**Preconditioned Conjugate Gradient Methods
for the Navier-Stokes Equations**

by

Kumud Ajmani

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in
Mechanical Engineering

APPROVED:

Dr. W. F. Ng, Chairman

Dr. H. L. Moses

Dr. C. J. Ribbens

Dr. D. J. Nelson

Dr. B. Vick

November 1991

Blacksburg, Virginia

R

Preconditioned Conjugate Gradient Methods for the Navier-Stokes Equations

by

Kumud Ajmani

Dr. W. F. Ng, Chairman

Mechanical Engineering

(ABSTRACT)

A generalized Conjugate Gradient like method is used to solve the linear systems of equations formed at each time-integration step of the unsteady, two-dimensional, compressible Navier-Stokes equations of fluid flow. The Navier-Stokes equations are cast in an implicit, upwind finite-volume, flux split formulation. Preconditioning techniques are employed with the Conjugate Gradient like method to enhance the stability and convergence rate of the overall iterative method. The superiority of the new solver is established by comparisons with a conventional Line Gauss-Seidel Relaxation (LGSR) solver. Comparisons are based on 'number of iterations required to converge to a steady-state solution' and 'total CPU time required for convergence'. Three test cases representing widely varying flow physics are chosen to investigate the performance of the solvers. Computational test results for very low speed (incompressible flow over a backward facing step at Mach 0.1), transonic flow (trailing edge flow in a transonic turbine cascade) and hypersonic flow (shock-on-shock interactions on a cylindrical leading edge at Mach 6.0) are presented. For the Mach 0.1 case, speed-up factors of 30 (in terms of iterations) and 20 (in terms of CPU time) are found in favor of the new solver when compared with the LGSR solver. The corresponding speed-up factors for the transonic flow case are 20 and 18, respectively. The hypersonic case shows relatively lower speed-up factors of 5 and 4, respectively. This study reveals that preconditioning can greatly enhance the range of applicability and improve the performance of Conjugate Gradient like methods.

Acknowledgements

This research effort has been made possible by the keen guidance and constant motivation provided by my faculty advisor, Dr. Wing-fai Ng. Dr. Ng has the distinct ability of providing a relaxed working atmosphere for his graduate students, which fosters independent thinking and free exchange of ideas. Dr. Ng's comments and insights proved to be extremely valuable during various stages of this work. I thank him for sharing his professional acumen with me, and hope that I shall continue to learn from him even after I graduate.

The four additional members of my examining committee have contributed to my professional development, and have provided various suggestions for improving this work. Dr. Moses has instructed me in the classroom, and taught me the fundamentals of fluid mechanics. The numerous discussions with Dr. Ribbens have helped me develop a better understanding of the numerical techniques used in this research. Dr. Ribbens also assisted me in finding additional literature in the areas of preconditioning and conjugate gradient methods. Dr. Nelson and Dr. Vick have endeavored to broaden the scope of this work by sharing their expertise in computational techniques with me, and have helped in adding depth and perspective to this research effort.

This research was funded in part by a special grant from the Internal Fluid Mechanics Division (IFMD) of NASA Lewis Research Center. The support of

Dr. Louis A. Povinelli, Dr. Meng-Sing Liou, and the research scientists of IFMD at NASA Lewis is gratefully acknowledged. The opportunity to work on-site at NASA Lewis towards the completion of this research was indeed a unique and stimulating opportunity, and was made possible by the efforts of Dr. Meng-Sing Liou at IFMD.

The past few years in graduate school have seen me share several exciting experiences with fellow students and friends. Dr. Arthur C. Taylor III, has been an excellent friend and source of encouragement during our years together as graduate students, and even after his graduation. My association with Jerry Brock and Philip Andrew has also been extremely fruitful and pleasant. I would also like to thank all my ex-roommates who bore the brunt of my moods and temperaments, and managed to smile through most of them! Special thanks are due to my good friends Kulin Parikh, Sanjay Chawla, Gopi Maliwal and Vidhu Nagpal for their emotional support over the years.

The members of the M. D. Bery family in India have always provided constant motivation and encouragement to me, and are sincerely thanked for their support. The family of Mr. Susheel Bery has often provided me with a home away from home in the United States, and deserves a special note of thanks.

This work is dedicated to the fond memory of my father, Vijayinder Kumar Ajmani, who inspired me to come to the United States for graduate studies, but did not live long enough to see me complete this work. The patience, love and dedication of my family in India — my mother, Ruma, brother, Naresh, his wife, Kirti, and especially my sister, Charu — has been instrumental in the completion of this dissertation.

Table of Contents

1.0 Introduction and Overview 1

2.0 Literature Review 8

2.1 Review of CFD Literature 8

2.2 Review of Iterative Solvers in CFD 11

3.0 Presentation of Theory 16

3.1 Governing Equations of Fluid Flow 16

3.2 Transformation to Generalized Coordinates 17

3.3 Treatment of Explicit Terms 20

3.4 Treatment of Implicit Terms 21

 3.4.1 Importance of Delta Formulation 23

 3.4.2 Structure of the Global Matrix 23

 3.4.3 Solution procedures for the linear system 24

4.0 Iterative Methods for the Navier Stokes Equations 27

4.1 Development of Iterative Methods 27

4.2 Relaxation Schemes 28

 4.2.1 Line Gauss-Seidel Relaxation (LGSR) Schemes 29

4.3 Approximate Factorization Methods 31

4.4 LGS Relaxation and AF 33

4.5 Conjugate Gradient Methods 35

4.6 The Generalized Minimal Residual Method 36

 4.6.1 Arnoldi’s Approach for Generation of Orthogonal Vectors 36

 4.6.2 Formulation and Solution of the Minimization Problem 38

 4.6.3 Characteristics of the GMRES Algorithm 40

5.0	Preconditioning of Conjugate Gradient like Methods	43
5.1	Concepts of Preconditioning	43
5.2	Preconditioning Techniques	45
5.3	Practical Preconditioners	49
5.3.1	Diagonal Preconditioners	49
5.3.2	Approximate Inversions as Preconditioners	50
5.3.3	Iterative Schemes as Preconditioners	53
6.0	Test Results and Discussion	59
6.1	Implementation Details	59
6.2	Framework for Comparison of Solvers	63
6.2.1	Operation Counts as a Performance Index	64
6.2.2	Number of Iterations as a Performance Index	65
6.2.3	Computational Time as a Performance Index	68
6.3	Overview of Test Cases	69
6.4	Low Speed Flow over a Backward Facing Step	72
6.5	Trailing Edge Flow in a Transonic Cascade	80
6.6	Shock-on-Shock Impingement in Hypersonic Flow	88
6.7	Summary of Results	94
6.8	Discussion of Results	102
7.0	Conclusions and Recommendations	112
7.1	Conclusions from Present Work	112
7.2	Recommendations for Future Work	117
Appendix A.	Comparisons Between AF and GMRES	121
Appendix B.	Some Standard CFD Techniques	131
B.1	The Finite Volume Formulation	131
B.2	Discretization of Inviscid Flux Terms	133
B.2.1	Concepts of Interpolation	133
B.2.2	Concepts of Upwinding and Flux-Splitting	135
B.3	Discretization of Viscous Flux Terms	137
B.4	Bandwidth and Structure of Implicit Matrix	138

Appendix C. Some Standard Iterative Techniques	140
C.1 Error Analysis and Convergence	140
C.2 Line Gauss-Seidel Relaxation (LGSR) Schemes	142
Appendix D. Conjugate Gradient Methods	147
D.1 Method Of Steepest Descent	147
D.2 General Search Directions	149
D.3 A-Conjugate Search Directions	150
D.4 Conjugate Gradient Search Directions	151
D.5 Remarks on the Conjugate Gradient Method	152
Appendix E. Techniques to Reduce Computational Time	154
References	161
Vita	167

List of Illustrations

Figure 1.	Computational Cell in Generalized Coordinates	19
Figure 2.	Nine Point Stencil and Global Implicit Matrix	25
Figure 3.	Computational Grid for Backward Facing Step	74
Figure 4.	Mach Number Contours for Backward Facing Step	76
Figure 5.	Velocity Vectors for Backward Facing Step	77
Figure 6.	‘Number of Iterations’ Comparison for Backward Facing Step . .	79
Figure 7.	‘CPU Time’ Comparison for Backward Facing Step	81
Figure 8.	Experimental Setup for Transonic Cascade [60]	83
Figure 9.	Computational Grid for Transonic Cascade	85
Figure 10.	Mach Number Contours for Transonic Cascade	86
Figure 11.	Velocity Vectors around trailing edge of Transonic Cascade . .	87
Figure 12.	‘Number of Iterations’ Comparison for Transonic Cascade . . .	89
Figure 13.	‘CPU Time’ Comparison for Transonic Cascade	90
Figure 14.	‘Type II’ Pattern defined by Edney [66] (not to scale)	92
Figure 15.	Computational Grid for Hypersonic Shock	93
Figure 16.	Mach Number Contours for Hypersonic Shock	95
Figure 17.	‘Number of Iterations’ Comparison for Hypersonic Shock . . .	96
Figure 18.	‘CPU Time’ Comparison for Hypersonic Shock	97
Figure 19.	‘Number of Iterations’ Comparison for all Three Test Cases . .	104
Figure 20.	‘CPU Time’ Comparison for all Three Test Cases	105
Figure 21.	‘Number of Iterations’ with AF for Backward Facing Step . . .	123

Figure 22.	'CPU Time' Comparison with AF for Backward Facing Step . .	124
Figure 23.	'Number of Iterations' with AF for Transonic Cascade	125
Figure 24.	'CPU Time' Comparison with AF for Transonic Cascade . . .	126
Figure 25.	'Number of Iterations' with AF for Hypersonic Shock	128
Figure 26.	'CPU Time' Comparison with AF for Hypersonic Shock	129
Figure 27.	CPU time (with RELU) Comparisons for Backward Facing Step	157
Figure 28.	CPU time (with RELU) Comparisons for Transonic Cascade .	158
Figure 29.	CPU time (with RELU) Comparisons for Hypersonic Shock . .	159

List of Tables

Table 1. Comprehensive Summary of all Three Test Cases 98

Table 2. Summary of Backward Facing Step Results 99

Table 3. Summary of Transonic Cascade Results 100

Table 4. Summary of Hypersonic Shock-on-Shock Results 101

Nomenclature

a	Speed of Sound
e	Thermodynamic Internal Energy
e_0	Total Energy per unit mass
$jmax, kmax$	Dimensions of computational problem
k	Size of Krylov Subspace or number of sub-iterations
p	Static Pressure
t	Time
u, v	Components of velocity in cartesian coordinates
x, y	Cartesian coordinates
J	Jacobian matrix of coordinate transformation
Q	Vector of conserved variables
$F(Q), G(Q)$	Inviscid flux vectors in cartesian coordinates
$F_v(Q), G_v(Q)$	Viscous flux vectors in cartesian coordinates
M	Mach Number
Pr	Prandtl Number
Re	Reynolds Number
$R(Q)$	Residual vector
V	Volume (area in 2-D) of computational cell
$ $	Absolute value of quantity inside vertical bars
$ $	Norm of quantity inside vertical bars

Greek Symbols

α	Order (or size) of matrix
β	Bandwidth of matrix
ϵ	Stopping criteria for sub-iterations
γ	Ratio of Specific Heats
$\kappa()$	Condition number of matrix inside brackets
λ	Eigen-value of matrix
μ	Molecular viscosity
ω	Relaxation parameter
∂	Partial derivative operator
ϕ, κ	Operators governing spatial accuracy

ρ	Density
σ	Singular-value of matrix
τ, θ	Shear stress terms
ξ, η	Generalized Computational Coordinates
ε	Error of iterative scheme
$\varrho()$	Spectral radius of matrix inside brackets
Δ, ∇	Forward and backward difference operators
Δt	Time-step
$\Delta \xi, \Delta \eta$	Incremental change in computational coordinates

Matrices and Vectors

A, x, b	Coefficient matrix, solution vector and load vector, respectively, of matrix vector problem
$[A]$ to $[I]$	Implicit 4*4 block coefficient matrices
G	Iteration matrix
K^n, L^n	Components of global matrix
L, U, D	Lower-triangular, Upper-triangular and Diagonal matrices
M	Preconditioning matrix
V^n	Global implicit coefficient matrix

Subscripts and Superscripts

∞	freestream value
0	stagnation value, initial value
2	Euclidean norm
c	complete orthogonalization
j, k	cell-centered value
$j \pm 1/2, k \pm 1/2$	cell-face value
L	reference length
<i>mass</i>	mass-averaged value
$-, +$	split-flux vectors or jacobian matrices
-1	inverse of matrix or vector
$\wedge$	generalized coordinates
$-$	true solution value, average value
n	time level
T	transpose of matrix or vector

Abbreviations

AF	Approximate Factorization
BILUF	Block Incomplete Lower-Upper Factorization
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit

GMILU	GMRES with BILUF preconditioning
GMLUS	GMRES with LUSSOR preconditioning
GMRES	Generalized Minimal RESidual method
LGSR	Line Gauss-Seidel Relaxation
LUSSOR	Lower-Upper Successive Symmetric Over-Relaxation
PCGM	Preconditioned Conjugate Gradient Method

1.0 Introduction and Overview

The interest of engineers and scientists in studying the aspects of fluid flow problems has motivated numerous experiments and numerical simulations over the years. For a long time, experimental data has been the primary means of gaining insights into the flow physics of fluid flow over and within complex configurations. It is only in recent years that numerical simulations of fluid flow have started to assist researchers and designers in analyzing features of complex flowfields.

The rapid growth in computer technology has made large scale and detailed simulations of fluid flow a distinct reality. This has led to extremely close interactions between experimental and numerical simulations, and has established a strong foothold for the science of Computational Fluid Dynamics (CFD) as a viable research tool. CFD is widely being used to assist in the design of experiments for new design configurations. As CFD techniques become more powerful, their utility as design tools will only increase.

The major advances in CFD in the past few decades have come chiefly as a result of breakthroughs in computer hardware. For example, the development of vector architectures, and more recently, parallel computers, has revolutionized the computer industry. This has in turn resulted in immense increases in the availability

of computing power, in the form of networked micro-computers, stand-alone workstations and supercomputers. Developments in CFD software, although significant, have simply not kept pace with hardware advances. As a result, the full power of the available computing resources remains to be harnessed.

In this research, an effort has been made to contribute to the development of software for state-of-the-art CFD codes and applications. The central idea is to improve the efficiency of performing CFD simulations of complex flowfields. The research concentrates its efforts in the area of development of algorithms for efficient solutions of large, sparse, linear systems of simultaneous equations. It will be shown that this translates into dramatic improvements in the overall efficiency of obtaining numerical solutions of fluid flow problems.

This research develops techniques to obtain steady-state solutions of the 2-D compressible Navier-Stokes equations using an implicit (time integration), upwind finite-volume (spatial discretization) strategy. Implicit integration in time requires the solution of a system of simultaneous linear equations for the entire computational domain. This linear system is created repeatedly at each step of the time integration, and the successive solution of such linear systems eventually produces the solution to the non-linear problem. The success and efficiency of the overall solution procedure depends heavily on a) the ability to obtain accurate and computationally efficient solutions for each linear system, and, b) the ability to achieve the steady-state solution in the minimum number of time-steps (or global iterations), i.e., a rapid convergence rate.

The system of linear equations can be solved by direct inversion of the coefficient matrix at each time step (or global iteration). However, this requires extensive com-

puter memory and computational effort at each time step. Even on current supercomputers, direct methods are impractical for 3-D applications. Iterative schemes are an attractive alternative to direct methods because of their relatively meagre memory requirements. Iterative schemes, however, can be slow in converging from the initial guess to the final solution — a problem which has been the topic of intensive research over the years.

The efficiency and convergence of iterative schemes have received considerable attention in the field of linear algebra. The CFD community has successfully adapted linear algebra techniques like Line Gauss-Seidel Relaxation (LGSR), and developed them into solvers for use in CFD software. In fact, LGSR has spawned a family of Upwind Relaxation solvers, for use with upwind methods. Upwind Relaxation solvers are relatively simple and efficient (per time step) but suffer from several well documented problems. The various potential problems with Upwind Relaxation solvers are

- a) The solver may suffer from a slow convergence rate if the relaxation parameter (ω) is not close to its optimal value. For complex flow problems, ω is difficult to estimate accurately. In CFD practice, it may be required to choose under-relaxation ($\omega < 1$), in order to stabilize the iterative scheme. Under-relaxation tends to slow down the convergence rate of the scheme.
- b) The greatest advantage of implicit schemes is the ability to employ large time-steps to accelerate convergence to the steady state. Although the Upwind Relaxation solvers were designed to accommodate large time-steps under all flow conditions, they often become unstable (as will be shown in this research) for large time-steps. This time-step restriction of Upwind Relaxation solvers need

to be overcome.

- c) On current vector architectures, the Upwind Relaxation solver is not completely vectorizable. This degrades the performance of the solver.
- d) The solver exhibits a smoothing property in its convergence rate, i.e., as the steady state approaches, the convergence rate decreases. This is particularly true when the solver is used with low time-steps (in accordance with time-step restrictions), and is clearly observed with the three test cases in this research.

Development of new solvers should be performed with the aim of overcoming the problems listed above. In addition, some other important criteria that need to be considered are:

- a) Efficient application in 3-D simulations
- b) Application to unstructured meshes
- c) Full exploitation of the power of vector/parallel architectures
- d) Widespread use over a range of problems, i.e., the ability to solve incompressible and compressible flows with the same solver.
- e) Ability to process ill-conditioned (e.g., weak diagonal dominance) linear systems, which occur frequently in CFD applications. This is particularly essential if CFD codes are to be used with design optimization procedures to construct engineering design tools.

The above list of criteria represents some of the motivating factors for this re-

search. A survey of the existing linear algebra literature reveals the existence of several alternatives to the Upwind Relaxation solvers. None of the alternatives, however, are as comprehensive in their ability to satisfy the requirements of a general purpose solver as the family of methods referred to as the Preconditioned Conjugate Gradient Methods (PCGMs). PCGMs have been adapted (from linear algebra) in this research to

- a) Develop a solver applicable to a wide range of flow problems (incompressible, transonic and hypersonic flows)
- b) Investigate preconditioning techniques to be used with Conjugate Gradient like methods
- c) Overcome some of the weaknesses of conventional solvers (like the Upwind Relaxation solver), and thus produce efficient steady state solutions for the Navier-Stokes equations

The above objectives combine into a single focus — to develop a reliable, robust and computationally efficient solver that can overcome the deficiencies of existing solvers and meet the requirements for solvers of the future. This research is a first step towards the development of such a solver for CFD codes.

As mentioned earlier, the Conjugate Gradient Methods (CGMs) are a family of methods, with most of the members of the family being mathematically equivalent. The particular CGM adapted for application in this research is the Generalized Minimal RESidual (GMRES) method. GMRES solves the system of linear equations by transforming the given problem into a minimization problem. This minimization approach is unique compared to conventional solvers, and is performed as a number

of sub-iterations at each time step. The number of sub-iterations controls the accuracy to which the solution to the linear system is obtained, and also determines the work per iteration and the storage requirements for GMRES.

The key to achieving the objectives of this research is the investigation of preconditioning techniques as they are applied to CGMs (GMRES in particular). It must be emphasized that the overall success and viability of PCGMs is critically determined by the efficiency and effectiveness of the applied preconditioner in providing stability and accelerated convergence of the overall iterative schemes. Preconditioning serves to greatly enhance the scope and applicability of CGMs, and is indispensable when the solver is applied to ill-conditioned problems.

The preconditioned GMRES algorithm is used in three test cases. The test cases represent complex, laminar flow problems in the very low speed (incompressible), transonic speed and hypersonic speed ranges. Only laminar flow cases have been considered, in order to eliminate the complexities associated with turbulence modeling. Three problems were chosen to demonstrate that the proposed new solver is competitive for a wide range of flow speeds, and can serve as the building block for a code for all flow regimes. This is particularly significant in efforts towards developing a unified approach for incompressible and compressible flows.

In this research, the performance of the preconditioned GMRES solver will be compared in detail with that of the Upwind Relaxation (or LGSR) solver. Detailed comparisons of relative convergence rates and computational times have been made for the two solvers. Since preconditioning is an integral part of this work, efforts have concentrated on identifying stable and efficient preconditioners. The comparisons amongst various solvers, and amongst preconditioning techniques, are an attempt

to demonstrate the competitiveness of using PCGMs as standard solvers in future CFD codes.

The organization of this work is as follows. Chapter 2 presents a survey of the existing literature in the areas of CFD techniques and iterative methods. Chapter 3 details the theory and approaches used to discretize the Navier-Stokes equations. A discussion of iterative methods follows in Chapter 4, and concentrates on describing the application of the GMRES solver. Preconditioning techniques are discussed in Chapter 5. Chapter 6 describes detailed results of flow physics and solver comparisons for the three test cases. The main body of the work concludes with Conclusions and Recommendations for future work in Chapter 7. Several appendices are attached to the main body of the work and include supplemental information, results and analyses in support of this work.

2.0 Literature Review

This literature review concentrates on developments in the field of CFD and methods for solving linear systems of equations which have large, sparse coefficient matrices. Efforts have been made to limit the discussion to relevant milestones in these two areas.

2.1 *Review of CFD Literature*

The time-integration of the unsteady Navier-Stokes equations can be traced to the early work of MacCormack [1], who developed the concept of explicit schemes. Explicit schemes are extremely cheap per time step, but generally require large numbers of time-steps to reach the steady state. This is because of the stability restrictions on the allowable time step, in accordance with the Courant-Friedrichs-Lewy [2] (CFL) stability criterion. The restrictive small time steps of explicit schemes are overcome by using implicit schemes, which allow large time steps as the steady state is approached. The increased computational effort per time step of implicit schemes (i.e., solution of block tridiagonal systems) is offset by the overall gains in computational efficiency (due to larger CFL or Courant numbers) achieved by implicit schemes. The earliest work for implicit schemes is attributed to Briley

and McDonald [3] and Pulliam and Steger [4].

It should be mentioned that efforts have been (and continue to be) made to accelerate the convergence of explicit methods. The multigrid method developed by Brandt [5] has sparked most of the developments in this area. The work of Jameson [6] in multigrid acceleration of explicit schemes is also a significant contribution in this field. Recent formulations of unstructured grid methods have also greatly revived interest in explicit schemes. The literature related to explicit methods is vast, and continues to grow.

The introduction of upwind schemes in the early 1980's led to the development of a new family of methods which were very different than the existing CFD approaches of the time. The ideas of upwinding originated from the monotone schemes proposed by Godunov [7]. Although upwind schemes require two-to-three times more computational work than equivalent central difference schemes, they have an improved convergence rate which offsets the additional work per time step. In addition, upwind schemes are naturally dissipative, and hence do not require the use of adjustable parameters (like artificial viscosity) to maintain stability, as is required in a central differencing approach. Upwind schemes were developed by several authors including Chakravarthy and Osher [8], Harten et. al [9] and Mulder and Van Leer [10].

The development of upwind schemes was accompanied by introduction of techniques for incorporating upwind differencing into conservative formulations of the Navier-Stokes equations. There are three main approaches in this area — Flux Difference Splitting (FDS), Flux Vector Splitting (FVS) and Total Variation Diminishing (TVD) schemes. The ideas of FDS were developed by Roe [11], and

involve the solution of a series of Riemann problems at each cell interface. The idea of FVS is to split the flux vector into two parts such that the individual split fluxes can be stably differenced using upwind differencing. Various FVS schemes are possible, with the most popular ones ascribed to Steger and Warming [12] and Van Leer [13]. TVD schemes are characterized by highly accurate differencing of the fluxes, and were developed by Osher and Chakravarthy [14]. Recent developments for flux formulas include Essentially Non-Oscillatory (ENO) schemes. An excellent review of numerical flux formulas can be found in the survey paper of Van Leer et. al [15].

Of the various techniques described in above, the present research has adopted an implicit, upwind finite-volume approach with Van Leer's FVS scheme. The reasons for choosing implicit time integration and upwind spatial discretization have already been discussed. A finite volume (instead of a regular finite-difference or finite-element) approach is selected since it is consistent with using upwind methods in a conservative discretization of the Navier-Stokes equations. The governing equations are thus interpreted as integral conservation laws [16], and shock waves and contact discontinuities evolve as part of the solution process. Since its inception, the FVS approach has enjoyed great popularity within the CFD community. FVS has been chosen as the numerical flux formula since it is the easiest to program, has the least computational time and provides easy calculations of the exact flux jacobians (derivatives of the split fluxes with respect to conserved variables) which are required to form the implicit coefficient matrix at each time step.

2.2 *Review of Iterative Solvers in CFD*

The widespread use of vector and parallel computers in CFD has impacted on the design of iterative methods for solving large, sparse linear systems of equations. Peterson [17] has discussed and surveyed the impact of supercomputing on aerodynamic research in much detail. Ortega and Voigt [18] have also surveyed the state-of-the-art in algorithms for supercomputing applications. The interested reader is referred to these two papers for further details on this topic. Some of the major milestones in this field are described below.

Algorithms for efficient implementation of implicit schemes are motivated by reducing the computational work per time step or towards reducing the total number of iterations to convergence or a combination of reduced work per time step and an increased convergence rate. The earliest implicit algorithms were inspired by the Alternating Direction Implicit (ADI) scheme pioneered by Douglas and Gunn [19] and Peaceman and Rachford [20]. The Strongly Implicit Procedure (SIP) was introduced by Stone [21] in 1968. The SIP methods led to spatial Approximate Factorization (AF) of the coefficient matrix of the linear system, and were adapted by Beam and Warming [22] to develop the AF algorithm. The AF method of Beam and Warming [22] was one of the earliest and very successful implicit algorithms.

The advent of upwind schemes led to a revival in the development of Relaxation techniques (based on the Gauss-Seidel method) as implicit solvers. Upwind Relaxation schemes were studied by Chakravarthy [23], Van Leer and Mulder [24], Thomas and Walters [25], and Walters and Dwoyer [26], to name a few. Thomas et. al [27] performed a detailed comparison between the AF and Upwind Relaxation schemes on a vector processor, and concluded that the relative efficiency of the two solvers

depends on the nature of the problem. Since neither AF or Upwind Relaxation is universally efficient, either one can be chosen to represent the current crop of CFD solvers. The Upwind Relaxation solver has been chosen as the representative solver against which the new solver developed in this research will be compared. For the sake of completeness, comparisons with the AF solver are presented in Appendix A.

The conventional (Upwind Relaxation and AF) solvers being used in CFD have several deficiencies (as described in Chapter 1), and are not universally efficient for all classes of flow problems. This has led the CFD community to develop better solvers for implicit algorithms. Krylov Subspace methods, of which the Conjugate Gradient Methods (CGMs) are a subset, have emerged as strong candidates for use as general purpose solvers. The classical Conjugate Gradient algorithm was presented by Hestenes and Steifel [28] in 1952. However, the method did not gain popularity till the late 1960s. The revival of interest in CGMs was sparked by the work of Reid [29]. The idea of using CGMs as iterative methods to solve large, sparse linear systems of equations was first discussed by Reid [29].

The discovery and popularization of preconditioning techniques greatly contributed to the renewed interest in CGMs, and led to development of preconditioned CGMs (PCGMs). Several preconditioners were proposed by the works of Axelsson [30], Concus et. al [31] and Gustaffson [32], to name a few. The use of Incomplete LU (ILU) decompositions as preconditioners was pioneered and promoted by Meijerink and van der Vorst [33], and has served as a reliable preconditioner for use with CGMs. This research uses a block version of the Incomplete LU decomposition and modifications of the implicit scheme of Jameson and Yoon [34] as the two major preconditioners.

