective of this section is to demonstrate the effectiveness of inexact Newton-Krylov solution techniques for steady state calculations of low Mach number compressible flow problems. Generally, a flow is considered incompressible if the Mach number is approximately less than 0.3 [8]. However, solution of the compressible flow equations in this Mach number regime is still important in instances where: the low Mach number region is imbedded within a high speed flow, and other flow situations where density variations are important, i.e., chemically reacting flow and flow with significant heat transfer. Thus, the purpose of this section is to investigate the efficiency of the numerical techniques described in Chapter 3 when confronted with these low Mach number flow regimes.

Newton-Krylov algorithms and finite volume discretization are used to solve the steady, compressible Navier-Stokes and energy equations that were presented in Section 2.2.1. The main problem of interest is the problem of low Mach number flow past a backward facing step that was described in Section 2.2.3. Solutions to this problem are used to investigate performance differences among the various Krylov algorithms considered and the to determine effective preconditioning strategies for these Krylov algorithms at low Mach numbers.

4.2.1. Important Computational Issues

Mathematically, the time dependent compressible Navier-Stokes system of equations becomes very stiff (i.e., a wide disparity in the system matrix eigenvalues) at low values of the reference Mach number [see 163]. This stiffness is observed in the form of a wide disparity in the time scales associated with convection and those associated with acoustic waves. Consequently, explicit numerical schemes must honor the more restrictive Courant

limit associated with the acoustic speed [164]. Thus, use of an explicit time marching scheme for steady state calculations becomes inefficient at low Mach numbers. An alternative is the use of implicit schemes that avoid this overly restrictive stability limit. However, many implicit schemes are based on some sort of approximate factorization technique [36, 61]. Typically, errors associated with the approximate factorization restrict the Courant-Fredricks-Lewy (CFL) [see 19] number to a certain value above which convergence slows. This optimal CFL number for low Mach number flows is elusive because the wide disparity in system eigenvalues causes a wide disparity in optimal CFL numbers [see 163, 165, 166, 167, 168]. As a result, the CFL number is often restricted to small values at low Mach numbers. The use of a fully coupled, simultaneous solution technique, without use of any approximate factorizations, introduces no such errors, and so its convergence is not strongly dependent upon the CFL number. However, the problem persists to some extent, in that the ill-conditioned linear systems that arise on each iteration may be difficult to solve, especially using iterative techniques. The source of this difficulty is the pressure dependence in the momentum equations. Pressure is not taken to be one of the dependent variables in this work, and so it is replaced with Equation (67), the state equation. The result of this substitution is that large off-diagonal terms appear in the Jacobian matrix on rows corresponding to the momentum equations. These terms are inversely proportional to the Mach number squared, whereas the other terms (if properly scaled) are of order unity. This situation leads to poorly conditioned Jacobian matrices that are difficult to solve iteratively, making selection of an appropriate preconditioner an important task. This selection is given considerable attention in Section 4.2.2.2 below. Additionally, the use of the pseudo-transient relaxation technique described in Section 3.1.4 can also be an effective way to alleviate problems associated with iteratively solving these ill-conditioned linear systems.

For completeness, it is important to mention other techniques for dealing with the disparate time scales associated with low Mach number flow. The first of these are perturbation methods, in which flow variables are expanded in terms of a small parameter, typically the Mach number or its square [see 167, 168]. The problem with these methods is that they are only valid at low Mach numbers and not at moderate or high Mach numbers. Other techniques, which are valid for a wide range of Mach numbers, are referred to as "preconditioning" methods [see 163, 164, 166, 165, 168, 169]. These methods should not be confused with the preconditioning discussion contained in Section 3.2.3. Typically these methods represent some sort of generalization of the artificial compressibility method to compressible flows, with the goal of reducing the acoustic speed to levels comparable to the fluid speed in order to overcome the stiffness problems mentioned previously [164]. The basic technique is to modify certain time terms in such a way that the alterations vanish at convergence [see 163, 164, 166, 165, 168, 169]. The reason for the term "preconditioning" is because the modified time term can often be expressed as a new matrix multiplying the original time term [163]. Because this investigation is primarily interested in direct steady state calculations (i.e., not time marching), these techniques for "preconditioning" certain troublesome time terms have not been considered. Although, it is noted that these techniques may be useful within the context of transient and pseudo-transient calculations.

Another troublesome issue that arises for low Mach number flows is associated with the absolute value of pressure. The magnitude of the pressure scales proportionately with the inverse of the square of the Mach number according to Equation (67). Thus, while other flow variables are typically of order unity, the pressure can become quite large [163, 164, 168]. The difficulty is then manifested in round-off errors that occur when differences in pressure are computed to approximate derivatives that are used in the momentum equations. These errors can accumulate to produce inaccurate solutions. Ramshaw and Mousseau [164] and Pletcher [163] have estimated that this issue is important (assuming reasonable machine

precision) when the Mach number is below approximately $10^{-6} - 10^{-5}$. Since the lowest Mach number considered in this work (2.5×10^{-3}) is well above these values, this issue is not an immediate concern. However, for smaller Mach numbers, remedies such as the introduction of the gauge pressure as a dependent variable should be considered [see 163, 164, 168].

4.2.2. Backward Facing Step Model Problem

Solutions to the low Mach number backward facing step model problem described in Section 2.2.3 are presented here. Algorithm performance is investigated using different Krylov algorithms and preconditioning strategies for various values of the reference flow Mach and Reynolds numbers. The Newton iteration convergence criteria specified by Equation (102) was employed, while the inner iteration convergence tolerance in Equation (127) is $\varepsilon^n = 1 \times 10^{-2}$. An upper limit of two-hundred is specified for the number of inner iterations on each Newton step.

4.2.2.1. Comparison of Different Krylov Algorithms

The advantages and disadvantages of the Lanczos-based and Arnoldi-based algorithms are compared and contrasted in this subsection. Figure 55 shows the outer Newton iteration convergence history for five different Newton-Krylov algorithms in solving a 16x80 problem [(x-cells) by (y-cells)] from a flat initial guess with an inlet Mach number of 0.25 and a Reynolds number of 100. Note that ILU(2) preconditioning was used to precondition each of the Krylov algorithms. Although convergence was obtained using each algorithm, the Lanczos-based Krylov algorithms enabled better convergence for this specific

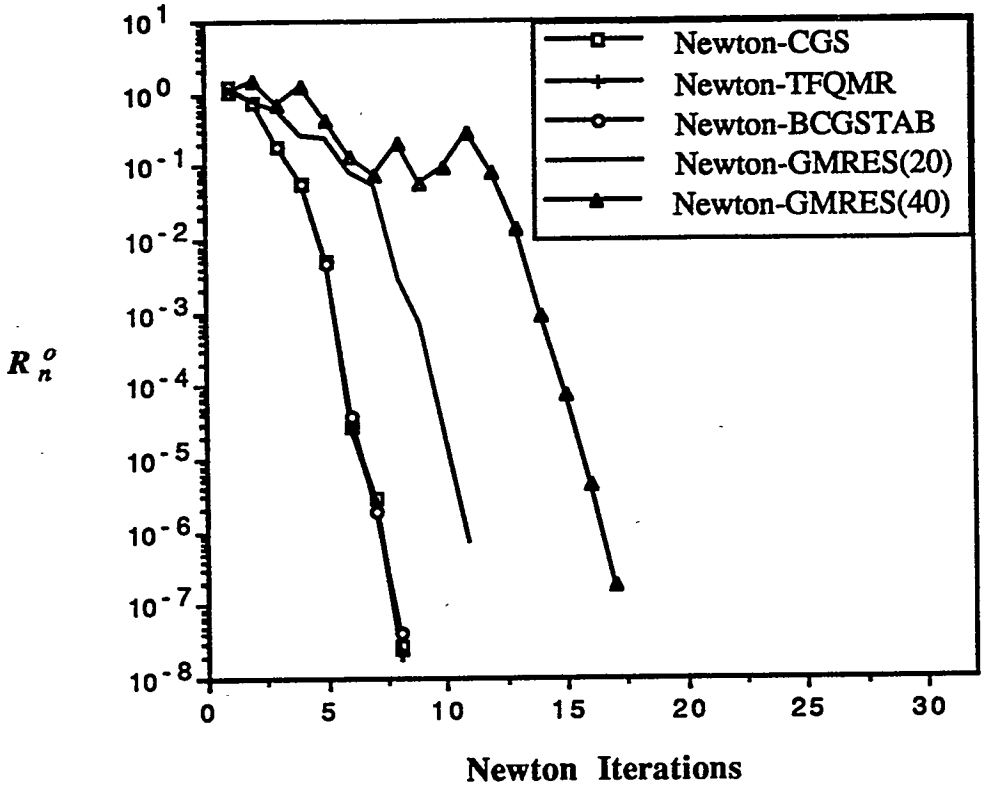


Figure 55. Convergence history of five different Newton-Krylov algorithms.

problem. This difference in convergence behavior was traced primarily to the second Newton step, which in all cases yielded an especially difficult linear system. The GMRES algorithms experienced stall because of their restricted Krylov subspace dimensions, and so did not converge within the allowed two-hundred inner iterations. In fact, the GMRES(20) algorithm also did not converge on the *first* Newton step. The Lanczos algorithms, although

requiring a large number of iterations, were able to converge within this inner iteration limit. This feature enabled these latter algorithms to move past this difficult linear system by the next Newton step. In contrast, the GMRES algorithms returned relatively poor Newton updates, which subsequently required damping. Consequently, more iterations were needed to move past this difficult part of the calculation. Damping was initiated for the GMRES(20) algorithm on the first Newton step so that the severity of the poor update returned on the second step was mitigated in comparison with the GMRES(40) algorithm. As a result, more stringent damping was needed in the case of the Newton-GMRES(40) algorithm, thus explaining why it required the most iterations. Keep in mind, however, that the initial guess strongly affects the convergence of Newton algorithm, and that preconditioning strongly affects the performance of the Krylov algorithms. In fact, results in the next section suggest that ILU preconditioning is likely not the most effective preconditioner for problems of this type.

The behavior discussed above demonstrates how the Krylov iteration can significantly affect the overall performance of the Newton-Krylov algorithm. Thus, it is instructive to examine the behavior of these different Krylov solvers. Figure 56 presents the convergence history of the different Krylov algorithms in solving the linear system on the first Newton step of this calculation. The first Newton step was selected because this is the only Newton iteration where each of the Krylov algorithms are solving the same linear system. Observations indicate that GMRES(p) converges rapidly if the required inner iterations are less than the specified dimension of the Krylov subspace, p. If frequent restarts are necessary, however, the convergence curve may flatten considerably to the point of stall. Specifically, notice that the convergence of GMRES(40) is very strong on this first Newton step, whereas the convergence of GMRES(20) is very poor beyond 20 iterations because of algorithm restarts. Recall from the discussion above, however, that GMRES(40) also encountered stall on the next Newton step. The CGS algorithm works well overall, but does

exhibit very erratic convergence behavior as shown in Figure 56. The Bi-CGSTAB exhibits more smoothly converging solutions, but can still exhibit oscillatory behavior. The TFQMR convergence curve is smooth, although somewhat flat during the intermediate iterations.

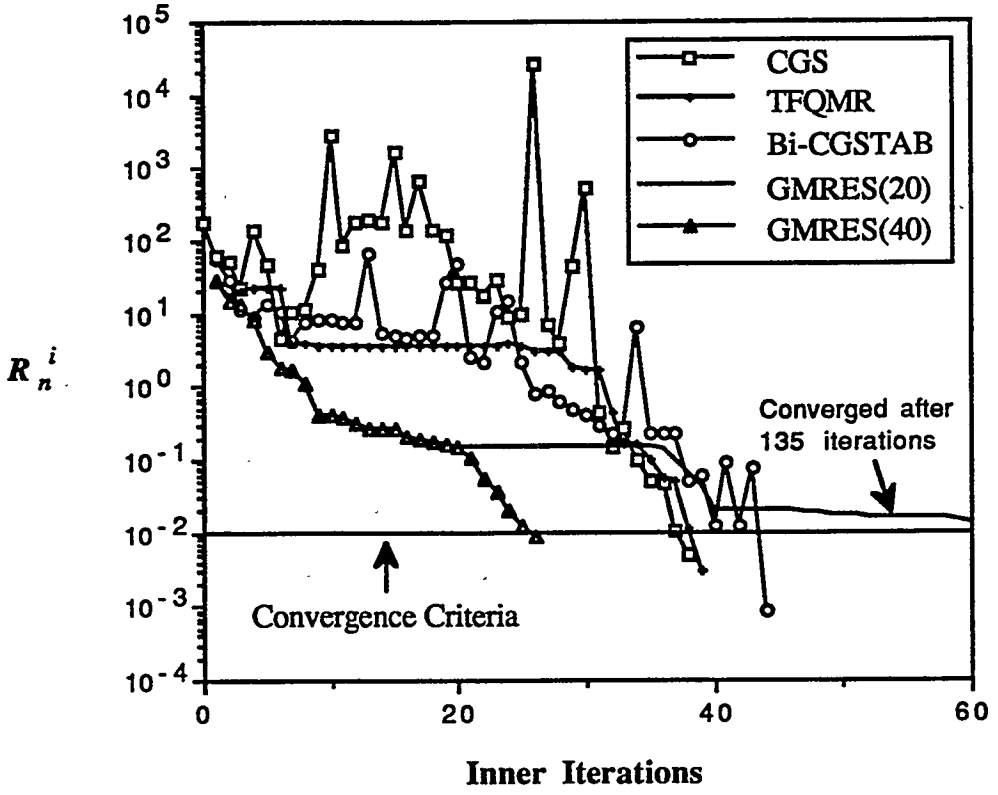


Figure 56. Comparison of convergence behavior of different Krylov solvers.

The TFQMR algorithm was selected for subsequent calculations in the next section because of its smaller storage requirements compared with the GMRES algorithms and its observed smoother convergence behavior compared with the other Lanczos based algorithms. Note that the storage requirements for the GMRES(40) algorithm is roughly four times that of the Lanczos-based algorithms and twice that of GMRES(20) (just considering Krylov algorithm memory requirements).

4.2.2.2. Preconditioner Effectiveness

The efficiency of the overall solution is strongly tied to preconditioner performance. Consequently, the effectiveness of ILU(k) and domain-based additive/multiplicative Schwarz preconditioners are investigated in this section for various values of the flow Mach and Reynolds numbers. Preconditioner effectiveness is measured not only by lower inner iteration counts, but also by CPU efficiency and memory cost. The latter two measures also being dependent upon either parallel or serial implementation; however, only a serial implementation is considered here. Note that ILU preconditioning was applied from the right, while the domain based preconditioners were applied from the left. Numerical experiments, however, suggested that solution performance was not sensitive to this choice, mainly because the true inner residual was computed on each iteration so that the preconditioner was not allowed to influence the inner iteration convergence criteria.

Table 22 presents the memory requirements for both ILU(k) and domain-based preconditioning on a uniform 16x80 grid. The ILU data assumes a reverse row type ordering (see Section 3.2.3.2). This ordering scheme performed better than the other schemes described in Section 3.2.3.2. Recall that these types of ordering schemes were the only ones considered here because of their simplicity and because they yielded banded matrix structures that were easily exploited using a non-zero diagonal storage scheme. The ILU preconditioner

memory requirements are presented as a function of the level of fill-in, k, and represent the storage required for the non-zero diagonals listed in the second column. Note that ILU(3) is equivalent to a full factorization of the Jacobian without pivoting, and as such it can no longer be considered an incomplete factorization.

The domain-based preconditioner storage requirements in Table 22 are listed as a function of the selected sub-domain blocking strategy and the amount of overlap shared by adjacent sub-domains. The data was obtained using a full LU factorization for each sub-domain using LINPACK banded Gaussian elimination [17]. The same preconditioner memory is required regardless whether additive or multiplicative Schwarz type preconditioning is selected. Reference names used to identify the different domain-based preconditioner selections in subsequent discussions are listed in column seven of Table 22. The direct solve memory requirements using the LINPACK routines is given by the limiting 1x1 blocking case (8.3MB). This number is larger than the ILU(3) case because of the additional memory used for pivoting. Note that the 1x5 blocking selection without overlap incurs the same storage requirement as the direct solve, and if a two-cell overlap is chosen the storage requirements actually exceed that of the direct solve. Although, from a storage point of view, these two selections are impractical on a single processor, they may be good selections on a distributed memory system where the per processor storage requirements would be considerably less. Note that the blocking strategies listed in Table 22 were selected to ensure that each subdomain contained the same number of cells in both the x and y directions. Although, this domain-decomposition is a convenient choice, it may or may not be the best choice regarding preconditioner efficiency. The optimal blocking strategy is likely problem and geometry dependent. Consequently, some numerical experimentation may be needed before the optimal domain decomposition is determined for a given problem and geometry.

Table 22. Preconditioner memory requirements for a uniform 16x80 grid.

ILU(k) Preconditioning (reverse row ordering)			Domain Based Preconditioners [Additive Schwarz (AS) and Multiplicative Schwarz (MS)]				
k	# non-zero diagonals	Memory (MB)	# blocks in x-dir.	# blocks in y-dir.	# overlap cells	Reference Name	Memory (MB)*
0	35	1.4	4	20	0	4x20-0-AS & 4x20-0-MS	2.4
1	59	2.4	2	10	0	2x10-0-AS & 2x10-0-MS	4.3
2	94	3.9	4	20	2	4x20-2-AS & 4x20-2-MS	5.3
3	138	5.7**	2	10	2	2x10-2-AS & 2x10-2-MS	6.8
			1	5	0	1x5-0-AS & 1x5-0-MS	8.3
			1	1	0	1x1	8.3
			1	5	2	1x5-2-AS & 1x5-2-MS	9.3

* Based on the use of LINPACK banded Gaussian elimination (with pivoting)
** Results in a full LU factorization (no pivoting).

Table 23 presents performance data for the preconditioners listed in Table 22 in solving six different flow conditions identified by three different inlet Mach numbers and two different flow Reynolds numbers. The required number of Newton iterations (n), the average number of TFQMR iterations per Newton step ($\overline{m}$), and the total CPU time (sec.) are presented for each preconditioner selection. This data was obtained on a single HP Model 735 workstation using a uniform 16x80 grid. The 'NS' abbreviation in Table 23 indicates that no solution was obtained within the allowed twenty-five Newton iterations, while a superscript on the Newton iterations counter indicates the number of times the inner iterations encountered the upper limit of two hundred iterations. Note that the preconditioner selections are listed in ascending order with respect to memory requirements as indicated in Table 22.

Table 23. Algorithm performance data for various flow Mach and Reynolds numbers on a uniform 16x80 grid (n ≡ total Newton iterations, $\overline{m}$ ≡ average inner iterations per Newton iteration, NS ≡ No Solution).

Re	Precond. Selection	Mach # = 0.25			Mach # = 0.025			Mach # = 0.0025		
		n	$\overline{m}$	CPU (sec)	n	$\overline{m}$	CPU (sec)	n	$\overline{m}$	CPU (sec)
100	ILU(0)	NS	NS	NS	NS	NS	NS	NS	NS	NS
	4x20-0-AS	8	93	178	7[1]	140	222	7[5]	184	325
	4x20-0-MS	7	50	124	7	73	168	NS	NS	NS
	ILU(1)	NS	NS	NS	NS	NS	NS	NS	NS	NS
	ILU(2)	8	39	431	NS	NS	NS	NS	NS	NS
	2x10-0-AS	8	41	120	7	62	145	7	110	235
	2x10-0-MS	7	21	81	7	30	103	NS	NS	NS
	4x20-2-AS	8	82	251	7	109	305	8	141	435
	4x20-2-MS	7	28	132	7	44	188	7	72	286
	ILU(3)	7	0	178	7	0	180	7	0	179
	2x10-2-AS	7	40	134	7	54	169	7	71	210
	2x10-2-MS	7	18	93	7	26	120	7	47	188
	1x5-0-AS	8	19	106	7	26	114	7	39	151
	1x5-0-MS	7	9	70	7	11	76	7	19	103
	1x1	7	0	43	7	0	43	7	0	43
	1x5-2-AS	7	18	96	7	21	107	7	33	142
	1x5-2-MS	7	7	70	7	8	72	7	12	147
10	ILU(0)	9	109	319	NS	NS	NS	NS	NS	NS
	4x20-0-AS	7	85	145	6	140	191	6[2]	166	222
	4x20-0-MS	7	48	121	6	64	131	NS	NS	NS
	ILU(1)	7	6	71	NS	NS	NS	NS	NS	NS
	ILU(2)	7	2	105	NS	NS	NS	NS	NS	NS
	2x10-0-AS	7	38	101	6	58	120	6	86	164
	2x10-0-MS	7	22	85	7	22	85	NS	NS	NS
	4x20-2-AS	7	58	179	6	73	184	5	97	197
	4x20-2-MS	7	22	115	7	25	125	5	40	126
	ILU(3)	6	0	153	4	0	103	5	0	128
	2x10-2-AS	6	30	96	6	39	114	5	49	115
	2x10-2-MS	6	14	69	6	15	74	5	28	92
	1x5-0-AS	7	15	86	6	21	89	5	23	79
	1x5-0-MS	6	8	63	5	9	56	6	12	73
	1x1	6	0	38	4	0	26	5	0	32
	1x5-2-AS	7	13	82	6	11	67	4	16	55
	1x5-2-MS	6	5	56	6	5	54	5	7	51

The ILU(k) preconditioners were generally less reliable for lower values of the flow Mach number as seen in Table 23. In fact for $k \leq 2$, solutions using ILU(k) preconditioning were obtained only for an inlet Mach number of 0.25, and only ILU(2) enabled a converged solution for Re=100 at that Mach number. Since the effectiveness of the ILU(k) preconditioners is dependent upon the problem size, the poor performance of these preconditioners worsens as the grid is refined. Comparing the full factorization, ILU(3) data with the full factorization, 1x1 blocking data shows that the LINPACK routines are about 4 times faster than the ILU routine when computing full LU factorizations. For this reason, ILU(3) is not a practical choice from a CPU efficiency point of view. In contrast to the ILU data, the domain based preconditioners generally performed well at the lower Mach numbers as indicated in Table 23.

