A Compressible Navier-Stokes Flow Solver Using the Newton-Krylov Method on Unstructured Grids

by

Peterson Wong

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Aerospace Science and Engineering
University of Toronto

© Copyright by Peterson Wong 2006

Abstract

A Compressible Navier-Stokes Flow Solver Using the Newton-Krylov Method on Unstructured Grids

Peterson Wong

Doctor of Philosophy

Graduate Department of Aerospace Science and Engineering

University of Toronto

2006

A Newton-Krylov algorithm is presented for the compressible Navier-Stokes equations on hybrid unstructured grids. The Spalart-Allmaras turbulence model is used for turbulent flows. The spatial discretization is based on a finite-volume matrix dissipation scheme. A preconditioned matrix-free generalized minimal residual method is used to solve the linear system that arises in the Newton iterations. The incomplete lower-upper factorization based on an approximate Jacobian is used as the preconditioner after applying the reverse Cuthill-McKee reordering. Various aspects of the Newton-Krylov algorithm are studied to improve efficiency and reliability. The inexact Newton method is studied to avoid over-solving of the linear system to reduce computational cost. The ILU(1) approach is selected in three dimensions, based on a comparison among various preconditioners. Approximate viscous formulations involving only the nearest neighboring terms are studied to reduce the cost of preconditioning. The resulting preconditioners are found to be effective and provide Newton-type convergence. Scaling of the linear system is studied to improve convergence of the inexact matrix-free approach. Numerical studies are performed for two-dimensional cases as well as flows over the ONERA M6 wing and the DLR-F6 wing-body configuration. A ten-order-of-magnitude residual reduction can be obtained with a computing cost equivalent to 4,000 residual function evaluations for two-dimensional cases, while the same convergence can be obtained in 5,500 and 8,000 function evaluations for the wing and wing-body configuration, respectively, on grids with a half million nodes.

Acknowledgements

These few years of study have been a wonderful learning experience with many exciting and enjoyable moments. I would like to sincerely thank each and everyone who have contributed to this great experience.

I would like to gratefully thank my supervisor Prof. D. W. Zingg in a special way for his excellent guidance and teaching. His generosity and patience to his students is very much appreciated. His enthusiasm and attitude toward research have encouraged and motivated me in a very positive way. I would like to thank him for many inspiring and insightful discussions, which have provided a strong foundation to this thesis.

I would like to deeply thank Prof. J. V. Lassaline for his long-term assistance with the Hurricane project. The many discussions and arguments have always been enjoyable and are much appreciated. I would like to thank my thesis committee Prof. C. P. T. Groth and Prof. Ö. L. Gülder for their advice and recommendations.

Thanks to Luis Manzano for his kind assistance with the Hurricane project during the start of the research, which is very helpful. Thanks to Dr. M. Nemec and Todd Chisholm for many discussions which have provided me a good understanding of the background theory. A big thanks to everyone in the CFD group, more notably John Gatsis and Jai Sachdev, for their valuable friendship which makes school life much more enjoyable.

To my family, I would like to express my gratitude for their ongoing support and encouragement.

Finally, I am very grateful to those who supported me financially during my study: the Government of Ontario (OGS), the University of Toronto, my supervisor and my parents.

Peterson Wong

University of Toronto
July 28, 2006

Contents

List of Tables

List of Figures

List of Symbols

Alphanumeric

a	speed of sound
b	right-hand side of the linear system
c	reference chord length
c_p	specific heat at constant pressure
d	distance to the closest wall
d_t	distance to the closest trip
e_1	first column of an identity matrix
f	time step parameter
k	thermal conductivity
k_t	turbulent thermal conductivity
lev	level of fill
n	magnitude of an area-weighted normal
p	pressure; incomplete factorization fill parameter
t	time
Δt	time step
u, v, w	Cartesian components of the velocity vector
x, y, z	Cartesian coordinates
x	solution of the linear system
y^+	non-dimensional normal distance

A	inviscid flux Jacobian
B	block size
C_L, C_D	lift and drag coefficients
C_p	pressure coefficient
D	dissipation operator
E	specific total energy
H	specific total enthalpy
$\bar{H}$	Hessenberg matrix
L	Laplacian operator