One major limitation of the original CGMs was that they required the underlying coefficient matrix to be symmetric and positive definite. Conjugate Gradient type methods for nonsymmetric systems were developed by Vinsome [35], Concus and Golub [36], and Widlund [37]. A unified, generalized approach for nonsymmetric systems was later presented by Young and Jea [38] and Axelsson [39]. The generalized CGM approach chosen in this research is based on the Generalized Minimal RESidual (GMRES) algorithm of Saad and Schultz [40]. In association with the concept of preconditioning, preconditioned GMRES is investigated as a general purpose solver in this research.

The ideas of preconditioning and CGMs have found use in several areas other than fluid dynamics. Applications may be found in the fields of electromagnetic scattering [41], meteorology [42], chemical engineering [43], signal processing [44], robotic manipulators [45], and finite element methods [46], to name a few. In fact, preconditioned CGMs are applicable in any area requiring efficient solutions of large, sparse linear systems of equations.

The earliest instances of the application of CGMs to fluid flow problems are found in the works of Wong and Hafez [47]. They used CGMs to solve the potential flow equations for calculations of flow over transonic airfoils. Wong [48] reported that PCGMs could be competitive with the AF solver for transonic calculations. The Incomplete LU preconditioner was employed by Wong [48] to accelerate convergence of the CGM.

Wigton et. al [49] used the GMRES algorithm to improve the robustness and convergence of existing CFD codes. The authors used GMRES to obtain efficient potential and Euler solutions for subsonic and transonic flow over airfoils. The

major preconditioner used by them is the Successive Line Over-Relaxation (SLOR) scheme. Comparisons are made between the performance of SLOR and SLOR preconditioned GMRES schemes, with the latter converging much faster than the former. The authors acknowledge that considerable extra storage is required to store the sub-iterates at each time-step, with the number of iterates varying from 20 to 80. Their paper demonstrated the possibility of using GMRES in CFD applications, and encouraged research into similar algorithms.

The work of Venkatakrishnan [50] has provided further evidence of the viability of preconditioned GMRES for the compressible Navier-Stokes equations. Venkatakrishnan's work has paralleled some of the efforts of this research in developing preconditioned GMRES for fluid flow problems. His work compares the use of preconditioned GMRES and the adaptive Chebychev algorithm [51], but does not compare the performance of preconditioned GMRES with conventional solvers. Block diagonal and Incomplete LU preconditioning have been adopted by Venkatakrishnan [50] in his work. It must be mentioned that his work was also limited to applications to subsonic and transonic flows. The number of sub-iterates requiring storage varied from 20 to 35 for most applications.

The above three paragraphs have summarized the work performed in the use of PCGMs for fluid flow problems over the past ten years. It is obvious that this powerful family of methods has not attracted the full attention of researchers working in the area of algorithm development for linear system solvers. (i) The power of generalized CGMs (like GMRES) has not been harnessed fully in the context of their applicability to all classes of flow problems (including incompressible and hypersonic flows). (ii) The development of new preconditioners, and efficient implementation of existing preconditioners needs to be undertaken. (iii) The issues of

how and when to terminate the preconditioned GMRES solver need to be examined. The termination criteria determines the number of sub-iterates at each global iteration, and directly affects the storage requirements and computational efficiency of the solver. Thus, efforts are required to limit the number of sub-iterates at each global iteration. (iv) The performance of preconditioned GMRES needs to be documented against the performance of present solvers for upwind schemes, in order to make a realistic evaluation of the power of these methods.

This research has endeavored to examine the four major points listed above. Although the afore-mentioned goals seem varied, they have all been successfully combined into a common focus in the present research effort. An attempt has thus been made to fill a gap in the existing literature by conducting a comprehensive investigation of preconditioned generalized Conjugate Gradient like methods, for solving the Navier-Stokes equations of fluid flow.

3.0 Presentation of Theory

This chapter describes some of the standard CFD techniques used in this research. The governing equations of fluid flow, and the formulation of the explicit and implicit terms are detailed here. Complete details of the finite-volume method, upwinding techniques, and the flux vector splitting method, are presented in Appendix B of this work.

3.1 Governing Equations of Fluid Flow

The complete Navier-Stokes equations governing 2-D fluid flow can be written in differential form as

$$\frac{\partial Q}{\partial t} + \frac{\partial F(Q)}{\partial x} + \frac{\partial G(Q)}{\partial y} = \frac{\partial F_v(Q)}{\partial x} + \frac{\partial G_v(Q)}{\partial y} \quad (3.1)$$

Q is the vector of independent conserved variables in equation 3.1. $F(Q)$ and $G(Q)$, which are non-linear functions of Q , and represent the inviscid flux vectors, are defined as

$$F(Q) = \begin{Bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (\rho e_0 + p)u \end{Bmatrix} \quad G(Q) = \begin{Bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (\rho e_0 + p)v \end{Bmatrix} \quad \text{with} \quad Q = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_0 \end{Bmatrix} \quad (3.2)$$

In equation 3.2 the pressure term, p , is defined by the equation of state as

$$p = (\gamma - 1)\rho e = (\gamma - 1)\rho \left\{ e_0 - \left(\frac{u^2 + v^2}{2} \right) \right\} \quad (3.3)$$

$F_v(Q)$ and $G_v(Q)$ are the viscous flux vectors, and are written as

$$F_v(Q) = \frac{1}{Re_L} \begin{Bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ u\tau_{xx} + v\tau_{xy} + \theta_x \end{Bmatrix} \quad G_v(Q) = \frac{1}{Re_L} \begin{Bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ u\tau_{xy} + v\tau_{yy} + \theta_y \end{Bmatrix} \quad (3.4)$$

The dependence of the flux vectors on Q is dropped from here on, for notational convenience. The terms τ_{xx} , τ_{xy} , τ_{yy} , θ_x and θ_y of equation 3.4 are defined as

$$\begin{aligned} \tau_{xx} &= \mu \left(\frac{4}{3} \frac{\partial u}{\partial x} - \frac{2}{3} \frac{\partial v}{\partial y} \right), \quad \tau_{xy} = \mu \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right), \quad \tau_{yy} = \mu \left(\frac{4}{3} \frac{\partial v}{\partial y} - \frac{2}{3} \frac{\partial u}{\partial x} \right) \\ \theta_x &= \mu \left(\frac{1}{Pr(\gamma-1)} \frac{\partial a^2}{\partial x} \right), \quad \theta_y = \mu \left(\frac{1}{Pr(\gamma-1)} \frac{\partial a^2}{\partial y} \right) \end{aligned} \quad (3.5)$$

For the purpose of this research, only laminar flow test cases have been considered, thus eliminating the need of dealing with turbulence modeling. The laminar molecular viscosity μ is obtained from Sutherland's law. Stokes' hypothesis for the bulk viscosity coefficient has been used. The Prandtl number Pr is assumed to be constant over the entire flowfield. The specific heat ratio, γ , is assigned a constant value of 1.4. a is the speed of sound and Re_L is the Reynolds number per unit reference length, L .

The equations above represent nondimensionalization with respect to the freestream density ρ_∞ and freestream velocity u_∞ . The physical coordinates (x,y) and the molecular viscosity μ have been nondimensionalized by a reference length L and the molecular viscosity of the freestream, μ_∞ , respectively.

3.2 Transformation to Generalized Coordinates

In order to deal with a wide variety of complex geometries and flowfields, the Navier-Stokes equations are transformed from their cartesian (x,y) coordinate form

to a generalized (ξ, η) coordinate form. This transformation greatly simplifies the discretization of the Navier-Stokes equations on arbitrary meshes. The transformed computational cell is shown in figure 1. The transformation is done by using the product chain-rule of differentiation, to convert all partial derivatives with respect to x and y into partial derivatives with respect to ξ and η as under:

$$\begin{aligned}\partial_x &= \partial_\xi \xi_x + \partial_\eta \eta_x \\ \partial_y &= \partial_\xi \xi_y + \partial_\eta \eta_y\end{aligned}\tag{3.6}$$

ξ_x, ξ_y, η_x and η_y are the metrics of the transformation from cartesian to generalized coordinates. Also, $\partial_x, \partial_y, \partial_\xi$ and ∂_η represent differentiation (of any variable) in the x, y, ξ and η directions, respectively. The complete Navier-Stokes equations in generalized coordinates are now written as

$$\frac{1}{J} \frac{\partial Q}{\partial t} + \frac{\partial \hat{F}}{\partial \xi} + \frac{\partial \hat{G}}{\partial \eta} = \frac{\partial \hat{F}_v}{\partial \xi} + \frac{\partial \hat{G}_v}{\partial \eta}\tag{3.7}$$

where $J = \xi_x \eta_y - \xi_y \eta_x$, is the determinant of the Jacobian of the transformation, and is required to be non-zero for a unique transformation.

The generalized inviscid flux vectors, $\hat{F}$ and $\hat{G}$ of equation 3.7, are defined as linear combinations of the cartesian flux vectors as

$$\hat{F} = \frac{\xi_x F}{J} + \frac{\xi_y G}{J} ; \quad \hat{G} = \frac{\eta_x F}{J} + \frac{\eta_y G}{J}\tag{3.8}$$

Similarly, the generalized viscous flux vectors are defined as

$$\hat{F}_v = \frac{\xi_x F_v}{J} + \frac{\xi_y G_v}{J} ; \quad \hat{G}_v = \frac{\eta_x F_v}{J} + \frac{\eta_y G_v}{J}\tag{3.9}$$

with $\tau_{xx}, \tau_{xy}, \tau_{yy}, \theta_x$ and θ_y of equation 3.5 being redefined for use in F_v and G_v

Figure 1. Computational Cell in Generalized Coordinates

as

$$\begin{aligned}
\tau_{xx} &= \mu \left[\frac{4}{3} \left(\frac{\partial u}{\partial \xi} \xi_x + \frac{\partial u}{\partial \eta} \eta_x \right) - \frac{2}{3} \left(\frac{\partial v}{\partial \xi} \xi_y + \frac{\partial v}{\partial \eta} \eta_y \right) \right] \\
\tau_{xy} &= \mu \left[\left(\frac{\partial u}{\partial \xi} \xi_y + \frac{\partial u}{\partial \eta} \eta_y \right) + \left(\frac{\partial v}{\partial \xi} \xi_x + \frac{\partial v}{\partial \eta} \eta_x \right) \right] \\
\tau_{yy} &= \mu \left[\frac{4}{3} \left(\frac{\partial v}{\partial \xi} \xi_y + \frac{\partial v}{\partial \eta} \eta_y \right) - \frac{2}{3} \left(\frac{\partial u}{\partial \xi} \xi_x + \frac{\partial u}{\partial \eta} \eta_x \right) \right] \\
\theta_x &= \mu \left[\frac{1}{Pr(\gamma - 1)} \left(\frac{\partial a^2}{\partial \xi} \xi_x + \frac{\partial a^2}{\partial \eta} \eta_x \right) \right] \\
\theta_y &= \mu \left[\frac{1}{Pr(\gamma - 1)} \left(\frac{\partial a^2}{\partial \xi} \xi_y + \frac{\partial a^2}{\partial \eta} \eta_y \right) \right]
\end{aligned} \tag{3.10}$$

It should be mentioned that all the results presented in this work employ the thin-layer approximation of the viscous terms. The thin-layer approximation is detailed in section B.3 of Appendix B.

3.3 Treatment of Explicit Terms

As has been stressed earlier, the main focus of this research is to devise techniques to obtain efficient steady-state solutions for equation 3.7. This equation can be rewritten as

$$\frac{1}{J} \frac{\partial Q}{\partial t} + \frac{\partial \hat{F}}{\partial \xi} + \frac{\partial \hat{G}}{\partial \eta} - \frac{\partial \hat{F}_v}{\partial \xi} - \frac{\partial \hat{G}_v}{\partial \eta} = 0 \quad \Leftrightarrow \quad \frac{1}{J} \frac{\partial Q}{\partial t} + R(Q) = 0 \tag{3.11}$$

$R(Q)$ is called the residual vector. Equation 3.11 is integrated in time till a steady-state solution is obtained. The steady state can be represented as a zero flux balance of inviscid and viscous fluxes across cell faces. For unit length of the cell faces (see Appendix B), this flux balance for each cell in the computational domain is written as

$$\begin{aligned}
(\hat{F}_{j+\frac{1}{2}} - \hat{F}_{j-\frac{1}{2}}) + (\hat{G}_{k+\frac{1}{2}} - \hat{G}_{k-\frac{1}{2}}) - (\hat{F}_{v_{j+\frac{1}{2}}} - \hat{F}_{v_{j-\frac{1}{2}}}) \\
- (\hat{G}_{v_{k+\frac{1}{2}}} - \hat{G}_{v_{k-\frac{1}{2}}}) = 0
\end{aligned} \tag{3.12}$$

Equivalently, the steady-state is achieved when the residual vector equals zero, i.e., $R(Q) = 0$. The evaluation of the residual vector thus requires the discretization of the inviscid and viscous flux derivatives. The details of the discretization have been presented in Appendix B.

3.4 Treatment of Implicit Terms

Equation 3.11 can be cast in terms of the steady state residual as

$$\frac{1}{J} \frac{\partial Q}{\partial t} = -R(Q) \quad (3.13)$$

The above equation can be integrated in time in an Euler explicit sense or an Euler implicit sense. Explicit schemes are computationally simple, but suffer from a limited time step restriction in order to maintain stability. Implicit schemes are attractive alternatives since they admit the use of very large time steps (after the initial transients have been overcome) to advance the solution from the initial conditions to the final steady state. This research concentrates on the use of implicit schemes and studies ways to improve the performance of implicit schemes.

The Euler implicit time integration of equation 3.13 yields

$$\frac{1}{J} \left(\frac{Q^{n+1} - Q^n}{\Delta t} \right) = -R(Q^{n+1}) \quad \Leftrightarrow \quad \frac{1}{J} \frac{\Delta Q^n}{\Delta t} = -R(Q^{n+1}) \quad (3.14)$$

where ΔQ^n is the incremental change in the cell-centered values of the vector Q between the $n + 1^{\text{th}}$ time level and the known n^{th} time level. It may be noted that equation 3.14 represents a non-linear equation in Q . This non-linearity is removed by linearizing the residual vector in time about the n^{th} time level as

$$R^{n+1} = R^n + \left(\frac{\partial R}{\partial Q} \right)^n \Delta Q^n + \text{higher order terms} \quad (3.15)$$

The substitution of equation 3.15 into equation 3.14 produces a system of simultaneous, linear, algebraic equations which are written as

$$\left(\frac{I}{J\Delta t} + \frac{\partial R}{\partial Q} \right)^n \Delta Q^n = -R^n \quad \text{or} \quad V^n \Delta Q^n = -R^n \quad (3.16)$$

where V^n is the global implicit matrix.

Time integration of equation 3.16 can proceed in a time-accurate way or a time-inaccurate way. Time-accurate integration is also called global time-stepping and involves the use of the same (and usually low) time-step (Δt) for every cell in the domain. Time-inaccurate integration or local time stepping involves the use of the same Courant number (which may lead to widely varying time steps across the domain) for all cells. Local time stepping is a popular technique used for obtaining rapid steady state solutions, and is used in this research. It is standard practice to increase the Courant number as the steady state is approached, and in the limit as $\Delta t \rightarrow \infty$, Newton's root finding method is recovered.

It is useful to rewrite equation 3.16 in terms of difference operators as

$$\left[\frac{I}{J\Delta t} + \delta_\xi \left(\frac{\partial \hat{F}}{\partial Q} - \frac{\partial \hat{F}_v}{\partial Q} \right) + \delta_\eta \left(\frac{\partial \hat{G}}{\partial Q} - \frac{\partial \hat{G}_v}{\partial Q} \right) \right]^n \Delta Q^n = -[\delta_\xi(\hat{F} - \hat{F}_v) + \delta_\eta(\hat{G} - \hat{G}_v)]^n \quad (3.17)$$

δ_ξ and δ_η represent spatial differencing in the ξ and η directions, respectively. Note that the differencing of the left-hand-side (LHS) of equation 3.17 may be done independently of the differencing of the right-hand-side (RHS). The spatial accuracy of the steady-state solution will depend only on the order of differencing adopted for the RHS. Thus, a lower-order accurately differenced LHS may be combined with a higher-order accurately differenced RHS. This approach is called inconsistent differencing, as opposed to consistent differencing (which involves the same order of

accuracy on the LHS and the RHS). A lower-order LHS is attractive since it reduces the amount of storage for the global implicit matrix. It also reduces the operational complexity associated with the solvers used to solve the linear system of equations. These points will be further detailed in the next chapter.

3.4.1 Importance of Delta Formulation

Equation 3.16 has been cast in the so-called ‘delta’ formulation. This means that the solution of this equation provides us with the vector ΔQ^n , which is then used in an update step to obtain the vector Q^{n+1} for the next iteration (recall $Q^{n+1} = Q^n + \Delta Q^n$). The casting of equation 3.16 in the ‘delta’ formulation is extremely significant in the context of this research. The delta form ensures that the final steady-state solution obtained after the time integration has been completed, will be independent of the algorithm or any other integration parameters used to solve the system of equations in equation 3.16. The same would not be true if one were solving for Q^{n+1} directly, instead of first solving for ΔQ^n and then obtaining Q^{n+1} . This is important since different algorithms can be evaluated for their efficiency in solving the system of equation 3.16, with the knowledge that the converged solutions obtained from the different algorithms will be identical.

3.4.2 Structure of the Global Matrix

It is important to comment on the structure of the global implicit matrix V^n . V^n is a linear combination of a block-diagonal matrix, $I/J\Delta t$ ($= K^n$), and the matrix formed by the block-diagonals of the various components of $\partial R/\partial Q$ ($= L^n$). K^n forms part of the main block-diagonal of the global matrix. It is evident from equation 3.17 that the structure of L^n will depend on the nature of the difference

operators (δ_ξ and δ_η) used for the LHS. It was shown in an earlier section that the residual vector, $R(Q)$, at each cell is a function of the Q 's at nine cell centers (for higher-order differencing of the residual). Similarly, the jacobian matrix $\partial R/\partial Q$ for each cell will be a function of nine individual jacobian matrices. Each of the nine matrices for a particular cell is a 4*4 matrix and represents the vector differentiation of the residual vector with respect to the vector of conserved variables.

When the nine block matrices from each grid point are assembled into the global matrix, the nature of the nine-point stencil creates a sparse, block, banded matrix with nine block-diagonals. The structure of the matrix thus formed is shown in figure 2. For structured meshes, the locations of these nine block-diagonals are well known. The sparsity structure of the matrix depends only on the nature of the differencing stencil, and is independent of the problem being solved. It is important to note that lower-order differencing of the LHS (as discussed previously) will create a five-point stencil for the LHS, and the global matrix will then consist only of five block-diagonals.

It is clear that the global matrix V^n is a sparse, block, banded matrix with a known sparsity pattern. The known information about the structure of V^n has been exploited to minimize storage and computational costs for the new solvers that have been developed in this research. For example, storage is greatly reduced by storing only the nine block diagonals (instead of the entire matrix). This corresponds to a meager 144 ($9*4*4$) storage locations per grid point.

3.4.3 Solution procedures for the linear system

The system of equation 3.16 represents a system of linear, simultaneous alge-

nine-point stencil

global implicit matrix

Figure 2. Nine Point Stencil and Global Implicit Matrix

braic equations that have to be solved for ΔQ^n at each time-step level n . This linearized system may be solved exactly at each time step, or alternately, to some arbitrary degree of approximation. The exact solution of this linear system can be obtained by direct inversion of the matrix V^n . However, direct inversion can be prohibitively expensive — both in terms of memory storage requirements and computational effort. This is particularly true for fine meshes in 2-D, and almost any 3-D application. For example, a solution being attempted on a 100*50 grid results in a matrix V^n of order $\alpha=20,000$ and bandwidth $\beta=400$. Since inversion of a banded matrix is an $O(\alpha\beta^2)$ operation, this would require $\approx 3.2*10^9$ operations, just to invert the matrix at each time step ! Moreover, inversion of V^n results in the appearance of non-zeros between the bands and thus destroys the sparsity structure of V^n . In 3-D applications, storage of the inverted matrix is not possible, and is thus a major stumbling block in the use of direct solvers.

The alternate ways to solve equation 3.16 are much more attractive — both from the viewpoint of storage requirements and operational complexity. Such methods are called iterative methods, and solve the linear system approximately at each time step. Each approximate solution is very cheap to compute as compared to the direct solver approach. Also, the additional storage required is of the same order as that required for V^n . Approximate solutions of equation 3.16 are acceptable since the linear system being solved is itself an approximate linearization of the true non-linear system. If the error due to the iteration is no greater than the order of the error introduced by linearization, the iterative method can provide good approximate solutions. Thus, by solving a series of such linear systems approximately, the solution to the original non-linear system can be obtained. Iterative methods for solving the Navier-Stokes equations are discussed in detail in the next chapter.

4.0 Iterative Methods for the Navier Stokes Equations

This chapter describes the theory of iterative methods, and their application as solvers in CFD codes. The conventional solvers, i.e., Upwind Relaxation and Approximate Factorization, are first described. The new solver investigated in this research, i.e., the Generalized Minimal Residual Method, is then detailed. Several issues related to iterative methods are also discussed in Appendix C of this work.

4.1 *Development of Iterative Methods*

This chapter describes iterative methods which are often used to solve systems of linear equations like $Ax = b$, for the solution vector x . In the discussion that follows, x^n will denote the solution at iteration level n . The general approach for any iterative method is to generate a sequence of increasingly accurate approximations, x^n to the exact solution $\bar{x}$ ($\bar{x} = A^{-1}b$ in exact arithmetic).

One major characteristic of iterative schemes is that the cost of obtaining each iterate x^n from the previous iterate x^{n-1} is relatively cheap — generally of the order of a few matrix-vector multiplies. The drawback is that the iterates may approach (or converge) slowly to $\bar{x}$, so that many iterations may be required. Hence, the number of iterations required for convergence is not known in advance.

The key to a successful iterative method is to be able to converge the iterates x^n to $\bar{x}$, while keeping the number of iterations to a minimum. This is done in general by picking an invertible matrix M such that

$$Mx + (A - M)x = b \quad \Rightarrow \quad x = M^{-1}b + (I - M^{-1}A)x \quad (4.1)$$

Now, if the x on the right-hand-side is an existing approximate solution, the x on the left-hand-side may be a better solution. This gives rise to a general iterative scheme

$$\begin{aligned} x^n &= M^{-1}b + (I - M^{-1}A)x^{n-1} \\ &= M^{-1}b + Gx^{n-1} \end{aligned} \quad (4.2)$$

where G is defined as the iteration matrix. It may be observed that if $M^{-1} = A^{-1}$, the direct solver approach is recovered. Hence, if M^{-1} is in some sense “close” to A^{-1} , an effective iterative scheme will be produced. The issue is how to select a good M and how to judge the rate of convergence of the iterates. The issues of error analysis and convergence behavior of iterative schemes is discussed in Appendix C.

The convergence rate of an iterative scheme depends on the eigenvalues of the iteration matrix, G , of equation 4.2. In particular, the smaller the spectral radius of G , the faster the convergence rate (see Appendix C). Iterative schemes can also be constructed to take advantage of any multiplicity in the eigenvalues of G , in order to dramatically improve the overall convergence rate. The ideas of preconditioning used in this research exploit this relation between the convergence rate of an iterative scheme and the distribution of eigenvalues of the iteration matrix.

4.2 Relaxation Schemes

Several different choices of the splitting matrix M have resulted in the development of some very popular iterative schemes. Consider the splitting of A as

$A = L + D + U$ where D , L and U are respectively the diagonal of A , the strictly lower triangle of A and the strictly upper triangle of A . For $M = D$, the well-known Jacobi (or simultaneous displacement) iteration results. Each Jacobi iteration requires inversion of a diagonal matrix. This is one of the simplest and computationally cheapest methods, but has a relatively slow convergence rate. For $M = D + L$, we get the Gauss-Seidel (GS) iteration

$$\begin{aligned} x^n &= (D + L)^{-1}b + (I - (D + L)^{-1}A)x^{n-1} \\ &= (D + L)^{-1}b + (I + D^{-1}L)D^{-1}Ux^{n-1} \end{aligned}$$

It can be seen that the GS iteration requires inversion of a lower triangular matrix, which makes each GS iteration more expensive than a corresponding Jacobi iteration. However, in most cases, the GS iterates converge faster than the Jacobi iterates. This is because the GS iteration allows updated elements of the solution vector x to be used as soon as they are available in the current iteration.

Variations of the GS method have been very popular as iterative methods in CFD research over the years. Line (Jacobi or GS) solvers converge faster than point (Jacobi or GS) schemes because the information at the n^{th} step propagates much more quickly (along lines) through the domain than in the point GS approach (one point at a time). Line GS as applied to this research, is detailed in section C.2 of Appendix C.

4.2.1 Line Gauss-Seidel Relaxation (LGSR) Schemes

Recall the nine-point stencil of figure 2. This stencil represents the cells which provide information for the evaluation of the residual vector, $R(Q)$. The same stencil also defines the nine block-diagonals ($[A]$ through $[I]$) of the global implicit

matrix of equation 3.16. This equation can be rewritten as

$$V^n \Delta Q^n = -R^n \quad \Leftrightarrow \quad [A, B, C, D, E, F, G, H, I]^n \Delta Q^n = -R^n \quad (4.3)$$

Several implicit solvers can be constructed by approximating the structure of the implicit matrix.

One class of implicit solvers was developed by Thomas et. al [27] for use with upwind, flux-split schemes, and called Upwind Relaxation solvers. Upwind Relaxation solvers use the ideas of relaxation, as discussed in the previous section. They are formed by including one or more of the nine block-diagonals in the implicit formulation. The LGSR solver used in this research belongs to this family of Upwind Relaxation solvers. The LGSR solver can be written as

$$[A, B, C, D, E]^n \Delta Q^n = -R^n - [F] \Delta Q_{j-1,k}^n - [I] \Delta Q_{j-2,k}^n \quad (4.4)$$

Note, that $\Delta Q_{j-1,k}^n$ and $\Delta Q_{j-2,k}^n$ are known because the solver computes all unknowns at a particular line simultaneously, and proceeds from $j=1$ to $j=jmax$.

The solution of equation 4.4 requires the inversion of a block penta-diagonal matrix. Recall, that for a first-order accurate differencing of the implicit operator the nine-point stencil reduces to a five-point stencil. For a five-point stencil, the LGSR solver of equation 4.4 can be rewritten as

$$[A, B, C]^n \Delta Q^n = -R^n - [F] \Delta Q_{j-1,k}^n \quad (4.5)$$

The solution of equation 4.5 requires inversion of a block tri-diagonal matrix (instead of a block penta-diagonal matrix). This considerably reduces the computational work per iteration for the family of LGSR solvers. The tradeoff is in the

(possible) reduction in convergence rate with the first-order implicit system. Details of convergence and stability characteristics of the LGSR solver are discussed in Appendix C.

A lower order differencing of the LHS has been adopted in this research. This has been done for the following reasons :

- i) Unconditional stability for any time step with the LGSR solver, for any problem (at least in theory)
- ii) Savings in storage (need to store only five (instead of nine) block diagonals)
- iii) Reduction in computational work — both during LU decomposition and back-substitution.