Multiplicative Schwarz preconditioning outperformed additive Schwarz preconditioning in most cases, except at the lowest inlet Mach number considered when using the 4x20 and 2x10 blocking strategies without overlap. In those cases, both additive and multiplicative Schwarz preconditioners experienced difficulty solving the linear system of the first Newton step. However, the TFQMR algorithm using the multiplicative Schwarz preconditioners returned a poor Newton update that resulted in a singular sub-domain matrix on the next Newton step. A possible remedy for this behavior might be to start the calculation from a better initial guess using mesh sequencing, parameter continuation, pseudo-transient relaxation, or some other convergence enhancement technique.

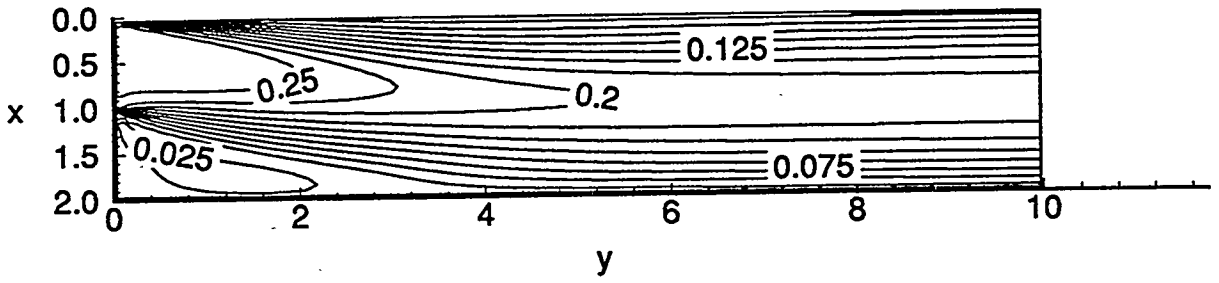
The use of overlap among sub-domains reduced inner iteration counts, but this effect did not always mean lower CPU times as shown in Table 23. Note, however, that only a small amount of overlap was allowed. Further investigation is needed before more specific observations regarding the benefits of overlap can be determined. Specifically, the optimum amount of overlap that balances preconditioner effectiveness with memory requirements and CPU cost needs to be determined.

Generally, the fewest number of sub-domains produced the best results on this coarse 16x80 grid. In fact the 1x1 blocking (single sub-domain) produced the best results for all 6 flow conditions. However, on finer grids this selection may be impractical for several reasons: first, the high memory storage cost; second, the full LU factorization becomes more expensive as the grid is refined [12] so that this trend is not expected to continue indefinitely; and third, a single domain is not amenable to parallel implementation. The last reason highlights another important advantage of the domain-based preconditioners, namely parallel implementation. The additive Schwarz preconditioners can be parallelized readily, while the multiplicative Schwarz algorithms can be parallelized using multi-coloring schemes. In this manner, both preconditioners allow the CPU and memory costs to be distributed over several processors. See Reference 16 for examples of parallel implementations of these specific domain-based preconditioners on both distributed and shared memory computers.

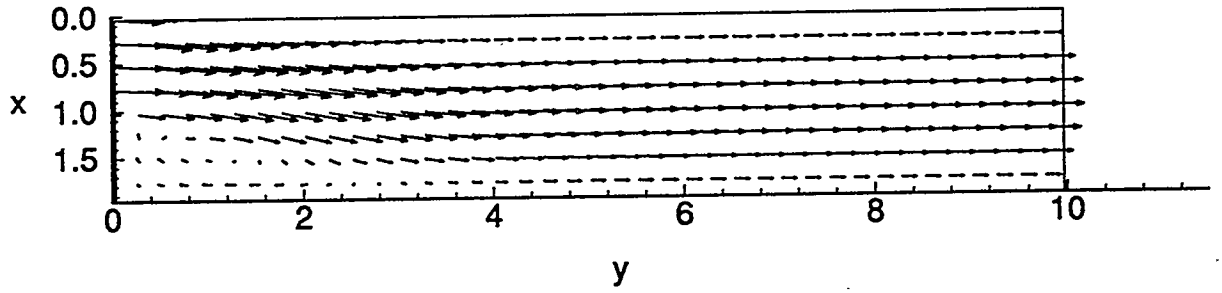
4.2.2.3. Solutions

This Section presents physical solutions for this backward facing step test problem using a 32x160 uniform grid. Solutions are presented for each of the considered Mach numbers appearing in Table 23 and a Reynolds number of 100. Based upon the results in Table 23, the 1x5 blocking strategy was selected with no overlap for use with the multiplicative Schwarz algorithm. Because of the low Mach numbers considered, the velocity field and Mach number variation are of primary interest because the scalar temperature, density, and pressure fields are relatively flat. Consequently, only flow Mach number contours and velocity vectors are presented for each Mach number at the fixed Reynolds number. Figure 57 presents these plots for an inlet Mach number of 0.25, while Figure 58 and Figure 59 present these plots for inlet Mach numbers of 0.025 and 0.0025, respectively. Note that the velocity vectors presented in these figures have been spaced

(four-cell spacing in each direction) for better clarity. Especially noticeable in the each of these plots is flow separation that occurs just downstream from the step and the subsequent zone of flow recirculation that occurs near the step corner. Note the marked reduction in the flow Mach number within the region of recirculating flow. The flow eventually reattaches (at approximately y=2.8), and at the outlet, the flow profile appears to be fully developed. Because all the Mach numbers considered lie within the incompressible flow regime, the velocity flow patterns do not change significantly as the Mach number is reduced (as one would expect). However, the computational effort required to obtain these solutions is different because of the higher condition numbers of the Jacobian matrices that arise for the lower Mach numbers. For an inlet Mach number of 0.25, 5 Newton iterations were required to converge to the 32x160 grid solution using the 16x80 grid solution as an initial guess. An average of 10 TFQMR iterations per Newton iteration and 409.8 seconds of CPU time were required . When the Mach number is dropped by an order of magnitude to 0.025, the required Newton iterations were 4, but the average TFQMR iterations increased to 17. The required CPU time increased by 6% to 434.6 seconds. This trend was continued when an inlet Mach number of 0.0025 was used. In this case 5 Newton iterations were required, while the average TFQMR iterations was 16 and the required CPU time was 693.4 seconds (a 70% increase from the highest Mach number case). This data and that presented in Table 23 clearly demonstrate that the higher condition numbers of the Jacobian matrices at low Mach numbers makes solutions of the linear systems at each Newton step more difficult. This feature, in turn, makes effective preconditioning a very important task at these low Mach numbers.

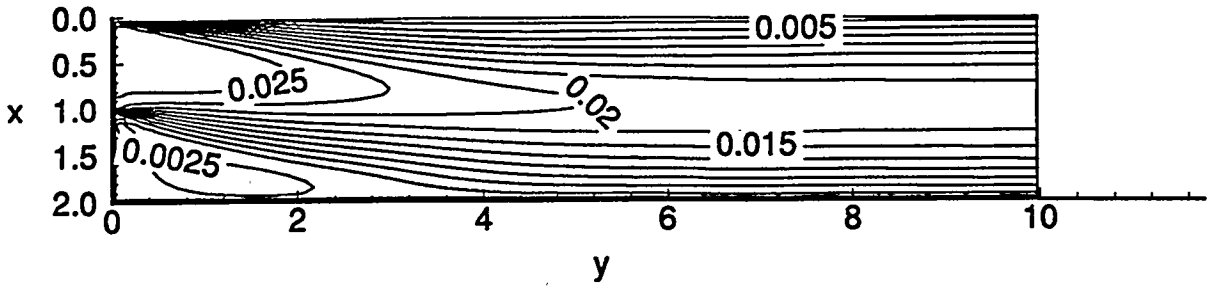


(a) Mach number contours.

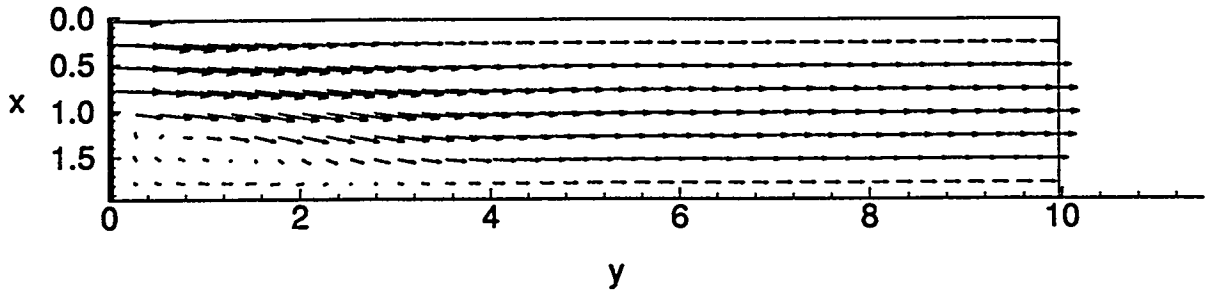


(b) Velocity Vectors (four-cell spacing used in both directions).

Figure 57. Mach number contours (a) and velocity vectors (b) for an inlet Mach number of 0.25 and flow Reynolds number of 100.

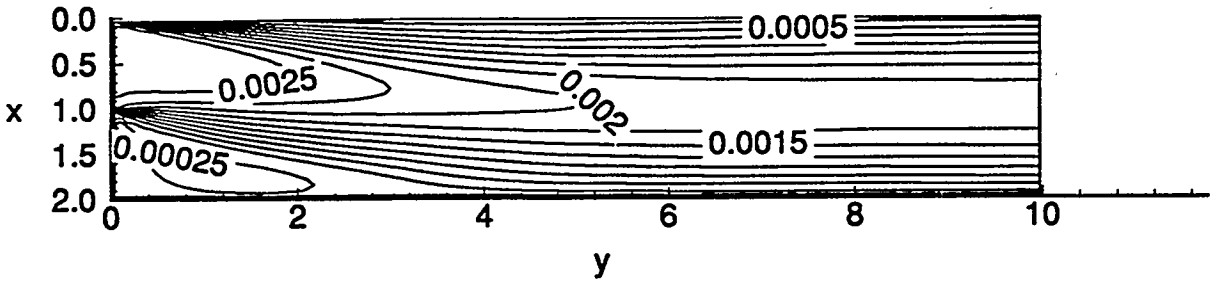


(a) Mach number contours.

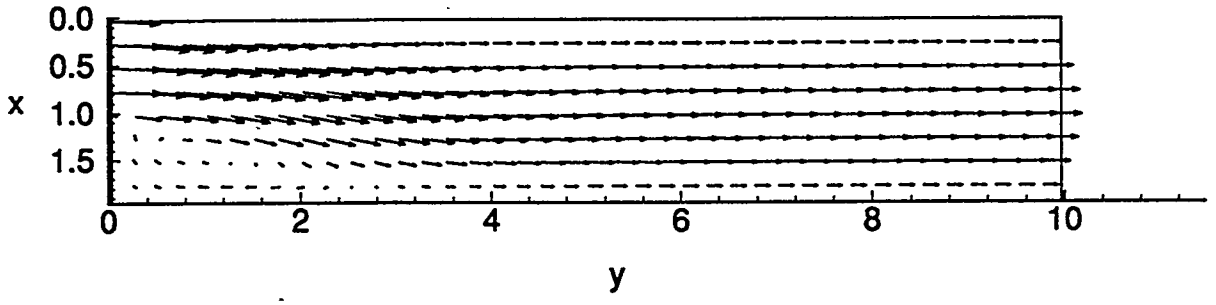


(b) Velocity Vectors (four-cell spacing used in both directions).

Figure 58. Mach number contours (a) and velocity vectors (b) for an inlet Mach number of 0.025 and flow Reynolds number of 100.



(a) Mach number contours.



(b) Velocity Vectors (four-cell spacing used in both directions).

Figure 59. Mach number contours (a) and velocity vectors (b) for an inlet Mach number of 0.0025 and flow Reynolds number of 100.

CHAPTER 5

CONCLUDING REMARKS

This final chapter is devoted to summarizing the information and results discussed in previous chapters. An overall summary of the important aspects of this work is presented in Section 5.1. Next, Section 5.2 reviews the important observations and conclusions made in Chapter 4. Finally, a list of suggested topics for further study is presented and discussed in Section 5.3.

5.1. SUMMARY

Fully coupled inexact Newton-Krylov algorithms were implemented and investigated for solving several strongly coupled, nonlinear systems of partial differential equations that arise in the field of computational fluid dynamics. Specifically considered were the steady state, incompressible and compressible Navier-Stokes and energy equations describing the flow of a laminar, Newtonian fluid in two-dimensions. These equations were solved numerically, in primitive variable form, by integrating them over discrete finite volumes. The resulting nonlinear algebraic equations were then solved using various fully coupled Newton-Krylov algorithms. Preconditioned Krylov subspace based iterative algorithms were used to invert the linear systems that arise on each Newton iteration. The Krylov subspace based algorithms considered in this study include the Generalized Minimal RESidual (GMRES) algorithm, the Conjugate Gradients Squared algorithm (CGS), the Bi-CGSTAB algorithm, and the Transpose-Free Quasi-Minimal Residual (TFQMR) algorithm. Note that this study was among the first of this sort to consider the latter two recently developed

Krylov algorithms. Both ILU and domain-based preconditioning strategies were studied for improving the performance of these Krylov algorithms.

The capabilities of the fully coupled Newton-Krylov algorithms were demonstrated in solving the coupled equations describing fluid flow and heat transfer. Specifically, natural, mixed, and forced convection of an incompressible fluid were investigated for different flow geometries and parameters. The natural convection problem consisted of a high Rayleigh number, thermally driven flow in an enclosed square cavity. The mixed and forced convection (high Reynolds number) test problems consisted of channel flow past a backward facing step. Additionally, the challenging problem of low Mach number subsonic flow was also addressed; specifically, low Mach number compressible flow past a backward facing step.

The Newton-Krylov algorithms were made more attractive by employing various numerical techniques to improve convergence and efficiency. These techniques include the use of mesh sequencing, adaptive damping, pseudo-transient relaxation, and parameter continuation. Useful observations and guidelines regarding implementation and use of these techniques were presented. Additionally, implementation was simplified considerably through the use of an efficient numerical Jacobian evaluation. Furthermore, the effect of varying the inexact Newton convergence tolerance was studied and suggested values were recommended.

The Krylov algorithms selected in this work did not require working with the matrix (Jacobian in this case) transpose. Consequently, matrix-free implementations of the various Newton-Krylov algorithms were possible. This implementation approximated the Jacobian-vector products that appear within the Krylov algorithm with finite difference projections. The advantages and disadvantages of this approach were discussed and the effectiveness of this technique was evaluated with respect to the use of different Krylov algorithms. Specifically, this research represented the first detailed comparison of Lanczos-based and

Arnoldi-based Krylov algorithms within the context of matrix-free Newton-Krylov solvers for incompressible fluid flow and heat transfer problems. Furthermore, practical matrix-free Newton-Krylov implementations, where the cost of periodic Jacobian and preconditioner evaluations are amortized over many pseudo-transient Newton steps, were demonstrated in solving the forced convection backstep model problem.

Efficient higher-order accurate solutions were obtained using the defect correction procedure. Specifically, research presented in this dissertation investigated algorithm efficiency issues associated with the use of the third order accurate cubic upwind interpolation (CUI) convection discretization scheme within a Newton-Krylov algorithm. Overall second order accurate solutions for a mixed convection benchmark problem were obtained. Recall that various CPU performance enhancement techniques were studied to improve the performance of the defect correction procedure, including the use of different mesh sequencing options.

Different preconditioning strategies were investigated to improved the performance of the Krylov algorithms. The first preconditioners considered were of the incomplete lower-upper factorization (ILU) type. These preconditioners were derived based upon the non-zero diagonal storage scheme employed in this work and allowed various levels of fill-in. Additionally, the effect of several different simple cell ordering strategies was investigated with respect to ILU preconditioner effectiveness. The second class of preconditioners studied were the domain-based Schwarz preconditioning methods. Both additive and multiplicative Schwarz methods were studied both with and without sub-domain overlap. Additionally, the memory requirements and effectiveness of these preconditioners were studied for various domain decomposition selections. These latter methods were specifically used to effectively precondition the linear systems arising during the solution of the low Mach number compressible flow test problem. This study was the first (to the author's knowledge) that demonstrates the superiority of domain-based preconditioning strategies

over more conventional incomplete lower upper (ILU) type preconditioning schemes for direct steady state calculations (i.e., no time stepping) of low Mach number compressible flows. Note also that the Mach numbers selected were well below those considered elsewhere using Newton-Krylov type algorithms.

5.2. OBSERVATIONS AND CONCLUSIONS

The first model problem considered in this work was the well known problem of natural convection in an enclosed cavity. The fluid in this problem was assumed incompressible even though the flow was thermally driven. Thus, the only density variations allowed were in the buoyancy force terms in the momentum equations using the Boussinesq approximation. This problem was solved for values of the Rayleigh number ranging from 10^4 to 10^6. Solutions to this problem demonstrated that the inexact Newton-Krylov algorithms were able to reduce work in solving the linear systems during the initial Newton iterations when far from the true solution, but still enabled more accurate solutions as the true solution was approached. Several choices for the convergence parameter, ε^n, which controls this behavior were investigated. Among these choices, $\varepsilon^n = (1/2)^{Min(n,10)}$ (where n is the Newton iteration number), was the best choice when a good initial guess was not available, but $\varepsilon^n = 10^{-2}$ worked best overall when a good initial guess was available.

Effective preconditioning is an essential ingredient in the successful use of the Krylov algorithms employed in this work. Thus, in the case of ILU(k) preconditioning, the optimal level of fill-in that balanced CPU efficiency with preconditioner effectiveness was determined for this natural convection problem. It was found that ILU(2) preconditioning provided a good compromise between CPU efficiency, memory considerations, and preconditioner effectiveness for moderately refined grids. For coarse grids (i.e., 15x15), ILU(0) preconditioning was determined to be sufficient. Using this convergence tolerance

and preconditioning recommendations, the inexact Newton algorithms were found to be CPU competitive with a direct Newton iteration using LINPACK banded Gaussian elimination, yet significantly more efficient from a memory standpoint.

In solving this same natural convection problem, the matrix-free Newton-Krylov implementation was compared to the standard Newton-Krylov implementation. In general, GMRES(20) outperformed the Lanczos based methods when the matrix-free approximation was employed. GMRES was able to maintain an acceptable level of performance when the standard implementation was replaced with the matrix-free approximation. In contrast, the Lanczos based methods Krylov algorithms considered (CGS, TFQMR, and Bi-CGSTAB) were not able to maintain the same level of performance. Among these methods, CGS was found to be poorly suited to matrix-free implementations of inexact Newton's method because of its erratic convergence behavior. TFQMR and Bi-CGSTAB performed better than CGS because of their smoother convergence behavior, but still suffered a notable drop in performance when the matrix-free approximation was used. The matrix-free implementation used in this work was primarily designed to compare and contrast the performance of the different Krylov algorithms when used in this context. As such, the Jacobian and preconditioner were formed on each Newton iteration. However, for problems where evaluating the Jacobian and preconditioner are CPU intensive operations, amortizing the cost of forming these matrices over several Newton iterations is an attractive alternative, especially during a pseudo-transient calculation. The matrix-free implementation enables this option without sacrificing the rapid convergence characteristics of the Newton-Krylov algorithm. This technique is well suited for strongly coupled systems with a large number of governing equations, such as those that arise in multi-species chemically reacting flow applications.

Standard implementations using the Lanczos based algorithms seemed to outperform the standard implementation using GMRES(20) when the grid was refined (number of

unknowns increased). Convergence of the GMRES(20) algorithm was not ensured within 20 iterations, the selected dimension of the Krylov subspace. Consequently, periodic algorithm restarts were necessary, leading to slower convergence. The GMRES(20) iteration frequently encountered the upper limit for the number of inner iterations on the finest grid. This resulted in the return of mediocre Newton updates and slower convergence of the outer Newton iteration compared with the use of the Lanczos based methods. In certain instances, the Newton-TFQMR algorithm was shown to be more robust than the Newton-CGS algorithm because of the smoother convergence of the TFQMR algorithm.

The second model problem considered was a mixed convection flow past a backward facing step. The fluid in this test problem was also considered incompressible with a Grashof number of 1000 and flow Reynolds numbers of 100 and 200. The important features of this problem included the combined effects of both free and forced convection, a different problem geometry (high aspect ratio), and inflow/outflow type velocity boundary conditions. The study of this problem focused on the efficient calculation of high-order accurate solutions. These solutions were obtained using the defect correction procedure, which used the third order CUI convection scheme in the evaluation of the residuals and the first order upwind scheme in the evaluation of the Jacobian. It was found that defect correction is a useful technique for obtaining higher order accurate solutions, but the CPU expense can be considerably higher that the cost for a lower order solution using a conventional Newton iteration. Therefore, its benefit lies in improving the accuracy of a solution using a grid, on which a low order discretization scheme could not achieve a comparable level of accuracy. The importance of obtaining a good initial guess, from which to start the defect correction procedure, was clearly demonstrated. Higher order solutions to this mixed convection model problem were used to isolate the effects of buoyancy through comparisons with an analogous solution that neglected buoyancy effects.