M	Mach number
N	number of nodes in mesh
P	incomplete factorization fill parameter; source term
Q	conservative flow variables
R_j	invariant quantities
S	magnitude of the vorticity
T	temperature
V	magnitude of the velocity vector
V_n, V_{tj}	normal and tangential components of the velocity vector
V_l, V_n	matrix dissipation parameters
X	eigenvectors of the inviscid Jacobian
$\mathcal{A}$	matrix of the linear system
$\widetilde{\mathcal{A}}$	matrix for preconditioning
$\mathcal{D}$	diagonal matrix
$\mathcal{K}$	Krylov subspace
$\mathcal{L}$	subspace of constraints; lower incomplete factor
$\mathcal{M}$	preconditioner
$\mathcal{P}$	permutation matrix; discrete source term
$\mathcal{P}r$	Prandtl number
$\mathcal{P}r_t$	turbulent Prandtl number
$\mathcal{Q}$	discrete flow variables
$\mathcal{R}$	discrete flow equations
$\mathcal{R}e$	Reynolds number
$\mathcal{U}$	upper incomplete factor
$\mathbf{F}$	inviscid flux tensor
$\mathbf{G}$	viscous flux tensor
$\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, $\hat{\mathbf{k}}$	unit vectors in the x, y, z directions
$\hat{\mathbf{n}}$	unit vector in the normal direction
$\vec{\mathbf{n}}$	area-weighted normal vector
$\hat{\mathbf{t}}_j$	unit vectors in the tangential directions
$\vec{\mathbf{V}}$	velocity vector
Greek	
α, β	time step parameters
γ	ratio of specific heats
ϵ	step size of the matrix-free approach
$\varepsilon^{(2)}$, $\varepsilon^{(4)}$	dissipation coefficients

η	linear system tolerance
κ	von Kármán constant
κ_2, κ_4	dissipation parameters
κ_t	turbulent kinetic energy
λ_j	characteristic speeds; modified eigenvalues of the absolute inviscid Jacobian
μ	dynamic viscosity
μ_t	dynamic eddy viscosity
ν	kinematic viscosity
ν_t	kinematic eddy viscosity
$\tilde{\nu}$	turbulence variable
ρ	density
σ	approximate Jacobian parameter; spectral radius
τ	drop tolerance
τ_{ij}	viscous stresses
τ_{ij}^R	Reynolds stresses
ω_t	magnitude of the vorticity at the trip
Λ	eigenvalues of the inviscid Jacobian
Φ	diagonal matrix of cell volumes divided by time steps
Ω_i	local cell volume
$\mathbf{\Omega}$	diagonal matrix of cell volumes

Superscripts

n	the n-th nonlinear iteration
$\sim$	dimensional
$\widehat{\ }$	scaled
$'$	reordered

Subscripts

b	boundary values after applying boundary conditions
e	extrapolated quantities at a boundary
$flow$	for the mean flow equations
i	at cell i
m	the m-th linear iteration
p	in the preconditioner
$turb$	for the turbulence model
x, y, z	components in the x, y, z directions
∞	free-stream quantities

Abbreviations

i-it	number of inner iterations
nnzB/N	number of nonzero blocks per block row
o-it	number of outer iterations
CFD	computational fluid dynamics
CPU-f.e.	CPU time in terms of equivalent function (RHS) evaluations
D1	distance-1; neglect next-to-nearest neighboring terms
D2	distance-2; include next-to-nearest neighboring terms
DLR	Deutsche forschungsanstalt für Luft and Raumfahrt
GMRES	generalized minimum residual
ILU	incomplete lower upper factorization
ILUT	incomplete lower upper factorization with threshold
ONERA	Office National d'Études et de Recherches Aérospatiales
RHS	right-hand side

Chapter 1

Introduction

1.1 Background

Aerodynamics is a branch of science that deals with the motion of air and the forces acting on bodies in motion relative to air. The subject has many applications, including the study of lift and drag over airfoils, the study of shock-wave locations in jet-engine turbines, and the prediction of aerodynamic heating of space shuttles during reentry to the earth's atmosphere.