The convergence rate of the LGSR solver depends on the value of the relaxation parameter (ω) used in the iteration. For simple mathematical problems, optimal values of ω lie between 0 and 2. Under-relaxation and over-relaxation refer to use of $\omega < 1$ and $\omega > 1$, respectively. This research does not use any form of over/under-relaxation for the Navier-Stokes equations. Over-relaxation often creates stability problems for the LGSR solver, when applied to the Navier-Stokes equations. Hence, $\omega > 1$ cannot be used to accelerate convergence of the LGSR solver. Under-relaxation is mainly used to stabilize the LGSR solver when it diverges, and is not a tool for accelerating convergence. Values of $\omega < 1$ produce slower convergence rates than $\omega=1$. Thus, use of $\omega=1$ will provide the best convergence rates for the test problems in this research. All results presented with the LGSR solver have been computed with $\omega=1$, i.e., no relaxation.

4.3 Approximate Factorization Methods

This class of methods is based on approximately factoring the global implicit matrix V^n into a product of two matrices, V_ξ^n and V_η^n ($V^n \approx V_\xi^n V_\eta^n$). The selection of the two factors determines the factorization error of the scheme. In the context of the Navier-Stokes equations and the discretizations adopted in Chapter 3, V_ξ^n and V_η^n are matrices representing a splitting of the implicit operator in the ξ and η spatial directions, respectively. This is also referred to as spatially split Approximate Factorization (AF) and was developed by Beam and Warming[22]. The two operators for the AF scheme can be derived from the operator notation of equation 3.17 as

$$\underbrace{\left[\frac{I}{J\Delta t} + \delta_\xi \left(\frac{\partial \hat{F}}{\partial Q} - \frac{\partial \hat{F}_v}{\partial Q} \right) \right]}_{V_\xi^n} \left(\frac{I}{J\Delta t} \right)^{-1} \underbrace{\left[\frac{I}{J\Delta t} + \delta_\eta \left(\frac{\partial \hat{G}}{\partial Q} - \frac{\partial \hat{G}_v}{\partial Q} \right) \right]}_{V_\eta^n} \quad (4.6)$$

The solution ΔQ^n is then determined in two steps as

$$\begin{aligned} \text{Solve } V_\xi^n \Delta Q^* &= -R^n \quad \text{for } \Delta Q^* \\ \text{Solve } V_\eta^n \Delta Q^n &= \left(\frac{I}{J\Delta t} \right) \Delta Q^* \quad \text{for } \Delta Q^n \end{aligned} \quad (4.7)$$

The solution of the linear system thus requires the inversion of two block tri-diagonal matrices (for a first-order differenced LHS) or two block-pentadiagonal matrices (for a higher-order differenced LHS). As reported in reference [27], the algorithm is insensitive to the accuracy of the differencing adopted for the LHS and the RHS. This implies that consistent and inconsistent differences, both yield the same convergence rates. It is thus desirable to use inconsistent first-order differencing for the implicit LHS, since it is considerably more computationally efficient than a higher-order differenced implicit operator.

One major disadvantage of the AF scheme is that the fastest convergence rate is obtained for an optimal time step. Convergence at lower (than optimal) time steps may be very slow. The solver may be unstable at time steps higher than the optimal time step. The optimal time step is not known a-priori, and varies from problem to problem. The only way to determine the optimal time step for a particular problem is by trial-and-error. The literature documents time steps of order ten as optimal. The time step restriction becomes even more severe when the scheme is used in 3-D with upwind differencing, due to the large factorization error. The convergence rate and stability of the AF scheme are thus sensitive to the time step used in the calculation.

4.4 *LGS Relaxation and AF*

The LGSR scheme has a higher convergence rate per iteration than the AF scheme [27]. It is to be noted here that one iteration refers to one solution update, i.e., one evaluation of ΔQ^n . For a first order differencing of the LHS operator, this requires one set of tridiagonal matrix inversions (for the entire domain) for a single LGSR sweep, and two tridiagonal matrix inversions (for the entire domain) for a solution with the AF solver. The operation count for the AF solver is hence roughly twice (after accounting for overheads) as much as that of the LGSR scheme, for each iteration.

Each tridiagonal matrix inversion can be divided into two steps — an LU (Lower-Upper) decomposition step by Gauss-Elimination and a back-substitution step. On a vector processor, the LU decomposition can be vectorized ‘over the number of lines’ in the domain, for both AF and LGSR. The back-substitution step is also

completely vectorizable for the AF scheme, since all the lines completely decouple from each other. However, for the LGSR scheme, the unknowns at a line depend on information from the previous line(s), and the line has to be updated sequentially (or in scalar mode). This severely affects the vectorization of the LGSR solver. Thus, in spite of the higher operation count of the AF scheme, potentially faster execution of the AF algorithm is possible on vector architectures. The use of parallel rather than vector processing can likewise benefit the LGSR scheme.

As discussed in earlier sections, the LGSR scheme is unconditionally stable in a linear stability analysis, both in 2-D and 3-D. In fact, the maximum damping of waves occurs at large time steps — a feature which is highly desirable for obtaining quick steady-state solutions. The AF scheme encounters optimal time stepping in 2-D and severe stability restrictions in 3-D. This makes the performance of the AF solver susceptible to the applied time step. The stability restrictions prevent the use of the AF solver in 3-D applications.

In the light of the above discussions, it can be concluded that both AF and LGSR have their merits and demerits as linear system solvers. Both solvers are routinely used with upwind, flux split schemes, and are extensively documented in the literature. For the purpose of this research, either of these two schemes can be chosen to represent the current school of solvers. Both solvers preserve the spirit of global implicit solvers, and represent the state-of-the-art in modern CFD algorithms. However, since neither scheme is universally applicable as a general purpose solver, only one of the schemes (LGS Relaxation) has been chosen for comparison with the Conjugate Gradient like solvers, in the main body of this work. For the sake of completeness, comparisons with AF are presented in Appendix A.

4.5 Conjugate Gradient Methods

The optimal application of standard iterative methods to obtain rapid convergence rates requires the estimation of parameters like the relaxation factor (for LGSR) or a stable time step (for AF). These parameters are often difficult to choose properly, and particularly so when the Navier-Stokes equations are solved. Hence, there exists a need to develop methods for which the parameters are easy to choose, and the parameters do not affect the global convergence behavior of the method.

An examination of the literature of linear system solvers reveals the existence of the Conjugate Gradient Method (CGM). This method was proposed by Hestenes and Steifel [28] for the solution of linear systems $Ax = b$, where A is a symmetric and positive-definite (SPD) matrix. The method has its origins in the method of steepest descent and proceeds by seeking to minimize the functional $\varphi(x)$ defined as

$$\varphi(x) \equiv \frac{1}{2}x^T Ax - x^T b \quad (4.8)$$

$\varphi(x)$ is minimized by setting $\nabla(\varphi(x)) = 0$. It can be easily shown that minimizing $\varphi(x)$ and solving $Ax = b$ for x are equivalent problems.

The details of the method of steepest descent, and the development of the ideas of the classical CGM are presented in Appendix D. For the purpose of this research, recall that we are interested in solutions of the linear system of equations presented in chapter 3, and reproduced below:

$$V^n \Delta Q^n = -R^n \quad \Leftrightarrow \quad Ax = b \quad (4.9)$$

The nature of the global implicit coefficient matrix, V^n , depends on the particular problem. V^n may not be symmetric, and may be indefinite in some CFD appli-

cations. Since the classical CGM requires the coefficient matrix to be symmetric and positive definite, the method may have limited practical utility for the Navier-Stokes equations. Fortunately, the ideas of the classical CGM have been generalized to solve systems with non-symmetric and/or indefinite coefficient matrices.

4.6 *The Generalized Minimal Residual Method*

Several generalizations of the Conjugate Gradient Method, most of which can be shown to be mathematically equivalent, are available in the literature. The generalization used in this research is the Generalized Minimal RESidual (GMRES) method of Saad and Schultz [40]. As the name of the method suggests, GMRES seeks to minimize the Euclidean norm of the computed residual vector, r^n , ($r^n \equiv b - Ax^n$) at each iteration.

The GMRES method is directly applicable to solve linear systems with non-symmetric coefficient matrices. The generalization is based on a two step procedure — the generation of a set of orthonormal vectors from a given initial guess, and the solution of a minimization problem. These two steps are detailed in the following two sections. The discussions are based on that of Saad and Schultz [40], and assume that the reader is familiar with some of the principles of linear algebra. Readers who are familiar with GMRES may skip to section 4.6.3, where the GMRES method is summarized.

4.6.1 *Arnoldi's Approach for Generation of Orthogonal Vectors*

Arnoldi's Method [52] uses the well known Gram-Schmidt algorithm for computing an orthonormal basis of vectors $\{v_1, v_2, \dots, v_k\}$, of the Krylov subspace

$K(A, v_1, k) \equiv \text{span}\{v_1, Av_1, \dots, A^{k-1}v_1\}$. The k -step Arnoldi algorithm proceeds as:

1. Choose an initial vector v_1 , with $\|v_1\| = 1$
2. For $j = 1, 2, \dots, k$ do:
 - i) $h_{i,j} = (Av_j, v_i), i = 1, 2, \dots, j$
 - ii) $\hat{v}_{j+1} = Av_j - \sum_{i=1}^j h_{i,j}v_i$
 - iii) $h_{j+1,j} = \|\hat{v}_{j+1}\|_2$
 - iv) $v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$
- enddo

Theoretically, if $v_{k+1} = 0$, then the orthonormalization process is complete and the set of orthonormal vectors $\{v_1, v_2, \dots, v_k\}$ represents an invariant subspace for the matrix A . In practice, the Gram-Schmidt algorithm of step 2 is replaced by the modified Gram-Schmidt algorithm, to avoid errors due to catastrophic cancellation.

Certain definitions are now made for use in later discussions. Let V_k be a rectangular matrix of size $N * k$, whose columns consist of the k individual vectors (each of length N) of the orthonormal basis of vectors formed by the Arnoldi process. It can be shown that $V_k^T V_k = I$ and $\|V_k\| = 1$. An upper Hessenberg matrix of size $k * k$ can be defined as $H_k \equiv V_k^T A V_k$, whose entries are the scalars $h_{i,j}$ generated by the Arnoldi process. Note, that the eigenvalues of A are approximated by those of H_k and that $AV_k = V_k H_k$.

It is possible to build sparse linear system methods based on the Arnoldi process. In order to solve the linear system $Ax = b$ using the orthonormal basis V_k , we seek an approximate solution x_k of the form $x_k = x_0 + z_k$, where x_0 is some arbitrary

initial guess to the exact solution $\bar{x}$. The vector z_k lies in the Krylov subspace collectively defined by A , r_0 ($= b - Ax_0$) and k , i.e.,

$$\begin{aligned} z_k \in K(A, r_0, k) &= \text{span}\{r_0, Ar_0, A^2r_0, \dots, A^{k-1}r_0\} \\ &= \text{span}\{r_0, r_1, r_2, \dots, r_k\} \end{aligned} \quad (4.10)$$

Such methods are referred to as Krylov subspace methods.

The iterative scheme based on Krylov subspace methods will be efficient when the iterate x_k minimizes the l_2 norm of the residual vector, r_k . This can be restated as a minimization of $\|r_k\|$ over $z_k \in K(A, r_0, k)$, i.e.,

$$\min_{z_k \in K(A, r_0, k)} \|r_k\| \quad \Leftrightarrow \quad \min_{z_k \in K(A, r_0, k)} \|(b - Ax_k)\| \quad (4.11)$$

The solution of this minimization problem forms the second part of the GMRES algorithm, and is discussed below.

4.6.2 Formulation and Solution of the Minimization Problem

Suppose that we have an arbitrary initial guess, x_0 . Determine $r_0 = b - Ax_0$. For $\beta = \|r_0\|_2$ set $v_1 = r_0/\beta$. Perform k steps of Arnoldi's process, to form the basis of orthonormal vectors $\{v_1, v_2, \dots, v_k, v_{k+1}\}$, i.e., V_{k+1} . Note, that by adding an additional row (whose only non-zero element is $h_{k+1,k}$) to the matrix H_k (defined earlier), we obtain the $(k+1) \times k$ matrix $\bar{H}_k$, such that $AV_k = V_{k+1}\bar{H}_k$. The

minimization problem of equation 4.11 is thus reformulated as

$$\begin{aligned}
\min_{z_k} \|b - Ax_k\| &= \min_{z_k} \|b - A(x_0 + z_k)\| \\
&= \min_{z_k} \|b - Ax_0 - Az_k\| \\
&= \min_{z_k} \|r_0 - Az_k\| \\
&= \min_{z_k} \|\beta v_1 - AV_k y_k\| \\
&= \min_{z_k} \|V_{k+1} \beta e_1 - V_{k+1} \bar{H}_k y_k\| \quad (v_1 = V_{k+1} e_1) \\
&= \min \| \beta e_1 - \bar{H}_k y_k \| \quad V_{k+1} \text{ is orthonormal} \\
&= \min \varphi(y_k)
\end{aligned} \tag{4.12}$$

Several definitions need to be made for equation 4.12. Recall that V_k is now the matrix composed of the orthogonal vectors $\{r_0, Ar_0, A^2 r_0, \dots, A^{k-1} r_0\}$. Thus, for z_k to lie in the Krylov subspace defined by V_k (i.e., z_k is a linear combination of the column vectors of V_k), we can write $z_k = V_k y_k$, for any arbitrary vector y_k . However, the vector y_k is not an arbitrary vector — instead, it is determined in a fashion that will lead to the solution of the minimization problem of equation 4.12. Hence, determining z_k is equivalent to solving for the vector y_k which minimizes $\varphi(y_k) \equiv \|\beta e_1 - \bar{H}_k y_k\|$. Note, that e_1 is the first column of the identity matrix of rank $(k + 1)$.

The minimization problem stated above is solved by performing a QR factorization of the matrix $\bar{H}_k$, i.e., forming $Q_k \bar{H}_k = R_k$, where Q_k is a $(k + 1) * (k + 1)$ orthogonal matrix, and R_k is a $(k + 1) * k$ upper-triangular matrix. The QR factorization is easy to compute and can be obtained by using one of several popular mathematical tools [53] for the $k * k$ Hessenberg matrix, $\bar{H}_k$. In this research, the QR factorization is obtained by a series of plane rotations. Since Q_k is orthogonal,

the function $\varphi(y_k)$ can be rewritten as

$$\varphi(y_k) = \|\beta e_1 - \bar{H}_k y_k\| = \|Q_k(\beta e_1 - \bar{H}_k y_k)\| = \|g_k - R_k y_k\| \quad (4.13)$$

where $g_k (=Q_k \beta e_1)$ is transposed to the right-hand-side to form a matrix system as $R_k y_k = g_k$. Finally, y_k may be obtained by solving the upper-triangular linear system which results from removing the last row of R_k and the last element of g_k .

4.6.3 Characteristics of the GMRES Algorithm

The complete GMRES algorithm can be summarized as follows:

1. For any starting vector x_0 , form $r_0 = b - Ax_0$; $\beta = \|r_0\|_2$; $v_1 = r_0/\beta$.
2. Perform Arnoldi's algorithm with the vector v_1 obtained from step 1.
3. Form the approximate solution:
 - a) Find the vector y_k which minimizes the function $\varphi(y_k) = \|\beta e_1 - \bar{H}_k y_k\|$
(involves QR factorization followed by an upper-triangular solve)
 - b) Compute $x_k = x_0 + V_k y_k$

Let k_c define the number of steps required by the Arnoldi algorithm for complete orthogonalization. In practice, the GMRES algorithm becomes increasingly expensive as k_c , increases. Precisely, the number of orthogonal vectors requiring storage increases like k_c , and the number of operations increases as $\frac{1}{2}k_c^2 N$ (N is the order of the matrix A). In order to limit the additional storage and to minimize the computational costs, the Arnoldi orthogonalization process is often truncated after $k (< k_c)$ steps (or after an equivalent 'stopping criteria' on the residual vectors is met). The algorithm is then restarted with information available at the k^{th} step. Restarted GMRES, referred to as GMRES(k) is similar to the GMRES algorithm presented earlier. For a choice of the starting vector x_0 , the size of the Krylov

subspace k and/or an equivalent ‘stopping criteria’, the GMRES(k) algorithm can be summarized as:

1. Do steps 1–3 of the non-restarted GMRES algorithm.
2. Restart: If ‘the stopping criteria’ is satisfied, then stop, else set $x_0 \leftarrow x_k$ and go to step 1.

It is important to highlight certain convergence features of the method. GMRES cannot breakdown, regardless of the positiveness of A . However, restarted GMRES, i.e., GMRES(k), converges if and only if A is positive real. GMRES(k) may not converge if the symmetric part of A , i.e., $(A + A^T)/2$, is not positive definite. GMRES(k) may be stationary for certain stiff problems. Further, the convergence of GMRES(k) is related only to the eigenvalue distribution of A , and is independent of the problem size N . Large values of k may be required to maintain stability in some cases.

The use of the GMRES algorithm thus requires the user to choose x_0 and k (and/or a ‘stopping criterion’, say $\|r_k\|/\|r_0\| < \varepsilon$). Since the method converges for any arbitrary initial vector, including the zero vector, the choice of x_0 is arbitrary, and hence easy. In order to guarantee convergence, a non-restart GMRES has been used in this research — the issue of if and when to restart, has thus been eliminated. Thus, the only significant parameter that can possibly affect the convergence rate is the size of the Krylov subspace k , or equivalently, the ‘stopping criteria’ for convergence enforced by the choice of ε . The Arnoldi process will stop after k steps, or earlier if the norm of the residual vector falls below the value of ε . As will be demonstrated in section 6.1, the convergence rate of the overall non-linear problem has been found to be remarkably insensitive to the choice of ε . Thus, the

use of GMRES to approximately solve the linear system of equation 4.9 provides an intrinsically parameter-free procedure to obtain efficient solutions of the Navier-Stokes equations. This is an important step in the development of the GMRES solver as a general purpose solver.

In summary, the GMRES method is a minimization process to solve linear matrix systems like $Ax = b$. The minimization proceeds as a sequence of sub-iterations, and the minimizer is obtained by a simple upper-triangular solve. The cost of each linear system solve (or minimization) increases as the number of sub-iterations increases. The method is used in conjunction with preconditioning, which is discussed in the next chapter.

5.0 Preconditioning of Conjugate Gradient like Methods

This chapter describes the concepts and techniques of preconditioning used in this research. Details of the two preconditioners selected for application with the GMRES solver are presented, and concepts and techniques of preconditioning are discussed.

5.1 Concepts of Preconditioning

One of the most effective iterative methods for solving large sparse linear systems is a combination of a generalized Conjugate Gradient like procedure with some appropriate preconditioning technique. Assuming that a preconditioner M is used on the left of the original unpreconditioned system, this involves solving the preconditioned linear system

$$M^{-1}Ax = M^{-1}b \quad \Leftrightarrow \quad \tilde{A}x = \tilde{b} \quad (5.1)$$

instead of the original system $Ax = b$.

The motivation of preconditioning is to reduce the overall computational effort required to solve linear systems of equations by increasing the convergence rate of the underlying iterative algorithm. Preconditioning will be effective only if the additional computational work incurred per iteration is compensated for by a reduction

in the total number of iterations to convergence — so that the total cost of solving the linear system is reduced.

The costs associated with preconditioning can be enumerated as

- i) Computing the preconditioning matrix M
- ii) Matrix-vector multiplies or equivalent linear system solves associated with M
- iii) Additional computer storage to store M — may be of the order of storage requirements for the coefficient matrix A

The selection of an ‘efficient’ preconditioner is motivated by the minimization of the afore-mentioned costs. The cheapest preconditioner in this context is the identity matrix (of appropriate rank). However, for $M = I$, the original, unpreconditioned iterative scheme is recovered! The costliest preconditioner results when $M = A$. This is equivalent to performing a direct solver approach and inverting A to obtain $x = A^{-1}b$ — an approach which is unacceptable in the context of iterative methods.

It is apparent that a practical preconditioner lies somewhere between the two extremal choices of $M = I$ and $M = A$. It is also evident that M should be in some sense “close” to A — so that $M^{-1}A$ is close to the identity matrix. For good preconditioning, this means that the eigenstructure of $\tilde{A}$ should be close to that of the identity matrix I , i.e. $\tilde{A}$ should be well-conditioned. There are two major criteria involved in determining the most suitable preconditioning of any matrix, say A :

- i) The ratio of the maximum singular-value of A to the minimum singular-value of

$\kappa_2(A)$, is defined as the condition number of the matrix A in the 2-norm, i.e.,

$$\kappa_2(A) \equiv \sigma_{\max}(A)/\sigma_{\min}(A) \equiv \|A\|_2 \|A^{-1}\|_2 \quad (5.2)$$

For a symmetric positive-definite matrix, singular-values and eigenvalues are identical. In the context of preconditioning, a good preconditioner ensures that the condition number of the preconditioned matrix is much less than that of the unpreconditioned matrix, i.e., $\kappa_2(\tilde{A}) \ll \kappa_2(A)$. A reduction in the condition number may be achieved by a matrix M which produces $\sigma_{\max}(\tilde{A}) < \sigma_{\max}(A)$ and/or $\sigma_{\min}(\tilde{A}) > \sigma_{\min}(A)$. Note, that such a reduction may not have any effect on the distribution of the intermediate singular-values.

- ii) The spectrum of A is defined as the set of eigenvalues of A . Assume that the σ are scaled in the range $[0,1]$. A broad spectrum refers to uniform distribution of σ in the range $[0,1]$. A banded spectrum refers to multiplicity of σ , i.e., certain eigenvalues are numerically equal, and occur repeatedly. A good preconditioner attempts to increase the multiplicity of the σ . This increased multiplicity of the σ may or may not affect the value of the condition number.

In practice, the distribution of eigenvalues has a greater impact on the convergence rate of the resulting algorithm than the value of the condition number. This is because the convergence rate of the algorithm is directly related to the number of distinct eigenvalues of the iteration matrix (see Appendix C). Thus, even though a low condition number is desirable (in order to minimize computational errors associated with matrix-vector multiplies), it is not critical to the success of preconditioned CGMs. This distinction has to be carefully borne in mind while evaluating the effectiveness of preconditioners, and should not be ignored.

5.2 Preconditioning Techniques

As discussed in the previous section, the choice of an appropriate preconditioner can greatly influence the performance of iterative schemes based on preconditioned, generalized CGMs. For a particular preconditioning matrix M , the preconditioned GMRES algorithm can be implemented as:

1. Start: Choose any x_0 and the size of the Krylov subspace k and/or a ‘stopping criterion’.
2. Arnoldi process:
 - a) $r_0 = M^{-1}(b - Ax_0)$; $\beta = \|r_0\|_2$; $v_1 = r_0/\beta$
 - b) For $j = 1, 2, \dots, k$ or till convergence is satisfied, do:
 - i) $h_{i,j} = (M^{-1}Av_j, v_i), i = 1, 2, \dots, j$
 - ii) $\hat{v}_{j+1} = M^{-1}Av_j - \sum_{i=1}^j h_{i,j}v_i$
 - iii) $h_{j+1,j} = \|\hat{v}_{j+1}\|_2$
 - iv) $v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$
 - enddo
3. Form the approximate solution:
 - a) Find the vector y_k which minimizes the function $\varphi(y_k) = \|\beta e_1 - \bar{H}_k y_k\|$
 - b) Compute $x_k = x_0 + V_k y_k$

Initial examination of the above algorithm reveals the necessity of forming M^{-1} . However, the ‘explicit’ computation of M^{-1} is not advisable in practice for two reasons:

- i) Even though M may be sparse (corresponding to the sparsity of A), M^{-1} may be a dense matrix. Storage requirements for M^{-1} may thus far exceed those

for M .

- ii) If M is ill-conditioned (corresponding to ill-conditioning of A), computation of M^{-1} in real computer arithmetic will be highly error-prone.

The non-availability of M^{-1} due to the above restrictions is not as catastrophic as it seems. Instead of ‘explicit’ preconditioning, an equivalent but highly effective technique called ‘implicit’ preconditioning is adopted.

On careful examination of the preconditioned GMRES algorithm, it is seen that M^{-1} is used only to form matrix-vector products of the type $M^{-1}(b - Ax_0)$ and $M^{-1}(Av_j)$. These matrix-vector products can be generalized as

$$M^{-1}u = \tilde{u} \quad \Leftrightarrow \quad M\tilde{u} = u \quad (5.3)$$

where u is any vector that needs to be multiplied by M^{-1} , and $\tilde{u}$ is the resulting vector of the multiplication. The concept of implicit preconditioning is to transform the ‘explicit’ problem of forming $M^{-1}u$ to an equivalent ‘implicit’ problem of solving a linear system for $\tilde{u}$, with M as the coefficient matrix. It may be noted that such linear systems have to be solved repeatedly for each application of the preconditioned GMRES algorithm. Thus, any matrix M which produces easy-to-solve (i.e., computationally efficient) linear systems of the type $M\tilde{u} = u$, is a potentially good preconditioner.

In the light of the preceding discussions, a good preconditioning matrix M should

- i) coalesce the singular-values of the matrix A , so that the singular-values of the resulting matrix $\tilde{A}$ occur in a few finite bands in the spectrum, and
- ii) produce easy-to-solve linear systems of the form $M\tilde{u} = u$.

The effectiveness of a preconditioner to coalesce the singular-values of a matrix can be judged by examining the respective eigenstructures of the original and the preconditioned matrices. However, it is extremely difficult to evaluate the success of a given preconditioner on the basis of eigenstructure information, because this information is not readily available. This is particularly true for the complex fluid-flow problems being solved in this research, with the Navier-Stokes equations. There is no existing theory which can give a cheap and accurate estimate of the extremal singular-values (σ_{max} and σ_{min}), for evaluation of the condition number. Thus, one can well imagine the impossibility of obtaining information about the entire spectrum of singular-values. It is for this reason that a complete eigenvalue analysis has not been attempted in the present work.

In contrast, it is relatively simple to quantify the computational costs associated with a particular preconditioner. This may be done by counting the number of operations involved in a) the formation of the preconditioner, and b) the linear system solves ($M\tilde{u} = u$). This can also be done by recording the difference in computational (CPU) time between the preconditioned and unpreconditioned GMRES algorithms. The operation count is a function of the problem size N , and the number of sub-iterations required to solve the global linear system at each time step. The CPU time is a function of the operation count and system overheads, and may be heavily influenced by the level of vectorization and/or parallelization of the implementation of the preconditioner. Since supercomputers are being routinely used in large scale computations of fluid flow problems, the issues of vectorization and parallelization are significant, and will be discussed in detail later.

Although not discussed above, the issue of storage requirements for M is also an important consideration during preconditioner selection. If M requires significantly

more storage than A , it may be undesirable as a preconditioner. This criteria is particularly valid for 3-D applications of the preconditioned algorithm. Thus, an acceptable preconditioner should match the sparsity pattern of A as closely as possible, so that the maximal additional storage required for M is of the same order as that for A itself.

5.3 Practical Preconditioners

The implementation of preconditioning in practical algorithms can be divided into two main categories:

- i) Preconditioners based on approximate inversions of the matrix A (i.e., diagonal and ILU preconditioners), and
- ii) Preconditioners based on iterative schemes for linear systems.

Some preconditioners derived from these two categories, and those that were used in this research, are now discussed.

5.3.1 Diagonal Preconditioners

Recall that M^{-1} should approximate A^{-1} in some sense for effective preconditioning. For non-singular A , any choice of a non-singular M is a valid preconditioner. For Diagonal Scalar (DS) preconditioning, M consists simply of the main scalar diagonal of A . M^{-1} can be formed from M by simple scalar inversions (or divisions), and is thus computed very cheaply. However, DS preconditioning is effective only if A is very strongly diagonally dominant, and has been found to have very limited practical utility with the test problems of this research.