The third incompressible flow test problem was forced convection past a backward facing step. The important different between this test problem and the first backward facing step problem was the use of a much higher Reynolds number (800), the neglect of buoyancy effects, and the use of specified heat flux type boundary conditions. The combined effects of a high aspect ratio geometry and a large flow Reynolds number made numerical solutions to this problem especially challenging. Several different numerical techniques were investigated to improve the efficiency of the solution algorithm for this problem. It was found that cell ordering can significantly affect the effectiveness of ILU type preconditioning, and can be used to help avoid the necessity to adjust pivots during the incomplete factorization process. It was found that reverse column ordering with ILU(1) preconditioning was a good combination for solutions to this model problem. Also considered was the effect of replacing the discrete continuity equation with a discrete pressure equation. This formulation also avoided pivot adjustments during the ILU factorization process and offered some CPU efficiency advantages, especially on finer meshes. A good cell ordering/preconditioner combination for this formulation appeared to be column ordering with ILU(1) preconditioning. The third technique used in the solution of this model problem was a pseudo-transient Newton iteration. This option was less CPU efficient than other techniques using mesh sequencing and adaptive damping, but it offered the advantage of initializing the calculation on the desired grid and did not require damping or mesh sequencing to enable convergence. Additionally, the potential advantages of using the matrix-free implementation to enable efficient amortization of the Jacobian and preconditioner evaluations over several iterations was demonstrated.

The final test problem included in this investigation was the low Mach number, subsonic flow past a backward facing step. This problem was solved for various values of the flow Mach and Reynolds numbers on several grids with different levels of refinement. The low subsonic Mach numbers considered in this work ranged from 0.0025 to 0.25. The

flow Reynolds number was varied between 10 and 100. Observations in solving this problem once again indicated that the Arnoldi-based algorithm, GMRES(m), converges rapidly if the required inner iterations are less than the specified dimension of the Krylov subspace, m. If frequent restarts were necessary, however, the convergence curve flattened considerably to the point of stall. The CGS algorithm worked well in most cases for this problem, but still exhibited erratic convergence behavior. The Bi-CGSTAB and TFQMR exhibited more smoothly converging solutions than CGS, but in certain instances exhibited either erratic or stalled convergence behavior. It was demonstrated that the performance of the Krylov solve can, in some instances, significantly affect the overall performance of the Newton-Krylov algorithm. The ILU(k) preconditioners were generally found to be less reliable for lower values of the flow Mach number, and exhibited strong sensitivity to cell ordering and grid size. The domain based preconditioners generally outperformed the ILU(k) preconditioners at the lower Mach numbers and were much less sensitive to the grid size. This observation was true for most domain-decomposition selections, including several that were competitive with the ILU preconditioners from a computer memory standpoint. Additionally, the favorable parallel aspects of the domain based preconditioners were discussed.

In summary, various Newton-Krylov solution algorithms were used to solve a variety of fluid flow and heat transfer problems. These algorithms coupled with various performance enhancement techniques and preconditioning strategies were found to be effective and efficient for solving problems of this type. The advantages of a fully coupled solution algorithm were maintained without the excessive memory requirements that are typically associated with algorithms of this type. The application and development of these fully coupled solution techniques will likely continue to benefit the topics considered in this work as well as other areas of computational fluid dynamics.

5.3. SUGGESTED TOPICS FOR FURTHER STUDY

The goal of this investigation was to investigate the different aspects of Newton-Krylov solution techniques for problems arising in computational fluid dynamics. Many features of these numerical techniques were studied on a variety of test problems in this work. However, research should continue to study global convergence strategies, alternative preconditioning techniques and iterative solvers, solution accuracy issues, and application to other computational fluid dynamics problems.

Global convergence issues were briefly discussed in the introduction. This work primarily used mesh sequencing, adaptive damping, and pseudo-transient relaxation to improve the global convergence properties of Newton's method. Other techniques not studied here that deserve attention include hybrid Picard-Newton type iteration schemes, and other general line-search based global convergence algorithms [see 3, 65, 66, 67, 70].

Several of the more popular and recently developed Krylov algorithms were used in this investigation. Note that these algorithms are amenable to other more general sparse storage schemes, other than the nonzero diagonal storage scheme used in this work. These more general schemes [140] should be considered in any future investigation because they will enable use and investigation of other more general cell ordering strategies [151] and also orderings produced by unstructured grids. Additionally, because some of these Krylov algorithms are so new, they still need further testing on a wider variety of applications and situations. It is also likely that new and possibly improved algorithms will be developed in the future. Furthermore, the Krylov algorithms considered here were used within the context of an inexact Newton iteration, but it is also possible to derive a nonlinear Krylov iteration using the matrix-free approximation discussed previously [see 170, 96, 171, 172]. The feasibility and dependability of these techniques deserves further study.

The importance of preconditioning was clearly demonstrated in the results of the previous chapter. Because preconditioning is so important, the search for better preconditioners should continue. Examples of other preconditioners not considered in this work include: block ILU type preconditioners [see 94] and polynomial type preconditioning [145, 144, 146, and 147]. Furthermore, additional study of the Schwarz preconditioners is need with respect to optimal blocking strategies (including amount of overlap), more efficient subdomain solvers, two-level algorithms [155], and efficient parallel implementations.

Because solution accuracy is of importance in every numerical investigation, research should continue in determining the most efficient techniques to obtain higher order accurate solutions using a Newton-Krylov solution algorithm. This research may include using the matrix-free implementation to capture the higher-order features in Jacobian-vector products, while the preconditioner is evaluated using a lower order accurate discretization scheme.

Finally, the study and use of Newton-Krylov solution techniques should continue in other areas of computational fluid dynamics. These areas should include (but are definitely not limited to) the effects of turbulence, complex geometries (i.e., unstructured meshes), large coupled equation sets such as those associated with multi-species reacting flows, and three-dimensional problems.

APPENDIX

AN OVERVIEW OF KRYLOV SUBSPACE-BASED METHODS FOR SOLVING SYSTEMS OF LINEAR EQUATIONS

A1. INTRODUCTION

Many numerical techniques for systems of partial differential equations require solutions to large linear systems of the form, $\mathbf{Ax} = \mathbf{b}$, of dimension n, where $\mathbf{A}$ is referred to as the system matrix, $\mathbf{x}$ is the unknown solution vector, and $\mathbf{b}$ is commonly referred to as the right hand side vector. Direct solution techniques often become impractical for large linear systems because of high memory and CPU cost. The alternative in these situations are the use of iterative techniques. Krylov subspace based methods are powerful iterative techniques for solving these types of linear systems. These methods compute new approximations to the solution, $\mathbf{x}_k$, from the affine (translated) subspace defined by

$$\mathbf{x}_0 + K_k(\mathbf{r}_0, \mathbf{A}), \tag{171}$$

where the Krylov subspace of dimension k is defined by

$$K_k(\mathbf{r}_0, \mathbf{A}) \equiv span(\mathbf{r}_0, \mathbf{Ar}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{k-1}\mathbf{r}_0), \tag{172}$$

and $\mathbf{r}_0$ is the initial residual determined from the initial solution guess, $\mathbf{x}_0$, i.e., $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$.

There are some excellent references discussing Krylov subspace based methods. Some of the more recent discussions are given in References 71, 80, 119, 123, 124, 125, and 126. Some earlier, but still extremely valuable, review articles are found in References 127, 128, 129, 130, and 131. A very interesting annotated, historical bibliography of the conjugate gradient and Lanczos methods is presented in Reference 132.

The classical conjugate gradient (CCG) method of Hestenes and Stiefel [133] is probably the best known Krylov subspace based method. Interestingly enough, this algorithm was originally derived as a direct method. Its full potential as an iterative

technique was not realized until its use by Reid in 1971 [134], and latter Concus in 1976 [135]. However, the idea of using the CCG algorithm as a direct method illustrates an important property of many Krylov subspace based algorithms not shared by other iterative techniques, namely a finite termination property. This property means that with exact precision mathematics the CCG method is guaranteed to converge within n iterations, but satisfactory convergence is likely for much less than n iterations. The CCG algorithm also does not require iteration parameter estimation to improve performance, unlike some Alternative Direction Implicit (ADI) schemes [173], Successive Over Relaxation (SOR), and Chebychev iteration [see 63]. Also, the CCG algorithm typically converges more rapidly than typical matrix-splitting iterative schemes such as Jacobi (J) and Gauss-Seidel (GS) iteration. CCG is optimal in the sense that the residual norm is minimized on each iteration and that new search directions are computed with economical vector recurrences so that work and storage requirements per iteration are small. In fact, these latter two properties define a "true" conjugate gradient method. However, the difficulty associated with the CCG algorithm is that it is applicable only to symmetric, positive definite matrices. As a result, considerable research has been devoted to generalizations of the conjugate gradient method to non symmetric and non positive-definite systems.

The two main options for generalization of the CCG ideas to nonsymmetric linear systems are the following:

1) Application of the CCG algorithm to the normal equations.
2) Development of conjugate gradient-like algorithms.

The normal equations option can be applied in two different ways. First, is the application of the CCG algorithm to the system, $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$, which results in what is referred to as the CGNR algorithm [133]. The capital 'N' in CGNR refers to the application of the conjugate gradient algorithm to the normal equations, and the capital 'R' indicates that

the residual norm is minimized over the Krylov subspace. Secondly, one can apply the CCG algorithm to the system, $\mathbf{A}\mathbf{A}^T\mathbf{y} = \mathbf{b}$, where $\mathbf{x} = \mathbf{A}^T\mathbf{y}$. This latter choice is referred to as the CGNE algorithm [136]. In this case the capital 'E' indicates that the norm of the error is minimized over the Krylov subspace. The disadvantage of the normal equation approach is that the condition number of the new system is squared, which can lead to very slow convergence in some instances. Additionally, working with the matrix transpose is often undesirable because it makes sparse storage and parallel/vector implementations more difficult, and because it prohibits use of finite difference projection techniques to approximate matrix-vector products within inexact Newton iterations [67]. These reasons have recently made the use of conjugate gradient-like algorithms a more attractive option.

Conjugate gradient-like algorithms are derived by relaxing either or both of the properties that define a "true" conjugate gradient method, namely optimality and economical vector recurrences. Some of the more popular and more recent conjugate gradient-like algorithms that will be discussed further in the subsequent sections include: the generalized minimal residual algorithm (GMRES) [4], the conjugate gradient squared algorithm (CGS) [5], the Bi-CGSTAB algorithm [6], and the transpose-free quasi-minimal residual algorithm (TFQMR) [7].

The purpose of this appendix is to adequately describe and review Krylov subspace based methods for solution of general linear systems. To this end, Section A2 will present a general description of these methods, including the two distinct approaches commonly used to develop the different Krylov algorithms. These are the *minimal residual approach* and the *orthogonal residual approach* [119]. Once this framework has been established, the next two sections will discuss two popular classes of Krylov subspace methods. The first class is the Arnoldi-Based [137] algorithms discussed in Section A3, and the second is the Lanczos-based [138] algorithms discussed in Section A4. Finally, a brief summary of Krylov subspace based methods is given in Section A5.

A2. GENERAL DESCRIPTION OF KRYLOV METHODS

Krylov subspace based methods can be viewed as polynomial-based iterative schemes for solving systems of the form $\mathbf{Ax} = \mathbf{b}$, of dimension n. The general form of a polynomial iterative scheme is given by [141]

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \sum_{j=0}^{k-1} \eta_{kj} \mathbf{r}_j, \tag{173}$$

where $\mathbf{r}_j = \mathbf{b} - \mathbf{A}\mathbf{x}_j$. Subtracting Equation (173) from the equation, x=x, where $\mathbf{x}$ is the true solution, yields an equation for the error, $\mathbf{e}$, given as,

$$\mathbf{e}_k = \mathbf{e}_{k-1} + \sum_{j=0}^{k-1} \eta_{kj} \mathbf{A}\mathbf{e}_j. \tag{174}$$

It can be shown via an induction proof [141] that this is equivalent to

$$\mathbf{e}_k = \mathbf{R}_k(\mathbf{A})\mathbf{e}_0; \ \mathbf{R}_k(0) = \mathbf{1}, \tag{175}$$

where $\mathbf{R}_k(\mathbf{A})$ is a polynomial of maximum degree, k, expressed as

$$\mathbf{R}_k(\mathbf{A}) = \mathbf{I} + \alpha_1 \mathbf{A} + \alpha_2 \mathbf{A}^2 + ... + \alpha_k \mathbf{A}^k. \tag{176}$$

Multiplying both sides of Equation (175) by A allows one to derive the residual polynomial expression as follows (i.e., $\mathbf{A}\mathbf{e}_k = \mathbf{A}(\mathbf{x} - \mathbf{x}_k) = \mathbf{b} - \mathbf{A}\mathbf{x}_k = \mathbf{r}_k$),

$$\mathbf{r}_k = \mathbf{R}_k(\mathbf{A})\mathbf{r}_0; \ \mathbf{R}_k(0) = \mathbf{1}. \tag{177}$$

It is instructive to use Equation (175) to express $\mathbf{x}_k$ as a polynomial relation as follows,

$$\mathbf{x}_k = \mathbf{x} - \mathbf{e}_k = \mathbf{x}_0 + \mathbf{e}_0 - \mathbf{R}_k(\mathbf{A})\mathbf{e}_0 = \mathbf{x}_0 + [\mathbf{I} - \mathbf{R}_k(\mathbf{A})]\mathbf{e}_0, \tag{178}$$

where the quantity in brackets can be defined as a new polynomial to give

$$\mathbf{x}_k = \mathbf{x}_0 + \mathbf{P}_k(\mathbf{A})\mathbf{e}_0; \ \mathbf{P}_k(0) = 0. \tag{179}$$

Now a factor of $\mathbf{A}$ can be factored out of the polynomial $\mathbf{P}$ to give

$$\mathbf{x}_k = \mathbf{x}_0 + \mathbf{Q}_k(\mathbf{A})\mathbf{A}\mathbf{e}_0 = \mathbf{x}_0 + \mathbf{Q}_k(\mathbf{A})\mathbf{r}_0; \ \textit{where} \ \mathbf{Q}_k(0) = \mathbf{1}. \tag{180}$$

Equivalently, Equation (180) can be expressed as

$$\mathbf{x}_k = \mathbf{x}_0 + \textit{Linear combination of} \ \{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{k-1}\mathbf{r}_0\}, \tag{181}$$

where the vectors in brackets are referred to as Krylov vectors. As indicated in the introduction, these vectors span a k-dimensional subspace of R^n referred to as the k^{th} Krylov subspace defined as in Equation (172). New approximations to the solution are then computed from the affine (i.e., translated by the vector $\mathbf{x}_0$) Krylov subspace as defined by Equation (172). As a result, algorithms of this type are typically referred to as Krylov subspace based methods. More generally, Equation (181) can be expressed as

$$\mathbf{x}_k = \mathbf{x}_0 + \mathbf{d}_k, \tag{182}$$

where

$$\mathbf{d}_k \in K_k(\mathbf{r}_0, \mathbf{A}) \equiv span\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, ..., \mathbf{A}^{k-1}\mathbf{r}_0\}. \tag{183}$$

The manner in which the $\mathbf{d}_k$ vector is computed defines a particular Krylov algorithm. Generally, there are two, often competing, objectives when choosing how to compute the $\mathbf{d}_k$ vectors. One objective is error minimization, while another is economical iterations obtained via short vector recurrence relationships. Note that a 'true' or 'ideal' conjugate gradient method satisfies both of these objectives. Methods for selecting the $\mathbf{d}_k$ vectors typically follow from taking either a *minimal residual approach* or an *orthogonal residual approach* [119]. The former approach picks the $\mathbf{d}_k$ vectors to minimize some norm of the residual, i.e.,

$$\underset{\mathbf{d}_k \in K_k(\mathbf{r}_0, \mathbf{A})}{Min} \|\mathbf{r}_k\| = \underset{\mathbf{d}_k \in K_k(\mathbf{r}_0, \mathbf{A})}{Min} \|\mathbf{r}_0 - \mathbf{A}\mathbf{d}_k\|. \tag{184}$$

Note that a unique iterate satisfying Equation (184) can always be found [119]. The latter approach requires the $\mathbf{d}_k$ vectors to satisfy some sort of Petrov-Galerkin condition, i.e.,

$$\mathbf{r}_k \perp L_k, \tag{185}$$

where L_k is some other Krylov subspace that may be different from K_k, and the symbol, $\perp$, indicates orthogonality. In contrast to the minimal residual approach, there is no guarantee that a unique iterate satisfying Equation (185) can be found [119]. The *minimal residual approach* is discussed further in Section A2.1, while the *orthogonal residual approach* is addressed in more detail in Section A2.2.

Note that for symmetric, positive definite matrices the two approaches are equivalent with $L_k \equiv K_k$. Additionally, the iterates can be computed efficiently using short vector recurrence relationships. This results in a 'true' or 'ideal' conjugate gradient method that exhibits both optimality (error reduction) and economical vector recurrences (constant work and storage requirements per iteration). However, for nonsymmetric matrices one cannot in general maintain optimality (with respect to a fixed norm) using short vector recurrences [129]. Consequently, maintaining optimality typically causes the work and storage requirements to increase with the iteration count. This often necessitates the use of algorithm restarts or truncation to keep iteration work and storage costs at a reasonable level. On the other hand, short vector recurrences can often be maintained to keep iteration work and storage costs low, but only at the expense of optimality. Both alternatives lead to what are commonly referred to as conjugate gradient-like algorithms. Specifically, algorithms that are derived by sacrificing optimality and/or economical vector recurrences. Derivation of several conjugate gradient-like algorithms will be discussed in Sections A2.1 and A2.2. However, discussion of conjugate gradient-like algorithms based upon the Arnoldi process [137] and the nonsymmetric Lanczos process [138] will be presented separately in Section A3 and Section A4, respectively.

A2.1. Minimal Residual Approach

The goal in the minimal residual approach is to minimize some *fixed* norm of the residual over the Krylov subspace, as indicated by Equation (184). For symmetric, positive definite (SPD) matrices, minimizing Equation (186) below leads to the conjugate gradient algorithm of Hestenes and Stiefel [119, 126, 133],

$$\|\mathbf{r}\|_{\mathbf{A}^{-1}} = \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_{\mathbf{A}^{-1}}. \tag{186}$$

Minimizing Equation (186) is equivalent to minimizing the functional given by

$$f(\mathbf{x}) = (\mathbf{b} - \mathbf{A}\mathbf{x})^T \mathbf{A}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x} - 2\mathbf{x}^T\mathbf{b} + \mathbf{b}^T\mathbf{A}\mathbf{b} \ . \tag{187}$$

Note that the negative gradient of this functional (defined with respect to $\mathbf{x}$) is $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$, so that minimization of Equation (187) is equivalent to solving the system, $\mathbf{A}\mathbf{x} = \mathbf{b}$ (at least for SPD matrices). Minimization problems of this type often assume the solution iterate can be computed simply as

$$\mathbf{x}_{k+1} = \mathbf{x}_0 + \sum_{j=0}^{k} \alpha_j \mathbf{p}_j = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \tag{188}$$

where α_k can be interpreted as a scale factor and $\mathbf{p}_k$ as a search direction. Note that Equation (188) is a simplified, economical vector recurrence relationship requiring information only from the previous solution iterate and the current search direction. Substituting Equation (188) into Equation (187) and minimizing with respect to α_k (i.e., $\partial f(\mathbf{x}_{k+1})/\partial \alpha_k = 0$) gives

$$\alpha_k = \frac{\mathbf{p}_k^T(\mathbf{b} - \mathbf{A}\mathbf{x}_k)}{\mathbf{p}_k^T\mathbf{A}\mathbf{p}_k} = \frac{\mathbf{p}_k^T\mathbf{r}_k}{\mathbf{p}_k^T\mathbf{A}\mathbf{p}_k} = \frac{(\mathbf{p}_k, \mathbf{r}_k)}{(\mathbf{p}_k, \mathbf{A}\mathbf{p}_k)}. \tag{189}$$

Since A is assumed SPD (i.e., $(\mathbf{p}_k, \mathbf{A}\mathbf{p}_k) > 0$), α_k is well defined. This choice for the scale factor and the assumed form of the solution update in Equation (188) minimizes $\|\mathbf{r}_{k+1}\|_{\mathbf{A}^{-1}}$ along the search direction, $\mathbf{p}_k$. The general form of the algorithm thus far can be expressed as [126, 128]:

Algorithm for SPD Matrices
1) *Choose* $\mathbf{x}_0$
2) *Compute* $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
3) *Set:* $\mathbf{p}_0 = \mathbf{r}_0$
For $k = 0,1,\ldots,$
4) $\alpha_k = \dfrac{(\mathbf{p}_k, \mathbf{r}_k)}{(\mathbf{p}_k, \mathbf{A}\mathbf{p}_k)}$ (190)
5) $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
6) $\mathbf{r}_{k+1} = \mathbf{r}_k + \alpha_k \mathbf{A}\mathbf{p}_k$
7) *If* $\|\mathbf{r}_{k+1}\| <$ *tolerance then quit*
8) *Compute* $\mathbf{p}_{k+1}$

The final requirement for specifying a particular algorithm is the computation of $\mathbf{p}_{k+1}$ in step (8) of the algorithm expressed in Equation (190). Possible choices for computing these search directions include the following [see 126]:

1) *Method of steepest descent* (also referred to as Richardson's Method). In this case,

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1}. \tag{191}$$

Recall that the residual is associated with the gradient of the functional given in Equation (187). This method is closely related to Jacobi iteration and as a consequence can exhibit rather slow convergence.

2) *Gauss-Seidel-type iteration.* In this case,

$$\mathbf{p}_k = \mathbf{e}_{k+1}, \tag{192}$$

where $\mathbf{e}_{k+1}$ is a vector whose only non zero value is a *1* in the $(k+1)$ component. Note that *n*-steps of the algorithm in Equation (190) would then be equivalent to one Gauss-Seidel iteration.