Before the invention of computers, solution to these problems relied on analytical and experimental methods. Analytical methods, such as the thin airfoil theory of Kutta and Joukowski, and Prandtl's lifting line and boundary-layer theories, are limited to flows with simplifying assumptions and to simple geometries due to the complicated mathematics involved. Experimental methods, on the other hand, are more applicable to complex flows. However, they can be long processes requiring many resources [2]. Alternatively, numerical simulations can be performed using computational fluid dynamics (CFD) methods. These methods have a lower cost than experiments and are less limited than analytical approaches. Nowadays, both experiment and CFD are used together to solve fluid dynamics problems. In particular, CFD is used extensively for external aerodynamic flows in modern aircraft design [3, 4, 5].

One objective of CFD is to accurately predict the drag coefficient for turbulent

flows over aircraft configurations. The Navier-Stokes equations which govern the physics of viscous compressible flows are solved to compute the total drag, including skin-friction drag due to viscous effects. Turbulence is modeled using a set of time-averaged flow equations and a turbulence model, which is adequate in many engineering applications with an affordable cost. Three-dimensional solvers and grid generators are used to handle the aircraft geometry. These are expensive computations due to the use of fine grids, which is necessary to accurately resolve the flow features such as shock waves and boundary layers. A typical computation on a grid with a few million nodes requires hours to obtain a solution on multiple processors. It is thus an area of research to improve the efficiency of algorithms to reduce computational time.

Two Drag Prediction Workshops were organized by the American Institute of Aeronautics and Astronautics to assess the capability of current solvers when applied to these flows [6]. Results from the most recent workshop show the state-of-the-art technology and some current issues, including inconclusive grid refinement studies and variation among solutions obtained using different codes. Many of the issues require further research. Grid refinement, mesh adaptation, and higher-order methods are suggested to improve accuracy and to reduce uncertainty.

CFD algorithms solve the governing equations on a grid to obtain a numerical solution. Two types of grids can be used. Structured grids represent the geometry of the fluid domain by an array of vertices connected to each other in an orderly fashion. This type of grid is used in many finite-difference and finite-volume schemes. The connectivity allows the grid to be represented as a multi-dimensional array in computer memory. This simplified data structure leads to faster algorithms with less memory usage in general. However, for complex geometries such as multi-element airfoils, it is difficult to generate a grid to conform to the geometry. The multi-block approach can be used to remedy the problem, but grid generation is time consuming and difficult to automate.

Unstructured grids are constructed by tessellating the fluid domain into simplicial elements or arbitrary-shaped elements. Simplicial elements refer to triangles in two dimensions and tetrahedra in three dimensions. Since the connectivity among

grid vertices does not follow a regular pattern, unstructured grids are used with finite-volume and finite-element schemes. Unstructured grids provide flexibility for meshing about complex geometries, especially in three dimensions. Grid generation can be achieved in a more automatic manner than the multi-block approach. Moreover, unstructured meshes are well suited to mesh adaptation. The drawback of the unstructured grid approach is the higher memory usage for the explicit storage of the grid connectivity information. The use of a more complicated data structure may lead to a higher computational cost as well.

As previously mentioned, one objective of current research is the development of faster algorithms to reduce computational time. Finer grids can be used with a faster algorithm to improve accuracy. Faster algorithms also improve efficiency of numerical optimization procedures that require repeated flow solves to determine an optimal design. Current efficient algorithms for steady-state problems include approximate factorization, multigrid, and the Newton-Krylov method. The Newton-Krylov method has been studied by many researchers [7]. The method has a fast convergence rate when the solution is close to the final result. A fully converged solution is thus more obtainable, which is beneficial to optimization in avoiding objective function uncertainties caused by inexact flow solves. Since Newton's method is implicit, the convergence rate is less sensitive to high-aspect-ratio cells, which are common in viscous grids. There are potential areas for improvement in various parts of the Newton-Krylov algorithm to develop a more efficient solver [7].