Since the global coefficient matrix of the linear system has a block structure (for the 2-D equations, each block is 4×4), Diagonal Block (DB) preconditioning may be adopted as a natural alternative to DS preconditioning. This is done by choosing M to be the main block-diagonal of A . Computation of M^{-1} now requires inversion of N block matrices of size 4×4 . This can be done in $\approx N * (4^3) = 64N$ operations (note, that matrix inversion is an $O(N^3)$ operation). It may be remarked that for large N (corresponding to fine meshes in 2-D), the operation count is linear in N . Hence, for large N , forming M^{-1} for the BD may be only marginally costlier than for the scalar case.

The DB preconditioner works well only when A is diagonally dominant in a block sense. It is useful for simple problems, but for large time steps (implying reduced diagonal dominance) and complex problems, this preconditioning may be unstable, i.e., the condition of the preconditioned system may be worse than that of the unpreconditioned system.

It is to be noted that diagonal (scalar and block) preconditioners are not used implicitly, but instead with explicit formation of M^{-1} . M^{-1} is computed and stored, and then used explicitly in the algorithm. This is acceptable because the use of M^{-1} in these case involves only vector-vector multiplies (each vector element is a scalar or a 4×4 matrix), which can be done much more efficiently on a vector computer than traditional linear system solves (of the type $M\tilde{u} = u$).

5.3.2 *Approximate Inversions as Preconditioners*

Another choice for M is that of $M \approx A$, with the provision that $M^{-1} \approx A^{-1}$, i.e., M^{-1} is calculated as an approximate, or more precisely, an incomplete inverse of A .

The incomplete inverse is formed by ignoring the “fill-in” that would be created during an exact Lower-Upper (LU) decomposition of A . It is well known that for any matrix A , an exact LU decomposition factors A into a lower triangular matrix (L) and a unit upper triangular matrix (U), by standard Gaussian Elimination (GE), i.e., $A = LU$. Also, if A has upper bandwidth β_u and lower bandwidth β_l , then the matrices U and L have bandwidths β_u and β_l , respectively, i.e., the bandwidth of the original matrix is preserved during GE. However, there is no guarantee that GE will not destroy the sparsity structure (location of zeros and nonzeros) of A . In fact, it is possible (and often the case) that L and U will be dense matrices, with a sparsity structure completely different from A .

Recall that, in this research, the global implicit matrix requires storage of only nine block-diagonals — no intermediate zeroes are stored. With such an existing storage format, the L and U matrices of the exact LU decomposition will require much more storage than that for the original global matrix. Thus, an exact LU decomposition by standard GE is undesirable, from the perspective of storage for the L and U factors.

The operation count for computing an exact LU is $O(N\beta^2)$, for a matrix of size N and bandwidth $\beta(= \beta_u = \beta_l)$. As has been discussed in section 3.4.3 and Appendix B, this is clearly unacceptable from a computational complexity standpoint, i.e., the cost of forming the preconditioning matrix will not be compensated for by the decrease in the total number of iterations to convergence.

An effective preconditioner can be created by requiring that $A = \tilde{L}\tilde{U} + R$, and then using $\tilde{L}$ and $\tilde{U}$ for preconditioning instead of the exact factors L and U . Thus, an Incomplete LU (ILU) is created instead of an exact one. The ILU is created by

simply ignoring all possible fill-in during GE, i.e., preserving the original sparsity structure of A . This is called ILU(0), corresponding to zero fill-in, and the $\tilde{L}$ and $\tilde{U}$ factors require additional storage of the same order as A . The computational cost is also reduced compared to that of an exact LU, as all operations that will produce fill-in are excluded from the computation.

As an extension of ILU(0), ILU(i) may allow i extra level(s) of fill-in, i.e., i additional diagonal band(s) may be filled-in in each of the LU factors. Note that, as i increases, the storage and complexity of the incomplete LU approaches that of the exact LU. It has been determined in the course of this research, as has also been reported in reference [50], that no additional gain in overall efficiency is found when i actually exceeds zero. Thus, ILU(0) has been used for all the results presented in this research.

The ILU is computed by a block GE procedure, which is consistent with the overall block-matrix approach of this research. The ILU for the global implicit matrix is computed and stored before the preconditioned GMRES algorithm is invoked. Each solve for the linear system (recall $M\tilde{u} = u$) is then performed by a forward substitution (with $\tilde{L}$) followed by a backward substitution (with $\tilde{U}$). It may be remarked that the forward and backward substitutions are inherently sequential processes, and create considerable bottlenecks when the ILU preconditioner is implemented on a vector processor.

The Block ILU Factorization (BILUF) of A , as discussed above, is one of the two preconditioners investigated in detail in this research. The original (scalar) ILU is a popular preconditioner for CGMs, and has been studied by several authors, as reported in references [50], [54], and [55]. The Block ILU Factorization used for the

present work represents an improvement on the conventional scalar ILU approach.

The algorithm used to compute the BILUF for this research is now described. Recall the global implicit matrix, and its structure consisting of nine block-diagonals. As mentioned earlier, this research uses a first-order differencing of the implicit operator. For a first-order differenced implicit matrix, only five of the nine block-diagonals (i.e., A , B , C , F and G) are required to completely define the global matrix. In the following algorithm, nb represents the number of blocks (or mesh points), β is the bandwidth of the first-order coefficient matrix, and L_b , U_b and Z are 4×4 matrices. The BILUF algorithm can be written as:

1. For $n = 1, nb - \beta$ do

Overwrite $B(n)$ with the exact LU of $B(n)$ ($L_b U_b = B(n)$)

Overwrite $C(n)$ with the solution to $L_b Z = C(n)$

Overwrite $G(n)$ with the solution to $L_b Z = G(n)$

Overwrite $A(n)$ with the solution to $U_b Z = A(n)$

Overwrite $F(n)$ with the solution to $U_b Z = F(n)$

$B(n+1) \leftarrow B(n+1) - A(n) * C(n)$

$B(n+\beta) \leftarrow B(n+\beta) - F(n) * G(n)$

enddo

2. For $n = nb - \beta + 1, nb - 1$ do

Overwrite $B(n)$ with the exact LU of $B(n)$ ($L_b U_b = B(n)$)

Overwrite $C(n)$ with the solution to $L_b Z = C(n)$

Overwrite $A(n)$ with the solution to $U_b Z = A(n)$

$B(n+1) \leftarrow B(n+1) - A(n) * C(n)$

enddo

3. Overwrite $B(nb)$ with the exact LU of $B(nb)$ ($L_b U_b = B(nb)$)

5.9.3 Iterative Schemes as Preconditioners

As discussed in section 5.1, preconditioning requires solutions for systems of the type $M\tilde{u} = u$. Standard iterative schemes for linear systems may be used to solve these systems. This approach is very attractive since, it allows the use of any existing iterative solver as a preconditioner. Hence, with minor modifications, iterative solvers available in a code may be employed to solve the preconditioning problem $M\tilde{u} = u$. This approach identifies a distinct class of preconditioners, and includes relaxation schemes, approximate factorization (AF), multigrid schemes, domain decomposition, and related techniques in CFD research.

The particular iterative scheme chosen for use as a preconditioner in this research is a Lower Upper Symmetric Successive Over-Relaxation (LUSSOR) method, based on the scheme formulated by Jameson and Yoon [34]. The LUSSOR scheme approximately splits the global implicit operator into two operators, called a lower (L) and an upper (U) operator. The L and U operators produced by LUSSOR are not based on any particular spatial orientation (recall that the Beam-Warming AF scheme produces two operators based on the spatial orientation of the problem).

Recall the global implicit operator (V^n), as was discussed in equation 3.16. In the discussion that follows, the viscous flux Jacobians have been temporarily ignored, for the sake of simplicity. The operator V^n can be algebraically manipulated, after

this simplification, as under:

$$\begin{aligned}
V^n &= \frac{I}{J\Delta t} + \delta_\xi \left[\frac{\partial \hat{F}}{\partial Q} \right] + \delta_\eta \left[\frac{\partial \hat{G}}{\partial Q} \right] \\
&= \frac{I}{J\Delta t} + \delta_\xi [\hat{A}] + \delta_\eta [\hat{B}] \\
&= \frac{I}{J\Delta t} + \delta_\xi [A^+ + A^-] + \delta_\eta [B^+ + B^-] \\
&= \left(\frac{I}{J\Delta t} + [A^+] - [A^-] + [B^+] - [B^-] \right) + (\delta_\xi^- [A^+] - [A^+]) \\
&\quad + (\delta_\xi^+ [A^-] + [A^-]) + (\delta_\eta^- [B^+] - [B^+]) + (\delta_\eta^+ [B^-] + [B^-]) \tag{5.4} \\
&= [S] + (\delta_\xi^- - 1)[A^+] + (\delta_\xi^+ + 1)[A^-] + (\delta_\eta^- - 1)[B^+] + (\delta_\eta^+ + 1)[B^-] \\
&= [S] - [A^+]_{j-1/2} - [B^+]_{k-1/2} + [A^-]_{j+1/2} + [B^-]_{k+1/2} \\
&\approx ([S] - [A^+]_{j-1/2} - [B^+]_{k-1/2})[S]^{-1}([S] + [A^-]_{j+1/2} + [B^-]_{k+1/2}) \\
&= \tilde{V}^n \equiv [L][S]^{-1}[U]
\end{aligned}$$

It may be noted that the flux Jacobian matrices have been split in step 3 according to Van Leer's flux-splitting scheme, as discussed in Appendix B. The superscripts (+ and -) on the difference operators (δ_ξ and δ_η) represent forward (+) and backward (-) differencing of the negative ($[A^-], [B^-]$) and positive ($[A^+], [B^+]$) flux Jacobians, respectively. Linear combinations of $[A^+]$, $[A^-]$, $[B^+]$ and $[B^-]$ are added and subtracted from the original operator V^n in step 4. This is done so that the matrix $[S]$ corresponds exactly to the linear combination of the individual flux Jacobian matrices appearing on the main block-diagonal of V^n . This holds true for the first order accurate differencing of the implicit operator, and is shown in equation 5.4.

The approximate LU factored matrix $\tilde{V}^n$ is adopted into an iterative scheme to determine ΔQ^n as

$$\tilde{V}^n \Delta Q^n = -R^n \quad \Leftrightarrow \quad [L][S]^{-1}[U] \Delta Q^n = -R^n \tag{5.5}$$

which is solved in a two-step procedure as

$$\begin{aligned} [L]\Delta Q^* &= -R^n \\ [U]\Delta Q^n &= [S]\Delta Q^* \end{aligned} \quad (5.6)$$

Using the definition of $[L]$, the first solve for the ΔQ^* of equation 5.5 can be written as

$$\begin{aligned} ([S] - [A^+]_{j-1/2} - [B^+]_{k-1/2})\Delta Q^* &= -R^n \\ [S]\Delta Q^*_{j,k} &= -R^n_{j,k} + [A^+]_{j-1/2}\Delta Q^*_{j-1,k} + [B^+]_{k-1/2}\Delta Q^*_{j,k-1} \end{aligned} \quad (5.7)$$

Note that if $\Delta Q^*_{j-1,k}$ and $\Delta Q^*_{j,k-1}$ are known, $\Delta Q^*_{j,k}$ can be determined by equation 5.7. This can be achieved in practice by starting at $\Delta Q^*_{1,1}$ ($\Delta Q^*_{1,0}$ and $\Delta Q^*_{0,1}$ are known from the boundary conditions) and proceeding towards $\Delta Q^*_{j_{max},k_{max}}$, in a pointwise solve for each ΔQ^* .

Similarly, the second solve step for the ΔQ^n of equation 5.5 is written as

$$\begin{aligned} ([S] + [A^-]_{j+1/2} + [B^-]_{k+1/2})\Delta Q &= [S]\Delta Q^* \\ [S]\Delta Q_{j,k} &= [S]\Delta Q^*_{j,k} - [A^-]_{j+1/2}\Delta Q_{j+1,k} - [B^-]_{k+1/2}\Delta Q_{j,k+1} \end{aligned} \quad (5.8)$$

Equation 5.8 is used to determine $\Delta Q^n_{j,k}$ by starting from $\Delta Q_{j_{max},k_{max}}$ and proceeding towards $\Delta Q_{1,1}$, i.e., by reversing the order of the solve for ΔQ^* .

It can be seen from equation 5.7 and equation 5.8 that the inverse of the matrix $[S]$ is required for the two solution steps. $[S]$ represents a linear combination of several 4*4 matrices at each grid point in the computational domain. In practice, $[S]$ is computed and stored before the solve for ΔQ^* is started. The LUSSOR scheme of Jameson and Yoon [34] proposes to avoid the block-matrix inversion by approximating the split flux Jacobians, so that the matrix $[S]$ becomes a scalar diagonal. This approximation is done by setting

$$[A^\pm] = 0.5(\hat{A} \pm \gamma_A I) \quad [B^\pm] = 0.5(\hat{B} \pm \gamma_B I) \quad (5.9)$$

where γ_A and γ_B are functions of the spectral radii of the flux Jacobian matrices ,
i.e.,

$$\gamma_A = \omega \max(|\lambda_A|) \quad \gamma_B = \omega \max(|\lambda_B|) \quad \omega = O(1) > 1 \quad (5.10)$$

The above diagonalization approach has not been used for preconditioning in this research for two main reasons:

- i) Determination of the optimal ω of equation 5.9 poses problems. The convergence of the diagonalized scheme is found to be sensitive to the choice of ω .
- ii) The block-matrix scheme is more stable than the diagonalized scheme at high Courant numbers, and is applicable over a wide range of problems.

The major computational effort for the block LUSSOR scheme consists of forming the inverse of $[S]$, solving for ΔQ^* , computing $[S]\Delta Q^*$ and finally solving for ΔQ . It may be remarked that the various flux Jacobian matrices and their linear combinations are available as the previously computed block-diagonals of the global matrix V^n , and thus require no additional computational effort.

The block LUSSOR scheme has been adopted for implementation on a vector processor. The inversion of $[S]$ has been vectorized over the number of blocks (N) in the domain. The pointwise solves for ΔQ^* and ΔQ have been converted to “line” solves — the line where $j + k = \text{constant}$ is identified, and all the unknowns on that line are solved for simultaneously. This translates to forward and backward sweeps along the main diagonal of the domain (e.g., lower-left to upper-right traverse for the ΔQ^* sweep, with the sum of j and k indices increasing from 2 to $j_{\max} + k_{\max}$). This form of sweeping is implemented by computing and storing the indices of each element belonging to a particular line, prior to the sweeps. The marginal

additional storage and computational effort are compensated by the gains in CPU time achieved over a scalar (or pointwise) implementation of the scheme.

The vectorized block LUSSOR scheme is used to solve systems like $M\tilde{u} = u$ for preconditioning the GMRES solver. As will be shown by the computational results, LUSSOR preconditioning is very stable and can be used successfully over a wide range of problems. It should be mentioned that the preliminary use of available solvers like LGS relaxation and spatially split AF as preconditioners did not produce the same success (in accelerating convergence), or range of applicability as the LUSSOR solver. Detailed investigations of iterative schemes as preconditioners are hence limited to the study of the block LUSSOR scheme as a preconditioner.

6.0 Test Results and Discussion

This chapter presents the various results obtained from the application of preconditioned GMRES to solve the Navier-Stokes equations of fluid flow. Several issues related to computational complexity and storage costs of the preconditioned GMRES solver are discussed. This is followed by a description of the framework used for comparisons between the preconditioned GMRES and conventional solvers. Details of computational results are then presented and discussed.

6.1 *Implementation Details*

This section details the techniques adopted in this research in order to minimize the storage and computational costs associated with the various solvers and preconditioners discussed in the previous chapters. The implementation of the iterative techniques can influence their performance, particularly in terms of overall CPU time. Most of the techniques adopted have been directed towards making the solvers and preconditioners more efficient when implemented with vector architectures.

Some issues related to storage requirements are first discussed. Recall that the global implicit matrix has a block, banded structure with a known sparsity pattern.

All results presented in this work involve a first-order accurate discretization of the implicit operator, thus creating an implicit matrix with five well-defined diagonals. Hence, for a problem size of N , storage corresponding to $5 \cdot N$ blocks (each of size 4×4) is required. Note, that no intermediate zeroes are stored. This storage for the implicit matrix is required of both solvers (GMRES and LGSR) tested in this research.

The use of preconditioning in conjunction with the GMRES solver necessitates the storage of the corresponding preconditioner. The additional storage depends on the particular preconditioner, and varies from N blocks (for block diagonal and LUSSOR preconditioning) to $5 \cdot N$ blocks (for block ILU preconditioning with no fill-in).

Recall, that each linear system solve with GMRES requires the generation of a set of orthogonal vectors at each global iterative step. For a set size of k orthogonal vectors, $k \cdot N \cdot 4$ additional storage locations are required for the set of vectors. As k becomes large (for high Courant numbers and stiff problems), the storage cost associated with GMRES may become significant, even on the current class of supercomputers. In this research, it has been demonstrated that effective preconditioning establishes an upper limit of $k = 10$. The corresponding storage of size $10 \cdot N \cdot 4$ locations has been extracted from existing 'temporary' storage in the code, i.e., storage is shared by GMRES and other subroutines in the code. Hence, the need for additional storage for the orthogonal vectors has been successfully eliminated in the implementation of preconditioned GMRES in this research. This resolves some of the issues highlighted by Wigton et. al [49], since their implementation of GMRES for different problems had k varying from 20 to 80 (see Chapter 2 for details).

Some issues related to minimization of computational costs are now discussed. The GMRES solver requires multiplication of a sparse matrix with a vector, with such matrix-vector multiplications being performed several times at each global iterative step. All matrix-vector multiplies are performed with the ‘multiplication by diagonals’ algorithm of Madsen et. al [56]. This is consistent with the storage format adopted for the implicit matrix.

One major parameter which determines the cost of using GMRES is the number of sub-iterations (which is equivalent to the size of the set of orthogonal vectors, k , for a non-restart GMRES) required at each global iteration. In other words, the issue is to determine the number of sub-iterations to be performed before the linear system is declared solved (to a particular accuracy) for the current time step. Clearly, k has to be large enough to allow for a stable and convergent overall iterative scheme. Also, k has to be small enough to minimize the storage and computational costs of GMRES. The number of sub-iterations has to strike a good balance between providing stability and minimizing costs. Unfortunately, there is no existing mathematical theory for determining an optimal number of sub-iterations.

One approach often adopted is to fix k , i.e., perform a predetermined (user-specified) number of sub-iterations at each time step. Another approach, which is used in this research, is to assign a ‘stopping criteria’ based on the reduction in the norm of the initial residual vector of the linear system, i.e., to terminate the sub-iterations when $\|r_m\|/\|r_0\| \leq \varepsilon$, where $\varepsilon < 1$. In practice, this translates to truncation of the orthogonalization process after k steps of the Arnoldi algorithm (see section 4.6.1 for details). The ‘stopping criteria’ provides a flexible, rather than a fixed k . Of course, the issue is now to pick an ε (or accuracy value) which will provide a stable and convergent iterative scheme.

The choice of an ideal ε for the Navier-Stokes equations is complicated by the following conditions :

- i) The range of Mach numbers across which the chosen ε has to be valid (Mach = 0.1 to Mach = 6.0 in this research)
- ii) The range of Courant numbers (of order 1 to 1000) for which the ε has to be valid. An increase in the Courant number generally requires an increase in the number of sub-iterations.

In the absence of a rigorous theory to determine an ideal ε to satisfy the above conditions, an empirical approach was adopted to determine the optimal ε . It was determined by numerical experimentation that the choice of ε can be made independently of the Mach number of the test problem. **This is a significant finding** because it implies that, for a particular Courant number, the same value of ε can be used as a stopping criteria for a wide range of Mach numbers (i.e., for incompressible and compressible flows). It was also determined by numerical experimentation and observation, that ε can be related to the Courant number, λ , by the following relation:

$$\begin{aligned} \varepsilon &= 0.5 & 0 < \lambda \leq 10 \\ &= \frac{1}{\log_{10}(\lambda^2)} & \lambda > 10 \end{aligned} \tag{6.1}$$

The use of the above criteria for ε provides a stable GMRES scheme for all of the test cases in this research. It is also valid for both the preconditioners (LUSSOR and block ILU) used with GMRES. It should be remarked that the use of equation 6.1 to specify ε as the stopping criterion successfully limits k to values below 10, for values of λ upto 1000 (as used in this research). The specific value of k depends

on the Courant number, the preconditioner, and also the particular test case. Since the value of k is variable, use of a fixed value of k for all possible applications of GMRES could affect the performance of the solver.

Numerical experiments were performed with values of ε lower than those suggested by equation 6.1. Lower values of ε translate to an increase in the number of sub-iterations. It was found that the convergence of the global iterative scheme is remarkably insensitive to the use of values of ε lower than those suggested by equation 6.1, i.e., solving the linear system more accurately (than necessary) at each time step does not improve the convergence rate of the global scheme. Instead, it only adds to the computational cost of each global iteration (in the form of increased sub-iterations) when preconditioned GMRES is used as a solver.

Hence, it can be concluded that equation 6.1 provides a good stopping criteria, valid over a wide range of applications of preconditioned GMRES. Even though the stopping criteria is developed empirically, it has been extremely effective in reducing the costs associated with GMRES, and contributes towards the practicability of employing GMRES as an algorithm over a wide range of fluid flow problems. In summary, k is a variable, with a maximum value of 10.

6.2 Framework for Comparison of Solvers

One of the main objectives of this research is to compare the performance of preconditioned Conjugate Gradient like methods with conventional iterative methods used in CFD. The particular methods chosen for a one-on-one comparison are preconditioned GMRES and Line Gauss-Seidel Relaxation (LGSR). It is sufficiently clear from the previous chapters that GMRES and LGSR are fundamentally dif-

ferent in their approach, when used to solve linear systems of equations. Hence, it becomes necessary to determine a common framework for comparing the two methods, in order to ensure a fair comparison of their relative performance.

6.2.1 Operation Counts as a Performance Index

The preceding discussion raises a basic issue often encountered in the field of algorithm development — how can the performance of a particular solver be judged against the performance of another solver ? The answer provided by computer scientists is to perform an ‘order-of-magnitude analysis of the operation count’ of the solvers, and use it as a performance index. This approach is the simplest and most straightforward in nature, and will be referred to as the ‘operation count’ method of comparing solvers. As an example, a solver requiring $O(\log N)$ operations is much better than a solver requiring $O(N)$ operations, when ‘operation counts’ are used as a performance index.

In the context of solving large, sparse, linear systems of size N , an operation count of $O(\log N)$ is improbable, if not impossible to achieve. Most iterative solvers used in modern CFD software have an operation count of $O(N)$ per time step. The same is also true of the two solvers being compared in this research. Thus, since both solvers require $O(N)$ operations, an order-of-magnitude comparison based on operation counts will not yield sufficient information about the relative strengths of the solvers. Another drawback of operation count comparisons is that they ignore important hardware and programming issues, which can severely affect the practical performance of a solver.

Thus, the ‘operation count’ performance index is of limited utility, and has

not been chosen for detailed comparisons of solvers and preconditioners. However, for the sake of completeness, the respective operation counts for GMRES and LGSR are now presented. Let N be the problem size of the linear system being solved. For a computational domain of size $jmax$ and $kmax$ in two dimensions, $N = jmax * kmax$ (note, that the actual number of unknowns is $N*4$, but the problem size equals N on the block level). The operation count is a function of the problem size (N) and the number of sub-iterations (k) performed at each time step. The operation count of all the solvers and preconditioners used in this research is summarized below.

LGSR	$N * 410$
GMRES	$N * (205k + 12k^2)$
LUSSOR	$N * (374 + 272k)$
BILUF	$N * (700 + 172k)$
GMRES/LUSSOR	$N * (374 + 477k + 12k^2)$
GMRES/BILUF	$N * (700 + 377k + 12k^2)$

It may be noted that the number of operations for a single global iteration of LGSR is constant, whereas that for GMRES is variable, and depends on the number of sub-iterations, k . Also, for the minimal possible value of $k=1$, preconditioned GMRES requires more operations per global iteration than the LGSR solver. However, it is also clear that both the competing solvers do have an operation count of $O(N)$ for large values of N (as will be shown later, N varies from 3000 to 10,000 for the three test cases in this research). Thus, it can be concluded that for large N (which is indeed true in this research), both competing solvers have similar operation counts as revealed by an order-of-magnitude analysis.

6.2.2 *Number of Iterations as a Performance Index*

Another standard and widely accepted approach to judge the performance of solvers is to study the convergence rate provided by the particular solver. The idea is to examine the rate of convergence of the global non-linear problem as a function of the number of global iterations (or time steps or solution updates), without regard of the work per iteration. This idea is very useful and significant because it provides a framework for a pure and direct one-on-one comparison between solvers. Simply stated, if one solver converges more rapidly (in terms of number of global iterations) to the desired steady-state solution, then it can be competitive as a solver in modern CFD.

The ‘number of iterations’ performance index has been chosen in this research to compare the two solvers. The comparison is based on the number of iterations required to reduce the l_2 norm of the residual vector (normalized by the norm of the initial residual vector) of the non-linear problem by ten orders-of-magnitude. The non-linear problem is declared solved when this residual reduction criterion is met. Ideally, one would like to examine the reduction in the non-linear error, rather than the reduction in the non-linear residual. However, the error vector is as difficult to determine as the true solution itself, and is impractical to compute at each iteration. On the other hand, the residual vector is already available at each iteration, since it is computed as the right-hand-side of the linear problem.

Recall, that the steady-state solution of equation 3.16 requires the residual vector to equal zero. This is consistent with the criteria of reducing the norm of the residual vector by ten orders-of-magnitude. It is often argued, and apparently correctly so, that a 2–3 order-of-magnitude reduction of the non-linear residual is sufficient to

obtain solutions within the limits of engineering accuracy. Hence, there seems to be no practical justification in imposing a seemingly strict criteria of a ten orders-of-magnitude reduction for the residual. However, there are several instances when this strict criteria is required, and very often necessary. Some of these instances are now discussed.

Firstly, if the initial guess to the solution is close to the true solution, then a greater (than 2–3 orders) reduction in the residual is necessary to provide the correct steady-state solution. Such instances arise when, say, a small perturbation (by changing the set of boundary conditions or the grid) is applied to a known solution (on a particular set of boundary conditions and grid). Secondly, very high residual reductions are often required when results from CFD codes are used as inputs to perform optimization of design parameters. In such cases, results to engineering accuracy are unacceptable since they may introduce large errors while evaluating sensitivities of critical design parameters. These errors arise because a 2–3 orders residual reduction only resolves global phenomena (like the pressure-field) but may not accurately resolve local phenomena (like heat transfer). Thirdly, if the underlying non-linear problem is stiff, then a 2–3 orders residual reduction may not be sufficient, even to provide results to engineering accuracy. This has been clearly illustrated by the results of the first (backward-facing step) test case in this research.

The final reason for performing the ten orders residual reduction is to ensure that the overall algorithm has been programmed correctly and that the physical problem has been correctly transformed into a discrete numerical problem. The true steady-state will not be achieved if there are any errors in the initial conditions, boundary conditions and/or spatial discretization of the problem. The ability of a solver to

converge the iterates to the true steady-state, for a wide range of problems, also provides positive evidence of the robustness of the solver.

In light of the above discussions, the ‘number of iterations required to obtain a ten orders residual reduction’ has been used as a performance index to evaluate the relative performance of competing solvers and preconditioners in this research.