3) *Conjugate direction methods.* These methods require that the search directions be A-orthogonal, or conjugate with respect to A. This requirement can be expressed as

$$\mathbf{p}_i^T \mathbf{A}\mathbf{p}_j = (\mathbf{p}_i, \mathbf{A}\mathbf{p}_j) = 0, \quad \textit{for } i \neq j, \tag{193}$$

The search directions then serve as an A-orthogonal basis for the Krylov subspace. This feature enables conjugate direction methods to assume the finite termination property that was mentioned in the introduction. Recall that this

property guarantees that the iteration will converge in no more that n-steps [126]. One possible choice for the conjugate search directions is the eigenvectors of A. Since A is assumed symmetric and nonsingular the eigenvectors, $\mathbf{s}_i$ (corresponding to the eigenvalues, λ_i), satisfy,

$$\mathbf{s}_i^T \mathbf{A}\mathbf{s}_j = \mathbf{s}_i^T\left(\lambda_j \mathbf{s}_j\right) = \lambda_j \mathbf{s}_i^T \mathbf{s}_j = 0, \ \textit{for } i \neq j. \tag{194}$$

Unfortunately, use of the eigenvectors as the conjugate search directions is not practical because their computation is often very expensive. A more practical choice is the use of the *conjugate gradient method* [133] where A-orthogonality is maintained via,

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k, \ \textit{where } \beta_k = -\frac{\left(\mathbf{p}_k, \mathbf{A}\mathbf{r}_{k+1}\right)}{\left(\mathbf{p}_k, \mathbf{A}\mathbf{p}_k\right)}. \tag{195}$$

Induction can be used to prove that Equation (195) satisfies the A-orthogonality condition expressed by Equation (193) for all previous search directions. The search directions based upon the conjugate gradient method can be viewed as the projection of the negative gradient of the functional in Equation (187) ($\mathbf{r}_{k+1}$) onto the Krylov subspace, $K_{k+1}(\mathbf{r}_0, \mathbf{A})$.

Note that the general form of the solution update is different from the simple form assumed in Equation (188), and it is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_0 + \sum_{j=0}^{k} \alpha_{kj} \mathbf{p}_j. \tag{196}$$

However, in the case of the conjugate direction methods, the A-orthogonality of the search directions allows the simpler form of Equation (188) to be used since substitution of Equation (196) into Equation (187) yields,

$$f(\mathbf{x}_{k+1}) = \sum_{j=0}^{k}\sum_{l=0}^{k} \alpha_{kj}\alpha_{kl}\left(\mathbf{p}_j^T \mathbf{A}\mathbf{p}_l\right) - 2\sum_{j=0}^{k} \alpha_{kj}\left(\mathbf{p}_j^T \mathbf{r}_0\right) - 2\mathbf{x}_0^T\mathbf{b} + \mathbf{x}_0^T\mathbf{A}\mathbf{x}_0 + \mathbf{b}^T\mathbf{A}\mathbf{b}. \tag{197}$$

Enforcing the A-orthogonality condition expressed in Equation (193) allows this expression to be simplified to

$$f(\mathbf{x}_{k+1}) = \sum_{j=0}^{k} \alpha_{kj}^2 \left(\mathbf{p}_j^T \mathbf{A} \mathbf{p}_j\right) - 2\sum_{j=0}^{k} \alpha_{kj} \left(\mathbf{p}_j^T \mathbf{r}_0\right) - 2\mathbf{x}_0^T \mathbf{b} + \mathbf{x}_0^T \mathbf{A} \mathbf{x}_0 + \mathbf{b}^T \mathbf{A} \mathbf{b}. \tag{198}$$

Now minimizing with respect to α_{kj} yields,

$$\frac{\partial f(\mathbf{x}_{k+1})}{\partial \alpha_{kj}} = 2\sum_{j=0}^{k} \left[\alpha_{kj} \left(\mathbf{p}_j^T \mathbf{A} \mathbf{p}_j\right) - \left(\mathbf{p}_j^T \mathbf{r}_0\right)\right] = 0. \tag{199}$$

From whence α_{kj} is given by

$$\alpha_{kj} = \frac{\left(\mathbf{p}_j^T \mathbf{r}_0\right)}{\left(\mathbf{p}_j^T \mathbf{A} \mathbf{p}_j\right)} = \frac{\left(\mathbf{p}_j, \mathbf{r}_0\right)}{\left(\mathbf{p}_j, \mathbf{A} \mathbf{p}_j\right)} = \frac{\left(\mathbf{p}_j, \mathbf{r}_j\right)}{\left(\mathbf{p}_j, \mathbf{A} \mathbf{p}_j\right)}, \tag{200}$$

where the last relation is due once again to the A-orthogonality of the search directions. Because α_{kj} does not depend on the k-subscript, this dependency can be removed to give

$$\alpha_j = \frac{\left(\mathbf{p}_j, \mathbf{r}_j\right)}{\left(\mathbf{p}_j, \mathbf{A} \mathbf{p}_j\right)}. \tag{201}$$

This result shows that use of Equation (188) is equivalent to the use of Equation (196) as long as the search directions are A-orthogonal. The consequence of this result is significant because it demonstrates that Equation (186) is minimized over all previous search directions, or equivalently over the entire affine Krylov subspace. This characteristic enables the conjugate gradient method to exhibit the optimality property mentioned previously. The first

two options listed above do not share this property, and consequently perform only a local minimization along the current search direction. This difference enables the conjugate gradient algorithm to display superior performance compared to the other options.

The drawback associated with the conjugate gradient method is that for nonsymmetric matrices, the norm defined by Equation (186) is not valid [119]. Consequently, the approach outlined above can not be used. However, it is legitimate to use the *L_2-norm*, i.e., minimize

$$\|\mathbf{r}\|_2 = \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2. \tag{202}$$

Minimizing Equation (202) is equivalent to minimizing the functional given by

$$g(\mathbf{x}) = (\mathbf{b} - \mathbf{A}\mathbf{x})^T(\mathbf{b} - \mathbf{A}\mathbf{x}) = \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\mathbf{b}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{b}\ . \tag{203}$$

Following the same procedure described above, namely substituting Equation (188) into Equation (203) and minimizing with respect to α_k yields,

$$\alpha_k = \frac{(\mathbf{A}\mathbf{p}_k, \mathbf{r}_k)}{(\mathbf{A}\mathbf{p}_k, \mathbf{A}\mathbf{p}_k)}, \tag{204}$$

which can be used to replace step 4 of the algorithm defined in Equation (190). Once again there are several options for computing the search directions:

1) *Minimal Residual Algorithm* (MR) (analogous to steepest descent). In this case, the search directions are computed from Equation (191) [128].

2) *Conjugate Residual Algorithm* (CR) [128, 174]. In this case, the search directions are computed from,

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k, \ \textit{where}\ \ \beta_k = -\frac{(\mathbf{A}\mathbf{r}_{k+1}, \mathbf{A}\mathbf{p}_i)}{(\mathbf{A}\mathbf{p}_k, \mathbf{A}\mathbf{p}_k)}. \tag{205}$$

If the matrix is symmetric, then Equation (205) guarantees $\mathbf{A}^T\mathbf{A}$-orthogonality of the search directions, i.e.,

$$\left(\mathbf{A}\mathbf{p}_i, \mathbf{A}\mathbf{p}_j\right) = 0 \quad for \ i \neq j, \tag{206}$$

and optimality is retained. However, for nonsymmetric systems Equation (205) ensures only that the current search direction is $\mathbf{A}^T\mathbf{A}$-orthogonal to the previous search direction. Orthogonality of all previous search directions is not guaranteed. Consequently, the conjugate residual algorithm loses its optimality property for nonsymmetric linear systems. Even if the search directions are $\mathbf{A}^T\mathbf{A}$-orthogonal, another potential problem occurs if $\alpha_k = 0$ before convergence is reached. This condition is referred to as algorithm stall, because according to step 5 of the algorithm in Equation (191) no further progress towards the solution is possible. This condition arises when

$$\left(\mathbf{r}_k, \mathbf{A}\mathbf{p}_k\right) = \left(\mathbf{r}_k, \mathbf{A}\mathbf{r}_k\right) = 0, \tag{207}$$

where the equality of the two expressions follows from the $\mathbf{A}^T\mathbf{A}$-orthogonality condition [see 128]. This condition is avoided if the matrix A is positive real so that

$$\left(\mathbf{r}_k, \mathbf{A}\mathbf{r}_k\right) > 0 \tag{208}$$

for all real $\mathbf{r}_k$, or equivalently the symmetric part of A ($=(\mathbf{A}+\mathbf{A}^T)/2$) must be positive definite. Consequently, these difficulties often make application of the conjugate residual algorithm to general nonsymmetric linear systems ineffective.

3) *Generalized Conjugate Residual Algorithm* (GCR) [128, 175]. This algorithm overcomes the orthogonality problems of the conjugate residual algorithm for nonsymmetric linear systems by using all previous search directions to compute the new search direction. $\mathbf{A}^T\mathbf{A}$-orthogonality of all the previous search directions is then enforced using the following expression,

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \sum_{j=0}^{k} \beta_{jk}\mathbf{p}_j, \ where \ \beta_{jk} = -\frac{\left(\mathbf{A}\mathbf{r}_{k+1}, \mathbf{A}\mathbf{p}_j\right)}{\left(\mathbf{A}\mathbf{p}_j, \mathbf{A}\mathbf{p}_j\right)}. \tag{209}$$

The problem with this approach is that the work and storage requirements increase with the iteration count. This disadvantage arises because all previous search directions must be stored in order to ensure that each new search direction is $\mathbf{A}^T\mathbf{A}$-orthogonal with all previous search directions using Equation (209). This situation was alluded to in the previous section. One potential remedy for this difficulty is to restart the algorithm after say *m*-iterations, thereby fixing the maximum dimension of the Krylov subspace to be *m*. The

restarted version of this algorithm is denoted GCR(m) [128]. Note that the GCR algorithm, like the CR algorithm, is still susceptible to stall for indefinite **A**, and so convergence is guaranteed only for positive real matrices.

4) *Orthomin(m)* [128, 176]. This algorithm is based upon GCR, but uses truncation instead of algorithm restarts to control the iteration work and storage requirements. In the case of truncation, orthogonality is maintained only among the previous m search directions,. This condition can be expressed by,

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \sum_{j=k-m+1}^{k} \beta_{jk} \mathbf{p}_j . \tag{210}$$

Once again, this algorithm can stall, and so convergence is guaranteed only for positive real matrices.

5) *Orthodir(m)* [128, 177]. The full Orthodir algorithm (no truncation) is another extension of CR whereby the $\mathbf{A}^T\mathbf{A}$-orthogonal search directions are computed from

$$\mathbf{p}_{k+1} = \mathbf{A}\mathbf{p}_k + \sum_{j=0}^{k} \beta_{jk} \mathbf{p}_j, \ \textit{where} \ \beta_{jk} = -\frac{\left(\mathbf{A}^2\mathbf{p}_k, \mathbf{A}\mathbf{p}_j\right)}{\left(\mathbf{A}\mathbf{p}_j, \mathbf{A}\mathbf{p}_j\right)}. \tag{211}$$

This slight modification to Equation (209) enables Orthodir to converge for any nonsingular **A**, thereby removing the restriction to positive real matrices. The truncated version of Orthodir is implemented using

$$\mathbf{p}_{k+1} = \mathbf{A}\mathbf{p}_k + \sum_{j=k-m+1}^{k} \beta_{jk} \mathbf{p}_j . \tag{212}$$

The one drawback associated with the Orthodir(m) algorithm is that scaling is sometimes necessary to ensure numerical stability [141].

In general, for nonsymmetric linear systems, one typically uses either GCR or Orthomin if the matrix is positive real, while Orthodir should be used if this condition is not satisfied [141]. However, since the development of GMRES and other more recent Krylov algorithms, these algorithms have become less popular [141].

A2.2. Orthogonal Residual Approach

The orthogonal residual approach is based upon enforcing some sort of Petrov-Galerkin condition. Typically, this condition takes the form

$$\mathbf{r}_k \perp L_k, \tag{213}$$

where $\perp$ indicates orthogonality, $\mathbf{r}_k$ is the residual at iteration k, and L_k is some Krylov subspace of dimension k [178]. This approach is also frequently referred to as a projection method [131, 178]; and since we are interested in Krylov subspaces, algorithms derived in this manner can further be classified as Krylov projection methods.

In describing the orthogonal residual approach it is convenient to use matrix notation. In this context, one can introduce two nxk matrices $\mathbf{P}_k$ and $\mathbf{W}_k$, whose columns span K_k and L_k, respectively [see 80, 131]:

$$\mathbf{P}_k = [\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k]; \quad \mathbf{W}_k = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k]. \tag{214}$$

Using this notation the general solution update can be expressed as

$$\mathbf{x}_k = \mathbf{x}_0 + \sum_{j=1}^{k} \alpha_{kj} \mathbf{p}_j = \mathbf{x}_0 + \mathbf{P}_k \mathbf{y}_k, \tag{215}$$

where

$$\mathbf{y}_k = (\alpha_{k1}, \alpha_{k2}, \ldots, \alpha_{kk})^T. \tag{216}$$

Note that the lower limit for the subscript on $\mathbf{p}$ is now assumed to be *one* for convenience instead of the previously used value of *zero*. Similarly, the residual at iteration k can be written as

$$\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k = \mathbf{r}_0 - \mathbf{A}\mathbf{P}_k\mathbf{y}_k. \tag{217}$$

With these definitions, the orthogonality condition of Equation (213) can be equivalently specified by

$$\mathbf{W}_k^T(\mathbf{r}_0 - \mathbf{A}\mathbf{P}_k\mathbf{y}_k) = 0. \tag{218}$$

This equation can then be solved for $\mathbf{y}_k$ to obtain

$$\mathbf{y}_k = \left[\mathbf{W}_k^T\mathbf{A}\mathbf{P}_k\right]^{-1}\mathbf{W}_k^T\mathbf{r}_0. \tag{219}$$

Substitution of this result into Equation (215) then specifies $\mathbf{x}_k$ in terms of $\mathbf{P}_k$ and $\mathbf{W}_k$. Equation (219) shows that the existence of a unique iterate is dependent upon the quantity $\left[\mathbf{W}_k^T\mathbf{A}\mathbf{P}_k\right]^{-1}$ being nonsingular. This requirement explains why a unique iterate satisfying Equation (213) cannot always be found when using the *orthogonal residual* approach.

The final steps in deriving a Krylov projection method consist of identifying L_k and selecting a procedure for computing $\mathbf{P}_k$ and $\mathbf{W}_k$. Saad [178] has concisely categorized several popular choices for L_k. Each choice gives rise to a class of Krylov projection techniques. These choices and examples of different Krylov projection techniques resulting from each choice are itemized below [see 178]:

1) $L_k = K_k(\mathbf{r}_0, \mathbf{A})$. This choice is equivalent to a Galerkin method [131]. Krylov projection techniques resulting from this choice include the classical conjugate gradient algorithm (CCG) [133], the generalized conjugate gradient algorithm (GCG) [135, 179], Orthores [177], and the full and incomplete orthogonalization methods [FOM, IOM] [130]. Recall from the previous section that the conjugate gradient algorithm was derived for SPD matrices. For this special case the minimal residual approach and this orthogonal residual choice are equivalent, as can be verified from the results of the previous section (via an induction proof). Consequently, the conjugate gradient algorithm can be derived from both perspectives. For general nonsymmetric matrices, however, the minimal residual approach and the orthogonal residual approach are not equivalent. As a result, for nonsymmetric systems, algorithms within this class do not exhibit a minimization property.

2) $L_k = \mathbf{A}K_k(\mathbf{r}_0, \mathbf{A})$. Saad [131] points out that this choice is similar to a least squares method or a variational method. It follows then that Krylov projection techniques derived from this orthogonalization choice, minimize the *L_2-norm* of the residual over the affine Krylov subspace. Thus, for both symmetric and nonsymmetric systems, this orthogonalization choice is equivalent to a minimal residual approach. Algorithms that are members of this class include the conjugate residual algorithm (CR) [174], the generalized conjugate residual algorithm (GCR) [175], Orthomin [176], Orthodir [177], Axelsson's method [180], and the generalized minimal residual algorithm (GMRES) [4].

3) $L_k = K_k(\mathbf{r}_0, \mathbf{A}^T)$. Lanczos-based [138] Krylov projection techniques fall into this class. Note that this choice is identical to the first for symmetric matrices. Thus, one can also view the classical conjugate gradient algorithm as a Lanczos-based method [133]. For nonsymmetric systems, though, several new Krylov projection techniques arise. Members of this class include the bi-conjugate gradient algorithm (BCG) [138, 139], the conjugate gradient squared algorithm (CGS) [5], the quasi-minimal residual (QMR) family of algorithms [see 7, 181, 182, 183, and 184], and Bi-CGSTAB and related algorithms [6, 185].

The last step in the derivation of a Krylov algorithm is selection of an appropriate procedure for computing $\mathbf{P}_k$ and $\mathbf{W}_k$. In the previous section, the vectors comprising the columns of these matrices were generated by enforcing either $\mathbf{A}$-orthogonality or $\mathbf{A}^T\mathbf{A}$-orthogonality of the search directions. The following sections consider two alternative processes for computing these vectors, namely the Arnoldi process [137] and the nonsymmetric Lanczos process [138].

A3. ARNOLDI-BASED KRYLOV ALGORITHMS

This section considers Krylov subspace methods based upon the Arnoldi process [130, 137]. This process uses the Gram-Schmidt orthogonalization procedure [186] to generate an orthonormal basis for the Krylov subspace. This technique is in contrast to previously discussed methods where the basis vectors were either $\mathbf{A}$-orthogonal or $\mathbf{A}^T\mathbf{A}$-orthogonal. The Arnoldi process also reduces the system matrix to upper Hessenberg form, which can be very useful for eigenvalue computations. The Arnoldi process is outlined below [see 80, 126, 141]:

$$
\begin{aligned}
&1)\ \mathbf{p}_1 = \frac{\mathbf{r}_0}{\|\mathbf{r}_0\|_2} \\
&\textit{For}\ \ k = 1,\ldots,n \\
&\quad 2)\ h_{lk} = (\mathbf{A}\mathbf{p}_l, \mathbf{p}_k)\quad l = 1,\ldots,k \\
&\quad 3)\ \tilde{\mathbf{p}}_{k+1} = \mathbf{A}\mathbf{p}_k - \sum_{l=1}^{k} h_{lk}\mathbf{p}_l \\
&\quad 4)\ h_{k+1,k} = \|\tilde{\mathbf{p}}_{k+1}\|_2 \\
&\quad 5)\ \mathbf{p}_{k+1} = \frac{\tilde{\mathbf{p}}_{k+1}}{h_{k+1,k}}
\end{aligned}
\tag{220}
$$

As mentioned above, this process results in an orthonormal basis (orthogonal and of unit L_2-*norm*) for the Krylov subspace. This can be expressed mathematically by the following relations:

$$\mathbf{p}_{k+1} \in K_{k+1}(\mathbf{r}_0, \mathbf{A}), \tag{221}$$

$$(\mathbf{p}_{k+1}, \mathbf{w}) = 0 \ \textit{for all}\ \mathbf{w} \in K_k(\mathbf{r}_0, \mathbf{A}), \text{ and} \tag{222}$$

$$\|\mathbf{p}_{k+1}\|_2 = 1. \tag{223}$$

Since the search directions are now I-orthogonal as indicated by Equation (222), and not A-orthogonal or $\mathbf{A}^T\mathbf{A}$-orthogonal, the general solution update form expressed in Equation (215) [also Equation (196)] must be used rather than the simpler form given in Equation (188). Step $(k+1)$ of the Arnoldi process in Equation (220) can be written recursively as

$$h_{k+1,k}\mathbf{p}_{k+1} = \mathbf{A}\mathbf{p}_k - \sum_{l=1}^{k} h_{lk}\mathbf{p}_l. \tag{224}$$

In matrix form, this recurrence relationship is given by

$$h_{k+1,k}\mathbf{p}_{k+1}\mathbf{e}_k^T = \mathbf{A}\mathbf{P}_k - \mathbf{P}_k\mathbf{H}_k, \tag{225}$$

where $\mathbf{e}_k^T$ is the transpose of the unit vector with a one in the k^{th} component and zeroes elsewhere. $\mathbf{H}_k$ represents an upper Hessenberg matrix with elements, h_{lk}. Specifically, $\mathbf{H}_k$ is the a $(k\text{x}k)$ matrix denoted by

$$\mathbf{H}_k = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \cdot & \cdot & \cdot & h_{1k} \\ h_{21} & h_{22} & h_{23} & & & & \cdot \\ 0 & h_{32} & h_{33} & \cdot & & & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & & & \cdot & \cdot & \cdot & h_{k-1,k} \\ 0 & \cdot & \cdot & \cdot & 0 & h_{k,k-1} & h_{kk} \end{bmatrix}. \tag{226}$$

Note that Equation (225) demonstrates the reduction to upper Hessenberg form that was mentioned at the beginning of this discussion. Once again, this reduction is very useful not only in deriving different Krylov algorithms, but also for approximating the eigenvalues of the system matrix.

At this point in the discussion, either of the two different approaches described in Section A2 can be applied; specifically, either the minimal residual approach or the orthogonal residual approach. Section A3.1 below will describe the use of the orthogonal residual approach in the development of the FOM and IOM(m) algorithms, while Section A3.2 will discuss the use of the minimal residual approach in the development of the GMRES algorithm.