1.2 Literature Review

1.2.1 Three-Dimensional Solvers

By the mid 1980's, CFD algorithms were capable of solving three-dimensional flows [19]. A summary of these solvers is given in Table 1.1. Barth [8] and Frink et al. [9] solved inviscid flows over the ONERA M6 wing. Venkatakrishnan and Mavriplis [10] solved an inviscid flow over the same wing using agglomeration multigrid with fast results. Only 20 minutes on a single processor are required to obtain

Year	Author(s)	Grid size	Processor(s)	Time hours	CPU hours
ONERA M6 wing inviscid					
1991	Barth [8]	105,177	Cray Y-MP		1.3
1991	Frink et al. [9]	231,507	Cray 2S		3.5
1995	Venkatakrishnan and Mavriplis [10]	357,900	Cray C-90		21 mins
1995	Nielsen et al. [11]	139,356	Cray Y-MP		1
ONERA M6 wing viscous					
1996	Frink [12]	578,556	Cray C-90		15.5
1998	Mavriplis [13]	1.6 million	8×Cray C-90		53
2003	Luo et al. [14]	504,790	16×SGI Origin 2000	74 mins	19.7
DLR F4 wing body					
2002	Mavriplis [15]	3.0 million	8×DEC Alpha 21264	8	64
		13 million	128×SGI Origin 2000	4	512
2002	Pirzadeh and Frink [16]	2,390,089	48×SGI Origin 2000	4.2	201.6
2003	Luo et al. [14]	1,641,451	16×SGI Origin 2000	7	112
DLR F6 wing body					
2004	Frink USM3D [6]	3,901,658	48×SGI Origin 3000	5.9	283.2
2004	Anderson et al. FUN3D [6]	3,010,307	24×Pentium 4 2.66 GHz	17	408
2004	Mavriplis NSU3D [6]	3,010,307	16×Pentium 1.7 GHz	5	80
2004	Luo et al. [17]	2,390,716	8×IBM Cluster 1600	12	96
2005	Mavriplis [18]	3,017,614	16×AMD Opteron 242	2.5	40
		72,183,630	128×SGI Altix	4.5	576

Table 1.1: A summary of three-dimensional flow solvers.

a nearly six orders of magnitude residual reduction on a grid with 357,900 nodes. Nielsen et al. [11] solved an inviscid flow over the ONERA M6 wing using the Newton-Krylov method. One hour was required to converge the residual to machine zero on a single processor on a grid with 139,356 nodes. Batina [20] solved an inviscid flow over the Boeing 747 configuration using an implicit scheme.

Three-dimensional solvers are also applied to viscous flows. Frink [21] solved a laminar flow over the ONERA M6 wing using an implicit method. The research is extended to turbulent flows [12] using the Spalart-Allmaras turbulence model. Wall functions are used to reduce the grid resolution required to resolve the viscous sublayer. Frink solved a turbulent flow over the M6 wing on a grid with 578,556 cells. The residual converges three orders of magnitude in 15 hours on a single processor. Leveling off of the residual occurs. Frink suggests this may be due to unsteady flow separation in the wing tip region. Anderson et al. [22] solved three-dimensional turbulent flows using an implicit method. Barth and Linton [1] use the Newton-Krylov method to solve three-dimensional turbulent flows, obtaining residual convergence to machine zero. The algorithm is extended to parallel computations. Mavriplis [23] solved three-dimensional turbulent flows using multigrid. Convergence is found to degrade when compared to inviscid computations. He suggests it is related to grid stretching in the viscous region. Improvements are developed later [13] using line implicit smoothing and directional coarse grids. Mavriplis [13] solved a turbulent flow over the M6 wing on a grid with 1.6 million nodes. The residual converges eight orders of magnitude in a total of 53 CPU hours using eight processors. Luo et al. [14] solved a turbulent flow over the wing on a grid with 504,790 nodes using a matrix-free implicit method. The residual converges four orders of magnitude in 74 minutes using 16 processors.