6.2.3 Computational Time as a Performance Index

One of the issues that can influence the success or failure of a solver is the actual amount of computational or Central Processing Unit (CPU) time required by the solver to perform each global iteration. The total CPU time consumed to obtain a converged solution can thus be used as a criteria to judge the performance of a solver. Some issues related to using CPU time as a performance index are now discussed.

The CPU time of an algorithm can be heavily influenced by the skills of the individual programmer. Efficient implementations often require investments in ‘real’ time and patience by the programmer. In addition, an implementation on one particular machine may run faster or slower on another machine, i.e., the CPU time may be machine dependent. Matters are further complicated by factors like compiler optimization, vectorization and parallelization of the algorithm.

The advances in machines with vector pipeline architectures have made scalar machines obsolete, particularly in the context of state-of-the-art CFD computations. The extent to which an algorithm is ‘vectorizable’ can reduce its total CPU time dramatically over that of a purely scalar implementation. All the test results

presented in this research were obtained with vector implementations of algorithms executed on a computer with vector processing facilities. Hence, the CPU time comparisons of the preconditioned GMRES and LGSR solvers (and the comparisons of preconditioners) do account for the intrinsic differences in their levels of vectorization.

Parallel and distributed computing can also afford rapid reductions in the total CPU time of an algorithm. The concept of parallelization is fundamentally different from that of vectorization, although both share the common objective of improving the viability of large-scale computing. This research has focussed on vectorization issues in detail, and no efforts have been presently made to parallelize any of the solvers or preconditioners. It is however well accepted that Conjugate Gradient like algorithms can be successfully adapted for use with parallel architectures [57].

It is clear that the CPU time of a solver may depend on several factors. Extensive efforts have been made in this research to implement the solvers and preconditioners in their most efficient and vectorizable forms. Since all comparisons have been done using the same computer and the same level of compiler optimization, the issue of machine dependence has been partly eliminated.

It may be remarked that the number of iterations to convergence is completely independent of all the factors that influence the CPU time, and can thus be said to reflect the true convergence characteristics of a solver. The CPU time comparisons are useful, and in conjunction with the number of iterations comparisons, provide an estimate of the practical utility of a solver. In summary, the effectiveness of a solver has been measured in terms of 'number of iterations' and 'CPU time' to convergence, with the two criteria complementing each other fully.

6.3 Overview of Test Cases

Three fluid flow problems representing flow in the low speed, transonic speed and hypersonic speed regimes were selected for this research. The flow physics for all three problems are fairly complex and encompass a wide range of flow features often encountered in fluid flow problems in engineering practice.

The first problem, low speed flow over a backward-facing step, illustrates the phenomena of flow separation and recirculation in internal flows. Extensive experimental and theoretical investigations have been performed for this flow by Armaly et. al [58]. The second test case, trailing edge flow in a convergent turbine cascade, has been the focus of extensive experimental investigations by Dietrichs et. al [59] and Sieverding et. al [60]. The third, hypersonic shock-on-shock interactions on a cylindrical leading edge, is of interest to researchers of the National AeroSpace Plane, and has been examined in detail by Wieting [61] and Tannehill et. al [62].

All calculations were performed with the upwind finite volume approach discussed in Chapter 3. Excellent qualitative comparisons with experimental data have been obtained for all the test cases examined. However, it is not the intent of this work to present detailed comparisons of computational and experimental data. It should be pointed out that detailed code validation results have been presented earlier (using other test problems) for the code being used in this research [63]. The results presented in this work stress the relative performance of the solvers and preconditioners being examined, which is consistent with the overall goals of this research. All computations have been performed with the following considerations :

- i) The thin-layer assumption to the full Navier-Stokes equations has been used in all the test cases.
- ii) All computations are performed using a single processor of a Cray-YMP/8. The code was compiled in fully vector mode.
- iii) All test cases are started with freestream flow as the initial guess. Primitive (rather than conserved) variables are used for interpolation.
- iv) All flow variables are second-order accurate, fully-upwinded in the ξ direction (i.e., $\kappa_\xi = -1$), and third-order accurate, upwind-biased in the η direction (i.e., $\kappa_\eta = 1/3$).
- v) The implicit, global operator is discretized in a first-order accurate manner. This is done to assure stability of the LGSR solver (see section 4.3.1), and to save on storage and computational costs.
- vi) Laminar flow only has been computed. This has been done to preclude the complexities arising in modeling of turbulent flows. The results presented can be extended to turbulent flows with appropriate turbulence modeling.
- vii) A maximum of 20,000 global iterations (or time steps) is permitted for all test cases. The sub-iterations for GMRES are controlled by equation 6.1. The initial vector for each GMRES global iteration is the solution vector from the previous global iteration. The relaxation parameter (ω) for LGSR is set equal to one. In the absence of a general theory for determining an 'optimal' ω for the Navier-Stokes equations, the value of $\omega = 1$ provides the fastest convergence rate for the LGSR solver. It must be remarked that $\omega > 1$ (i.e., over-relaxation) is not

a stable choice for the LGSR solver, and $\omega < 1$ (i.e., under-relaxation) can only serve to slow down the convergence rate. Alternate sweeps in the horizontal and vertical directions are used for stability of the LGSR solver.

viii) Local time-stepping is used for the time-integration. The initial solution is iterated with Courant numbers of order one, till the initial transients in the solution are eliminated. The Courant number is then ramped to a value which provides the asymptotic convergence rate for each solver. For the LGSR solver, the asymptotic λ represents the stability limit, i.e., use of λ greater than the asymptotic λ causes the iterations to diverge. For the GMRES solver, the Courant number is increased till the maximum of ten sub-iterations (as discussed in section 6.1) is reached. The use of the asymptotic λ provides the true asymptotic rate of convergence for the solvers being compared.

6.4 *Low Speed Flow over a Backward Facing Step*

Armaly et. al [58] have presented detailed measurements of velocity distribution and reattachment length for the incompressible flow of air downstream of a single backward facing step in a 2-D channel. The results show that the various flow regimes (in the Reynolds number range of $70 < Re < 8000$), are characterized by typical variations of separation length with Reynolds number. The Reynolds number is based on the height of the step, and two-thirds of the maximum inflow velocity at the step. The particular test case chosen for this research corresponds to $Re \approx 400$, since the experimental data suggests that $Re > 400$ produces 3-D variations in the flowfield.

The numerical computations are performed on a mesh with 61 and 51 points in

the ξ (streamwise) and η (normal) directions, respectively. The grid is shown in Figure 3. Grid points are clustered, both in the normal and streamwise directions, to resolve the various viscous gradients and the reattachment point of the separated flow. A freestream Mach number of $M_\infty = 0.1$ and Reynolds number of $Re_\infty = 387$ is specified. Conserved (rather than primitive) variables are used for interpolation, in this test case. The full Navier-Stokes terms are not included during the computations because the coarseness of the grid in the η direction will prevent resolution of these terms. Moreover, as will be shown later, extremely accurate physical results are obtained even without the full Navier-Stokes terms.

Adiabatic, no-slip boundary conditions are specified on the top and bottom walls forming the boundaries of the channel, and on the lower portion (which defines the step) of the inflow boundary. For fully developed subsonic flow at the outflow boundary, three variables (ρ , u and v) are extrapolated and one variable (stagnation enthalpy) is held constant. The inflow boundary condition presented a considerable challenge for this problem and is now discussed.

According to characteristic wave theory, the proper specification of boundary conditions for subsonic inflow requires one physical condition and three numerical conditions to be satisfied. Since only one physical quantity can be fixed, and the inflow (parabolic) velocity profile requires both components of velocity (u and v) to be fixed, it seems unlikely that a purely parabolic velocity profile can be maintained at the inflow.

One approach used in external flow calculations is to specify a variation of stagnation enthalpy at the inflow, which in turn provides the desired velocity profile. This, however, does not prove to be a stable inflow boundary condition for this

BACKWARD FACING STEP
61x51 GRID

Figure 3. Computational Grid for Backward Facing Step

internal flow case. The problem is resolved by imposing a profile of Reimann invariants at the inflow boundary. The velocity profile obtained with this boundary condition does vary in time, but is sufficiently stable to simulate the incoming flow in an accurate manner. The Reimann invariants act as a non-reflecting boundary condition [64], and are thus successful in rapidly eliminating the initial transients in the solution, as will be seen in the convergence history plots.

Figure 4 shows Mach number contours obtained from the computations on the 61×51 grid. The nature and size of the separation and recirculation behind the step closely matches the physical description of the flow as obtained in the experiments of Armaly et. al [58]. It should be remarked that the reattachment point is the primary flow feature used to characterize the flow in the experimental data [58]. For $Re = 387$, the experimental results suggest a downstream reattachment length (X_R) to step height (S) ratio of $X_R/S = 7.9$. The numerical computation predicts $X_R/S = 8.0$. Thus, the experimental and numerical reattachment lengths are extremely close to each other. This represents a particularly good numerical solution, in light of the fact that onset of reattachment is more difficult to predict than onset of separation. The reattachment point is confirmed by examining the plot of velocity vectors shown in figure 5.

Figure 6 presents the convergence history comparisons between GMRES (with LUSSOR and BILUF preconditioning) and the Line Gauss-Seidel Relaxation (LGSR) solver. The ‘logarithm of the l_2 norm of the residual’ has been plotted against the ‘number of global iterations’ (or time steps). The curves marked ‘A’ and ‘B’ show the convergence of GMRES with LUSSOR (abbreviated as GMLUS) and GMRES with BILUF (abbreviated as GMILU) preconditioning, respectively. The curve marked ‘C’ shows the convergence rate of the LGSR solver. Similar

BACKWARD FACING STEP MACH NUMBER CONTOURS

Figure 4. Mach Number Contours for Backward Facing Step

**BACKWARD FACING STEP
VELOCITY VECTORS
SEPARATION REGION**

Figure 5. Velocity Vectors for Backward Facing Step

markings will also be seen on plots for the other two test cases. It can be clearly seen that both GMLUS and GMILU converge at a much faster rate than the LGSR solver. The asymptotic Courant numbers (λ) for GMLUS, GMILU and LGSR are 200, 200 and 10, respectively. Several features of the convergence rate histories are now discussed.

The correct physical solution (i.e., the proper reattachment point) is obtained after a six order reduction of the residual is achieved. The rapid reduction in the initial residual (from zero to three orders) required the use of Reimman invariants at the inflow boundary (as discussed earlier). This rapid reduction represents elimination of part of the initial transient. The need to reduce the residual by six orders to obtain the correct reattachment point reinforces the earlier argument (of section 6.2), that residual reductions of greater than 2–3 orders are often necessary.

Curves A (for GMLUS) and B (for GMILU) are generated with an initial Courant number (λ) of 100, with an increase to $\lambda = 200$ after 100 global iterations. The LGSR solver (Curve C) permits an initial Courant number of $\lambda = 10$, and does not permit any increase in the Courant number (i.e., the iterates diverge at higher values of λ), even after the correct physical solution has been obtained. GMLUS and GMILU converge in approximately 2000 global iterations. LGSR, however, reaches a six order reduction (i.e., the correct physical solution) in 15,000 global iterations and is estimated to require about 56,000 iterations (according to convergence rate estimates) to attain a ten order reduction (i.e., convergence). Hence, in terms of the number of overall global iterations to convergence, GMRES (with either preconditioner) is significantly (about 28 times) faster than the LGSR solver.

The CPU time comparison of the solvers is shown in figure 7. The superior

Figure 6. 'Number of Iterations' Comparison for Backward Facing Step

efficiency of GMLUS (curve A) is seen in this comparison. GMILU requires almost five times as much CPU time as GMLUS. The LGSR solver is again extremely slow in terms of total CPU time, as it was in terms of number of iterations. LGSR is estimated to require about 15,000 seconds of CPU time for convergence. Thus, LGSR requires about 20 times more overall CPU time than that required by GMLUS.

One of the goals of this research is to develop stable and efficient preconditioning for the GMRES algorithm, for use with the Navier-Stokes equations. The results of figures 6 and 7 contribute to this development. The ‘number of iterations’ comparison shows that both the LUSSOR and BILUF preconditioners are stable, and are equally effective in converging to the steady state. The CPU time comparison shows that the use of LUSSOR preconditioning can afford considerable gains in CPU time over the use of the BILUF preconditioner, while maintaining a competitive convergence rate. This is a significant step towards the identification of a stable and efficient preconditioner.

The above preconditioner comparison reveals a significant result — any stable preconditioner can be used with GMRES, to obtain a better convergence rate than the LGSR solver. Even though the overhead cost for one preconditioner (BILUF) may be more than that for another (LUSSOR), the use of either one guarantees an improved convergence rate. Hence, for implementing preconditioned GMRES in new or existing codes, any readily available, stable preconditioner can be employed.

6.5 *Trailing Edge Flow in a Transonic Cascade*

In order to fully understand the flow physics near the trailing-edge of a transonic turbine cascade, Sieverding et. al [60] conducted experimental tests on a model

Figure 7. 'CPU Time' Comparison for Backward Facing Step

simulating the flow in the overhang section of convergent turbine cascades. The experimental setup is shown in figure 8. A primary goal of the experiment was to study the flow physics around and behind the blunt trailing edge of the cascade (as modeled by the trailing edge of the flat plate). Correspondingly, the computational results presented here concentrate on resolution of the flow in the trailing edge region. Of the various experimental configurations tested in reference [60], the one chosen for numerical simulation in this work corresponds to a flat plate overhang length (l_s) of 37mm, and maximum inclination of the tailboard (simulating complete loading of the cascade). It must be mentioned that the experiment [60] was performed with fully turbulent flow. However, the numerical simulation of this work is restricted to laminar flow only, and hence detailed comparisons of surface pressure etc. are not presented here.

The overhang length of $l_s = 37\text{mm}$ corresponds to a maximum suction side Mach number of 1.3 at the trailing edge. The maximum pressure side Mach number is 1.0. The suction side flow undergoes a shock free expansion upto the trailing edge. The blade pressure distribution on the pressure side is independent of the overhang length. The Reynolds number of the flow is $3 * 10^7 m^{-1}$, based on the unit length of the flat plate.

The computational results are obtained on a mesh with 207 and 51 points in the ξ and η directions, respectively. The mesh is generated with the GRAPE code [65], and is shown in figure 9. Grid points are clustered around solid surfaces (i.e., the flat plate and the outer walls of the setup) and particularly packed around the trailing edge of the flat plate. A freestream Mach number (based on the upstream total pressure, the pressure distribution on the pressure side, and the maximum suction side Mach number of 1.3) is calculated from inviscid theory and specified

Figure 8. Experimental Setup for Transonic Cascade [60]

as 0.6. The freestream Reynolds number is set to $3 * 10^7 m^{-1}$.

Adiabatic, no-slip boundary conditions are specified for all solid surfaces. The inflow boundary (consisting of the cells forming the semi-circular part of the grid) is specified with subsonic inflow conditions ($v = 0$, fixed entropy and stagnation enthalpy, extrapolation of u from inside the domain). Standard periodic boundary conditions for C-meshes are applied across the centerline of the wake cut. At the outflow boundary, a back-pressure, $P_b = 0.4P_\infty(\text{freestream})$ is initially specified, in order to start the problem. After the initial transient is overcome, this boundary condition is changed to enforce extrapolation of all flow variables to the outflow boundary. This change is necessary to maintain stability of the overall solution algorithm.

Figure 10 shows a comparison of Mach number contours generated from computational results and an experimental shadowgraph for the shock system at the trailing edge of the flat plate. The computed trailing edge shocks and expansion waves, along with the location and inclination of the wake centerline, compare very well with the experimental data. Figure 11 further details the flow around the trailing edge, and the various flow features seen are consistent with the findings of Dietrichs et. al [59].

Figure 12 details the convergence characteristics of the GMRES/LUSSOR (GMLUS), GMRES/BILUF (GMILU) and LGSR solvers. For this test case, the correct physical solution is obtained after a three order residual reduction. The asymptotic Courant numbers are then used (to obtain the asymptotic convergence rate), and equal 1000, 1000, and 5 for GMLUS, GMILU and LGSR, respectively. Similar to the previous test case, both GMLUS and GMILU converge to the steady-state so-

TRANSONIC CASCADE
207x51 GRID

Figure 9. Computational Grid for Transonic Cascade

TRANSONIC CASCADE
MACH NUMBER CONTOURS
TRAILING EDGE REGION

Figure 10. Mach Number Contours for Transonic Cascade

TRANSONIC CASCADE
VELOCITY VECTORS
TRAILING EDGE REGION

Figure 11. Velocity Vectors around trailing edge of Transonic Cascade

lution in about 1300 global iterations. LGSR is estimated to converge (on the basis of its asymptotic convergence rate) in about 27,000 global iterations. Thus, GMRES (with either preconditioner) converges 20 times faster than LGSR, in terms of overall number of global iterations to convergence.

Figure 13 presents the CPU time comparison of the GMRES (with BILUF and LUSSOR preconditioning) and LGSR solvers. The tremendous gains in using preconditioned GMRES instead of LGSR are again clearly evident from this figure. As for the previous test case, BILUF preconditioning proves to be more expensive than LUSSOR preconditioning, and requires about five times more CPU time. However, both GMLUS and GMILU are more efficient than LGSR, which is estimated to require roughly 24,000 seconds of CPU time to converge. Thus, in terms of overall CPU time to convergence, GMLUS (1300 seconds) and GMILU (6500 seconds) are roughly 20 and 4 times faster than LGSR, respectively.

6.6 Shock-on-Shock Impingement in Hypersonic Flow

Extensive studies of shock-on-shock impingement, and its effect on heat transfer rates and related phenomena on a cylindrical leading edge, were conducted by Wieting [61]. The study was motivated by a need to gain insights into the flow physics of (ramp) shocks impinging on a two-dimensional viscous inlet cowl. The flowfield has complex shock-shock and shock-shear layer interactions, and is difficult to simulate numerically [62].

Edney [66] has classified shock-shock interactions into six categories, based on the interference nature of the impinging shock with the main bow shock. For the purpose of this research, the ‘Type II’ interference pattern of Edney [66] has been

Figure 12. 'Number of Iterations' Comparison for Transonic Cascade

Figure 13. 'CPU Time' Comparison for Transonic Cascade

chosen for numerical simulation. A schematic of this pattern is shown in figure 14. The pattern is created when the impinging shock interacts with the bow shock at a shock angle of 15° . The flow is characterized by a displaced bow shock, a transmitted shock and formation of supersonic shear layers. The Reynolds number of the flow is $1.8 * 10^5$, per unit diameter of the cowl lip.

Computations are performed on a mesh with 101 points each in the ξ and η directions. The mesh is shown in figure 15. Grid points are clustered near the cowl surface, to resolve the viscous boundary layer. A freestream Mach number of 5.94 is specified for the computation. Due to the presence of strong shocks in the flowfield, flux-limiting is required to limit the flux-gradients. Since it is difficult to establish the exact location of the impinging shock from the experiment [67], trial-and-error is used till the desired interference pattern for a 'Type II' interaction is established.

The incoming oblique shock at the inflow boundary is simulated by imposing the appropriate jump conditions (with respect to freestream values) calculated from inviscid shock theory. Approximately 75% of the points at the inflow boundary are assigned fixed freestream values, and the remaining 25% of the points are 'overspecified' with the jump conditions for a 15° shock angle. The two outflow boundaries are treated with simple extrapolation of all variables to the respective outflow boundaries. Adiabatic, no-slip boundary conditions are imposed at the cowl surface.

The Mach number contours obtained from the computations are shown in figure 16. The shear layers and displacement of the bow shock are captured well by the computation. The results compare excellently with similar calculations done by Kloptex and Yee [67]. The transmitted impinging shock (as seen in the schematic of figure 14), which is very weak, is not captured. Extensive local grid refinement

Figure 14. 'Type II' Pattern defined by Edney [66] (not to scale)

SHOCK ON SHOCK IMPINGEMENT
101x101 GRID

Figure 15. Computational Grid for Hypersonic Shock

and grid adaptation is probably required to capture this weak shock.

Figure 17 presents the ‘number of iterations’ comparison of the GMLUS, GMILU and LGSR solvers. The correct physical solution is established after a four order reduction in the residual. The asymptotic Courant numbers for the GMLUS, GMILU and LGSR solvers are 50, 50 and 10, respectively. The LGSR solver converges in about 14,000 iterations for this test case. Contrary to what was observed in the two previous test cases, the GMRES scheme converges at different rates for the two different preconditioners. The reason for this difference in convergence rates is not clear yet. Specifically, GMLUS and GMILU require 3150 and 1870 global iterations, respectively, for a ten orders residual reduction. Thus, BILUF preconditioning of GMRES has a better asymptotic rate of convergence than LUSSOR preconditioning, for this hypersonic flow test case. When compared with the LGSR solver, GMLUS and GMILU are 4 and 7 times faster, in terms of number of iterations to convergence.

CPU time comparisons for the different solvers are shown in figure 18. It is interesting to note that although the BILUF preconditioner took the least number of global iterations to converge, it requires almost the same amount of CPU time as the LGSR solver. In contrast, the LUSSOR preconditioned GMRES solver is very efficient in terms of CPU time, and is four times quicker than either GMILU or LGSR. Specifically, GMLUS, GMILU and LGSR require 2600, 8500 and 10,200 seconds of CPU time for convergence, respectively.

6.7 *Summary of Results*

A comprehensive summary of key results is presented in table 1. Detailed results

SHOCK ON SHOCK IMPINGEMENT MACH NUMBER CONTOURS

Figure 16. Mach Number Contours for Hypersonic Shock

Figure 17. 'Number of Iterations' Comparison for Hypersonic Shock

Figure 18. 'CPU Time' Comparison for Hypersonic Shock

Table 1. Comprehensive Summary of all Three Test Cases

Backward Facing Step			
	GMLUS	GMILU	LGSR
Total Iterations	1825	2095	56,555
Total CPU time	723	3699	14,761

Transonic Cascade			
	GMLUS	GMILU	LGSR
Total Iterations	1269	1289	26,656
Total CPU time	1300	6472	23,497

Hypersonic Shock-on-Shock			
	GMLUS	GMILU	LGSR
Total Iterations	3144	1870	14,040
Total CPU time	2655	8518	10,109

Table 2. Summary of Backward Facing Step Results

		GMLUS	GMILU	LGSR
Total	Iterations	1825	2095	56,555
Total	CPU time	723	3699	14,761
Speed-up	(iterations)	31	27	—
Speed-up	(CPU time)	20.4	4.0	—
Iteration	per order	277	338	10,470
Asym. Speed-up	(Iter)	37.8	31	—
CPU time	per order	110	597	2733
Asym. Speed-up	(CPU)	24.8	4.6	—

Table 3. Summary of Transonic Cascade Results

		GMLUS	GMILU	LGSR
Total	Iterations	1269	1289	26,656
Total	CPU time	1300	6472	23,497
Speed-up	(iterations)	21	20.7	—
Speed-up	(CPU time)	18.1	3.6	—
Iteration	per order	77	81	5377
Asym. Speed-up	(Iter)	69.8	66.4	—
CPU time	per order	93	435	4740
Asym. Speed-up	(CPU)	51	10.9	—

Table 4. Summary of Hypersonic Shock-on-Shock Results

		GMLUS	GMILU	LGSR
Total	Iterations	3144	1870	14,040
Total	CPU time	2655	8518	10,109
Speed-up	(iterations)	4.5	7.5	—
Speed-up	(CPU time)	3.8	1.2	—
Iteration	per order	326	133	2014
Asym. Speed-up	(Iter)	6.2	15.1	—
CPU time	per order	285	638	1450
Asym. Speed-up	(CPU)	5.1	2.3	—

for the three test cases are tabulated in tables 2, 3 and 4, respectively. The various parameters used in the summary are listed below:

1. Total number of iterations required for convergence to the steady state (i.e., ten orders-of-magnitude reduction of the residual).
2. Total amount of CPU time consumed by each solver for convergence.
3. Overall speed-up factor calculated as the ratio of LGSR iterations to preconditioned GMRES global iterations.
4. Overall speed-up factor calculated as the ratio of LGSR CPU time to preconditioned GMRES CPU time.
5. Iterations required (based on the asymptotic convergence rate) for a one order-of-magnitude reduction in the residual, i.e., slope of the convergence rate curve.
6. Asymptotic Speed-up factor in terms of number of Iterations to convergence.
7. CPU time required (based on the asymptotic convergence rate) for a one order-of-magnitude reduction in the residual, i.e., slope of the CPU time curve.
8. Asymptotic Speed-up factor in terms of number of CPU time to convergence.

6.8 Discussion of Results

The discussions in this section are motivated by the need to further qualify some of the results presented and summarized in the previous two sections. Attempts have

also been made to explain some of the results obtained with the different solvers and preconditioners. In order to facilitate the discussions, figures 6, 12 and 17, and figures 7, 13 and 18, have been combined into figures 19 and 20, respectively.

It should be remarked that all the comparisons of solvers and preconditioners have been done for actual, complex problems of engineering interest. This approach is refreshing in that it avoids the use of ‘model’ problems and ‘model’ equations for the comparisons — for it is very often the case that results obtained under ‘model’ conditions do not apply under ‘real’ conditions.

An examination of the ‘number of iterations’ convergence histories of the three test cases reveals that the convergence histories can be divided into two distinct phases. The first phase is called the ‘initial transient’ phase. This phase represents the period in the time-integration when the high-frequency errors are eliminated from the solution. This phase is marked by a rapid reduction in the initial residual (or error), and resolves certain ‘global’ phenomena in the physical solution. The completion of the ‘initial transient’ phase is marked by a rapid drop-off in the rate of residual reduction per time-step, and signals the beginning of the second phase of the convergence history.

The second phase is called the ‘asymptotic’ phase of residual reduction. This phase represents the period when the low-frequency errors are eliminated, and ‘local’ phenomena affecting the physical solution are resolved. The elimination of the low-frequency errors can dominate the total amount of computational work performed to achieve the final steady-state solution, particularly if low time-steps are used in this phase of the time-integration. The use of implicit methods is designed to use large time-steps in the ‘asymptotic’ phase, to accelerate convergence to the

Figure 19. ‘Number of Iterations’ Comparison for all Three Test Cases

Figure 20. 'CPU Time' Comparison for all Three Test Cases

steady-state, and hence to reduce the total number of iterations to convergence.

The transition point between the initial transient phase and the asymptotic phase is not clearly defined, and depends on the nature of the physics of the problem. One approach often adopted is to switch to the asymptotic phase after the residual has dropped by 2–3 orders-of-magnitude. It is desirable to switch to the asymptotic phase (and raise the time-step to accelerate convergence) as quickly as possible. The time-step (or Courant number) is then raised to the maximum allowable limit (within the limits of stability), in inverse proportion to the decrease in the norm of the residual, i.e., a one order residual reduction causes a ten-fold increase in the Courant number. This strategy, though popular, may be unstable if the residual is not decreasing in a smooth and regular fashion.

For this research, the switch to the asymptotic phase has been made as soon as it is so “allowed” by the stability requirements of the problem. The subsequent iterations are performed with a fixed, maximum allowable Courant number. The switch is “allowed” after 100, 700 and 1000 iterations for the backward-facing step, the transonic cascade and the hypersonic flow case, respectively. It must be mentioned that the progress of the iterations during the initial transient phase has to be carefully monitored, to reach the switch-over point as quickly as possible. Hence, the convergence history upto the switch-over point can vary considerably, depending on how the initial transient is eliminated. However, the convergence history in the asymptotic phase is obtained with a fixed Courant number, and the iterations can proceed smoothly without any input from the user. Thus, the repeatability of the asymptotic convergence rate is guaranteed with the respective asymptotic Courant numbers (λ).