A3.1. The FOM and IOM(m) Algorithms

The full orthogonalization (FOM) and incomplete orthogonalization [IOM(m)] algorithms [130] are derived via the orthogonal residual approach. Recall from Section A2.2 that use of the orthogonal residual approach requires enforcing some sort of orthogonality condition on the residual at each iteration. It was indicated in Section A2.2 that these algorithms were members of the class of algorithms for which,

$$L_k = K_k(\mathbf{r}_0, \mathbf{A}). \tag{227}$$

Thus, with this choice, the orthogonality condition defined by Equation (218) becomes

$$\mathbf{P}_k^T(\mathbf{r}_0 - \mathbf{A}\mathbf{P}_k\mathbf{y}_k) = 0, \tag{228}$$

since the Arnoldi vectors, which comprise the columns of $\mathbf{P}_k$, span both K_k and L_k. Equation (228) can be rearranged to give

$$\mathbf{P}_k^T\mathbf{A}\mathbf{P}_k\mathbf{y}_k = \mathbf{P}_k^T\mathbf{r}_0. \tag{229}$$

From the Arnoldi algorithm in Equation (220) note that $\mathbf{r}_0 = \|\mathbf{r}_0\|_2 \mathbf{p}_1$. Substitution then yields,

$$\mathbf{P}_k^T \mathbf{A} \mathbf{P}_k \mathbf{y}_k = \|\mathbf{r}_0\|_2 \mathbf{P}_k^T \mathbf{p}_1, \tag{230}$$

but since the Arnoldi vectors are orthonormal $\left(\mathbf{P}_k^T \mathbf{p}_1 = \mathbf{e}_1\right)$ this can be further simplified to

$$\mathbf{P}_k^T \mathbf{A} \mathbf{P}_k \mathbf{y}_k = \|\mathbf{r}_0\|_2 \mathbf{e}_1. \tag{231}$$

Equation (231) represents an (nxk) linear system that must be solved for the unknown vector $\mathbf{y}_k$. Equation (225) can be used to reduce this (nxk) linear system to a smaller (kxk)linear system that is much easier to solve. Solving Equation (225) for $\mathbf{A}\mathbf{P}_k$ and multiplying by $\mathbf{P}_k^T$ yields,

$$\mathbf{P}_k^T \mathbf{A} \mathbf{P}_k = h_{k+1,k}\left(\mathbf{P}_k^T \mathbf{p}_{k+1}\right)\mathbf{e}_k^T + \mathbf{P}_k^T \mathbf{P}_k \mathbf{H}_k = \mathbf{H}_k. \tag{232}$$

This result, which follows from the Arnoldi vectors being orthonormal, demonstrates the iterative reduction of the (nxn) matrix $\mathbf{A}$ to a (kxk) upper Hessenberg matrix, $\mathbf{H}_k$. Historically the advantage of this reduction was that on each iteration the eigenvalues of $\mathbf{H}_k$ approximate those of $\mathbf{A}$, but are much easier to compute [80, 137]. The advantage in this discussion, however, is the reduction of the (nxk) linear system in Equation (231) to the (kxk) linear system given below,

$$\mathbf{H}_k \mathbf{y}_k = \|\mathbf{r}_0\|_2 \mathbf{e}_1. \tag{233}$$

Equation (233) can now be efficiently solved for $\mathbf{y}_k$, which in turn defines the solution update through Equation (215) [see 130]. The procedure outlined above yields the FOM of Saad [130]. Note that the work and storage requirements of this algorithm increase with the iteration count. This drawback often requires restarts or truncation for practical implementations similar to the case of the GCR algorithm described in Section A2.1. In this case, the use of truncation requires the following modification of step three of the Arnoldi algorithm [Equation (220)],

$$\tilde{\mathbf{p}}_{k+1} = \mathbf{A}\mathbf{p}_k - \sum_{l=k-m+1}^{k} h_{lk}\mathbf{p}_l. \tag{234}$$

This modification is analogous to the one used by Orthomin(m) in Equation (210). Use of Equation (234) results in the IOM(m) algorithm [130].

Recall from Section A2.2 that for general nonsymmetric matrices, both FOM and IOM(m) do not posses any minimization property. Additionally, since these algorithms are derived via the orthogonal residual approach there is no guarantee that a unique iterate can be found to satisfy the orthogonality condition. Consequently, these algorithms are susceptible to breakdown. The GMRES algorithm described below, which is also based upon the Arnoldi process, avoids these disadvantages.

A3.2. The GMRES Algorithm

Section A2.2 indicated that the GMRES algorithm is a member of a class of algorithms that can be viewed equivalently from either the minimal residual or the orthogonal residual approach. In this section, the minimal residual approach is chosen in order to highlight the minimization property possessed by GMRES. Analogous with the algorithms

in Section A2.1, the goal is to derive an algorithm that minimizes the $L_2 - norm$ of the residual [see 4, 80, 126, 141]. The difference in this case is that the search directions are generated via the Arnoldi process. Consequently, the quantity to be minimized follows from Equation (217), and is given by,

$$\|\mathbf{r}_k\|_2 = \|\mathbf{r}_0 - \mathbf{A}\mathbf{P}_k\mathbf{y}_k\|_2 = \|\mathbf{r}_0 - (\mathbf{P}_k\mathbf{H}_k + h_{k+1,k}\mathbf{p}_{k+1}\mathbf{e}_k^T)\mathbf{y}_k\|_2, \tag{235}$$

using Equation (225). Minimization of Equation (235) is equivalent to an (nxk) least squares problem. The goal is to reduce the order of this least squares problem to allow efficient computation of $\mathbf{y}_k$. This is accomplished by noting that

$$(\mathbf{P}_k\mathbf{H}_k + h_{k+1,k}\mathbf{p}_{k+1}\mathbf{e}_k^T) = \mathbf{P}_{k+1}\tilde{\mathbf{H}}_k, \tag{236}$$

where $\tilde{\mathbf{H}}_k$ is a $(k+1)xk$ matrix identical to $\mathbf{H}_k$ except for the $(k+1)^{st}$ row, in which $\tilde{\mathbf{H}}_k$ has $h_{k+1,k}$ as its only non zero entry, i.e.,

$$\tilde{\mathbf{H}}_k = \begin{bmatrix} h_{11} & h_{12} & h_{13} & . & . & . & h_{1k} \\ h_{21} & h_{22} & h_{23} & & & & . \\ 0 & h_{32} & h_{33} & . & & & . \\ . & . & . & . & . & & . \\ . & & . & . & . & . & . \\ . & & & . & . & . & h_{k-1,k} \\ 0 & . & . & . & 0 & h_{k,k-1} & h_{kk} \\ 0 & 0 & . & . & . & 0 & h_{k+1,k} \end{bmatrix}. \tag{237}$$

Additionally, from the first step in the Arnoldi algorithm of Equation (220),

$$\mathbf{r}_0 = \|\mathbf{r}_0\|_2\mathbf{p}_1 = \|\mathbf{r}_0\|_2\mathbf{P}_{k+1}\mathbf{e}_1. \tag{238}$$

Substitution of Equation (236) and Equation (238) into Equation (235) then yields,

$$\|\mathbf{r}_k\|_2 = \left\|\mathbf{P}_{k+1}\left(\|\mathbf{r}_0\|_2 \mathbf{e}_1 - \tilde{\mathbf{H}}_k \mathbf{y}_k\right)\right\|_2. \tag{239}$$

However, since the columns of $\mathbf{P}_{k+1}$ are orthonormal it has unit L_2-*norm* and so it makes no contribution to the $L_2 - norm$ of the residual. Therefore, it can be subsequently dropped out of Equation (239) to give,

$$\|\mathbf{r}_k\|_2 = \left\|\left(\|\mathbf{r}_0\|_2 \mathbf{e}_1 - \tilde{\mathbf{H}}_k \mathbf{y}_k\right)\right\|_2. \tag{240}$$

This simplification to Equation (240) is very important since it reduces the original $(n\text{x}k)$ least squares problem to a much simpler $(k+1)\text{x}k$ least squares problem. This reduction results in considerably less work for $k << n$. This type of minimization problem can be solved very efficiently using QR-factorization [see 126], especially since $\tilde{\mathbf{H}}_k$ is an upper Hessenberg matrix [4]. Another advantage of this formulation is that the $L_2 - norm$ of the residual can be computed as a byproduct of the QR-solution of the least squares problem [4, 126]. This enables one to avoid computing the solution using Equation (215) until the end of the calculation.

The GMRES algorithm can be expressed in algorithmic form as follows [4]:

GMRES Algorithm
1) *Choose* $\mathbf{x}_0$
2) *Compute* $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
3) *Set* $\mathbf{p}_1 = \dfrac{\mathbf{r}_0}{\|\mathbf{r}_0\|_2}$
For $k = 1,\ldots,n$
Arnoldi Process
4) $h_{lk} = (\mathbf{A}\mathbf{p}_l, \mathbf{p}_k) \quad l = 1,\ldots,k$
5) $\tilde{\mathbf{p}}_{k+1} = \mathbf{A}\mathbf{p}_k - \sum_{l=1}^{k} h_{lk}\mathbf{p}_l$
6) $h_{k+1,k} = \|\tilde{\mathbf{p}}_{k+1}\|_2$
7) $\mathbf{p}_{k+1} = \dfrac{\tilde{\mathbf{p}}_{k+1}}{h_{k+1,k}}$
Mimimization Step
8) *Update* $\tilde{\mathbf{H}}_k$ *and its QR factorization for solving*:
$$\underset{\mathbf{y}_k \in K_k}{Min} \left\| \|\mathbf{r}_0\|_2 \mathbf{e}_1 - \tilde{\mathbf{H}}_k \mathbf{y}_k \right\|_2 \qquad (241)$$
Convergence Check
9) *If* $\|\mathbf{r}_k\|_2 <$ *tolerance then compute solution from* $\mathbf{x}_k = \mathbf{x}_0 + \mathbf{P}_k\mathbf{y}_k$ *and quit, otherwise continue.*

Note that since GMRES is derived via a least squares type minimization, a unique iterate satisfying the minimization of Equation (240) will always exist. Thus, GMRES will not breakdown as can the FOM and IOM(m) algorithms of the previous section. Additionally, GMRES should be expected to produce smaller residuals on each iteration than these other Arnoldi-based algorithms. Also, in general, GMRES is also more efficient than other minimization type algorithms such as GCR and Orthodir [80]. These features have made GMRES a very popular algorithm. A significant drawback associated with GMRES, is that, like FOM, the work and storage requirements increase with the iteration count. This often requires use of the restarted version, GMRES(k), for practical implementations, where k is the specified maximum dimension of the Krylov subspace [4].

A4. LANCZOS-BASED KRYLOV ALGORITHMS

This section considers Krylov subspace methods based upon building pairs of biorthogonal bases using the nonsymmetric Lanczos biorthogonalization procedure [138]. This technique is in contrast to the technique of the previous section where an orthonormal basis was generated for the Krylov subspace via the Arnoldi process. In this case, the vectors within a given subspace are not orthogonal. Instead, they are generated by maintaining orthogonality with the vectors in another different, but related, Krylov subspace. The nonsymmetric Lanczos biorthogonalization procedure is a process for efficiently generating these vectors using economical vector recurrence relationships. Another feature of this procedure is the iterative reduction of the system matrix to tridiagonal form. Thus, on each Lanczos iteration the eigenvalues of the system matrix can be approximated by computing the eigenvalues of this tridiagonal matrix. However, the interest here is the use of the Lanczos process as a foundation for the development of different Krylov algorithms for the solution of general nonsymmetric linear systems.

Lanczos-based Krylov techniques are typically derived using the orthogonal residual approach, and more specifically the third choice listed in Section A2.2. This orthogonality condition can be expressed as

$$\mathbf{r}_k \perp K_k(\mathbf{r}_0, \mathbf{A}^T). \tag{242}$$

Following the notation in Section A2.2, a basis for $K_k(\mathbf{r}_0, \mathbf{A}^T)$ is given by the columns of $\mathbf{W}_k$, and so Equation (242) is equivalently expressed by Equation (218). The next step is to determine a procedure for computing the vectors that span $K_k(\mathbf{r}_0, \mathbf{A})$ and $K_k(\mathbf{r}_0, \mathbf{A}^T)$. These vectors can be generated efficiently with three term recurrences using the nonsymmetric

Lanczos process. This process requires the vectors, $\mathbf{p}$, that span $K_k(\mathbf{r}_0, \mathbf{A})$ and the vectors, $\mathbf{w}$, that span $K_k(\mathbf{r}_0, \mathbf{A}^T)$ be biorthogonal. This condition requires

$$\mathbf{w}_i^T \mathbf{p}_j = (\mathbf{w}_i, \mathbf{p}_j) = \begin{Bmatrix} d_i, & i = j \\ 0, & i \neq j \end{Bmatrix}. \tag{243}$$

In matrix form this condition is given by

$$\mathbf{W}_i^T \mathbf{P}_j = diag(d_i) = \mathbf{D}_i, \tag{244}$$

where the quantities to the right of the equality signs denote a diagonal matrix whose diagonal entry on row i is equal to d_i. Given initial non zero starting vectors $\mathbf{p}_1$ and $\mathbf{w}_1$, the nonsymmetric Lanczos process generates two sequences of vectors using the following three term recurrence relationships [71, 119, 138]:

$$\mathbf{p}_{k+1} = \mathbf{A}\mathbf{p}_k - \alpha_k \mathbf{p}_k - \beta_k \mathbf{p}_{k-1}, \tag{245}$$

and

$$\mathbf{w}_{k+1} = \mathbf{A}^T \mathbf{w}_k - \alpha_k \mathbf{w}_k - \beta_k \mathbf{w}_{k-1}, \tag{246}$$

where

$$\alpha_k = \frac{(\mathbf{w}_k, A\mathbf{p}_k)}{(\mathbf{w}_k, \mathbf{p}_k)}, \tag{247}$$

$$\beta_k = \frac{d_k}{d_{k-1}}, \tag{248}$$

and β_1, $\mathbf{p}_0$,and $\mathbf{w}_0$ are set equal to zero. An important concern associated with the nonsymmetric Lanczos process is that $(\mathbf{w}_k, \mathbf{p}_k)$ may equal zero in Equation (247) before the process is finished iterating. This is referred to as a "serious breakdown", while if this quantity is only approximately zero it is called a "near breakdown." Algorithm restarts and/or look-ahead strategies [see 187] can be used to mitigate this problem, but these techniques are not included in this discussion.

Note that in matrix form the recurrence relationships given in Equation (245) and Equation (246) can be written as [following References 80, 126]:

$$\mathbf{A}\mathbf{P}_k = \mathbf{P}_k\mathbf{T}_k + \mathbf{p}_{k+1}\mathbf{e}_k^T, \tag{249}$$

and

$$\mathbf{A}^T\mathbf{W}_k = \mathbf{W}_k\mathbf{T}_k + \mathbf{w}_{k+1}\mathbf{e}_k^T, \tag{250}$$

where $\mathbf{T}_k$ is a *kxk* tridiagonal matrix defined by

$$\mathbf{T}_k = \begin{bmatrix} \alpha_1 & \beta_2 & 0 & . & . & 0 \\ 1 & \alpha_2 & \beta_3 & & & . \\ 0 & . & . & . & & . \\ . & & . & . & . & 0 \\ . & & & . & . & \beta_k \\ 0 & . & . & 0 & 1 & \alpha_k \end{bmatrix}. \tag{251}$$

Equation (249) and Equation (250) more clearly illustrate the reduction of A to a tridiagonal form. The Lanczos process naturally terminates when either $\mathbf{p}_{k+1}$ or $\mathbf{w}_{k+1}$ equals zero, at which point an invariant Krylov subspace is spanned by that set of vectors.

A4.1. The Bi-Conjugate Gradient Algorithm (BCG)

The first Krylov subspace based algorithm derived using the nonsymmetric Lanczos process was the bi-conjugate gradient algorithm (BCG) [138, 139]. This section discusses the development of the BCG algorithm since it forms the foundation for many other Lanczos based Krylov algorithms.

Substituting the value of $\mathbf{AP}_k$ from Equation (249) into the orthogonality condition of Equation (218) gives,

$$\mathbf{W}_k^T\mathbf{r}_0 - \mathbf{W}_k^T\mathbf{P}_k\mathbf{T}_k\mathbf{y}_k - \mathbf{W}_k^T\mathbf{p}_{k+1}\mathbf{e}_k^T\mathbf{y}_k = 0. \tag{252}$$

Defining $\mathbf{p}_1 = \mathbf{w}_1 = \mathbf{r}_0 / \|\mathbf{r}_0\|_2$ and using the biorthogonality condition expressed by Equations (243) and (244), the first term in Equation (252) can be written as

$$\mathbf{W}_k^T\mathbf{r}_0 = \mathbf{W}_k^T\left(\|\mathbf{r}_0\|_2\,\mathbf{p}_1\right) = \|\mathbf{r}_0\|_2(\mathbf{w}_1,\mathbf{p}_1)\mathbf{e}_1 = \|\mathbf{r}_0\|_2 d_1\mathbf{e}_1. \tag{253}$$

Additionally, the biorthogonality conditions expressed by Equation (243) and Equation (244) can also be used to simplify the second term and eliminate the third term. The simplified form of Equation (252) then becomes,

$$\mathbf{D}_k\mathbf{T}_k\mathbf{y}_k = \|\mathbf{r}_0\|_2 d_1\mathbf{e}_1, \tag{254}$$

but from Equation (244) this is equivalent to

$$\mathbf{T}_k \mathbf{y}_k = \|\mathbf{r}_0\|_2 \mathbf{e}_1, \tag{255}$$

which can be efficiently solved for $\mathbf{y}_k$ because $\mathbf{T}_k$ is tridiagonal. The new solution iterate can then be computed from Equation (215). However, like GMRES, the residual can actually be computed without computing the new solution iterate. This enables one to monitor convergence and then compute the new solution via Equation (215) only after convergence is reached. The residual equation is derived as follows:

$$\begin{aligned}
\mathbf{r}_k &= \mathbf{r}_0 - \mathbf{A}\mathbf{P}_k \mathbf{y}_k \\
&= \mathbf{r}_0 - \mathbf{P}_k \mathbf{T}_k \mathbf{y}_k - \mathbf{p}_{k+1} \mathbf{e}_k^T \mathbf{y}_k \\
&= \mathbf{r}_0 - \mathbf{P}_k \|\mathbf{r}_0\|_2 \mathbf{e}_1 - \mathbf{p}_{k+1} \mathbf{e}_k^T \mathbf{y}_k \\
&= \mathbf{r}_0 - \mathbf{p}_1 \|\mathbf{r}_0\|_2 - \mathbf{p}_{k+1} \mathbf{e}_k^T \mathbf{y}_k \\
&= \mathbf{r}_0 - \mathbf{r}_0 - \mathbf{p}_{k+1} \mathbf{e}_k^T \mathbf{y}_k \\
&= -\mathbf{p}_{k+1} \mathbf{e}_k^T \mathbf{y}_k .
\end{aligned} \tag{256}$$

Thus, the *L_2-norm* of the residual can be computed from

$$\|\mathbf{r}_k\|_2 = \left| \mathbf{e}_k^T \mathbf{y}_k \right| \cdot \|\mathbf{p}_{k+1}\|_2 . \tag{257}$$

Following Golub [126], the BCG procedure can be summarized by the following algorithm:

BCG Algorithm (first form)
1) *Choose* $\mathbf{x}_0$
2) *Compute* $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
3) *Set:* $\beta_0 = \|\mathbf{r}_0\|_2$; $\mathbf{p}_0 = \mathbf{w}_0 = 0$; $\mathbf{p}_1 = \mathbf{w}_1 = \dfrac{\mathbf{r}_0}{\beta_0}$; $d_0 = 1$

For $k = 1,2,...$
Nonsymmetric Lanczos Biorthogonalization Process
4) $d_k = \mathbf{w}_k^T \mathbf{p}_k$
5) $\alpha_k = \dfrac{\mathbf{w}_k^T \mathbf{A}\mathbf{p}_k}{d_k}$
6) $\beta_k = \dfrac{d_k}{d_{k-1}}$
7) $\mathbf{p}_{k+1} = \mathbf{A}\mathbf{p}_k - \alpha_k \mathbf{p}_k - \beta_k \mathbf{p}_{k-1}$
8) $\mathbf{w}_{k+1} = \mathbf{A}^T \mathbf{w}_k - \alpha_k \mathbf{w}_k - \beta_k \mathbf{w}_{k-1}$
Orthogonal Residual Approach
9) $\mathbf{y}_k = \beta_0 \mathbf{T}_k^{-1} \mathbf{e}_1$
10) $\|\mathbf{r}_k\|_2 = \left|\mathbf{e}_k^T \mathbf{y}_k\right| \cdot \|\mathbf{p}_{k+1}\|_2$
11) *If* $\|\mathbf{r}_k\|_2$ < *tolerance then compute*
$\mathbf{x}_k = \mathbf{x}_0 + \mathbf{P}_k \mathbf{y}_k$
otherwise continue to iterate.

(258)

Note that the main part of the BCG algorithm as presented in Equation (258) is separated into two distinct steps, each of which is susceptible to breakdown. The Lanczos process breaks down if $d_k = \mathbf{w}_k^T \mathbf{p}_k = 0$, and the orthogonal residual approach can fail if a unique iterate satisfying the orthogonality condition expressed by Equation (218) does not exist. In terms of the steps of the algorithm in Equation (258), this condition is equivalent to the Lanczos matrix, $\mathbf{T}_k$, being singular [119]. Other difficulties associated with the use of BCG include: sometimes very erratic convergence behavior stemming from the lack of a minimization property, and the necessity of working with the matrix transpose. The advantages of BCG over Arnoldi-based techniques are the short vector recurrence relationships that require only small work and storage requirements on each iteration. Consequently, algorithm restarts and truncation are not necessary. Thus, in the absence of round-off error and assuming no breakdowns occur, the algorithm exhibits the finite

termination property mentioned in the introduction. The Arnoldi-based algorithms exhibit this property only if restarts and truncation are avoided, which means the work and storage requirements are allowed to increase on each iteration.