1.2.2 Drag-Prediction Workshops

The first workshop was held at Anaheim, California in 2001. The objective was to assess the capability of algorithms for practical flows with emphasis on accurate prediction of the drag coefficient. A transonic turbulent flow over the DLR-F4 wing-body

configuration was used as a standard test case. Mavriplis and Levy [15] applied multigrid to solve this wing-body case. The residual converges five orders of magnitude in eight hours using eight processors on a three-million node grid. A finer grid with 13 million nodes is solved to the same tolerance in four hours using 128 processors. Convergence degradation is observed for cases involving increased flow separation. A fully converged solution could not be obtained on the grid with 13 million nodes. Issues such as grid resolution and the effect of turbulence modeling are raised in the study. Pirzadeh and Frink [16] solved a range of flows over the wing-body configuration using an implicit scheme. The cases are solved on a grid with 2,390,089 cells which is designed to be used with wall functions. The flow solutions converge four orders of magnitude in 1.7 to 4.2 hours on 48 processors, depending on the extent of flow separation. The grid is generated using the VGRIDns grid generator, which allows stretching in the span-wise direction. This can reduce the number of cells by a factor of three without losing resolution in the normal and chord-wise directions. Luo et al. [14] solved the case on a grid with 1,641,451 nodes. The residual converges four orders of magnitude in seven hours using 16 processors. Levy et al. [24] summarized the results obtained in the first workshop from a total of 18 participants using 14 different codes. The study shows a reasonable agreement between computational and experimental results, but a large statistical variation among solutions obtained using different codes. They recommend future work to identify the sources of the variation, which may be due to the use of various grids and turbulence models. Lee-Rausch et al. [25] performed a grid convergence study to evaluate if discretization error is the source of the variation. The study on two unstructured codes shows an increase in the variation in forces and moments from solver to solver when the grid is refined. Some solutions converge to different shock locations and wing root flow separation patterns. It is inconclusive whether the increase is due to discretization error or differences in numerical formulation, such as thin-layer versus full viscous calculations.

The second workshop was held at Orlando, Florida in 2003. The objectives were to further investigate the code-to-code uncertainties, study the effect of grid resolution on drag prediction, and assess the capability of drag prediction for a wing-body-nacelle-pylon configuration. The DLR-F6 wing-body configuration with and

without the nacelle and pylon attached were used as standard test cases. Solutions from the second workshop show a smaller statistical variation when compared to the first workshop [26]. Lee-Rausch et al. [6] summarized the results from three unstructured solvers: USM3D of Frink [27], FUN3D of Anderson et al. [22] and NSU3D of Mavriplis [28]. For a wing-body case, USM3D converges drag to 10^{-5} in 5.9 hours using 48 processors on a grid with 3,901,658 cells. FUN3D obtains the same convergence in 17 hours using 24 processors on a grid with 3,010,307 nodes, while NSU3D obtains the same convergence in five hours using 16 processors on the same grid. The solutions show a reduction in code-to-code variation with grid refinement for the wing-body case, but an increase in variation for the wing-body-nacelle-pylon case. Luo et al. [17] solved the case on a grid with 2,390,716 nodes. The residual converges three orders of magnitude in 12 hours using eight processors. May et al. [29] reported the results from two structured solvers. The solution for the wing-body case has a better agreement with experimental data than that for the wing-body-nacelle-pylon case. The solution for the wing-body case is not grid converged even on grids with up to ten million nodes. Improvements in grid quality and density are suggested. Mavriplis [18] performed a grid convergence study and a sensitivity study of the force coefficients to viscous and artificial dissipation formulations. The wing-body case is solved on grids up to 72,183,630 nodes. Convergence on the 72 million node grid is obtained in 4.5 hours using 128 processors, or in 20 minutes when using 2008 processors. The solution from the 72 million node grid is quite different from that on a 65 million node non-similar grid. Grid resolution may still be insufficient on these high resolution grids.

Two trends are apparent in the summary in Table 1.1. The grid size for computation has increased over ten fold over the past decade. Furthermore, there is a shift towards parallel computation in CFD algorithms.

1.2.3 Unstructured Grid Methods

Unstructured grids provide flexibility for meshing about complex geometries and for mesh adaptation. Reviews of unstructured grid methods can be found in the

works by Venkatakrishnan [19] and Mavriplis [30].

Jameson and Mavriplis [31] extended the structured grid solver of Jameson et al. [32] to regular triangular grids. Mavriplis [33] extended the work to general triangular grids. The numerical scheme is based on a central-difference finite-volume discretization with added artificial dissipation. Dissipation is added as approximations to a Laplacian operator and a biharmonic operator, which is analogous to the scheme of Jameson et al. [32] with blended second and fourth differences. The algorithm is spatially second-order accurate and produces similar accuracy and convergence when compared to the solver of Jameson et al. [32]. The work is extended to three dimensions and to viscous flows [34, 23], and also to matrix dissipation [35].