The use of the different values of λ for the preconditioned GMRES and LGSR solvers may raise some question about the ‘fairness’ of the convergence rate comparisons, particularly since comparatively higher values of λ are used with preconditioned GMRES. The superior convergence rate of the preconditioned GMRES solver can be attributed to two factors — the higher “allowed” values of λ , and the use of preconditioning. The role of preconditioning in accelerating convergence can be determined by using preconditioned GMRES at the same λ as the LGSR solver — the result is that preconditioned GMRES continues to have a superior convergence rate than the LGSR solver, although the speed-up factor is reduced.

It must be remarked that the use of preconditioning enables the GMRES solver to accept higher values of λ than those “allowed” by the LGSR solver. It is hence only prudent to take advantage of the increased stability afforded by the use of preconditioning, and increase the convergence rate of the preconditioned GMRES solver by using large Courant numbers. The maximum speed-up is thus obtained by using the maximum allowable values of λ . The preconditioned GMRES solver can accept even higher values of the Courant number than those used in this research, but this requires an increase in the number of sub-iterations (and hence, increase in the storage and CPU time per time-step), without any appreciable increase in the overall convergence rate. The choice of λ is thus influenced by the need to obtain the best computational efficiency with the preconditioned GMRES solver, and by the need to limit the storage for the sub-iterates (as discussed in section 6.1).

In the absence of any preconditioning, it was observed that the GMRES solver requires considerably more sub-iterations (than the upper limit of ten sub-iterations with preconditioned GMRES) to provide an overall stable iterative scheme (even at low Courant numbers of order one). The unpreconditioned GMRES is thus

impractical as a solver at low Courant numbers, and is unstable at high Courant numbers.

One particular characteristic of the LGSR solver is the distinct decrease in the residual reduction per time-step at some point in the asymptotic convergence history. The change in convergence rate is seen in all the three test cases, and is particularly pronounced in the transonic and hypersonic cases, as can be seen in figure 19. This phenomenon is due to the ‘smoothing property’ of the solver. The dependence of the convergence rate of an iterative scheme on the distribution of eigenvalues of the iteration matrix, was detailed in section 4.1. It is clear that as the components of error associated with some eigenvalues are eliminated, the spectral radius of the iteration matrix changes, and so does the convergence rate. The existence of such a varying asymptotic convergence rate is characteristic of all solvers based on the Gauss-Seidel relaxation method. The preconditioned GMRES solver does not have this ‘smoothing’ property because the use of preconditioning continually provides an iteration matrix with a fairly constant spectral radius, so that the asymptotic convergence rate is fairly constant.

The asymptotic speed-up factors (iterations) for the three test cases are very different from one another. The factor for the transonic cascade is the highest, amongst the three test cases. This is consistent with the observations of Wong [48], Wigton et. al [49], and Venkatakrishnan [50], that preconditioned Conjugate Gradient like methods are effective in accelerating convergence when applied to subsonic and transonic flows, i.e., when the flow is mainly elliptic in nature. The distribution of eigenvalues encountered in such flows is very well suited for acceleration with preconditioned Conjugate Gradient like methods. As the eigenvalue distribution becomes uniform, the gains with Conjugate Gradient like methods decrease,

because the problem cannot be ‘accelerated’. This is clearly corroborated by the drop-off in the speed-up factor for the hypersonic case.

The differences in the distribution of eigenvalues can be seen by considering the eigenvalues of a simplified Euler problem. In two dimensions, the eigenvalues of the Euler equations are u , u , $u + a$ and $u - a$. For $M=0.6$ (transonic case), $u = 0.6a$, and for $M=6.0$ (hypersonic case), $u = 6.0a$. Thus, the eigenvalues for the transonic case are $0.6a$, $0.6a$, $1.4a$ and $-0.4a$, and those for the hypersonic case are $6.0a$, $6.0a$, $7.0a$ and $5.0a$, respectively. It is thus clear that the transonic case has a more non-uniform distribution of eigenvalues than the hypersonic case, and is thus more suited to acceleration by Conjugate Gradient like methods.

The backward-facing step test case also has a non-uniform spectrum of eigenvalues ($0.1a$, $0.1a$, $1.1a$ and $-0.9a$), and thus benefits greatly from the use of preconditioned GMRES. One particular observation with this case is that the speed-up factor is fairly close to the overall speed-up (in iterations) factor. This is because the initial Courant number (100) is very close to λ (200). This test case shows that preconditioned GMRES handles the underlying stiffness characteristics of incompressible problems very well. The speed-up factor in this case is lower than that of the transonic case, partly because the λ ratio (200/10) is lower by a factor of 10 (1000/5) with respect to the transonic case.

A thorough mathematical analysis of the competing solvers in this research could provide an in-depth explanations of why one particular solver (preconditioned GMRES) works better than another (Upwind Relaxation), or why one preconditioner is better than another. Such an analysis of the solvers (or preconditioners) would require a detailed eigenvalue analysis of each iterative scheme. The problems with

doing such an analysis are varied. The respective sizes of the implicit matrices for the three test cases makes it virtually impossible to do a conventional eigenvalue determination with standard software like LINPACK [68]. It may be mentioned that it is practical to do such an analysis only for small matrices, as may be encountered in 1-D problems or in 2-D problems with coarse meshes. A coarse mesh for the test problems of this research could be adopted for this purpose, but then there is no theory to provide a one-on-one correspondence between eigenvalues for a coarse and fine mesh.

It is possible to obtain estimates of extremal (maximum and minimum) eigenvalues for the GMRES solver [69]. However, the usefulness of such estimates is limited and questionable since a) the estimates may not be accurate, b) the estimates are not available when GMRES is preconditioned, and, c) extremal eigenvalues may provide an accurate condition number estimate, but more information is required to judge the effectiveness of any particular preconditioner (see section 5.1 for details). Thus, even though a complete eigenvalue analysis is desirable to fully explain the differences in convergence rates of the solvers, it is impractical for the large problems being tested in this research. The development of cheap and accurate methods to perform such eigenvalue analyses could serve as a good source of future work.

The BILUF preconditioner has a higher asymptotic convergence rate than the LUSSOR preconditioner, for the hypersonic test case. This is because the BILUF preconditioner retains more information from the global implicit matrix (V^n) than the LUSSOR preconditioner, and is hence ‘closer’ to $(V^n)^{-1}$ in character. Thus, BILUF is more ‘effective’ in this test case, but is still less ‘efficient’ than LUSSOR. The BILUF preconditioner does deserve further investigation into its role as a preconditioner, particularly for high-speed flow applications.

The CPU time comparisons reveal that the speed-up factors with the LUSSOR preconditioner are much higher than those for the BILUF preconditioner. The major reason for this difference is the lack of vectorizability of the BILUF preconditioner. The CPU time for the BILUF preconditioner can be reduced by reusing the preconditioner over a (user fixed) number of global iterations. This translates to performing the incomplete LU decomposition only every i^{th} iteration, and then reusing the incomplete LU for the next $(i-1)$ global iterations. This greatly reduces the overhead cost of computing the BILUF preconditioner at each time-step, and makes BILUF more competitive in terms of CPU time.

The CPU time for the CFD code used in this research can be reduced by using the ‘reuse of LU’ strategy. The flux Jacobian matrices are usually calculated at every time-step, to form the block-diagonals of the global implicit matrix. The calculation of these Jacobian matrices is a major computational task at each time-step. After the initial transient has been overcome, these matrices remain fairly constant over a number of time-steps. Thus, the Jacobian matrices can be stored and reused for a number of time-steps, without affecting the overall convergence rate of the iterative methods involved. The ReUse of LU (RELU) strategy, and its effect on CPU time, is discussed in Appendix E. The conclusion from the comparisons in Appendix E is that GMRES with LUSSOR preconditioning (i.e., GMLUS) is the most efficient solver.

7.0 Conclusions and Recommendations

This chapter serves to highlight the important efforts undertaken in this research to develop a general purpose solver for the Navier-Stokes equations. Conclusions are drawn from the results obtained and put into perspective with the original goals of the work. Certain recommendations and suggestions for future work are also included in this chapter.

7.1 *Conclusions from Present Work*

The central idea of this research has been to use a Generalized Conjugate Gradient Method to construct a general purpose solver for the solution of large, sparse systems of simultaneous equations encountered in CFD. The Generalized Minimal RESidual (GMRES) method of Saad and Schultz [40] has been used in this research as a representative method of the family of Conjugate Gradient Methods. It was remarked in Chapter 1 that preconditioning plays a key role in enhancing the viability of CGMs. The indispensability of preconditioning has been clearly established in this work, and the detailed examination of two preconditioners has greatly assisted in establishing this fact. The accelerated convergence afforded by the use of preconditioned GMRES has played a vital role in justifying the use of preconditioned

GMRES against conventional, popular solvers like Line Gauss-Seidel Relaxation (LGSR) and Approximate Factorization (AF).

The widespread applicability of the preconditioned GMRES solver has been demonstrated in this research by using three diverse test problems — incompressible flow over a backward facing step, transonic flow through a turbine cascade, and hypersonic flow on a cylindrical leading edge. The solver developed in this research performs with uniform success with all the test cases. This is a positive step towards developing a universal solver, and satisfies one of the original goals of this research.

Another aim of this research was to investigate preconditioning techniques for use with Conjugate Gradient like methods. Preconditioning transforms the original system of linear equations into an equivalent system, and is aimed towards reducing the overall computational cost of attaining a converged solution. This research establishes that the choice of an effective, stable preconditioner is extremely important to the success of GMRES. For this work, several preconditioners were investigated for their effectiveness and stability. An ‘effective’ preconditioner is defined as one that assists GMRES in obtaining rapid convergence, while requiring a minimal overhead cost for the preconditioning. A ‘stable’ preconditioner guarantees convergence of GMRES at large Courant numbers and ensures that preconditioned GMRES does not fail when applied to different flow problems.

Diagonal preconditioning — both scalar and block versions — was found to be unstable when applied to the three test cases. LGS relaxation and spatial AF, representing use of iterative schemes as preconditioners, were found to be stable preconditioners at low Courant numbers. However, they were unstable and ineffective in accelerating convergence for large Courant numbers. The LUSSOR (Lower-Upper

Symmetric Successive Over-Relaxation) scheme of Jameson and Yoon [34] and the BILUF (Block Incomplete Lower-Upper Factorization with zero fill-in) preconditioner emerged as two of the most stable, effective and widely applicable preconditioners. The LUSSOR and BILUF preconditioners were thus selected for detailed analysis and application in this research.

This research represents the first documented use of the LUSSOR scheme as a preconditioner in conjunction with generalized CGMs (GMRES in particular). The studies in this research reveal that both preconditioners (LUSSOR and BILUF) are equally effective in their ability to assist GMRES in achieving an excellent convergence rate. This shows that ‘effective’ preconditioning can be as useful as ‘optimal’ preconditioning. An ‘optimal’ preconditioner is defined here as the best possible, universal preconditioner that is applicable to all types of flow problems. The time and effort spent in finding the optimal preconditioner may not be worth the potential gains it might bring over an effective preconditioner. It is thus suggested that, as a first step, any readily available stable preconditioner be used to implement GMRES as a solver in an existing code.

The ability of the LUSSOR preconditioner to be as effective in accelerating convergence as the BILUF preconditioner is a significant finding related to development of preconditioners. BILUF has been one of the most popular preconditioning techniques used by researchers working in this field. The present work has shown that the LUSSOR technique can match BILUF in stability and effectiveness, but is considerably more efficient in terms of the overhead costs associated with the preconditioning effort. As seen from the CPU time comparisons in chapter 6, LUSSOR is clearly more efficient than BILUF as a preconditioner. Another advantage of LUSSOR is the lower storage requirements for the preconditioner, as has been

discussed in section 6.1 of this work. It must be noted that LUSSOR generally requires more sub-iterations per global iteration than the BILUF preconditioner, but the overall greater efficiency of the LUSSOR preconditioner demands that it be considered as a viable and better alternative to the currently popular BILUF preconditioning.

One of the concurrent goals of this research was to develop a solver which overcomes some of the deficiencies (e.g., Courant number limitations, optimal parameter selection) of conventional solvers. The enhanced stability provided by preconditioned GMRES as compared to the LGSR and AF solvers is a significant contribution towards overcoming the Courant number restrictions of the LGSR and AF solvers. Preconditioned GMRES permits the use of larger Courant numbers (of order 100) than those allowed by the LGSR solver (of order 10). The stability of preconditioned GMRES at large Courant numbers helps in accelerating convergence to the steady state and is consistent with the underlying idea of using implicit schemes. Recall, that the major advantage of implicit schemes over explicit schemes is the (potential) ability to achieve rapid convergence by using large Courant numbers as the steady state is approached. The LGSR and AF solvers require limitations on the maximum Courant numbers for all three test cases in this research, which negates this important advantage. It must be remarked that unpreconditioned GMRES also suffers from such Courant number restrictions. It is only the introduction of preconditioning that enables the resulting preconditioned GMRES solver to perform well at higher Courant numbers.

One of the major issues raised in context with methods which require sub-iterations to solve linear systems is the stopping criteria used to terminate the sub-iterations. Since the number of sub-iterations determines both the computa-

tional complexity and the storage requirements for Conjugate Gradient like methods, the stopping criteria becomes an important parameter governing the success of the preconditioned GMRES solvers. The development of a uniformly applicable stopping criteria for a variety of flow phenomena represents an important step in using preconditioned GMRES as a general purpose solver.

A method for selection of a stopping criteria has been developed in this research (see section 6.1). The criteria is based on the Courant number of the global iteration, and can be easily automated for different problems. The criteria is based on empirical evidence, but works excellently for all the test cases in this research. Thus, even though parameter selection is required for the preconditioned GMRES solvers, the parameter is easily determined from equation 6.1. This is in contrast to difficulties encountered in selecting the optimal parameters for conventional solvers, i.e., the relaxation parameter for LGSR, and the optimal time step for AF.

One major goal of this research was to compare the relative performance of the new preconditioned GMRES solvers and the conventional solvers. The LGSR solver is chosen to represent conventional solvers, for detailed performance comparisons. The superiority of the preconditioned GMRES solvers is clearly demonstrated by the comparisons made for the three diverse test cases. Remarkable improvements in convergence rate are afforded by the new solver. When the 'number of iterations to convergence' is used as a comparison criterion, the preconditioned GMRES solvers are 30, 20, and 7 times faster than the LGSR solver, for the three respective test cases (incompressible, transonic and hypersonic). The reductions in overall computational or CPU time are equally dramatic, and clearly indicate that significant CPU time savings can be realized by using the preconditioned GMRES solver. For 'CPU time to convergence' as a comparison criterion, GMRES with LUSSOR

preconditioning is the most efficient solver as it is 20, 18 and 4 times faster (on a Cray-YMP/8) for the three test cases, respectively, than the LGSR solver. The detailed comparisons presented in Chapter 6 provide sufficient evidence that preconditioned GMRES is extremely desirable as a solver over conventional solvers (like Line Gauss-Seidel Relaxation and Approximate Factorization) used currently in CFD applications.

7.2 *Recommendations for Future Work*

It is natural to ask whether it is possible to improve the performance of the LGSR solver. The recent work of Taylor [70] does exhibit some evidence that the use of relaxation with ‘inner iterations’ (to solve each linear system more accurately at each time step) can improve the performance of the LGSR solver in some cases. However, reference [70] states that ‘inner iterations’ work better for supersonic flow than for subsonic flow, and may thus have limited applicability. The performance could be improved by use of concepts like multigrid, mesh sequencing or similar preconditioning techniques. All these possibilities are beyond the scope of the present work, but could serve as excellent sources for future work.

A parallel issue concerns improvement of the performance of the preconditioned GMRES solver. Any improvements in the design of preconditioners will be reflected in improved performance. It has been stressed by Young [71] that even though a particular linear system may not permit convergence acceleration (because of a particular eigenvalue distribution), such a system can be solved efficiently by good preconditioning. Preconditioning can thus be a useful tool where conventional acceleration techniques (like multigrid, under/over relaxation, polynomial acceleration)

may fail. In the context of this research, the hypersonic flow test case represents a situation where conventional acceleration with a basic iterative method may not work. Hence, efforts need to continue to identify effective, stable preconditioners, particularly for use in high-speed flows.

This research has concentrated on implementations of the preconditioners and solvers on vector machines. The recent progress in parallel processing has necessitated the identification of solvers for parallel architectures. It has been suggested by Meurant [57] and Saad [72] that Conjugate Gradient like algorithms are very well suited for parallelization. This is indeed encouraging because as CFD moves towards increased use of parallel computing, the ideas of the general purpose solver (based on Conjugate Gradient like concepts) developed in this research can be used to conform to the needs of parallel algorithms.

Another recent popular technique is the use of unstructured meshes to increase the flexibility of CFD codes. The nature of the implicit matrix (large, sparse with no fixed sparsity pattern) obtained from an unstructured grid formulation precludes the possibility of using conventional implicit solvers (which require a structured, well defined, sparse matrix). Consequently, most unstructured grid methods currently employ an explicit, pointwise solver. The development of implicit algorithms for unstructured grids is thus an important area of future research. Conjugate Gradient like methods can be used to develop such algorithms as they employ a matrix based approach — the method can be employed independently of the sparsity pattern of the implicit matrix. Exploratory work towards developing this idea has recently been reported by Venkatakrishnan [73].

Preconditioned GMRES can be successfully employed in 3-D applications with-

out major modifications of the 2-D solver. The key, again, is that all operations with Conjugate Gradient like methods involve the implicit matrix as a whole, and not specific parts from within the matrix. Thus, there is no ‘choice of splitting the implicit matrix’ which can affect the stability and convergence of the solver in 3-D. The payoffs of using Conjugate Gradient like algorithms in 3-D can be tremendous in terms of reductions in CPU time — even a modest 50% reduction can be significant since 3-D calculations usually require several hours of CPU time (as compared to several minutes for 2-D solutions). The results of this work indicate that such a 50% reduction (or factor of two speed-up) may be achievable for 3-D applications (as evidenced by speed-up factors ranging from four to twenty for the 2-D results presented). Further work towards the development of 3-D solvers is needed to verify these claims.

The continuing development of new algorithms for use as solvers in CFD will involve the comparison of new and existing solvers. The basis for performing such comparisons is not clearly available in the current CFD literature. A need exists for the establishment of certain performance indices for the evaluation of future solvers. The lack of such indices prompted the consideration of criteria like number of iterations to convergence, overall CPU time and total operation count, for comparing the solvers and preconditioners in this research. The relative merits and demerits of these criteria are discussed in section 5.2. The development of standard test cases and performance indices can greatly help in conducting future research efforts, and provide a uniform baseline for evaluation of CFD codes. This is clearly an area of research which will yield long term benefits to CFD researchers engaged in algorithm development.

Another area of future work is the study of the performance of the solvers used in

this research for turbulent flows. It would be interesting to investigate the changes, if any, in the relative efficiencies of the solvers and preconditioners studied in this work, when turbulence modeling is incorporated in the overall solution algorithm. The preconditioned GMRES and LGSR solver require no modification to be used with turbulent flow test cases. In principle, the results obtained in turbulent flow cases should follow the trends for the present laminar flow test cases.

As mentioned in the literature survey of Chapter 2, Conjugate Gradient like algorithms are gradually finding use in areas other than CFD. Hence, the methods employed in this research can be extended to any application requiring the solution of large, sparse, linear systems of equations. One particular area where the concepts of preconditioning and Conjugate Gradient minimization can be used to assist CFD is the area of design optimization. Ill-conditioned systems are often encountered in calculations of design sensitivities [74], and necessitate the use of preconditioning. The use of preconditioned GMRES as a solver thus provides a potentially wider scope of application than that afforded by conventional solvers.

In summary, it can be said that the efforts of this research have contributed to the development of a general purpose solver for use in CFD codes. Stable, effective and efficient preconditioning is determined to be critical to the success of the preconditioned GMRES solver. The new solver offers a superior convergence rate, improved stability properties, and reduced computational times, for the solution of the Navier-Stokes equations over a wide range of complex flow problems. Preconditioned Conjugate Gradient like methods thus emerge as a viable alternative to popular current solvers, and reveal considerable promise for use as solvers in future CFD codes on supercomputers.

Appendix A. Comparisons Between AF and GMRES

This appendix compares the performance of the Approximate Factorization (AF) scheme of Beam and Warming [22] and the preconditioned GMRES solvers developed in this research. This comparison is made for the sake of completeness, and complements the comparisons between Line Gauss-Seidel Relaxation (LGSR) and preconditioned GMRES, made in the main body of this work. The major purpose of this appendix is to demonstrate that comparisons with AF (instead of with LGSR) do not affect the major conclusions of this work.

All the results for the AF solver are obtained by employing a Courant number (λ) of the same order as was used earlier with the LGSR scheme. As discussed in section 4.4 of chapter 4, the AF scheme has a stability restriction on the maximum λ , and the optimal λ is of order 10. The results presented for the AF solver in this appendix are consistent with this optimal value of λ . It must be mentioned that considerable slowdown in convergence rates was observed when AF was used with either lower or higher λ (than the optimal λ).

An optimal value of $\lambda=10$ is used for the Backward Facing Step test case. The ‘number of iterations’ comparison is shown in figure 21. The AF solver reaches a six order residual reduction (i.e., the correct physical solution) in 4500 iterations, as

compared to 1000 iterations for the two preconditioned GMRES solvers (GMLUS and GMILU). AF is estimated to converge to ten orders in 27,000 iterations, as compared to 1825 and 2095 iterations for GMLUS and GMILU, respectively. The preconditioned GMRES solvers are thus 13 times faster than the AF solver, in terms of number of iterations to convergence. The CPU time comparison is shown in figure 22. The AF solver is estimated to converge in 5500 seconds, as compared to 723 and 3699 seconds for GMLUS and GMILU, respectively. It may be noted, that GMLUS is the most efficient solver in terms of CPU time. GMLUS and GMILU are 8 and 1.5 times faster, respectively, than the AF solver, for this test case.

The optimal value of λ for the Transonic Cascade test case equals 5. Figure 23 shows that the AF scheme converges in 13,000 iterations for this case. The convergence rate of the AF solver deteriorates after 2000 iterations, revealing the same ‘smoothing property’ as was seen with the LGSR solver. A speed-up factor of 10 is seen in favor of the two preconditioned GMRES solvers. The CPU times for the three solvers are compared in figure 24. AF is estimated to converge in 10,600 seconds, as compared to the 1300 and 6472 seconds required by GMLUS and GMILU. Thus, GMLUS and GMILU are 8 and 1.5 times faster, respectively, than the AF solver, in terms of overall CPU time.

Recall, that a maximum value of $\lambda = 10$ was used with the LGSR solver for the Hypersonic Shock case. However, AF was unstable at $\lambda = 10$, and a value of $\lambda=5$ had to be used for this test case. The convergence rate comparison is shown in figure 25. AF converges in 9000 iterations, compared with 3144 and 1870 iterations for GMLUS and GMILU, respectively. The CPU time comparison of figure 26 shows that AF is faster than GMILU, but slower than the GMLUS solver. AF requires 6300 seconds of CPU time, whereas GMLUS and GMILU require 2655 and 8518

Figure 21. 'Number of Iterations' with AF for Backward Facing Step

Figure 22. 'CPU Time' Comparison with AF for Backward Facing Step

Figure 23. 'Number of Iterations' with AF for Transonic Cascade

Figure 24. 'CPU Time' Comparison with AF for Transonic Cascade

seconds, respectively. GMLUS is thus 2.5 faster than AF, whereas AF is 1.35 times faster than GMILU in terms of CPU time, for this test case.

Discussion of Results

It was mentioned in section 4.4 of chapter 4 that the operation count for the AF solver is roughly twice as much as that of the LGSR solver. However, the AF solver is more ‘vectorizable’ than the LGSR solver, and this is the primary reason for the dramatically lower CPU time per iteration for the AF solver. Specifically, the AF solver requires 805 operations per iteration per grid point (compared with 410 for LGSR), but requires 10% lesser CPU time per iteration per grid point than LGSR for the vector implementations used in this research! The above fact needs to be kept in perspective while examining the relative performance of the LGSR and AF solvers.

Thomas et. al [27] showed that one iteration of AF is equivalent to two iterations of LGSR. This implies that in order to make a fair ‘number of iterations’ comparison between AF and LGSR, the number of iterations for AF need to be doubled. When this is done, the ‘number of iterations’ to convergence for AF and LGSR becomes remarkably similar for all the test cases. Hence, the convergence rate comparisons between LGSR and preconditioned GMRES in the main body of this work are also representative of comparisons between AF and preconditioned GMRES.

The CPU times for AF are better than those for the LGSR solver for all the three test cases. The role of vectorization in the lower CPU times for AF has already been pointed out. Even though AF is more efficient (CPU time wise) than LGSR, GMLUS emerges as the most efficient solver in terms of CPU time, for all the three

Figure 25. 'Number of Iterations' with AF for Hypersonic Shock

Figure 26. 'CPU Time' Comparison with AF for Hypersonic Shock

test cases.

In conclusion, it may be remarked that the comparisons with AF are instructive, and reveal some possible merits of using the AF solver over the LGSR solver on vector architectures. However, the comparisons in this appendix provide no reason to question the overall superiority of the preconditioned GMRES solvers developed in this research.

Appendix B. Some Standard CFD Techniques

This appendix contains details which support the theory presented in Chapter 3. Several standard CFD techniques, including details of the finite-volume formulation, discretization of inviscid and viscous fluxes, and some issues relating to the structure of the implicit matrix, are included in this appendix.

B.1 The Finite Volume Formulation

In this research, the set of equations formed by equations 3.7 to 3.10 are discretized in computational (ξ, η) space by using the cell-centered finite-volume method. There are several advantages to using the finite-volume method. As the equations are originally cast in their integral, strong conservation law form, they are valid across shocks and other discontinuities in the flowfield. The evaluation of the metric terms is also simplified — the metric terms and the jacobian for each cell can be easily defined in terms of the x and y coordinate locations of the points defining the cell. The method is also capable of maintaining a given freestream flow on arbitrary meshes, i.e., the difference equations at each cell are satisfied exactly when evaluated at uniform freestream conditions.

Recall, that figure 1 shows a generalized computational space surrounding the

jk^{th} cell of an arbitrary mesh. In the discussion that follows, subscripts j and k shall refer to cell-centered quantities indexed in the ξ and η directions, respectively. The subscripts $j \pm \frac{1}{2}$ and $k \pm \frac{1}{2}$ shall refer to quantities at cell faces formed by lines of constant η and cell faces formed by lines of constant ξ , respectively. For notational convenience, the appropriate j and k subscripts shall be omitted when cell-face quantities are referenced.

In this research, the transformation from cartesian to generalized coordinates has been performed in a manner that produces a uniform mesh in computational space. This implies that

$$\begin{aligned}\xi_{j+1/2} - \xi_{j-1/2} &= \Delta\xi = 1 \\ \eta_{k+1/2} - \eta_{k-1/2} &= \Delta\eta = 1\end{aligned}\tag{B.1}$$

As will be shown later, this considerably simplifies the discretization of all differenced terms.