Curfman [80] points out that the form of the BCG algorithm presented in Equation (258) is the one first presented by Lanczos [138] and is also known as the Biorthogonal Lanczos Method [80]. The form of BCG proposed later by Fletcher [139] given below in Equation (259) is mathematically equivalent to the first form, but is more frequently cited as the BCG algorithm in the literature [see 5, 6, 80, 119, and 120]:

BCG Algorithm (second form)
1) *Choose* $\mathbf{x}_0, \tilde{\mathbf{r}}_0$ $(\tilde{\mathbf{r}}_0 \neq 0)$
2) *Compute* $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
3) *Set:* $\mathbf{v}_0 = \mathbf{r}_0$, $\tilde{\mathbf{v}}_0 = \tilde{\mathbf{r}}_0$, $\mathbf{v}_{-1} = \tilde{\mathbf{v}}_{-1} = 0$, $\rho_{-1} = 1$
For $k = 0,1,2,...$
4) $\rho_k = \tilde{\mathbf{r}}_k^T \mathbf{r}_k$
5) $\beta_k = \dfrac{\rho_k}{\rho_{k-1}}$
6) $\mathbf{v}_k = \mathbf{r}_k + \beta_k \mathbf{v}_{k-1}$
7) $\tilde{\mathbf{v}}_k = \tilde{\mathbf{r}}_k + \beta_k \tilde{\mathbf{v}}_{k-1}$
8) $\sigma_k = \tilde{\mathbf{v}}_k^T \mathbf{A}\mathbf{v}_k$
9) $\alpha_k = \dfrac{\rho_k}{\sigma_k}$
10) $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{v}_k$
11) $\mathbf{r}_{k+1} = \mathbf{r}_k + \alpha_k \mathbf{A}\mathbf{v}_k$
12) $\tilde{\mathbf{r}}_{k+1} = \tilde{\mathbf{r}}_k + \alpha_k \mathbf{A}^T \tilde{\mathbf{v}}_k$
13) *If* $\|\mathbf{r}_{k+1}\|_2 <$ *tolerance then quit*

(259)

Within this second form, $\mathbf{r}_k$ and $\tilde{\mathbf{r}}_k$ are analogous to the vectors $\mathbf{p}_k$ and $\mathbf{w}_k$ in Equation (258) (i.e., they are biorthogonal). Note that the search vectors in Equation (259) given by $\mathbf{v}_k$ and $\tilde{\mathbf{v}}_k$ are *bi-conjugate*, meaning they satisfy

$$(\mathbf{A}\mathbf{v}_i, \tilde{\mathbf{v}}_j) = 0 \quad \textit{for } i \neq j, \tag{260}$$

which explains why this algorithm was given the name it now assumes.

A4.2. The Conjugate Gradients Squared Algorithm (CGS)

One drawback associated with the use of the BCG algorithm is the required use of the matrix transpose. This drawback has motivated the development of several transpose-free Lanczos-based Krylov algorithms. The first such algorithm was the conjugate gradients squared algorithm (CGS) of Sonneveld [5]. In order to better describe the CGS algorithm it is helpful to look at the polynomial representation of the BCG algorithm. Recall from Section A2 that all of the Krylov algorithms discussed here are members of a more general class of polynomial based iterative techniques. With this in mind, the residuals and search directions of the BCG algorithm expressed in Equation (259) can be represented by

$$\begin{aligned} \mathbf{r}_k &= \mathbf{R}_k(\mathbf{A})\mathbf{r}_0 \\ \tilde{\mathbf{r}}_k &= \mathbf{R}_k(\mathbf{A}^T)\tilde{\mathbf{r}}_0 \\ \mathbf{v}_k &= \mathbf{S}_k(\mathbf{A})\mathbf{r}_0 \\ \tilde{\mathbf{v}}_k &= \mathbf{S}_k(\mathbf{A}^T)\tilde{\mathbf{r}}_0, \end{aligned} \tag{261}$$

where $\mathbf{R}_k$ and $\mathbf{S}_k$ are polynomials of maximum degree, k. In the BCG algorithm, the matrix transpose appears on step 12 in Equation (259) for the calculation of $\tilde{\mathbf{r}}_k$. This vector then subsequently influences the update of $\tilde{\mathbf{v}}_k$ and calculation of ρ_k and σ_k [119]. Sonneveld observed that by using the polynomial representations in Equation (261), calculation of the latter two quantities could be rewritten to eliminate the matrix transpose [5]. For example, step 4 of Equation (259) can be written as [5],

$$\rho_k = \left[\mathbf{R}_k(\mathbf{A}^T)\tilde{\mathbf{r}}_0\right]^T\left[\mathbf{R}_k(\mathbf{A})\mathbf{r}_0\right] = \tilde{\mathbf{r}}_0^T\left[\mathbf{R}_k(\mathbf{A})\right]^2\mathbf{r}_0 = \tilde{\mathbf{r}}_0^T\mathbf{r}_k^{CGS}, \tag{262}$$

where

$$\mathbf{r}_k^{CGS} = \left[\mathbf{R}_k(\mathbf{A})\right]^2 \mathbf{r}_0. \tag{263}$$

Similarly, σ_k on step 8 can be expressed as in [5] by,

$$\sigma_k = \left[\mathbf{S}_k(\mathbf{A}^T)\tilde{\mathbf{r}}_0\right]^T \mathbf{A}\left[\mathbf{S}_k(\mathbf{A})\mathbf{r}_0\right] = \tilde{\mathbf{r}}_0^T \mathbf{A}\left[\mathbf{S}_k(\mathbf{A})\right]^2 \mathbf{r}_0 = \tilde{\mathbf{r}}_0^T \mathbf{A}\mathbf{v}_k^{CGS}, \tag{264}$$

where

$$\mathbf{v}_k^{CGS} = \left[\mathbf{S}_k(\mathbf{A})\right]^2 \mathbf{r}_0. \tag{265}$$

Equation (262) and Equation (264) demonstrate that both ρ_k and σ_k can be computed without the matrix transpose [5, 119]. Sonneveld further noted from the form of these equations, that generating the polynomials $\left[\mathbf{R}_k(\mathbf{A})\right]^2$ and $\left[\mathbf{S}_k(\mathbf{A})\right]^2$ eliminates the need for generating $\tilde{\mathbf{r}}_k$ and $\tilde{\mathbf{v}}_k$ altogether. Sonneveld accomplished this goal by squaring the polynomial forms of the BCG update formulae. The corresponding matrix form of the resulting CGS algorithm is listed in Equation (266) [5]. Note that like BCG, the CGS algorithm is still based on short vector recurrence relationships. However, unlike BCG, the matrix transpose does not appear within the CGS algorithm. The negative effect of the squaring process, however, is a worsening of the erratic convergence behavior exhibited by BCG. In many instances, this erratic convergence behavior does not affect the final convergence of CGS, and in fact when CGS converges it converges roughly twice as fast as BCG. In other cases, though, van der Vorst has shown that the erratic convergence behavior of CGS can result in a loss of orthogonality among the updated residuals [6, 80], which can lead to inaccurate solutions.

CGS Algorithm
1) *Choose* $\mathbf{x}_0$, $\tilde{\mathbf{r}}_0$ $(\tilde{\mathbf{r}}_0 \neq 0)$
2) *Compute* $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
3) *Set*: $\mathbf{q}_0 = \mathbf{p}_{-1} = 0$, $\rho_{-1} = 1$
For $k = 0,1,2,...$
4) $\rho_k = \tilde{\mathbf{r}}_0^T \mathbf{r}_k$
5) $\beta_k = \dfrac{\rho_k}{\rho_{k-1}}$
6) $\mathbf{u}_k = \mathbf{r}_k + \beta_k \mathbf{q}_k$
7) $\mathbf{p}_k = \mathbf{u}_k + \beta_k(\mathbf{q}_k + \beta_k \mathbf{p}_{k-1})$
8) $\mathbf{v}_k = \mathbf{A}\mathbf{p}_k$
9) $\sigma_k = \tilde{\mathbf{r}}_0^T \mathbf{v}_k$
10) $\alpha_k = \dfrac{\rho_k}{\sigma_k}$
11) $\mathbf{q}_{k+1} = \mathbf{u}_k - \alpha_k \mathbf{v}_k$
12) $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}(\mathbf{u}_k + \mathbf{q}_{k+1})$
13) $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k(\mathbf{u}_k + \mathbf{q}_{k+1})$
14) *If* $\|\mathbf{r}_{k+1}\|_2 <$ *tolerance then quit*

(266)

A4.3. The Bi-CGSTAB Algorithm

The Bi-CGSTAB algorithm was developed in an effort to obtain more smoothly convergent CGS-like solutions [6]. van der Vorst observed that the BCG polynomial relations could be modified in ways other than squaring to yield a different transpose-free algorithm. In contrast to the polynomial form of the CGS residual in Equation (263), the polynomial form of the Bi-CGSTAB residual is given by [6],

$$\mathbf{r}_k^{Bi-CGSTAB} = \mathbf{T}_k(\mathbf{A})\mathbf{R}_k(\mathbf{A})\mathbf{r}_0, \tag{267}$$

where $\mathbf{T}_k(\mathbf{A})$ is a polynomial of maximum degree k that is updated on each iteration according to,

$$\mathbf{T}_k(\mathbf{A}) = (\mathbf{I} - \eta_k \mathbf{A})\mathbf{T}_{k-1}(\mathbf{A}) = (\mathbf{I} - \eta_1 \mathbf{A})(\mathbf{I} - \eta_2 \mathbf{A})...(\mathbf{I} - \eta_k \mathbf{A}). \tag{268}$$

The weighting parameter, η_k, at each step is chosen so as to minimize

$$\left\| \mathbf{r}_k^{Bi-CGSTAB} \right\|_2 = \left\| (\mathbf{I} - \eta_k \mathbf{A})\mathbf{T}_{k-1}(\mathbf{A})\mathbf{R}_k(\mathbf{A})\mathbf{r}_0 \right\|_2, \tag{269}$$

which can be interpreted as a local steepest descent step [6]. The resulting matrix form of the Bi-CGSTAB algorithm is listed below [6]:

$$\begin{aligned}
&\textit{Bi-CGSTAB Algorithm} \\
&1)\ \textit{Choose}\ \mathbf{x}_0,\ \tilde{\mathbf{r}}_0\ (\tilde{\mathbf{r}}_0 \neq 0) \\
&2)\ \textit{Compute}\ \mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0 \\
&3)\ \textit{Set:}\ \rho_0 = \alpha = \omega_0 = 1,\ \mathbf{v}_0 = \mathbf{p}_0 = 0 \\
&\textit{For}\ k = 1,2,3,... \\
&\quad 4)\ \rho_k = \tilde{\mathbf{r}}_0^T \mathbf{r}_{k-1} \\
&\quad 5)\ \beta_k = \frac{\rho_k}{\rho_{k-1}} \frac{\alpha}{\omega_{k-1}} \\
&\quad 6)\ \mathbf{p}_k = \mathbf{r}_{k-1} + \beta_k (\mathbf{p}_{k-1} - \omega_{k-1}\mathbf{v}_{k-1}) \\
&\quad 7)\ \mathbf{v}_k = \mathbf{A}\mathbf{p}_k \\
&\quad 8)\ \alpha = \frac{\rho_k}{\tilde{\mathbf{r}}_0^T \mathbf{v}_k} \\
&\quad 9)\ \mathbf{s} = \mathbf{r}_{k-1} - \alpha \mathbf{v}_k \\
&\quad 10)\ \mathbf{t} = \mathbf{A}\mathbf{s} \\
&\quad 11)\ \omega_k = \frac{\mathbf{t}^T \mathbf{s}}{\mathbf{t}^T \mathbf{t}} \\
&\quad 12)\ \mathbf{x}_k = \mathbf{x}_{k-1} + \alpha \mathbf{p}_k + \omega_k \mathbf{s} \\
&\quad 13)\ \mathbf{r}_k = \mathbf{s} - \omega_k \mathbf{t} \\
&\quad 14)\ \textit{If}\ \left\| \mathbf{r}_k \right\|_2 < \textit{tolerance then quit}
\end{aligned} \tag{270}$$

Note that the steepest descent steps used by Bi-CGSTAB typically result in much smoother convergence than that exhibited by CGS.

A4.4. The Transpose-Free Quasi-Minimal Residual Algorithm (TFQMR)

The transpose-free quasi-minimal residual algorithm of Freund [7] is another Lanczos-base Krylov algorithm that tries to control the erratic convergence behavior of the CGS algorithm. Recall that one fundamental drawback of the Lanczos based methods is the lack of a true minimization property. Optimality is thereby sacrificed to obtain short vector recurrence relationships. In fact, the Faber-Manteuffel theorem [129] proves that this sacrifice is necessary because one cannot in general satisfy an optimality condition (minimization of the residual with respect to a fixed norm) for nonsymmetric systems using short vector recurrence relationships. Previously, the Bi-CGSTAB algorithm applied local steepest descent steps in order to control the erratic CGS convergence behavior. Unfortunately, however, Bi-CGSTAB residual norms can still undergo considerable oscillations. This observation motivated Freund and Nachtigal to develop the quasi-minimal residual algorithm (QMR) [181], which applies the quasi-minimization technique to the BCG algorithm (as described below). Since the development of QMR, the quasi-minimization technique has been used to develop a whole family of QMR-like algorithms [see 7, 120, 182, 183, 187, and 188]. Among the algorithms of this family, the transpose-free quasi-minimal residual algorithm (TFQMR) [7], which is generally less expensive per iteration than the other transpose-free algorithms of this family [120], is specifically considered.

The basic idea of quasi-minimization is the same as the least squares minimization technique used in deriving the GMRES algorithm in Section A3.2, with one very important difference, as will be seen shortly. From Equation (256) the norm of the BCG residual can be written as,

$$\begin{aligned}\|\mathbf{r}_k\|_2 &= \|\mathbf{r}_0 - \mathbf{A}\mathbf{P}_k\mathbf{y}_k\|_2 \\ &= \left\|\left(\|\mathbf{r}_0\|_2\mathbf{P}_{k+1}\mathbf{e}_1 - \mathbf{P}_k\mathbf{T}_k\mathbf{y}_k - \mathbf{p}_{k+1}\mathbf{e}_k^T\mathbf{y}_k\right)\right\|_2 \\ &= \left\|\mathbf{P}_{k+1}\left(\|\mathbf{r}_0\|_2\mathbf{e}_1 - \tilde{\mathbf{T}}_k\mathbf{y}_k\right)\right\|_2,\end{aligned} \tag{271}$$

where

$$\mathbf{T}_k = \begin{bmatrix} \alpha_1 & \beta_2 & 0 & . & . & 0 \\ 1 & \alpha_2 & \beta_3 & & & . \\ 0 & . & . & . & & . \\ . & & . & . & . & 0 \\ . & & & . & . & \beta_k \\ . & . & . & 0 & 1 & \alpha_k \\ 0 & . & . & . & 0 & 1 \end{bmatrix}. \tag{272}$$

Notice the strong similarities between Equation (271) and Equation (239), which corresponds the GMRES least squares problem. The obvious difference between these two equations is the appearance of $\tilde{\mathbf{T}}_k$ in Equation (271) instead of $\tilde{\mathbf{H}}_k$. The more important difference, however, is that in the GMRES case $\mathbf{P}_{k+1}$ has orthonormal columns and so $\|\mathbf{P}_{k+1}\|_2 = 1$. This condition is no longer true in Equation (271), because the columns of $\mathbf{P}_{k+1}$ are generated via the nonsymmetric Lanczos process and not the Arnoldi process. Using the multiplicative property of matrix norms [126], Equation (271) can be expressed by the following inequality,

$$\|\mathbf{r}_k\|_2 \le \left\|\left(\|\mathbf{r}_0\|_2\mathbf{e}_1 - \tilde{\mathbf{T}}_k\mathbf{y}_k\right)\right\|_2 \|\mathbf{P}_{k+1}\|_2. \tag{273}$$

The idea behind the quasi-minimization technique is to minimize the coefficient of $\|\mathbf{P}_{k+1}\|_2$ in Equation (273) by solving a $(k+1)$xk least squares problem [181]. This least squares problem is very similar to the one solved by GMRES with $\tilde{\mathbf{H}}_k$ replaced by $\tilde{\mathbf{T}}_k$. Consequently, since $\tilde{\mathbf{T}}_k$ is a simple tridiagonal matrix, this least squares problem can be

solved very efficiently by QR factorization techniques [126]. Thus, the QMR update is obtained from Equation (215) after $\mathbf{y}_k$ is determined from the above mentioned least squares minimization problem [141, 181]. This determination of $\mathbf{y}_k$ in effect replaces step 9 of the BCG algorithm in Equation (258). Consequently, the QMR algorithm is not derived via the orthogonal residual approach as were the previous Lanczoz-based algorithms. As a result, the QMR algorithm is not susceptible to the breakdown discussed in Section A4.1 that occurs when a unique iterate satisfying Equation (218) does not exist. However, QMR is still susceptible to breakdown caused by the nonsymmetric Lanczos process itself. Additionally, the QMR algorithm still requires working with the matrix transpose since only the determination of $\mathbf{y}_k$ is different from the original BCG algorithm.

The QMR least squares problem minimizes the coefficient of $\|\mathbf{P}_{k+1}\|_2$ in Equation (273) but it does not strictly minimize the residual in the $L_2 - norm$ or any other fixed norm. As a result, QMR is not a minimal residual type algorithm according to the strict definition presented in Section A2.1. However, Manteuffel shows that QMR actually minimizes the residual in a norm that varies with the iteration number; i.e., QMR minimizes [see 119, 141],

$$\|\mathbf{D}_k^{-1}\mathbf{W}_k^T\mathbf{r}_k\|_2. \tag{274}$$

Recall from the discussion at the beginning of Section A4 that the columns of $\mathbf{W}_k$ span $K_k(\mathbf{r}_0,\mathbf{A}^T)$, and that $\mathbf{D}_k$ is defined by Equation (244). At first glance, this may seem to violate the Faber-Manteuffel theorem [129]. However, Manteuffel points out that this theorem applies only to a *fixed* norm and not to one that varies with the iteration [141]. Consequently, Barth and Manteuffel classify QMR as a *variable metric* method [124].

Freund combined the underlying ideas of CGS with the quasi-minimization technique described above to develop a new transpose-free algorithm, namely the transpose-free quasi-minimal residual algorithm (TFQMR). This algorithm is listed in Equation (275) [see 7,

189]. The TFQMR algorithm uses the same search directions generated by CGS, but then applies the quasi-minimal residual technique to determine $\mathbf{y}_k$. As a result, TFQMR is *not* mathematically equivalent to the standard QMR algorithm described above [181]. Another interesting feature of TFQMR is that it actually produces two solution updates on each iteration. This feature arises because the squaring process used by CGS actually yields two search directions, $\mathbf{u}_k$ and $\mathbf{q}_{k+1}$, on each CGS iteration. However, CGS computes only one solution update given by step 13 of the CGS algorithm in Equation (266). TFQMR actually makes use of both search directions to produce two solution iterates. This feature appears on steps 13-20 of the TFQMR algorithm in Equation (275). Note that steps 4-12 of TFQMR are very similar to steps 4-12 of the CGS algorithm given in Equation (266). Thus, TFQMR can be viewed as an extension or modification of the CGS algorithm that yields more smoothly convergent CGS-like solutions.

Note the conspicuous absence of an update expression for the residual associated with the solution update in the TFQMR algorithm expressed in Equation (275). Unfortunately, the residual is not readily available in the TFQMR algorithm. However, an estimate for the upper bound of the residual is available as indicated in parenthesis on step 19. This upper bound may be used to monitor convergence of the algorithm. If this upper bound is not used, then the residual must be computed on each iteration, which is significantly more expensive. In practice, a combination of both options may be necessary to monitor convergence of TFQMR.

TFQMR Algorithm

1) *Choose* $\mathbf{x}_0$, $\tilde{\mathbf{r}}_0$ $(\tilde{\mathbf{r}}_0 \neq 0)$

2) *Compute* $\mathbf{r}_0^{CGS} = \mathbf{b} - \mathbf{A}\mathbf{x}_0$

3) *Set:* $\mathbf{q}_0 = \mathbf{d}_0 = \mathbf{p}_{-1} = 0,\ \rho_{-1} = 1,\ \nu_0 = \eta_0 = 0,\ \tau_0 = \left\|\mathbf{r}_0^{CGS}\right\|_2$

For $k = 0,1,2,\ldots$

4) $\rho_k = \tilde{\mathbf{r}}_0^T \mathbf{r}_k^{CGS}$

5) $\beta_k = \dfrac{\rho_k}{\rho_{k-1}}$

6) $\mathbf{u}_k = \mathbf{r}_k^{CGS} + \beta_k \mathbf{q}_k$

7) $\mathbf{p}_k = \mathbf{u}_k + \beta_k(\mathbf{q}_k + \beta_k \mathbf{p}_{k-1})$

8) $\mathbf{v}_k = \mathbf{A}\mathbf{p}_k$

9) $\sigma_k = \tilde{\mathbf{r}}_0^T \mathbf{v}_k$

10) $\alpha_k = \dfrac{\rho_k}{\sigma_k}$

11) $\mathbf{q}_{k+1} = \mathbf{u}_k - \alpha_k \mathbf{v}_k$

12) $\mathbf{r}_{k+1}^{CGS} = \mathbf{r}_k^{CGS} - \alpha_k \mathbf{A}(\mathbf{u}_k + \mathbf{q}_{k+1})$

For $m = 2k+1, 2k+2$

13) $\nu_m = \begin{cases} \sqrt{\left\|\mathbf{r}_k^{CGS}\right\|_2 \left\|\mathbf{r}_{k+1}^{CGS}\right\|_2} \Big/ \tau_{m-1} & ;\ m \text{ is odd} \\ \left\|\mathbf{r}_{k+1}^{CGS}\right\|_2 / \tau_{m-1} & ;\ m \text{ is even} \end{cases}$

14) $c_m = \dfrac{1}{\sqrt{1+\nu_m}}$

15) $\tau_m = \tau_{m-1} \nu_m c_m$

16) $\eta_m = c_m^2 \alpha_k$

17) $\mathbf{d}_m = \begin{cases} \mathbf{u}_k + \dfrac{\nu_{m-1}^2 \eta_{m-1}}{\alpha_k} \mathbf{d}_{m-1} & ;\ m \text{ is odd} \\ \mathbf{q}_k + \dfrac{\nu_{m-1}^2 \eta_{m-1}}{\alpha_k} \mathbf{d}_{m-1} & ;\ m \text{ is even} \end{cases}$ (275)

18) $\mathbf{x}_m = \mathbf{x}_{m-1} + \eta_m \mathbf{d}_m$

19) *Compute* $\left\|\mathbf{r}_m\right\|_2$ *or approximate it from* $\left\|\mathbf{r}_m\right\|_2 \leq \left(\sqrt{m+1}\right)\tau_m$

20) *If* $\left\|\mathbf{r}_m\right\|_2$ < *tolerance then quit*

A5. SUMMARY

The goals of this appendix were to first motivate the use of Krylov subspace based methods for solving systems of linear equations, and then to adequately describe the various

Krylov subspace based methods available. The latter required a general description of Krylov techniques, including a discussion of the two primary approaches taken in the derivation of Krylov algorithms; namely, the minimal residual approach and the orthogonal residual approach. Examples of both were included in order to demonstrate the required processes.