Upwind methods produce more accurate solutions than scalar artificial dissipation schemes. The formulation is based on the local flow physics governed by the Riemann problem. Godunov-type schemes solve the Riemann problem exactly, which are computationally expensive. Approximate Riemann solvers provide more efficient upwind schemes [36, 37]. Research on unstructured upwind solvers includes the work of Barth [8], Batina [20], Ollivier-Gooch [38], and Frink et al. [9]. Analogous to upwind schemes, matrix dissipation schemes [39] are used to improve the accuracy of artificial dissipation schemes [35, 17].

Finite-volume schemes can be classified as vertex-based or cell-centered. For vertex-based schemes, the flow variables are stored at grid vertices. Various approaches are used to construct the control volumes around the vertices. Examples include the union of cells containing the vertex [33, 40], and a centroidal or a median dual of the vertex [8, 41]. The containment dual [42] is suggested for better accuracy on triangular grids. Haselbacher et al. [43] showed the median dual approach is less accurate on triangular grids than on quadrilateral grids. Luo et al. [14] reported the same issue in three dimensions. Both show improvements with the containment dual approach. For cell-centered schemes, the flow variables are stored at cell centers. The cells are used as control volumes. The ratio of cells to vertices is two on triangular grids. The ratio is approximately five to six on tetrahedral grids. Thus cell-centered schemes have more unknowns on the same grid but they may be potentially more accurate [30]. Cell-centered schemes have smaller fixed-sized stencils. For example,

the number of cell neighbors is three on triangular grids and is four on quadrilateral and tetrahedral grids. On the other hand, vertex-based schemes have larger stencils in general, and the number of vertex neighbors is a variable.

A fine grid is required to resolve the boundary layers for viscous flows. Grid refinement is necessary in the normal direction to the body but not in the tangential directions. Resolution of $y^+ \sim 1$, which can be as small as 10^{-6} relative to the chord length, is typically required for aerodynamic flows, where y^+ is a non-dimensional normal distance based on the law of the wall. Refinement of the mesh isotropically leads to a large increase in grid size which is undesirable. Alternatively, the grid can be directionally refined, leading to stretched grids with high-aspect-ratio cells. Stretched triangular grids are used by Barth [44], Anderson and Bonhaus [45], and Walsh and Zingg [40] in viscous computations. Holmes and Connell [46] developed a weighted Laplacian in the artificial dissipation that improves stability on stretched grids. Aftosmis et al. [41] show that on triangular grids, the solution becomes more inaccurate when the cell aspect ratio is increased. On quadrilateral grids, the accuracy is independent of the cell aspect ratio. Furthermore, they found the additional edges in stretched triangular meshes do not improve accuracy but cause a higher computational cost. This suggests hybrid grids, by replacing stretched triangles with quadrilaterals in the viscous region. Stretched tetrahedra can be replaced by prisms in three dimensions. The use of hybrid grids increases the complexity of numerical schemes, but with a potential improvement in accuracy and computational cost. Hybrid grids are used by Mavriplis and Venkatakrishnan [28], Haselbacher et al. [43], and Lassaline and Zingg [47].

The viscous terms in the Navier-Stokes equations involve second-order derivatives. For finite-difference methods on structured grids, first-order derivatives can be precomputed at half-node locations, leading to a compact viscous stencil. A similar approach can be used for vertex-based triangular solvers by pre-computing the gradients at triangular cells, which leads to a compact viscous stencil as well. However, for hybrid grids, the viscous stencil is no longer compact and involves next-to-nearest neighboring terms [28, 48, 49]. This increases computational cost of algorithms with potentially higher memory usage. The issue becomes more significant in

three dimensions [1]. Alternative viscous formulations are suggested. Mavriplis and Venkatakrishnan [28] replace the diffusion terms by a Laplacian operator following an incompressible flow assumption. Haselbacher et al. [43] developed an approximate viscous flux that can be applied to both hybrid and triangular grids. Haselbacher and Blazek [48] compared gradient formulations using simple averaging and directional finite differences. They show simple averaging has a higher truncation error due to decoupling, and suggest the finite-difference approach. The finite-difference approach is used by Smith et al. [49] to develop a preconditioner for the Newton-Krylov method. Coirier [50] studied gradient formulations by integration over diamond-shaped control volumes and over grid cells in addition to the simple averaging and the finite-difference approaches.