The metrics of the transformation are defined as

$$\xi_x = +Jy_\eta \quad \xi_y = -Jx_\eta \quad \eta_x = -Jy_\xi \quad \eta_y = +Jx_\xi \tag{B.2}$$

As mentioned earlier, the finite-volume formulation permits geometric evaluation of the metric terms, which is facilitated by the following observations:

$$\begin{aligned}+y_{\eta_{j+\frac{1}{2}}} &= +\Delta y_{j+\frac{1}{2}} = y_2 - y_1 & -x_{\eta_{j+\frac{1}{2}}} &= -\Delta x_{j+\frac{1}{2}} = x_1 - x_2 \\ +y_{\eta_{j-\frac{1}{2}}} &= +\Delta y_{j-\frac{1}{2}} = y_3 - y_4 & -x_{\eta_{j-\frac{1}{2}}} &= -\Delta x_{j-\frac{1}{2}} = x_4 - x_3 \\ -y_{\xi_{k+\frac{1}{2}}} &= -\Delta y_{k+\frac{1}{2}} = y_3 - y_2 & +x_{\xi_{k+\frac{1}{2}}} &= +\Delta x_{k+\frac{1}{2}} = x_2 - x_3 \\ -y_{\xi_{k-\frac{1}{2}}} &= -\Delta y_{k-\frac{1}{2}} = y_4 - y_1 & +x_{\xi_{k-\frac{1}{2}}} &= +\Delta x_{k-\frac{1}{2}} = x_1 - x_4\end{aligned}\tag{B.3}$$

The determinant of the jacobian of the transformation, J (at each cell face), can be defined as the reciprocal of the average of the cell-volumes (areas in 2-D) of the cells

adjoining the particular face (e.g., $J_{j+1/2} = 1/2(V_{j,k} + V_{j+1,k})$) The volume (area in 2-D) of each cell, is computed as one-half of the cross-product of the diagonals defining the cell, i.e.,

$$V_{j,k} = \frac{1}{2}[(x_2 - x_4)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_4)] \quad (\text{B.4})$$

As shall be seen later, the length of each cell face (ΔS) is required for the computation of the total flux balance of each cell. After ξ_x , ξ_y , η_x , η_y and J have been computed, the lengths of the cell faces defining a computational cell can be determined from the following relationships :

$$\begin{aligned} \Delta S_{j\pm 1/2} &= \sqrt{(+\Delta x)^2 + (-\Delta y)^2} \Big|_{j\pm 1/2} \\ &= \sqrt{\left(\frac{\xi_x}{J}\right)^2 + \left(\frac{\xi_y}{J}\right)^2} \Big|_{j\pm 1/2} \equiv \frac{|\nabla \xi|}{J} \Big|_{j\pm 1/2} \\ \Delta S_{k\pm 1/2} &= \sqrt{(-\Delta x)^2 + (+\Delta y)^2} \Big|_{k\pm 1/2} \\ &= \sqrt{\left(\frac{\eta_x}{J}\right)^2 + \left(\frac{\eta_y}{J}\right)^2} \Big|_{k\pm 1/2} \equiv \frac{|\nabla \eta|}{J} \Big|_{k\pm 1/2} \end{aligned} \quad (\text{B.5})$$

B.2 Discretization of Inviscid Flux Terms

B.2.1 Concepts of Interpolation

A Monotone Upstream-centered Scheme for the Conservation Laws (MUSCL) approach has been used to evaluate all inviscid flux derivatives. For example,

$$\frac{\partial \hat{F}}{\partial \xi} = \frac{\hat{F}_{j+1/2} - \hat{F}_{j-1/2}}{\Delta \xi} = \hat{F}_{j+1/2} - \hat{F}_{j-1/2} \quad (\text{B.6})$$

where the interface flux is evaluated as

$$\hat{F}_{j+1/2} = \hat{F}(Q^-, Q^+)_{j+1/2} = \hat{F}(Q_{j-1,k}, Q_{j,k}, Q_{j+1,k}, Q_{j+2,k}) \quad (\text{B.7})$$

$Q^\pm$ denotes variables on cell interfaces determined from ‘upwind’ (to be discussed later) interpolation of variables at the appropriate cell centers. This interpolation is defined by

$$\begin{aligned} Q_{j+1/2}^- &= Q_{j,k} + \frac{\phi_\xi}{4} [(1 - \kappa_\xi) \nabla_\xi + (1 + \kappa_\xi) \Delta_\xi] Q_{j,k} \\ Q_{j+1/2}^+ &= Q_{j+1,k} - \frac{\phi_\xi}{4} [(1 + \kappa_\xi) \nabla_\xi + (1 - \kappa_\xi) \Delta_\xi] Q_{j+1,k} \end{aligned} \quad (\text{B.8})$$

In equation B.8, ∇_ξ and Δ_ξ are operators on $Q_{j,k}$ and represent standard backward ($Q_{j,k} - Q_{j-1,k}$) and forward ($Q_{j+1,k} - Q_{j,k}$) gradients in the ξ direction, respectively. ϕ_ξ and κ_ξ are parameters which control the spatial accuracy of the discretization of the inviscid terms for the residual vector. In the context of upwind schemes, some special (and popular) choices, and the associated spatial accuracies of these parameters are:

$\phi_\xi = 0$	First-order accurate
$\phi_\xi = 1, \kappa_\xi = -1$	Fully-upwind, second-order accurate
$\phi_\xi = 1, \kappa_\xi = 1/3$	Upwind-biased, third-order accurate

It must be mentioned here that ϕ_ξ and κ_ξ may be varied independently of ϕ_η and κ_η (parameters controlling interpolation for the $\hat{G}$ flux), according to the nature of the flowfield being computed and the accuracy desired. For higher-order accurate calculations, flux limiting is often needed to enforce monotonicity of the inviscid fluxes across strong gradients (e.g., shocks). This helps to reduce spurious oscillations in the numerical computations.

A higher-order accurate ($\phi_{\xi,\eta} = 1$) evaluation of the inviscid flux-balance for the j, k^{th} cell in the flowfield will require the evaluation of $\hat{F}_{j\pm 1/2}^{\pm}$ and $\hat{G}_{k\pm 1/2}^{\pm}$. This, according to equation B.7, will require interpolation involving $Q_{j-2,k}$, $Q_{j-1,k}$, $Q_{j,k}$, $Q_{j+1,k}$ and $Q_{j+2,k}$ for the $\hat{F}$ -flux, and similarly involve $Q_{j,k-2}$, $Q_{j,k-1}$, $Q_{j,k}$, $Q_{j,k+1}$ and $Q_{j,k+2}$ for the $\hat{G}$ -flux. The viscous flux-balance can also be adequately computed from the values of Q at these cell-centers. Thus, the residual vector, $R(Q)$, is in fact a function of the Q values at nine cells (note that $Q_{j,k}$ is common to the $\hat{F}$ and $\hat{G}$ fluxes). This gives rise to a nine-point stencil for the calculation of $R(Q)$ at each cell in the grid. This nine-point stencil was shown earlier in figure 2. It may be observed that this nine-point stencil reduces to a five-point stencil for a first-order ($\phi_{\xi} = 0, \phi_{\eta} = 0$) spatially accurate calculation.

B.2.2 Concepts of Upwinding and Flux-Splitting

Since the inviscid fluxes are primarily convective in nature, they are treated with an upwind scheme, in this research. Upwind schemes are naturally dissipative, and hence do not require addition of user-specified and problem-dependent terms like artificial viscosity to maintain stability. They do require more computational work per time step than central differencing, which is compensated for by their stability and versatility, as also their increased convergence rate.

In order to identify which direction is ‘upwind’ with respect to the direction of propagation of the unsteady waves in the computational domain, the concept of flux-splitting is employed. The upwinded inviscid fluxes have been evaluated in this research with the Flux Vector Splitting (FVS) technique of Van Leer [13]. The fluxes are split into two parts according to the signs (+ or –) of the eigenvalues of

the Jacobian matrices of the respective split fluxes. For example, $\hat{F}_{j+1/2}$ is split as

$$\hat{F}_{j+1/2} = \hat{F}^+(Q_{j+1/2}^-) + \hat{F}^-(Q_{j+1/2}^+) \quad (\text{B.9})$$

The concept of flux splitting is thus consistent with, and fully complements the ideas of upwinding.

Van Leer's FVS scheme provides continuous eigenvalues across the Mach number range, and is applicable to steady or unsteady flows. The scheme provides sharp resolution of shocks, but smears contact discontinuities. The split fluxes and their Jacobians are easy to compute, and the exact Jacobians are available for use in implicit linearizations.

Van Leer's scheme of splitting the fluxes is based on ensuring that the resulting flux-split Jacobian matrices, $[\frac{\partial \hat{F}^+}{\partial Q^-}]$ and $[\frac{\partial \hat{F}^-}{\partial Q^+}]$ (say, for the $\hat{F}$ fluxes) have only non-negative real and non-positive real eigenvalues, respectively. This is achieved by defining the splitting for $\hat{F}^\pm$ as

$$\hat{F}^\pm = \frac{|\nabla \xi|}{J} \left\{ \begin{array}{l} f_{mass}^\pm \\ f_{mass}^\pm \left\{ \frac{\xi_x}{|\nabla \xi|} \frac{(\pm 2a - \bar{u})}{\gamma} + u \right\} \\ f_{mass}^\pm \left\{ \frac{\xi_y}{|\nabla \xi|} \frac{(\pm 2a - \bar{u})}{\gamma} + v \right\} \\ f_{mass}^\pm \left\{ \frac{\bar{u}(\gamma - 1)(\pm 2a - \bar{u}) + 2a^2}{\gamma^2 - 1} + \frac{u^2 + v^2}{2} \right\} \end{array} \right\} \quad (\text{B.10})$$

Here, $|\nabla \xi| / J$ is simply the length of the cell interface, as defined in equation B.5. $\xi_x / |\nabla \xi|$ and $\xi_y / |\nabla \xi|$ are components of the unit outward normal at constant η faces, and can be easily computed from known values of ξ_x , ξ_y , $|\nabla \xi| / J$ and J .

The above splitting is used with the following definitions:

$$f_{mass}^\pm = \frac{\pm \rho a}{4} (M_\xi \pm 1)^2 \quad M_\xi = \frac{\bar{u}}{a} \quad \bar{u} = (u\hat{i} + v\hat{j}) \cdot \left(\frac{\nabla \xi}{|\nabla \xi|} \right) \quad (\text{B.11})$$

For $M_\xi \geq 1$, $\hat{F}^+ = \hat{F}$, $\hat{F}^- = 0$ and for $M_\xi \leq 1$, $\hat{F}^+ = 0$, $\hat{F}^- = \hat{F}$, i.e., the full flux is used if the flow is fully supersonic or fully subsonic. When $|M_\xi| < 1$, $\hat{F}^+$ and $\hat{F}^-$ are defined by the splitting of equation B.10. As a final note, the unit outward normal in equation B.11 is simply defined as

$$\left(\frac{\nabla \xi}{|\nabla \xi|} \right) = \left(\frac{\xi_x}{J} i + \frac{\xi_y}{J} j \right) / \frac{|\nabla \xi|}{J} \quad (\text{B.12})$$

The equivalent splitting for $\hat{G}^\pm$ is obtained by simply replacing all occurrences of ξ with η in equation B.10 through equation B.12 .

B.3 Discretization of Viscous Flux Terms

The nature of the viscous flux terms (diffusive) is fundamentally different from their inviscid (convective) counterparts. The viscous fluxes are thus evaluated with a central differencing approach, rather than upwinding. These flux terms are computed with second order spatially accurate central differencing of quantities from cell centers adjacent to the particular cell face, as shown below:

$$\frac{\partial \hat{F}_v}{\partial \xi} = \frac{\hat{F}_{v_{j+1/2}} - \hat{F}_{v_{j-1/2}}}{\Delta \xi} = \hat{F}_{v_{j+1/2}} - \hat{F}_{v_{j-1/2}} \quad (\text{B.13})$$

where the viscous interface flux is computed as

$$\hat{F}_{v_{j+1/2}} = \hat{F}_v(Q_{j,k}, Q_{j+1,k}) \quad \hat{F}_{v_{j-1/2}} = \hat{F}_v(Q_{j-1,k}, Q_{j,k}) \quad (\text{B.14})$$

The terms appearing in $\hat{F}_v$ (see equation 3.4 and equation 3.10) are further evaluated as

$$\begin{aligned} u_{j+1/2} &= (u_{j,k} + u_{j+1,k})/2 \\ \left. \frac{\partial u}{\partial \xi} \right|_{j+1/2} &= u_{j+1,k} - u_{j,k} \end{aligned} \quad (\text{B.15})$$

In this research, the thin-layer approximation of the complete Navier-Stokes equations has been invoked, wherever appropriate. This is done without any loss in generality of the results obtained for comparisons between different solvers. The thin-layer approximation considerably simplifies the evaluation of the viscous fluxes, with very little loss in accuracy of the solution obtained (particularly for high Reynolds number flows and/or coarse meshes). The approximation is invoked by neglecting all viscous terms which involve a gradient in the streamwise, i.e., ξ direction. This is achieved by dropping $\hat{F}_v$ entirely and modifying $\hat{G}_v$ to omit all terms which involve differentiation with respect to ξ .

B.4 Bandwidth and Structure of Implicit Matrix

This section supports the discussions in section 3.4.3 of chapter 3. The ‘global matrix’ referred to here is the matrix V^n of equation 3.16.

The bandwidth of the global matrix is controlled by the order in which the individual jacobian matrices for each grid point are assembled into the global matrix. To illustrate this point, consider a problem being solved on a 100*50 grid — with 100 and 50 points in the ξ and η directions, respectively. The minimum bandwidth for the global matrix will be achieved when the individual matrices are loaded preferentially in the ξ direction, i.e., for j -index of 1, load k -index from 1 to 50, move to $j=2$ and so on. The global matrix V^n then has a full bandwidth of $50*4*2=400$ (compared with the order of the global matrix which is 20,000). If the matrices were loaded preferentially in the η direction, the full bandwidth would be 800. It is desirable to minimize the bandwidth of the global matrix, particularly if a direct solver approach is being adopted, as is discussed in the next section.

The structure of the global matrix is thus determined by the nature of the differencing stencil and is independent of the problem being solved (the only exception being when periodic boundary conditions are used with consistent differencing). The discussion that follows assumes that the matrices have been loaded in the preferential ξ direction. The matrices corresponding to the $\xi=\text{constant}$ cells, i.e., $(j-2, k)$, $(j-1, k)$, (j, k) , $(j+1, k)$ and $(j+2, k)$, form a centrally located block penta-diagonal matrix, within the global matrix. Two other block diagonals are located a half bandwidth away each from the main diagonal (in the upper and lower triangles of the global matrix), and represent the matrices for the $(j, k+1)$ and $(j, k-1)$ cells, respectively. The two remaining diagonals are located a full bandwidth away each from the main diagonal (in the upper and lower triangles of the global matrix), and represent the matrices for the $(j, k+2)$ and $(j, k-2)$ cells, respectively.

Appendix C. Some Standard Iterative Techniques

This appendix contains details which support the theory of iterative methods presented in chapter 4. Issues related to error analysis and convergence of iterative methods are discussed, followed by details of the the Line Relaxation scheme used in this research.

C.1 Error Analysis and Convergence

The exact solution $\bar{x}$ can be shown to satisfy the general iteration

$$\bar{x} = M^{-1}b + (I - M^{-1}A)\bar{x} \quad (\text{C.1})$$

The above equation can be used to establish the relation between the error vector at the n^{th} step and the error vector at the $(n - 1)^{\text{th}}$ step as

$$\varepsilon^n \equiv x^n - \bar{x} = (I - M^{-1}A)(x^{n-1} - \bar{x}) = (I - M^{-1}A)\varepsilon^{n-1} = G\varepsilon^{n-1} \quad (\text{C.2})$$

Thus, the rate at which the error is reduced (i.e., $\varepsilon^n \rightarrow 0$) is determined in some sense by the “size” of the iteration matrix G . It can be shown that an iterative method converges for any starting vector x^0 , if the spectral radius of the iteration matrix is less than one, i.e., $\varrho(G) < 1$. This is a necessary and sufficient condition for convergence. Also, for any iteration matrix G , the spectral radius is defined as

$$\varrho(G) \equiv \max\{|\lambda| : \lambda \text{ is an eigenvalue of } G\}$$

The norm of the error at the n^{th} iteration can be written as a function of the error of the starting vector as

$$\|\varepsilon^n\| = [\varrho(G)]^n \|\varepsilon^0\| \quad (\text{C.3})$$

If $\varrho(G)$ is known, equation C.3 can be used to determine the number of iterations required to achieve a specified error reduction. For example, in order to reduce the error by m orders of magnitude (i.e., $\varepsilon^n = 10^{-m} \varepsilon^0$),

$$n = \frac{\log_{10}(10)^{-m}}{\log_{10}[\varrho(G)]} = \frac{m}{-\log_{10}[\varrho(G)]} \quad (\text{C.4})$$

where $-\log_{10}[\varrho(G)]$ is defined as the asymptotic linear rate of convergence.

A marginal decrease in the asymptotic rate of convergence can produce a major decrease in the total number of iterations required to achieve a specified reduction in the error. To illustrate this point, let us consider a three orders of magnitude error reduction, i.e., $m=3$. If $\varrho(G)=0.99$, then $n = 687$. However, if $\varrho(G)=0.9$, then $n = 65$. Hence, a 10% reduction in the spectral radius translates to a 90% reduction in the number of iterations for a three orders error reduction !

In practice, it is difficult to determine $\varrho(G)$ since information about the eigenvalues of G is not readily available. In order to use equation C.4, approximations to $\varrho(G)$ are used. It can be shown that for any consistent matrix norm, $\varrho(G) \leq \|G\|$. Hence, by evaluating $\|G\|$, an upper bound for $\varrho(G)$ is established. Another approach often used is to use information from the iterative procedure itself to approximate $\varrho(G)$ from the relation $\log_{10}[\varrho(G)] = \log_{10} \|\varepsilon^n\| - \log_{10} \|\varepsilon^{n-1}\|$.

The rate of convergence is determined by $\varrho(G)$, or by definition, by the maximum modulus eigenvalue of G . As the iterations proceed and the components of

error associated with the largest eigenvalue are annihilated, the convergence rate changes (as $\rho(G)$ changes) and is successively determined by the magnitude of the next smaller eigenvalue(s). The nature of the distribution of the eigenvalues in the spectrum can hence play an important role in the overall convergence behavior of the iterative scheme. This is further explained in the following paragraph.

Let us consider a matrix G of size or order k . Let the eigenvalues of G be scaled in the range $[0,1]$ and arranged in descending order of magnitude as $\lambda_1, \lambda_2, \dots, \lambda_k$. If all the eigenvalues of G are distinct, the rate of convergence will be successively determined by $\lambda_1, \lambda_2, \dots, \lambda_k$. Further, if $\lambda_1 \gg \lambda_2 \approx \lambda_3 \dots \approx \lambda_k$, then once λ_1 is annihilated, the iterative scheme will converge as λ_2 . However, if $\lambda_1, \lambda_2, \dots, \lambda_k$ are uniformly distributed over the range $[0,1]$, the scheme will be overall slow. If all the eigenvalues of G are not distinct (i.e., multiple eigenvalues exist), then G may have, say, j distinct eigenvalues, where $j < k$. This multiplicity of eigenvalues can be exploited to produce efficient iterative methods.

C.2 Line Gauss-Seidel Relaxation (LGSR) Schemes

Recall the nine-point stencil of figure 2. This stencil represents the cells which provide information for the evaluation of the residual vector, $R(Q)$. The same stencil also defines the nine block-diagonals ($[A]$ through $[I]$) of the global implicit matrix of equation 3.16. This equation can be rewritten as

$$V^n \Delta Q^n = -R^n \quad \Leftrightarrow \quad [A, B, C, D, E, F, G, H, I]^n \Delta Q^n = -R^n \quad (\text{C.5})$$

Several implicit solvers can be constructed by approximating the structure of the implicit matrix.

One class of implicit solvers was developed by Thomas et. al [27] for use with upwind, flux-split schemes, and called Upwind Relaxation (UR) solvers. UR solvers use the ideas of relaxation, as discussed in the previous section. They are formed by including one or more of the nine block-diagonals in the implicit formulation. The various solvers thus obtained, and the components of the original implicit matrix included in the formulation are given below:

Point Jacobi	$[B]$
Point Gauss-Seidel	$[B]$, $[A]^*$ and $[F]^*$
Line Jacobi	$[A]$ through $[E]$
Vertical Line Gauss-Seidel	$[A]$ through $[E]$, $[F]^*$ and $[H]^*$ (forward)
Vertical Line Gauss-Seidel	$[A]$ through $[E]$, $[G]^*$ and $[I]^*$ (backward)
Horizontal Line Gauss-Seidel	$[B]$, $[F]$ through $[I]$, $[A]^*$ and $[D]^*$ (upward)
Horizontal Line Gauss-Seidel	$[B]$, $[F]$ through $[I]$, $[C]^*$ and $[E]^*$ (downward)

All matrices denoted by an asterisk (*) in the above summary represent cells for which the solutions are already known, and hence can be transferred to the right-hand-side of equation C.5. For example, the Vertical Line Gauss-Seidel solver can be written as

$$[A, B, C, D, E]^n \Delta Q^n = -R^n - [F] \Delta Q_{j-1,k}^n - [I] \Delta Q_{j-2,k}^n \quad (\text{C.6})$$

for a forward sweeping strategy. $\Delta Q_{j-1,k}^n$ and $\Delta Q_{j-2,k}^n$ are already known because the solver solves simultaneously for all unknowns at a line, and proceeds along lines from $j=1$ to $j=jmax$. Note that the solution of equation C.6 requires the inversion of a block penta-diagonal matrix.

Recall, that for a first-order accurate differencing of the implicit operator (as

adopted in this research), the nine-point stencil reduces to a five-point stencil. The corresponding UR solvers can be obtained by eliminating the matrices $[D]$, $[E]$, $[H]$ and $[I]$ in the above discussion. Equation C.6 can now be rewritten as

$$[A, B, C]^n \Delta Q^n = -R^n - [F] \Delta Q_{j-1, k}^n \quad (\text{C.7})$$

The solution of equation C.7 requires inversion of a block tri-diagonal matrix (instead of a block penta-diagonal matrix). This considerably reduces the computational work per iteration for the family of LGSR solvers. The tradeoff is in the (possible) reduction in convergence rate with the first-order implicit system.

The process of inverting a block tri-diagonal matrix can be split into two major steps:

- a) Construction of a Lower-Upper (LU) decomposition by standard Gaussian elimination. This factors the tridiagonal matrix into the product of a lower triangular matrix (L) and an upper triangular matrix (U).
- b) Use of the L and U matrices obtained from step (a) to perform back-substitution and forward-substitution, respectively, to obtain ΔQ for equation C.7.

These two steps will be referred to as the LU decomposition step and the back-substitution step. The LU decomposition is computationally more complex than the back-substitution step. This fact can be used to obtain considerable savings in CPU time while using this two-step process for the solution of equation C.7. Several approaches to obtain CPU time reductions are discussed in Appendix E.

The UR solvers possess the property that for first-order differencing, the coefficient matrix is diagonally dominant for any time step. This assured diagonal

dominance guarantees convergence for first order differencing. Another excellent feature of the scheme is that the convergence rate depends solely on the characteristics of the global coefficient matrix and the particular relaxation scheme used, and is independent of how the residual vector $R(Q)$ is evaluated. This research uses the Line Gauss-Seidel Relaxation solver with a first-order differenced implicit operator for the LHS (to minimize computational work per iteration and guarantee convergence) while using a higher-order differenced evaluation of the residual (to maximize accuracy). This approach, (as also discussed in section 3.4.3) is referred to as inconsistent differencing.

For a higher-order differencing of the implicit operator, the coefficient matrix is no longer guaranteed to be diagonally dominant. Loss of diagonal dominance can cause the LGSR solver to diverge. The loss of diagonal dominance is particularly pronounced when large time steps are used. The scheme can be made convergent by using an approach called symmetric GS (involves alternate diagonal sweeps across the domain) and/or under-relaxation. It may be argued that a higher-order LHS is desirable since it introduces more implicitness into the global matrix, and will hence improve the convergence rate compared to a first order LHS. This may be true for low time-steps, wherein the resulting matrix may be marginally diagonally dominant. However, the additional work required to solve the higher-order system (inversion of block penta-diagonals instead of block tri-diagonals), and the corresponding increase in CPU time per iteration, will probably offset the gain in the number of iterations to convergence. The choice of using a higher-order LHS versus a lower-order LHS is thus unclear, and requires some experimentation with the problem being solved.

Several combinations of sweeps across the computational domain are possible

with the various LGSR schemes. A sweep along Vertical ($\xi=\text{constant}$) lines (VLGS) may be combined with a sweep along Horizontal ($\eta=\text{constant}$) lines (HLGS). For subsonic flows, alternate forward and backward VLGS sweeps across the domain are required to maintain stability, and hence ensure convergence. For purely supersonic inviscid flows, alternate sweeps are not required. In this case, the numerical domain of dependence matches the physical domain of dependence closely for consistent first or second order fully upwind differencing, and quadratic convergence (matching the performance of a direct solver) can be obtained. This research has used an alternating forward and backward VLGS sweep, followed by an alternating upward and downward HLGS sweep, to form a multi-direction sweep cycle. The same sweep cycle is used for all the three test problems in this research.

Appendix D. Conjugate Gradient Methods

Convergence acceleration of standard iterative methods requires estimation of certain parameters like maximum/minimum eigenvalues, as discussed in Chapter 4. The estimation of acceleration parameters is a difficult, and at times, computationally expensive procedure. The Conjugate Gradient (CG) method does not require the calculation of any such parameters for accelerating convergence. The CG method is based on the ideas of Steepest Descent, and is now discussed. The discussions below follow from reference 53.

D.1 Method Of Steepest Descent

Consider minimizing the functional, $\varphi(x)$

$$\varphi(x) \equiv \frac{1}{2}x^T Ax - x^T b \quad (\text{D.1})$$

where $b \in R^n$, and $A \in R^{n \times n}$. Also, A is a Symmetric, Positive Definite (SPD) matrix. For minimizing $\varphi(x)$, set $\nabla\varphi(x) = 0$, which gives

$$\frac{1}{2}Ax + \frac{1}{2}x^T A - b = 0 \quad \Rightarrow \quad Ax = b \quad (\text{D.2})$$

Hence, minimizing $\varphi(x)$ and solving $Ax = b$ are equivalent problems.

According to differential calculus, $\nabla\varphi(x)$ gives the direction of steepest ascent ; $-\nabla\varphi(x)$ will thus provide the steepest descent direction for the computed residual vector, i.e.,

$$-\nabla\varphi(x_k) = b - Ax_k \equiv r_k \quad (\text{D.3})$$

Let x_k be a linear combination of x_{k-1} and r_{k-1} . This implies that $x_k = x_{k-1} + \alpha_k r_{k-1}$, for some positive integer α_k . If $r_k \neq 0$, then there exists a positive α_k such that $\varphi(x_k) < \varphi(x_{k-1})$. In the steepest descent method, $\alpha_k = r_k^T r_k / r_k^T A r_k$. This provides the following Steepest Descent algorithm:

```

k = 0 ; x0 = 0 ; r0 = b
while rk ≠ 0 do
    k = k + 1
    αk = rk-1T rk-1 / rk-1T A xk-1
    xk = xk-1 + αk rk-1
    rk = b - A xk
enddo

```

The speed of convergence of the above algorithm may be very slow if $\kappa_2(A)$ is large, i.e., if A is poorly conditioned. The problem created by poor conditioning is that the gradient directions (i.e., r_k) that arise for poorly conditioned A are too close to another, thus slowing progress towards the minimum point. The root of the problem is that the search directions provided by steepest descent are locally the best, but globally questionable. The solution is to seek alternate search directions, i.e., to choose other minimization strategies.