Transpose-free Krylov algorithms that are applicable to general nonsymmetric linear systems were given particular attention in this work. Accordingly, true conjugate gradient methods that exhibit both optimality and economical vector recurrences were discussed only to provide a framework for discussing more generally applicable conjugate gradient-like algorithms. Consequently, the discussion progressed to more recently developed Krylov algorithms based upon both the Arnoldi process (i.e., GMRES) and the nonsymmetric Lanczos process (i.e., CGS, Bi-CGSTAB, and TFQMR). The advantages and disadvantages of these different algorithms were discussed.

Specifically, recall that the Arnoldi-based GMRES algorithm [4] was derived so as to maintain optimality, but at the expense of economical vector recurrences. Consequently, the work and storage requirements of GMRES increase with the iteration count. Therefore, practical implementations frequently require use of the restarted version, GMRES(m), where m is the maximum dimension of the Krylov subspace. The restarted algorithm is then only optimal within a cycle, and so frequent restarts can lead to slow convergence or even algorithm stall.

In contrast, the Lanczos-based algorithms were derived so as to maintain economical vector recurrences, but at the expense of optimality. Additionally, it was noted that the nonsymmetric Lanczos process itself is susceptible to breakdowns, making algorithms derived based upon this process also susceptible to breakdown. CGS was the first transpose-free algorithm of this type developed [5]. It was derived by squaring the BCG [138, 139] polynomial relations. CGS can exhibit very rapid convergence compared to BCG, but its

convergence is marred by sometimes very wild oscillations, which under certain conditions can lead to inaccurate solutions [6]. This difficulty led to the development of both Bi-CGSTAB [6], which uses local steepest descent steps, and TFQMR [7], which uses the quasi-minimization idea, to obtain more smoothly convergent CGS-like solutions.

The four algorithms discussed above were given considerable attention because of their recent popularity. This area of numerical linear algebra continues to be a very active research topic. In fact, many recent studies, in addition to those already cited, have been devoted to the improvement and investigation of these different Krylov algorithms [i.e., see 190, 191, and 192]. Thus, one can anticipate future development of new algorithms as well as the continued improvement of the algorithms described in this overview.

REFERENCES

1. White, F.M., *Viscous Fluid Flow*, McGraw-Hill, Inc., New York (1974).

2. Burmeister, L.C., *Convective Heat Transfer*, John Wiley & Sons, New York (1983).

3. Ortega, J.M. and Rheinboldt, W.C., *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, Inc., San Diego, CA. , *Computer Science and Applied Mathematics Series* (1970).

4. Saad, Y. and Schultz, M.H., "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM J Sci. Stat. Comput. 7*, 856-869 (1986).

5. Sonneveld, P., "CGS, a Fast Lanczos-type Solver for Nonsymmetric Linear Systems," *SIAM J. Sci. Stat. Comput. 10*, 36-52 (1989).

6. van der Vorst, H.A., "Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems," *SIAM J. Sci. Stat. Comput. 13*, 631-644 (1992).

7. Freund, R.W., "A Transpose-Free Quasi-Minimal Residual Algorithm for non-Hermitian Linear Systems," *SIAM J. Sci. Comput. 14*, 470-482 (1993).

8. White, F.M., *Fluid Mechanics*, McGraw-Hill, Inc. , 2nd (1986).

9. McHugh, P.R. and Knoll, D.A., "Fully Implicit Solution of the Benchmark Backward Facing Step Problem Using Finite Volume Differencing and Inexact Newton's Method." In *Benchmark Problems for Heat Transfer Codes,* Blackwell, B. and Pepper, D.W. (Eds.), 1992 ASME Winter Annual Meeting, Anaheim CA., ASME HTD-Vol. 222, Nov. 8-13 1992, pp. 77-87.

10. McHugh, P.R. and Knoll, D.A., "Inexact Newton's Method Solutions to the Incompressible Navier-Stokes and Energy Equations Using Standard and Matrix-Free Implementations." In *Proc. of 11th AIAA Computational Fluid Dynamics Conference,* Orlando, FL., AIAA-93-3332, July 6-9 1993, pp. 385-393.

11. McHugh, P.R., Knoll, D.A., and Johnson, R.W., "Fully Implicit Solutions of the Benchmark Problem Using Inexact Newton's Method." In *Computational Aspects of Heat Transfer Benchmark Problems,* Blackwell, B.F. and Armaly, B.F. (Eds.), 1993 ASME Winter Annual Meeting, New Orleans, LA., HTD-Vol. 258, Nov. 28 - Dec. 3 1993, pp. 83-91.

12. McHugh, P.R. and Knoll, D.A., "Fully Coupled Finite Volume Solutions of the Incompressible Navier Stokes and Energy Equations Using Inexact Newton's Method," *Int. J. Numer. Meth. Fluids 19*, 439-455 (1994).

13. McHugh, P.R. and Knoll, D.A., "Comparison of Standard and Matrix-Free Implementations of Several Newton-Krylov Solvers," *AIAA J. 32*, N12, 2394-2400 (Dec. 1994).

14. Johnson, R.W., McHugh, P.R., and Knoll, D.A., "Defect Correction with a Fully Coupled Inexact Newton Method," *Numer. Heat Transfer., Part B 26*, 173-188 (1994).

15. Knoll, D.A., McHugh, P.R., and Mousseau, V.A., "Newton-Krylov-Schwarz Methods Applied to the Tokamak Edge Plasma Fluid Equations." In *Domain-Based Parallelism and Problem Decomposition Methods in Computational Science and Engineering (Minnesota Supercomputer Institute, April 25-26, 1994),* Keyes, D.E., Y.S., and Truhlar, D.G. (Eds.), Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, (to appear in 1995, approx. 300 pages), 1995.

16. McHugh, P.R., Knoll, D.A., Mousseau, V.A., and Hansen, G.A. *An Investigation of Newton-Krylov Solution Techniques for Low Mach Number Compressible flow,* accepted for presentation at ASME FED Summer Meeting, Hilton Head Island, S.C., August 13-18 1995.

17. Dongarra, J.J., Bunch, J., Moler, C., and Stewart, G., *LINPACK User's Guide*, SIAM, Philadelphia, PA., 1979.

18. Dongarra, J.J., *Performance of Various Computers Using Standard Linear Equation Software* CS-89-85, Computer Science Department, University of Tennessee, Knoxville, TN 37996-1301, February 1995.

19. Anderson, D.A., Tannehill, J.C., and Pletcher, R.H., *Computational Fluid Mechanics and Heat Transfer*, Hemisphere Publishing Corp., New York (1984).

20. Roache, P.J., *Computational Fluid Dynamics*, Hermosa Publishers, Albuquerque, N.M. (1982).

21. Patankar, S.V., *Numerical Heat Transfer and Fluid Flow*, Hemisphere, New York (1980).

22. Harlow, F.H. and Amsden, A.A., "A Numerical Fluid Dynamics Calculation Method for All Flow Speeds," *J. Comut. Phys. 8*, 197-213 (1971).

23. Knoll, D.A. and McHugh, P.R., "A Fully Implicit Direct Newton Solver for the Navier-Stokes Equations," *Int. J. Num. Meth. Fluids 17*, 449-461 (1993).

24. MacArthur, J.W. and Patankar, S.V., "Robust Semidirect Finite Difference Methods for Solving the Navier-Stokes and Energy Equations," *Int. J. Num. Meth. Fluids 9*, 325-340 (1989).

25. MacArthur, J.W. *Development and Implementation of Robust Direct Finite-Difference Methods for the Solution fo Strongly Coupled Elliptic Transport*, Ph.D. dissertation, University of Minnesota, 1986.

26. Schreiber, R. and Keller, H.B., "Driven Cavity Flows by Efficient Numerical Techniques," *J. of Comput. Phys. 49*, 310-333 (1983).

27. Fornberg, B., "Steady Viscous Flow Past a Circular Cylinder up to Reynolds Number 600," *J. Comp. Physics 61*, 297 (1985).

28. Jackson, C.P., "A Finite-Element Study of the Onset of Vortex Shedding in Flow Past Variously Shaped Bodies," *J. Fluid Mech. 182*, 23-45 (1987).

29. Vanka, S.P. and Leaf, G.K., *Fully-Coupled Solution of Pressure-Linked Fluid-Flow Equations*, Technical Report ANL-83-73, Argonne National Laboratory 1983.

30. Vanka, S.P., "Block-Implicit Calculation of Steady Turbulent Recirculating Flows," *Int. J. Heat Mass Transfer 28*, 2093-2103 (1985).

31. Landau, L.D. and Lifshitz, E.M., *Fluid Mechanics*, Pergamon Press, Elmsford, N.Y., Vol. 6, 2nd (1987).

32. Bailey, H.E. and Beam, R.M., "Newton's Method Applied to Finite-Difference Approximations for the Steady-State Compressible Navier-Stokes Equations," *J. Comput. Phys. 93*, 108-127 (1991).

33. Venkatakrishnan, V., "Newton Solution of Inviscid and Viscous Problems," *AIAA J. 27*, 885- 891 (1989).

34. Venkatakrishnan, V., "Viscous Computations Using a Direct Solver," *Computers & Fluids 18*, 191-204 (1990).

35. Gustafsson, B. and Wahlund, P., "Finite-Difference Methods for Computing the Steady Flow about Blunt Bodies," *J. Comp. Physics 36*, 327 (1980).

36. Beam, R.M. and Warming, R.F., "An Implicit Factored Scheme for the Compressible Navier-Stokes Equations," *AIAA J. 16*, 393-401 (1978).

37. Zienkiewicz, O.C., *The Finite Element Method*, McGraw-Hill, Inc., London , 3rd ed. (1977).

38. Habashi, W.G., Robichaud, M., Nguyen, V.N., Ghaly, W.S., Fortin, M., and Liu, J.W.H., "Large-Scale Computational Fluid Dynamics by the Finite Element Method," *Int. J. Numer. Meth. Fluids 18*, 1083-1105 (1994).

39. Peeters, M.F., Habashi, W.G., Nguyen, B.Q., and Kotiuga, P.L., "Finite Element Solutions of the Navier-Stokes Equations for Compressible Internal Flows," *J. Propulsion 8*, N1, 192-198 (1990).

40. Smooke, M.D., "Solution of Burner-Stabilized Premixed Laminar Flames by Boundary Value Methods," *J. Comp. Phys. 48*, 72-105 (1982).

41. Smooke, M.D., "Error Estimate for the Modified Newton Method with Applications to the Solution of Nonlinear, Two-Point Boundary-Value Problems," *Journal of Optimization Theory and Applications 39*, 489-511 (1983).

42. Smooke, M.D., Miller, J.A., and Kee, R.J., *Solution of Premixed and Counterflow Diffusion Flame Problems by Adaptive Boundary Value Methods,* Birkhuser Boston Inc., Vol. 5 (1985).

43. Puri, K.I., Seshadri, K., Smooke, M.D., and Keyes, D.E., "A Comparison Between Numerical Calculations and Experimental Measurements of the Structure of a Counterflow Methane-Air Diffusion Flame," *Combust. Sci. and Tech. 56*, 1-22 (1987).

44. Knoll, D.A., Prinja, A.K., and Campbell, R.B., "A Direct Newton Solver for the Two-Dimensional Tokamak Edge Plasma Fluid Equations," *J. Comput. Phys. 104*, 418-426 (1993).

45. Iliev, O.P., Makarov, M.M., and Vassilevski, P.S., "Performance of Certain Iterative Methods in Solving Implicit Difference Schemes for 2-D Navier-Stokes Equations," *Int. J. for Numerical Methods in Engineering 33*, 1485-1479 (1992).

46. Langtangen, H.P., "Conjugate Gradient Methods and ILU Preconditioning of Non-Symmetric Matrix Systems with Arbitrary Sparsity Patterns," *Int. J. Numer. Meth. Fluids 9*, 213-233 (1989).

47. Natarajan, R., "A Numerical Method for Incompressible Viscous Flow Simulation," *J. Comput. Phys. 100*, 384-395 (1992).

48. Noll, B. and Wittig, S., "Generalized Conjugate Gradient Method for the Efficient Solution of Three-Dimensional Fluid Flow Problems," *Numerical Heat Transfer, Part B 20*, 207 - 221 (1991).

49. Reddy, M.P., Reddy, J.N., and Akay, H.U., "Penalty Finite Element Analysis of Incompressible Flows Using Element by Element Solution Algorithms," *Computer Meth. Appl. Mech. and Eng. 100*, 169-205 (1992).

50. Reddy, J.N., *An Introduction to the Finite Element Method*, McGraw-Hill, Inc., New York (1993).

51. Reddy, M.P., Reifschneider, L.G., Reddy, J.N., and Akay, H.U., "Accuracy and Convergence of Element-By-Element Iterative Solvers For Incompressible Fluid Flows Using Penalty Finite Element Model," *Int. J. Numer. Meth. in Fluids 17*, 1019-1033 (1993).

52. Vincent, C. and Boyer, R., "A Preconditioned Conjugate Gradient Uzawa-Type Method for the Solution of the Stokes Problem by Mixed Q1-P0 Stabilized Finite Elements," *Int. J. Numerical Methods in Fluids 14*, 289 - 298 (1992).

53. Joly, P. and Eymard, R., "Preconditioned Biconjugate Gradient Methods for Numerical Reservoir Simulation," *J. of Comput. Physics 91*, 298 - 309 (1990).

54. Jordan, S.A., "An Iterative Scheme for Numerical Solution of Steady Incompressible Viscous Flows," *Computers Fluids 21*, 503-517 (1992).

55. Sjögreen, B., "Iterative Methods for Stationary Solutions to the Steady-State Compressible Navier-Stokes Equations," *Computers Fluids 21*, 627-645 (1992).

56. Ern, A. and Smooke, M.D., "Vorticity-Velocity Formulation for Three-Dimensional Steady Compressible Flows," *J. Comput. Phys. 105*, 58-71 (1993).

57. Smooke, M.D. and Giovangigli, V., "Numerical Modeling of Axisymmetric Laminar Diffusion Flames by a Parallel Boundary Value Method," *Int. J. Supercomputer Appl. 5*, 72-105 (1982).

58. Xu, Y. and Smooke, M.D., "Application of a Primitive Variable Newton's Method for the Calculation of an Axisymmetric Laminar Diffusion Flame," *J. Comput. Phys. 104*, 99-109 (1993).

59. Chorin, A.J., "A Numerical Method for Solving Incompressible Viscous Flow Problems," *J. Comput. Phys. 2*, 12-26 (1967).

60. Peyret, R., "Unsteady Evolution of a Horizontal Jet in a Stratified Fluid," *J. Fluid Mech. 78*, 49-63 (1976).

61. Briley, W.R. and McDonald, H., "On the Structure and Use of Linearized Block Implicit Schemes," *J. Comput. Phys. 34*, 54-73 (1980).

62. Stone, H.L., "Iterative Solution of Implicit Approximations of Multidimensional Partial Differential Equations," *SIAM J. Numer. Anal. 5*, 530 (1968).

63. Hageman, L.A. and Young, D.M., *Applied Iterative Methods*, Academic Press, San Diego, CA. , *Computer Science and Applied Mathematics Series* (1981).

64. Howard, D., Connolley, W.M., and Rollett, J.S., "Unsymmetric Conjugate Gradient Methods and Sparse Direct Methods in Finite Element Flow Simulation," *Int. J. Numer. Meth. Fluids 10*, 925-945 (1990).

65. Clift, S.S. and Forsyth, P.A., "Linear and Non-Linear Iterative Methods for the Incompressible Navier-Stokes Equations," *Int. J. Numer. Meth. Fluids 16*, 229-256 (1994).

66. J.E. Dennis, J. and Schnabel, R.R., *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall Inc., Englewood Cliffs, N.J. (1983).

67. Brown, P.N. and Saad, Y., "Hybrid Krylov Methods for Nonlinear Systems of Equations," *SIAM J. Sci. Stat. Comput. 11*, 450-481 (1990).

68. Carey, G.F., Wang, K.C., and Joubert, W.D., "Performance of Iterative Methods for Newtonian and Generalized Newtonian Flows," *Int. J. Numer. Meth. Fluids 9*, 127-150 (1989).

69. Eisenstat, S.C. and Walker, H.F. *Globally Convergent Inexact Newton Methods*. to appear in SIAM J. Optimization.

70. Walker, H.F., "A GMRES-Backtracking Newton Iterative Method." In *Proc. of the Copper Mountain Conference on Iterative Methods*, April 1992.

71. Zhou, L. *Krylov Subspace Methods for Linear and Nonlinear Systems*, Ph.D. dissertation, Utah State University, 1993.

72. Blue, J., "Robust Methods for Solving Systems of Nonlinear Equations," *SIAM J. Sci. Stat. Comput. 1*, 22-33 (1980).

73. Papadrakakis, M. and Pantazopoulos, G., "A Survey of Quasi-Newton Methods with Reduced Storage," *Int. J. Numer. Meth. Engineering 36*, 1573-1596 (1993).

74. Broyden, C.G., "A Class of Methods for Solving Nonlinear Simultaneous Equations," *Math. Comp. 19*, 577-593 (1965).

75. Edwards, J.R. and McRae, D.S., "Nonlinear Relaxation/Quasi-Newton Algorithm for the Compressible Navier-Stokes Equations," *AIAA J. 31*, 57 - 60 (1993).

76. Chin, P., D'Azevedo, E.F., Forsyth, P.A., and Tang, W.-P., "Preconditioned Conjugate Gradient Methods for the Incompressible Navier-Stokes Equations," *Int. J. Num. Meth. Fluids 15*, 273-295 (1992).

77. Chin, P. and Forsyth, P.A., "A Comparison of GMRES and CGSTAB Accelerations for Incompressible Navier-Stokes Problems," *J. of Computational and Applied Mathematics 46*, 415-426 (1993).

78. Gropp, W.D. and Keyes, D.E., "Domain Decomposition Methods in Computational Fluid Dynamics," *Int. J. Numer. Meth. Fluids 14*, 147-165 (1992).

79. Dahl, O. and Wille, S.Ø., "An ILU Precondioner with Coupled Node Fill-In for Iterative Solution of the Mixed Finite Element Formulation of the 2D and 3D Navier-Stokes Equations," *Int. J. Numer. Meth. Fluids 15*, 525-544 (1992).

80. Curfman, L.V. *Solution of Convective-Diffusive Flow Problems with Newton-Like Methods*, Ph.D. dissertation, University of Virginia, 1993.

81. Einset, E.O. and Jensen, K.F., "A Finite Element Solution of Three-Dimensional Mixed Convection Gas Flows in Horizontal Channels Using Preconditioned Iterative Matrix Methods," *Int. J. Numer. Meth. in Fluids 14*, 817-841 (1992).

82. Hood, P., "Frontal Solution Program for Unsymmetric Matrices," *Int. J. Num. Meth Eng. 10*, 379-399 (1976).

83. Edwards, W.S., Tuckerman, L.S., Friesner, R.A., and Sorensen, D.C., "Krylov Methods for the Incompressible Navier-Stokes Equations," *J. Comput. Phys. 110*, 82-102 (1994).

84. Venkatakrishnan, V., "Preconditioned Conjugate Gradient Methods for the Compressible Navier-Stokes Equations," *AIAA J. 29*, 1092-1100 (1991).

85. Venkatakrishnan, V. and Mavriplis, D.J., "Implicit Solvers for Unstructured Meshes," *J. Comput. Phys. 105*, 83-91 (1993).

86. Ajmani, K. and Liou, M.S., "Generalized Conjugate-Gradient Methods for the Navier-Stokes Equations." In *Proc. of 10th AIAA Computational Fluid Dynamics Conference,* Honolulu, Hawaii, AIAA Paper 93-881, June 24-26 1991.

87. Ajmani, K., Ng, W.F., and Liou, M.S., *Preconditioned Conjugate-Gradient Methods for Low-Speed Flow Calculations*, NASA Tech. Memorandum 105929, ICOMP-92-22, Institute for Computational Mechanics in Propulsion (ICOMP), NASA Lewis Research Center, Cleveland, OH., (AIAA-93-0881) 1993.