D.2 General Search Directions

Consider the successive minimization of φ along a set of directions $\{p_1, p_2, \dots\}$ that do not necessarily correspond to the residual directions, $\{r_1, r_2, \dots\}$ of steepest descent. In order to minimize $\varphi(x_k)$ ($= \varphi(x_{k-1} + \alpha_k p_k)$) with respect to α_k , it is required that

$$\alpha_k = p_k^T r_{k-1} / p_k^T A p_k \quad (\text{D.4})$$

It can be shown that this choice of α_k redefines $\varphi(x_k)$ as

$$\varphi(x_k) = \varphi(x_{k-1} + \alpha_k p_k) = \varphi(x_{k-1}) - \frac{1}{2} (p_k^T r_{k-1})^2 p_k^T A p_k \quad (\text{D.5})$$

Hence, to ensure a reduction in the size of φ , p_k and r_{k-1} should not be orthogonal to each other, i.e., $p_k^T r_{k-1} \neq 0$ (otherwise $\varphi(x_k) = \varphi(x_{k-1})$). This leads to a generalized minimization strategy as under:

$k = 0$; $x_0 = 0$; $r_0 = b$

while $r_k \neq 0$ do

$k = k + 1$

 choose p_k such that $p_k^T r_{k-1} \neq 0$

$\alpha_k = p_k^T r_{k-1} / p_k^T A p_k$

$x_k = x_{k-1} + \alpha_k p_k$

$r_k = b - A x_k$

enddo

The issue is how to choose p_k so as to guarantee global minimization of $\varphi(x_k)$, while avoiding the drawbacks of steepest descent.

D.3 A-Conjugate Search Directions

The ideal approach to minimize $\varphi(x_k)$ is to choose linearly independent p_k with the property that each x_k solves

$$\min_{x \in \text{span}(p_1, \dots, p_k)} \varphi(x) \quad (\text{D.6})$$

This choice of p_k guarantees global convergence and finite termination. Hence, the aim is to seek a vector p_k such that when the one-dimensional minimization problem given below is solved,

$$\min_{\alpha_k} \varphi(x_{k-1} + \alpha_k p_k) \quad (\text{D.7})$$

it also solves the k -dimensional minimization of equation D.6.

Let $P_k = \{p_1, \dots, p_k\} \in R^{n \times k}$ be the matrix of search directions. If $x_{k-1} \in \text{range}(P_{k-1})$, then $x_k = P_{k-1}y + \alpha_k p_k$ for some vector $y \in R^{k-1}$ and $\alpha_k \in R$. For x_k in this form, it can be shown (by using the definition of $\varphi(x_k)$) that

$$\varphi(x_k) = \varphi(P_{k-1}y) + \alpha_k y^T P_{k-1}^T A p_k + \underbrace{\frac{\alpha_k^2}{2} p_k^T A p_k - \alpha_k p_k^T b}_{f(\alpha_k)} \quad (\text{D.8})$$

The minimization of $\varphi(x_k)$ in this form is greatly simplified if the condition $P_{k-1}^T A p_k = 0$ is imposed. This reduces the global minimization problem to two independent minimization problems — minimize $\varphi(P_{k-1}y)$ over y , and minimize $f(\alpha_k)$ over α_k . Hence,

$$\min \varphi(x_k) = \min_{y, \alpha_k} \varphi(P_{k-1}y + \alpha_k p_k) = \min_y \varphi(P_{k-1}y) + \min_{\alpha_k} f(\alpha_k) \quad (\text{D.9})$$

The first minimization problem is solved by setting $P_{k-1}y = x_{k-1}$ and solving for y . The second minimization problem is easily solved by setting $-\nabla(f(\alpha_k)) = 0$,

which gives

$$\begin{aligned}
\alpha_k &= p_k^T b / p_k^T A p_k \\
&= p_k^T (r_{k-1} + A x_{k-1}) / p_k^T A p_k \\
&= p_k^T r_{k-1} / p_k^T A p_k + p_k^T A x_{k-1} / p_k^T A p_k \\
&= p_k^T r_{k-1} / p_k^T A p_k
\end{aligned} \tag{D.10}$$

This expression for α_k is used in the Conjugate Gradient algorithm.

D.4 Conjugate Gradient Search Directions

As discussed above, search directions p_k are chosen such that $P_{k-1}^T A p_k = 0$ and $p_k^T r_{k-1} \neq 0$. In order to bring about a swift reduction in the size of the residuals, p_k is chosen to be the closest vector to r_{k-1} that is A-conjugate to $\{p_1, \dots, p_{k-1}\}$. This strategy defines the following Conjugate Gradient search algorithm:

```

k = 0 ; x0 = 0 ; r0 = b
while rk ≠ 0 do
  k = k + 1
  if k = 1 then
    p1 = r0
  else
    choose pk to minimize ||pk - rk-1||2 over all
    vectors pk ∈ span {A p1, ..., A pk-1}⊥
  endif
  αk = pkT rk-1 / pkT A pk
  xk = xk-1 + αk pk
  rk = b - A xk
enddo
x = xk

```

The Conjugate Gradient method is obtained from the above algorithm. It may be remarked that p_k can be expressed as a linear combination of p_{k-1} and r_{k-1} as

$$p_k = r_{k-1} + \beta_{k-1}p_{k-1} \quad \text{since } \{p_k \in \text{span}(r_{k-1}, p_{k-1})\} \quad (\text{D.11})$$

The following definitions for β_k and α_k are used:

$$\beta_k = -\frac{p_{k-1}^T A r_{k-1}}{p_{k-1}^T A p_{k-1}} \quad \alpha_k = \frac{r_{k-1}^T r_{k-1}}{p_k^T A p_k} \quad (\text{D.12})$$

This gives rise to the following algorithm for the Conjugate Gradient method:

```

k = 0 ; x0 = 0 ; r0 = b
while rk ≠ 0 do
    k = k + 1
    if k = 1 then
        p1 = r0
    else
        βk = rk-1T rk-1 / rk-2T rk-2
        pk = rk-1 + βk pk-1
    endif
    αk = rk-1T rk-1 / pkT A pk
    xk = xk-1 + αk pk
    rk = rk-1 - αk A pk
enddo
x = xk

```

D.5 Remarks on the Conjugate Gradient Method

1. In exact arithmetic, the algorithm will find the solution in no more than n steps, where n is the order of the matrix A . In finite precision, more than n steps may be required.

2. If the Conjugate Gradient Method produces iterates $\{x_k\}$ and $\kappa = \kappa_2(A)$, and $\|x\|_A \equiv \sqrt{x^T A x}$, then the convergence of the iterates is governed by

$$\|x - x_k\|_A \leq 2\|x - x_0\|_A \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \quad (\text{D.13})$$

3. If A has only $m(< n)$ distinct eigenvalues, the method will converge in m steps, as has been proved in reference [75]. Hence, preconditioning is vital to the success of the method.
4. In practice, the effective condition number may be reduced by preconditioning. This is done by choosing a preconditioning matrix M , setting $M^{-1}Ax = M^{-1}b$, and then solving $\tilde{A}x = \tilde{b}$, where $\tilde{A} = M^{-1}A$ and $\tilde{b} = M^{-1}b$.

Appendix E. Techniques to Reduce Computational Time

It was remarked in section 6.2.3 of Chapter 6 that the CPU time of a solver is influenced by several factors, including vectorization and/or parallelization of the solver. The role of vectorization in reducing the CPU time of the AF solver was described in Appendix A. Section 6.8 of Chapter 6 introduced the idea of reusing the flux jacobian matrices over a fixed number of time-steps, in order to further reduce the CPU time per iteration. This particular idea, which is a fairly popular CFD technique, is discussed in this appendix.

A major part of the computational work in implicit CFD codes involves the computation of the individual flux-jacobian matrices, and the solution of large linear systems of equations at each time-step. Linear combinations of the individual matrices are assembled to form the block-diagonal matrices ($[A]$ through $[I]$ of equation 4.3) for the global implicit matrix. As the solution progresses towards the steady-state, and the norm of the residual vector ($R(Q)$) approaches zero, the nature of the global matrix $\left[\frac{\partial R}{\partial Q}\right]$ remains fairly constant over a large number of time-steps. Hence, once the solution is ‘close’ to the steady-state solution, the flux-jacobians can be ‘frozen’ over a number of time-steps. This distributes the computational work for one time step over the number of ‘frozen’ time-steps, and thus reduces the CPU time per time-step.

Recall, that the relaxation scheme (LGSR) used in this research requires inversion of a block-tridiagonal matrix (which is a part of the global matrix) at each time-step. The inversion is done in a two-step process — an LU decomposition step and a back-substitution step. When the global matrix is ‘frozen’, it is obvious that the LU decomposition can also be ‘frozen’ for the corresponding number of time-steps. Each iteration with a ‘frozen’ or reused LU requires the back-substitution step only. This results in a significant reduction in the computational work per iteration.

The ideas of ‘frozen’ jacobians and reusing the LU thus complement each other, and can decrease the CPU time per iteration of a CFD code. The combined strategy will henceforth be called the RELU approach. There are some complexities in using the RELU approach. Care has to be taken to use the RELU strategy only after the solution is close to the steady-state, or else the iterative method may diverge. A 2–3 order reduction in the residual is generally a good point to begin use of the strategy. The number of time-steps over which the RELU strategy is used (i.e., NRELU) is problem dependent. Acceptable values for NRELU range from 1 to 100. A constant value of NRELU=10 has been used for all the test cases in this research. The RELU strategy is used at the expense of additional storage requirements for the ‘frozen’ flux jacobian matrices. This storage cost may be unacceptable in large 2-D problems and in 3-D problems. Finally, it is to be noted that the use of the RELU strategy does not affect the convergence rate of any of the solvers used in this research, i.e., the number of iterations to convergence is unaffected by the RELU strategy.

The RELU strategy has varying effects on the CPU time per iteration of the various solvers used in this research. Recall, that the LU decomposition step is

vectorizable ‘over the number of lines in the domain’, for LGSR and AF. However, the back-substitution step is recursive for LGSR, but completely vectorizable for AF. When the RELU strategy is used, it eliminates the LU decomposition step for both solvers. Hence, with the RELU strategy, LGSR runs in scalar mode and AF runs in vector mode. Thus, the LGSR solver does not show the same reduction in CPU time as the AF solver. This point is clearly illustrated in the results of the three test cases — the CPU time per time-step for LGSR and AF is reduced by a factor of 3 and 6, respectively. Thus, inspite of requiring twice as much work per time-step (see section 4.4), the AF solver is twice as fast as LGSR when the RELU strategy is used. Recall, that the CPU times per time-step without the RELU strategy are almost equal for LGSR and AF.

The two GMRES solvers — GMRES with LUSSOR preconditioning (GMLUS) and GMRES with block ILU factored preconditioning (GMILU) — are also affected differently by the RELU strategy. Recall, that the work per time-step for the two GMRES solvers can be divided into a fixed cost per time-step and a variable cost per time-step (i.e., cost for the sub-iterations). For the GMLUS solver, the fixed cost is reduced by a factor of 3, but the variable cost remains unaffected by the RELU approach. The GMILU solver enjoys a factor of 6 reduction in the fixed cost and a factor of two drop in the variable cost. Thus, GMILU benefits more from the use of the RELU strategy than the GMLUS solver. This is primarily because the cost of computing the incomplete LU (which dominates the cost of using the ILU preconditioner) is eliminated for NRELU time-steps. It may be remarked that the GMILU solver is competitive with other solvers examined in this research only when it is used in conjunction with the RELU strategy.

Since the RELU approach affects the various solvers differently, it is difficult

Figure 27. CPU time (with RELU) Comparisons for Backward Facing Step

Figure 28. CPU time (with RELU) Comparisons for Transonic Cascade

Figure 29. CPU time (with RELU) Comparisons for Hypersonic Shock

to perform a fair comparison of the CPU times of the solvers, with RELU as a framework for comparison. However, reuse of the flux jacobians is often employed in CFD codes, and such comparisons will contribute to the completeness of this work. Figures 27, 28 and 29 present comprehensive comparisons for the three test cases. The RELU strategy is ‘switched on’ after residual reductions of roughly 6, 2 and 2 orders of magnitude are achieved for the backward facing step, transonic cascade and hypersonic shock cases, respectively. The effect of the RELU strategy on the various solvers can be seen by comparing the CPU time figures in this appendix with the plots of CPU time presented earlier. It is clear that the maximum decrease in CPU time is observed for AF, followed by GMILU, LGSR and GMLUS, in decreasing order. The major conclusion from these comparisons is that the GMLUS solver is the best solver amongst all the solvers, in terms of number of iterations and CPU time (with or without the RELU strategy).

References

1. MacCormack, R. W., "The Effect of Viscosity on Hypervelocity Impact Cratering," AIAA Paper 69-0354, April 1969.
2. Anderson, D. A., Tannehill, J. C., and Pletcher, R. H., *Computational Fluid Mechanics and Heat Transfer*, Hemisphere Publishing Company, 1984, p. 75.
3. Briley, W. R., and McDonald, H., "Solution of the Multi-Dimensional Compressible Navier-Stokes equations by a Generalized Implicit Method," *Journal of Computational Physics*, Vol. 24, August 1977, pp. 372-397.
4. Pulliam, T. H., and Steger, J. L., "On Implicit Finite Difference Simulations of Three Dimensional Flow," AIAA Paper 78-0010, January 1978.
5. Brandt, A., "Multi-Level Adaptive Computations in Fluid Dynamics," *AIAA Journal*, Vol. 18, October 1990, pp. 1165-1172.
6. Jameson, A., "Solution of the Euler Equations by a Multigrid Method," *Applied Math. and Computation*, Vol. 13, 1983, pp. 327-356.
7. Godunov, S. K., "A Finite Difference Method for the Numerical Computation of Discontinuous Solutions of the Equations of Fluid Dynamics," *Mat. Sbornik*, Vol. 47, 1959, pp. 357-393.
8. Chakravarthy, S. R., and Osher, S., "Numerical Experiments with the Osher Upwind Scheme for the Euler Equations," *AIAA Journal*, Vol. 21, September 1983, pp. 1241-1248.
9. Harten, A., Lax, P. D., and Van Leer, B., "On Upstream Differencing and Godunov-Type Schemes for Hyperbolic Conservation Laws," *SIAM Review*, Vol. 25, pp. 35-61.
10. Mulder, W. A., and Van Leer, B., "Implicit Upwind Methods in Partial Differential Equations," AIAA Paper 83-1930, July 1983.
11. Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference

Schemes," *Journal of Computational Physics*, Vol. 43, 1981, pp. 357-372.

12. Steger, J. L., and Warming, R. F., "Flux Vector Splitting of the Inviscid Gas-dynamic Equations with Application to Finite Difference Methods," *Journal of Computational Physics*, Vol. 40, 1981, pp. 263-293.
13. Van Leer, B., "Flux Vector Splitting for the Euler Equations," *Lecture Notes in Physics*, Vol. 170, 1982, pp. 507-512 (also ICASE Report 82-30, September 1982).
14. Osher, S., and Chakravarthy, S. R., "Very High Order Accurate TVD Schemes," UCLA Math Report, 1984.
15. Van Leer, B., Thomas, J. L., Roe, P. L., and Newsome, R. W., "A Comparison of Numerical Flux Formulas for the Euler and Navier-Stokes equations," AIAA Paper 87-1104, June 1987.
16. Walters, R. W., and Thomas, J. L., "Advances in Upwind Relaxation Methods," in *State of the Art Surveys of Computational Mechanics*, ed. A. K. Noor, ASME Publication, 1988.
17. Peterson, V. L., "Impact of Computers on Aerodynamics Research and Development," *Proceedings of the IEEE*, Vol. 72, 1984, pp. 68-79.
18. Ortega, J. M., and Voigt, R. G., "Solution of Partial Differential Equations on Parallel and Vector Computers," *SIAM Review*, Vol. 27, 1985, pp. 149-240.
19. Douglas, J., and Gunn, J. E., "A General Formulation of Alternating Direction Methods," *Numerische Mathematik*, Vol. 6, 1964.
20. Peaceman, D. W., and Rachford, H. H., "The Numerical Solution of Parabolic and Elliptic Differential Equations," *SIAM Journal*, Vol. 3, No. 1, 1955, pp. 28-41.
21. Stone, H. L., "Iterative Solution of Implicit Approximations of Multidimensional Partial Differential Equations," *SIAM Journal of Numerical Analysis*, Vol. 5, No. 3, September 1968.
22. Beam, R. M., and Warming, R. F., "An Implicit Factored Scheme for the Compressible Navier-Stokes Equations," *AIAA Journal*, Vol. 16, No. 4, April 1978, pp. 393-402.
23. Chakravarthy, S. R., "Relaxation Methods for Unfactored Implicit Upwind Schemes," AIAA Paper 84-0165, January 1984.
24. Van Leer, B., and Mulder, W. A., "Relaxation Methods for Hyperbolic Equations," Delft University of Technology Report 84-20, 1984.

25. Thomas, J. L., and Walters, R. W., "Upwind Relaxation Algorithms for the Navier-Stokes Equations," AIAA Paper 85-1501, July 1985.
26. Walters, R. W., and Dwoyer, "An Efficient Iteration Strategy Based on Upwind Relaxation Schemes for the Euler Equations," AIAA Paper 85-1529, July 1985.
27. Thomas, J. L., Van Leer, B., and Walters, R. W., "Implicit Flux-Split Schemes of the Euler Equations," AIAA Paper 85-1680, July 1985.
28. Hestenes, M. R., and Steifel, E., "Methods of Conjugate Gradients for Solving Linear Systems," *J. Res. Nat. Bur. Stand.*, Vol. 49, 1952, pp. 409-436.
29. Reid, J. K., "On the Method of Conjugate Gradients for the Solution of Large Sparse Linear Equations," in *Large Sparse Sets of Linear Equations*, Academic Press, New York, 1971, pp. 231-254.
30. Axelsson, "A Survey of Preconditioned Iterative Methods for Linear Systems of Algebraic Equations," *BIT*, Vol. 25, 1985, pp. 166-187.
31. Concus, P., Golub, G. H., and Meurant, G., "Block Preconditioning for the Conjugate Gradient Method," *SIAM Journal of Scientific and Statistical Computing*, Vol. 6, 1985, pp. 220-231.
32. Gustaffson, I., "Modified Incomplete Cholesky Methods," in *Preconditioning Methods: Analysis and Applications* (editor D. J. Evans), pp. 265-293, Gordon and Breach, New York, 1983.
33. Meijerink, J. A., and Van der Vorst, H. A., "An Iterative Solution for Linear Systems of which the Coefficient Matrix is a Symmetric M-Matrix," *Mathematics of Computation*, Vol. 31, 1977, pp. 148-162.
34. Yoon, S., and Jameson, A., "An LU-SSOR Scheme for the Euler and Navier-Stokes Equations," AIAA Paper 87-0600, January 1987.
35. Vinsome, P. K. W., "Orthomin, An Iterative Method for Solving Sparse Sets of Simultaneous Linear Equations," SPE Paper 5729, 1976.
36. Concus, P., and Golub, G. H., "A Generalized Conjugate Gradient Method for Nonsymmetric Systems of Linear Equations," Report STAN-CS-76-535, Computer Science Dept., Stanford Univ., 1976.
37. Widlund, O., "A Lanczos Method for a Class of Nonsymmetric Systems of Linear Equations," *SIAM Journal of Numerical Analysis*, Vol. 15, 1978, pp. 801-812.
38. Young, D., and Jea, K. C., "On the Simplification of Generalized Conjugate Gradient Methods for Nonsymmetrizable Linear Systems," *Linear Algebra and its Applications*, Vol. 52, 1983, pp. 399-471.

39. Axelsson, O., "Conjugate Gradient Type Methods for Unsymmetric and Inconsistent Systems of Linear Equations," *Linear Algebra and its Applications*, Vol. 29, 1980, pp. 1-16.
40. Saad, Y., and Schultz, M. H., "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM Journal of Scientific and Statistical Computing*, Vol. 7, 1986, pp. 856-869.
41. Smith, C. F., Peterson, A. F., and Mittra, R., "The Bi-Conjugate Gradient Method for Electromagnetic Scattering," *IEEE Transactions on Antennas and Propagation*, Vol. 38, June 1990, pp. 938-940.
42. Navon, I. M., Phua, P. K. H., and Ramamurthy, M., "Vectorization of Conjugate Gradient Methods for Large Scale Minimization," in *Supercomputing '88: Proc. of the 1st Meeting*, IEEE Computer Society Press, 1988, pp. 410-418.
43. Storey, C., "A New Look at Conjugate Direction Methods of Optimization", in *Proc. of the 2nd North America/German Workshop on Chemical Engg. Math. and Computation*, July 1990, pp. 15-20.
44. Baehre, M. D., "Neural Networks Applied to Signal Processing," M. S. Thesis, Dept. of Electrical and Computer Engg., Naval Postgraduate School, Monterey, CA.
45. Fijany, A., and Scheid, R. E., "Efficient Conjugate Gradient Algorithms for Computation of the Manipulator Forward Dynamics," in *Proc. of the NASA Conference on Space Telerobotics*, Vol. 4, 1989, pp. 329-340.
46. Habashi, W. G., Robichaud, M., Nguyen, V. N., Ghaly, W. S., and Fortin, M., "Large Scale Computational Fluid Dynamics by the Finite Element Method," AIAA Paper 91-0120, January 1991.
47. Wong, Y. S., and Hafez, M. M., "A Minimal Residual Method for Transonic Potential Flow," ICASE Report 81-15, June 1982.
48. Wong, Y. S., "Newton-like Minimal Residual Methods Applied to Transonic Flow Calculations," NASA CR-172305, February 1984.
49. Wigton, L. B., Yu, N. J., and Young, D. P., "GMRES Acceleration of Computational Fluid Dynamics Codes," AIAA Paper 85-1494, January 1985 (also in *AIAA Journal*, Vol. 29, No. 7, July 1991, pp. 1092-1100).
50. Venkatakrishnan, V., "Preconditioned Conjugate Gradient Methods for the Compressible Navier-Stokes Equations," AIAA Paper 90-0586, January 1990.
51. Manteuffel, T. A., "Adaptive Procedures for Estimating Parameters for the Nonsymmetric Chebychev Iteration," *Numerische Mathematik*, Vol. 31, 1978,

pp. 183–208.

52. Arnoldi, W. E., "The Principle of Minimized Iteration in the Solution of the Matrix Eigenvalue Problem," *Quart. Appl. Math.*, Vol. 9, 1951, pp. 17–29.
53. Golub, G. H., and Van Loan, C. F., *Matrix Computations*, 2nd Ed., The John Hopkins University Press, 1989.
54. Kershaw, D. S., "The Incomplete Choleski Conjugate Gradient Method for the Iterative Solution of systems of linear equations," *Journal of Computational Physics*, Vol. 26, 1978, pp. 43–65.
55. Meijerink, J. A., and Van der Vorst, H. A., "Guidelines for Usage of Incomplete Decompositions in Solving Sets of Linear Equations as they occur in Practical Problems," *Journal of Computational Physics*, Vol. 44, 1981, pp. 134–155.
56. Madsen, N. K., Rodrigue, G. H., and Karush, J. I., "Matrix Multiplication by Diagonals on a Vector/Parallel Processor," *Info. Proc. Letters*, Vol. 5, 1976, pp. 41–55.
57. Meurant, G., "Practical Use of the Conjugate Gradient Method on Parallel Supercomputers," *Computer Physics Communications*, Vol. 53, 1989, pp. 467–477.
58. Armaly, B. F., Durst, F., Pereira, J. C. F., and Schonung, B., "Experimental and Theoretical Investigation of Backward Facing Step Flow," *Journal of Fluid Mechanics*, Vol. 127, 1983, pp. 473–496.
59. Dietrichs, H. J., Hourmouziadis, J., Malzacher, F., and Braunling, W., "Flow Phenomena in Transonic Turbine Cascades — Detailed Experimental and Numerical Investigation," International Symposium on Airbreathing Engines, IS-ABE 87-7031, 1987.
60. Sieverding, C., Decuyper, M., Colpin, J., and Amana, O., "Model Tests for the Detailed Investigation of the Trailing Edge Flow in Convergent Transonic Turbine Cascades," ASME Paper 76-GT-30, 1976.
61. Wieting, A. R., "Experimental Study of Shock Wave Interference Heating on a Cylindrical Leading Edge," NASA TM-100484, May 1987.
62. Tannehill, J. C., Holst, T. L., and Rakich, J. V., "Numerical Computation of Two-Dimensional Viscous Blunt Body Flows with an Impinging Shock," *AIAA Journal*, Vol. 14, No. 2, February 1976, pp. 204–211.
63. Ng, W. F., Mitchell, C. R., Ajmani, K., Taylor, A. C., and Brock, J. S., "Viscous Analysis of High Speed Flows Using an Upwind Finite Volume Technique," AIAA Paper 89-0001, January 1989.

64. Giles, M. B., "Non-reflecting Boundary Conditions for Euler Equation Calculations," AIAA Paper 89-1942, June 1989.
65. Sorensen, R. L., "A Computer Program to Generate Two-Dimensional Grids about Airfoils and other Shapes by the use of Poisson's Equation," NASA TM-81198, 1980.
66. Edney, B. E., "Anomalous Heat Transfer and Pressure Distributions on Blunt Bodies at Hypersonic Speeds in the Presence of an Impinging Shock," FFA Report 115, Aeronautical Research Institute of Sweden, 1968.
67. Kloptex, G. M., Yee, M. C., "Viscous Hypersonic Shock-on-Shock Interaction on Blunt Cowl Lips," AIAA Paper 88-0233, January 1989.
68. Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W., *LINPACK Users Guide*, SIAM Publications, Philadelphia, PA, 1978.
69. Saad, Y., "Variations on Arnoldi's Method for Computing Eigenvalues of Large Unsymmetric Matrices," *Linear Algebra and its Applications*, Vol. 34, 1980, pp. 269-295.
70. Taylor, A. C., Ng, W. F., and Walters, R. W., "An Improved Upwind Finite Volume Relaxation Method for High Speed Viscous Flows," AIAA Paper 89-0549, January 1989.
71. Young, D. M., "A Historical Overview of Iterative Methods," *Computer Physics Communications*, Vol. 53, 1989, pp. 1-17.
72. Saad, Y., "Krylov Subspace Methods on Supercomputers," *SIAM Journal of Scientific and Statistical Computing*, Vol. 10, No. 6, November 1989, pp. 1200-1232.
73. Venkatakrishnan, V., "Implicit Solvers for Unstructured Meshes," AIAA Paper 91-1537, June 1991.
74. Taylor, A. C., Hou, G. W., and Korivi, V. M., "An Efficient Method for Estimating Neighboring Steady-State Numerical Solutions to the Euler Equations," AIAA Paper 91-1680, June 1991.
75. Chandra, R., "Conjugate Gradient Methods for Partial Differential Equations," Ph.D. Dissertation, Yale University, 1978.

Vita

The author was born on January 16, 1965, in Pune, India. He obtained his undergraduate degree in Mechanical Engineering from Delhi College of Engineering, Delhi, India in June 1986. He then pursued his graduate education at Virginia Polytechnic Institute and State University, Blacksburg, Virginia. He graduated with an M. S. degree in Mechanical Engineering in November, 1987, and entered the Ph.D. program in Mechanical Engineering in March 1988. He moved to Cleveland, Ohio, in February 1991, to work at NASA Lewis Research Center towards the completion of his Ph.D. research. The author plans to continue working at NASA Lewis Research Center after his graduation, and has accepted a post-doctoral position with the Institute of Computational Mechanics of NASA Lewis.