88. Ajmani, K., Liou, M.S., and Dyson, R.W., *Preconditioned Implicit Solvers for the Navier-Stokes Equations on Distributed-Memory Machines*, NASA Tech. Memorandum 106449, ICOMP-93-49, Institute for Computational Mechanics in Propulsion (ICOMP), NASA Lewis Research Center, Cleveland, OH., (AIAA-94-0408) 1994.

89. Orkwis, P. and George, J.H., "A Comparison of CGS Preconditioning Methods for Newton's Method Solvers." In *Proc. 11th AIAA CFD Conference,* Orlando, FL., AIAA paper 93-3327, July 6-9 1993.

90. Orkwis, P., "Comparison of Newton's and Quasi-Newton's Method Solvers for Navier-Stokes Equations," *AIAA J. 31,* 832-836 (1993).

91. Cai, X.C., Gropp, W.D., Keyes, D.E., and Tidriri, M.D., "Parallel Implicit Methods for Aerodynamics." In *Proc. of the 7th Int. Conf. on Domain Decomposition Methods in Scientific and Engineering Computing,* The Pennsylvania State University, Oct. 27-30 1993.

92. Cai, X.C., Gropp, W.D., Keyes, D.E., and Tidriri, M.D., "Newton-Krylov-Schwarz Methods in CFD." In *Proc. of the Int. Workshop on Num. Meth. for the Navier-Stokes Equations,* Hebeker, F. and Rannacher, R. (Eds.), Vieweg Verlag, Brainishwig, Notes on Numerical Fluid Mechanics, 1994, pp. 17-30.

93. Nielsen, E.J. *Application of Newton-Krylov Methodology to a Three-Dimensional Unstructured Euler Code.* paper in preparation.

94. Ern, A., Giovangigli, V., Keyes, D.E., and Smooke, M.D., "Towards Polyalgorithmic Linear System Solvers for Nonlinear Elliptic Problems," *SIAM J. Sci. Comput. 15,* 681-703 (1994).

95. Keyes, D.E., "Domain Decomposition Methods for the Parallel Computation of Reacting Flows," *Computer Physics Communications 53,* 181-200 (1989).

96. Dutto, L.C., "The Effect of Ordering on Preconditioned GMRES Algorithm for Solving the Compressible Navier-Stokes Equations," *Int. J. Numer. Meth. in Eng. 36,* 457-497 (1993).

97. Dutto, L.C., Habashi, W.G., Robichaud, M., and Fortin, M., "A Parallel Strategy for the Solution of the Fully-Coupled Compressible Navier-Stokes Equations." In *Advances in Finite Element Analysis in Fluid Dynamics,* Dhaubhadel, M.N., Engelman, M.S., and Habashi, W.G. (Eds.), 1993 ASME Winter Annual Meeting, New Orleans, LA., FED-Vol. 171, Nov. 28 - Dec. 3 1993.

98. Dutto, L.C., Habashi, W.G., and Fortin, M., "Parallelizable Block Diagonal Preconditioners for 3D Viscous Compressible Flow Calculations." In *Proc. of 11th AIAA Computational Fluid Dynamics Conference,* Orlando, FL., AIAA-93-3309-CP, July 6-9 1993.

99. Dutto, L.C., Habashi, W.G., and Fortin, M., "Parallelizable Block Diagonal Preconditioners for the Compressible Navier-Stokes Equations," *Comput. Methods Appl. Mech. and Engrg. 117*, 15-47 (1994).

100. Dutto, L.C., Habashi, W.G., Robichaud, M., and Fortin, M., "A Method for Finite Element Parallel Viscous Compressible Flow Calculations," *Int. J. Numer. Meth. Fluids 19*, 275-294 (1994).

101. Ramshaw, J.D. and Dukowicz, J.K., *APACHE: A Generalized-Mesh Eulerian Computer Code for Multicomponent Chemically Reactive Fluid Flow*, Technical Report LA-7427, Los Alamos Scientific Laboratory 1979.

102. McHugh, P.R. and Ramshaw, J.D., *A Computational Model for Viscous Fluid Flow, Heat Transfer, and Melting in In Situ Vitrification Melt Pools*, Technical Report EGG-WTD-9845, Idaho National Engineering Laboratory, Idaho Falls, ID. 1991.

103. Kreyszig, E., *Advanced Engineering Mathematics*, John Wiley & Sons, New York, 5th ed. (1983).

104. Gresho, P.M., "Some Current CFD Issues Relevant to the Incompressible Navier-Stokes Equations," *Computer Meth. in Appl. Mech. and Eng. 87*, 201-252 (1991).

105. Gaskell, P.H. and Lau, A.K.C., "Curvature-Compensated Convective Transport: SMART, A New Boundedness-Preserving Transport Algorithm," *Int. J. Num. Meth. Fluids 8*, 617-641 (1988).

106. Johnson, R.W. and MacKinnon, R.J., "Equivalent Versions of the Quick Scheme for Finite-Difference and Finite-Volume Numerical Methods," *Comm. Appl. Numer. Meth. 8*, 841-847 (1992).

107. Hayase, J.A., Humphrey, J.A.C., and Greif, R., "A Consistently Formulated QUICK Scheme for Fast and Stable Convergence Using Finite-Volume Iterative Calculation Procedures," *J. Comput. Phys. 98*, 108-118 (1992).

108. Leonard, B.P., "A Stable and Accurate Convective Modelling Procedure Based on Quadratic Upstream Interpolation," *Computer Methods in Applied Mechanics and Engineering 19*, 59-58 (1979).

109. de Vahl Davis, G., "Natural Convection of Air in a Square Cavity: A Benchmark Numerical Solution," *Int. J. Num. Meth. Fluids 3*, 249-264 (1983).

110. Blackwell, B.F. and Armaly, B.F. (Eds). *Computational Aspects of Heat Transfer Benchmark Problems*. 1993 ASME Winter Annual Meeting, New Orleans, Louisiana, November 28-December 3 1993.

111. Incropera, F.P. and DeWitt, D.P., *Introduction to Heat Transfer*, John Wiley & Sons, New York (1990).

112. Blackwell, B.F. and Pepper, D.W. (Eds). *Benchmark Problems for Heat Transfer Codes*. 1992 ASME Winter Annual Meeting, Anaheim, CA., November 8-13 1992.

113. Gartling, D.K., "A Test Problem for Outflow Boundary Conditions—Flow Over a Backward Facing Step," *Int. J. Num. Meth. Fluids 11*, 953-967 (1990).

114. Curtis, A.R., Powell, M.J.D., and Reid, J.K., "On the Estimation of Sparse Jacobian Matrices," *J. Inst. Maths. Applics. 13*, 117-119 (1974).

115. Gerald, C.F. and Wheatley, P.O., *Applied Numerical Analysis*, Addison-Wesley Publishing Company, Reading, Massachusetts , 3rd ed. (1984).

116. Mulder, W.A. and Leer, B.V., *Implicit Upwind Methods for the Euler Equations*, AIAA Paper 83-1930 1983.

117. Darwish, M.S., "A New High-Resolution Scheme Based on the Normalized Variable Formulation," *Numer. Heat Transfer, Part B 24*, 353-371 (1993).

118. Rubin, S.G. and Khosla, P.K., "Polynomial Interpolation Method for Viscous Flow Calculations," *J. Comput. Phys. 27*, 153-168 (1982).

119. Freund, R.W., Golub, G.H., and Nachtigal, N., "Iterative Solution of Linear Systems," *Acta Numerica* , 57-100 (1991).

120. Tong, C.H., *A Comparative Study of Preconditioned Lanczos Methods for Nonsymmetric Linear Systems*, Technical Report SAND91-8240, UC-404, Sandia National Laboratories Report, January 1992.

121. Averick, B.M. and Ortega, J.M., "Solutions of nonlinear Poisson-type equations," *Applied Numerical Mathematics 8*, 443-455 (1991).

122. Dembo, R.S., Eisenstat, S.C., and Steihaug, T., "Inexact Newton methods," *SIAM J. Numer. Anal. 19*, 400-408 (1982).

123. Ashby, S.F., Manteuffel, T., and Saylor, P., "A Taxonomy for Conjugate Gradient Methods," *SIAM J. Numer. Anal. 27*, 1542-1568 (1990).

124. Barth, T. and Manteuffel, T., *Variable Metric Conjugate Gradient Methods*, Center for Nonlinear Studies Newsletter LALP-94-003, Los Alamos National Lab. 1994.

125. Bramble, J.H., Leyk, Z., and Pasciak, J.E., "Iterative Schemes for Nonsymmetric and Indefinite Elliptic Boundary Value Problems," *Mathematics of Computation 60*, 1-22 (1993).

126. Golub, G. and Ortega, J.M., *Scientific Computing, An Introduction with Parallel Computing*, Academic Press, Inc., New York (1993).

127. Axelsson, O., "A Survey of Preconditioned Iterative Methods for Linear Systems of Algebraic Equations," *BIT 25*, 166-187 (1985).

128. Eisenstat, S.C., Elman, H.C., and Schultz, M.H., "Variational Iterative Methods for Nonsymmetric Systems of Linear Equations," *SIAM J. Numer. Anal. 20*, 345-361 (1983).

129. Faber, V. and Manteuffel, T., "Necessary and Sufficient Conditions for the Existence of a Conjugate Gradient Method," *SIAM J. Numer. Anal. 21*, 352-362 (1984).

130. Saad, Y., "Krylov Subspace Methods for Solving Large Unsymmetric Linear Systems," *Mathematics of Computation 37*, N155, 105-126 (1981).

131. Saad, Y. and Schultz, M.H., "Conjugate Gradient-Like Algorithms for Solving Nonsymmetric Linear Systems," *Mathematics of Computation 44*, N170, 417-424 (1985).

132. Golub, G.H. and O'Leary, D.P., "Some History of the Conjugate Gradient and Lanczos Algorithms: 1948-1976," *SIAM Review 31*, 50-102 (1989).

133. Hestenes, M.R. and Stiefel, E., "Methods of Conjugate Gradients for Solving Linear Systems," *J. Res. Natl. Bur. Stand. 49*, 409-436 (1952).

134. Reid, J.K., *On the Method of Conjugate Gradients for the Solution of Large Sparse Systems of Linear Equations*, Reid, J.K. (Ed.), Academic Press: New York (1971), pp. 231-253.

135. Concus, P. and Golub, G., *A Generalized Conjugate Gradient Method for Nonsymmetric Systems of Linear Equations*, Glowinski, R. and Lions, J.L. (Eds.), Springer-Verlag: Berlin, Vol. 134 (1976), pp. 56-65.

136. Craig, E.J., "The N-Step Iteration Procedures," *J. Math. Phys. 34*, 64-73 (1955).

137. Arnoldi, W.E., "The Principal of Minimized Iterations in the Solution of Matrix Eigenvalue Problems," *Quart. Appl. Math. 9*, 17-29 (1951).

138. Lanczos, C., "Solution of Systems of Linear Equations by Minimized Iterations," *J. Res. Natl. Natl. Bur. Stand. 49*, 33-53 (1952).

139. Fletcher, R., *Conjugate Gradient Methods for Indefinite Systems*, Watson, G.A. (Ed.), Springer-Verlag: Berlin, Vol. 506 (1976), pp. 73-89.

140. Saad, Y., *SPARSKIT, A Basic Tool Kit for Sparse Matrix Computations*, RIACS Technical Report 90.20, Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center, Moffet Field, CA. 1990.

141. Ashby, S., Manteuffel, T., and Saylor, P., *Preconditioned Polynomial Iterative Methods, A Tutorial*, Universityof Colorado, Denver, CO., April 7-8, 1992.

142. Concus, P., Golub, G.H., and Meurant, G., "Block Preconditioning for the Conjugate Gradient Method," *SIAM J. Sci. Stat. Comput. 6*, 220-252 (1985).

143. Meijerink, J.A. and van der Vorst, H.A., "An Iterative Solution Method for Linear Systems of Which the Coefficient Matrix is a Symmetric M-Matrix," *Mathematics of Computation 31*, N137, 148-162 (1977).

144. Ashby, S., Manteuffel, T., and Otto, J., "A Comparison of Adaptive Chebyshev and Least Squares Polynomial Preconditioning for Hermitian Positive Definite Linear Systems," *SIAM J. Sci. Stat. Comput. 13*, 1-29 (1992).

145. Johnson, O.G., Micchelli, C.A., and Paul, G., "Polynomial Preconditioners for Conjugate Gradient Calculations," *SIAM J. Numer. Anal. 20*, 362-376 (1983).

146. Joubert, W., "A Robust GMRES-Based Adaptive Polynomial Preconditioning Algorithm for Nonsymmetric Linear Systems," *SIAM J. Sci Comput. 15*, 427-439 (1994).

147. Manteuffel, T. and Otto, J., "Optimal Equivalent Preconditioners," *SIAM J. Numer. Anal. 30*, 790-812 (1993).

148. Watts, J.W., "A Conjugate Gradient-Truncated Direct Method for the Iterative Solution of the Reservoir Simulation Pressure Equation," *Soc. Pet. Eng. J. 21*, 345-353 (1981).

149. Sangback, M. and Chronopoulos, A.T., "Implementation of Iterative Methods for Large Sparse Nonsymmetric Linear Systems on A Parallel Vector Machine," *Int. J. Supercomputer Applications 4*, 9-24 (1990).

150. Kershaw, D.S., "On the Problem of Unstable Pivots in the Incomplete LU-Conjugate Gradient Method," *J. Comp. Phys. 38*, 114-123 (1980).

151. Duff, I.S. and Meurant, G.A., "The Effect of Ordering on Preconditioned Conjugate Gradients," *BIT 29*, 635-657 (1989).

152. Cai, X.C. and Widlund, O.B., "Multiplicative Schwarz Algorithms for Some Nonsymmetric and Indefinite Problems," *SIAM J. Numer. Anal. 30*, 936-952 (1993).

153. Cai, X.C. and Saad, Y., *Overlapping Domain Decompositon Algorithms for General Sparse Matrices*, Preprint 93-27, Army High Performance Computing Research Center, University of Minnesota 1993.

154. Cai, X.C., Gropp, W.D., and Keyes, D.E. *A Comparison of Some Domain Decomposition and ILU Preconditioned Iterative Methods For Nonsymmetric Elliptic Problems*. to appear in *J. Numer. Lin. Alg. Applic.*

155. Dryja, M. and Widlund, O.B., "Domain Decomposition Algorithms with Small Overlap," *SIAM J. Sci. Comput. 15*, 604-620 (1994).

156. Keyes, D.E., "Domain Decomposition: A Bridge Between Nature and Parallel Computers." In *Proc. of the Symposium on Adaptive, Multilevel and Hierarchial Computation Strategies*, ASME Winter Annual Meeting, Anaheim, CA., Nov. 8-13 1992.

157. Brown, P.N. and Hindmarsh, A.C., "Matrix-Free Methods for Stiff Systems of ODE's," *SIAM J. Numer. Anal. 23*, 610-638 (1986).

158. Brown, P.N., "A Local Convergence Theory for Combined Inexact-Newton/Finite Difference Projection Methods," *SIAM J. Numer. Anal. 24*, 407-435 (1987).

159. Brown, P.N. and Hindmarsh, A.C., "Reduced Storage Matrix Methods in Stiff ODE Systems," *Applied Mathematics and Computation 31*, 40-91 (1989).

160. Gear, C.W. and Saad, Y., "Iterative Solution of Linear Equations in ODE Codes," *SIAM J. Sci. Stat. Comp. 4*, 583-601 (1983).

161. Eisenstat, S.C., Gursky, M.C., Schultz, M.H., and Sherman, A.H., *The Yale Sparse Matrix Package, II. Nonsymmetric Codes*, Technical Report 114, Department of Computer Science, Yale University 1978.

162. Lin, J.T., Armaly, B.F., and Chen, T.S., "Mixed Convection in Buoyancy-Assisting, Vertical Backward-Facing Step Flows," *Int. J. Heat Mass Transfer 33*, N10, 2121-2132 (1990).

163. Pletcher, R.H. and Chen, K.H., "On Solving the Compressible Navier-Stokes Equations for Unsteady Flows at Very Low Mach Numbers." In *Proc. 11th AIAA CFD Conference,* Orlando, FL, AIAA-93-3368-CP, July 6-9 1993.

164. Ramshaw, J.D. and Mousseau, V.A., "Damped Artificial Compressibility Method for Steady-State Low-Speed Flow Calculations," *Computers Fluids 20*, N2, 177-186 (1991).

165. Briley, W.R., McDonald, H., and Shamroth, S.J., "A Low Mach Number Euler Formulation and Application to Time-Iterative LBI Schemes," *AIAA Journal 21*, N10, 1467-1469 (1983).

166. Choi, D. and Merkle, C.L., "Application of Time-Iterative Schemes to Incompressible Flow," *AIAA J. 23*, N10, 1518 (October 1985).

167. Merkle, C.L. and Choi, Y.H., "Computation of Low-Speed Flow with Heat Addition," *AIAA J. 25*, N6, 831 (June 1987).

168. Shuen, J.S., Chen, K.H., and Choi, Y., "A Coupled Implicit Method for Chemical Non-Equilibrium Flows at all Speeds," *J. Comput. Phys. 106*, 306-318 (1993).

169. Turkel, E., "Preconditioned Methods for Solving the Incompressible and Low Speed Compressible Equations," *J. Comput. Phys.* *72*, 277 (1987).

170. Daniel, J.W., "The Conjugate Gradient Method for Linear and Nonlinear Operator Equations," *Numer. Math.* *10*, 10-26 (1967).

171. Chronopoulos, A.T., "Nonlinear CG-Like Iterative Methods," *J. Comput. and Appl. Math.* *40*, 73-89 (1992).

172. Chronopoulos, A.T., "Iterative Methods for Nonlinear Operator Equations," *Appl. Math. and Comput.* *51*, 167-180 (1992).

173. Peaceman, D.W. and Rachford, H.H., "The Numerical Solution of Parabolic and Elliptic Differential Equations," *J. Soc. Ind. Appl. Math.* *3*, 28 (1955).

174. Chandra, R. *Conjugate Gradient Methods for Partial Differential Equations*, Ph.D. dissertation, Computer Science Dept., Yale University, New Haven, CT., 1978.

175. Elman, H.C. *Iterative Methods for Large Sparse Nonsymmetric Systems of Linear Equations*, Ph.D. dissertation, Computer Science Department, Yale University, New Haven, CT., 1982.

176. Vinsome, P.K.W., "ORTHOMIN, An Iterative Method for Solving Sparse Sets of Simultaneous Linear Equations." In *Proc. Fourth Symposium on Reservoir Simulation,* Soc.of Petroleum Engineering of AIME, 1976, pp. 149-159.

177. Young, D.M. and Jea, K.C., "Generalized Conjugate Gradient Acceleration of Nonsymmetrizable Iterative Methods," *Linear Alg. Appl.* *34*, 159-194 (1980).

178. Saad, Y., "Krylov Subspace Methods on Supercomputers," *SIAM J Sci. Stat. Comput.* *10*, 1200-1232 (1989).

179. Widlund, O., "A Lanczos Method for a Class of Nonsymmetric Systems of Linear Equations," *SIAM J. Numer. Anal.* *15*, 801-812 (1978).

180. Axelsson, O., "Conjugate Gradient-Type Methods for Unsymmetric and Inconsistent Systems of Linear Equations," *Linear Alg. Appl.* *29*, 1-16 (1980).

181. Freund, R.W. and Nachtigal, N.M., "QMR: a Quasi-Minimal Residual Method for Non-Hermitian Linear Systems," *Numer. Math. 60*, 315-339 (1991).

182. Freund, R.W. and Szeto, T., *A Quasi-minimal Residual Squared Algorithm for non-Hermitian Linear Systems*, RIACS Technical Report 91.26, Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center, Moffett Field, CA., December 1991.

183. Freund, R.W. and Nachtigal, N.M., "An Implementation of the QMR Method Based on Coupled Two-Term Recurrences," *SIAM J. Sci. Comput. 15*, 313-337 (1994).

184. Tong, C.H., "A Family of Quasi-Minimal Residual Methods for Nonsymmetric Linear Systems," *SIAM J. Sci. Comput. 15*, 89-105 (1994).

185. Gutknecht, M.H., "Variants of BiCGSTAB for Matrices With Complex Spectrum," *SIAM J. Sci. Comput. 14*, 1020-1033 (1993).

186. Hoffman, K. and Kunze, R., *Linear Algebra*, Prentice-Hall Inc., Englewood Cliffs, New Jersey , 2nd (1971).

187. Freund, R.W., Gutknecht, M.H., and Nachtigal, N.M., "An Implementation of the Look-ahead Lanczos Algorithm for Non-Hermitian Matrices," *SIAM J. Sci. Comput. 14*, 137-158 (1993).

188. Chan, T.F., Gallopoulos, E., Simoncini, V., Szeto, T., and Tong, C.H., "A Quasi-Minimal Residual Variant of the Bi-CGSTAB Algorithm for Nonsymmetric Systems," *SIAM J. Sci. Comput. 15*, 338-347 (1994).

189. Freund, R.W., *Transpose-Free Quasi-Minimal Residual Methods for Non-Hermitian Linear Systems*, Numerical Analysis Manuscript 92-7, AT&T Bell Laboratories, Murray Hill, NJ., July 1992.

190. Shadid, J.N. and Tuminaro, R.S., "A Comparison of Preconditioned Nonsymmetric Krylov Methods on a Large-Scale MIMD Machine," *SIAM J. Sci. Comput. 15*, 440-459 (1994).

191. Brezinski, C. and Sadok, H., "Lanczos-Type Algorithms for Solving Systems of Linear Equations," *Applied Numer. Math. 11*, 443-473 (1993).

192. Pommerell, C. and Fichtner, W., "Memory Aspects and Performance of Iterative Solvers," *SIAM J. Sci. Comput. 15*, 460-473 (1994).