1.2.4 Implicit Methods

Newton's method is used to solve the steady-state Euler and Navier-Stokes equations by Venkatakrishnan [51, 52]. A direct method is used to solve the linear system. The Jacobian is constructed by a complete linearization of the flow equations after spatial discretization. The resulting scheme is unconditionally stable. Large time steps can be used for a Newton-type convergence rate. The time step sequence of Mulder and van Leer [53] is used. Convergence to machine zero in only a few iterations is demonstrated. However, the algorithm requires high computational cost and memory usage for matrix inversions. Application to larger problems is prohibitive.

More efficient schemes are used to alleviate the expensive cost of direct solvers. The Alternating Direction Implicit approach approximates the implicit operator by one-dimensional operators involving tri-diagonal or penta-diagonal matrices. Inversion of these matrices of small bandwidths is more efficient than that of the full matrix. Beam and Warming [54] developed an approximate factorization scheme using this approach. Pulliam and Chaussee [55] further reduce the matrix into diagonal form which can be solved more efficiently. Approximate factorization techniques have a slower convergence rate than Newton's method, due to the approximations introduced in the implicit operator. Constant time steps based on a Courant-Friedrichs-Lewy

number can be used to improve robustness. Moreover, approximate factorization techniques cannot be used with unstructured grids. Nevertheless, the stability time step requirement is less restrictive for implicit methods than explicit methods.

Besides direct methods, iterative methods can also be used to solve the linear system of implicit schemes. They require less storage and are less robust than direct methods. They are originally developed for systems with diagonally dominant matrices. The Jacobi method is used by Anderson and Bonhaus [56] and Frink [27]. Gauss-Seidel with multigrid acceleration is used by Nielsen et al. [11] and Weiss et al. [57]. Symmetric successive over-relaxation is used by Venkatakrishnan and Mavriplis [58]. Iterative methods like Jacobi, Gauss-Seidel, and successive over-relaxation are stationary iterative methods. They may have a slow convergence rate. Small time steps can be used to improve diagonal dominance of the linear system to improve the convergence rate.

Krylov subspace methods are non-stationary iterative methods where the computation involves information that changes at each iteration. Krylov methods find an approximate solution of the linear system from a Krylov subspace. Orthogonality conditions are imposed to extract the approximate solution. These methods are shown to perform better than stationery iterative methods in a number of studies, including Nielsen et al. [11], Ajmani et al. [59], and Anderson et al. [22]. Venkatakrishnan and Mavriplis [58] show faster convergence using a Krylov method than the symmetric successive over-relaxation and the incomplete lower-upper factorization (ILU) as a linear solver of an approximate Newton algorithm. Pueyo and Zingg [60] show the Newton-Krylov method converges faster than an approximate factorization algorithm accelerated by multigrid. The generalized minimum residual method (GMRES) is a popular Krylov method for non-symmetric indefinite systems such as those obtained from the Euler and the Navier-Stokes equations. For symmetric positive-definite systems, the conjugate gradient algorithm is widely used.

Other Krylov methods have also been used. Details about Krylov methods can be found in the book by Saad [61] and the review paper by Saad and van der Vorst [62], which includes historical perspectives. The conjugate gradient squared algorithm is used by Orkwis [63], and the stabilized conjugate gradient algorithm is used by

Forsyth and Jiang [64]. Geuzaine and Essers [65] studied the stabilized bi-conjugate gradient and the GMRES method. McHugh and Knoll [66] compared the conjugate gradient squared, the transpose-free quasi-minimal-residual, the stabilized bi-conjugate gradient, and the GMRES method. All four approaches are Krylov methods developed for non-symmetric indefinite systems. The first three are based on the Lanczos bi-orthogonalization procedure, where short vector recurrence relationships exist. Thus work and storage are constant at each iteration. The GMRES method is based on the Arnoldi orthogonalization procedure, where storage and work increase with the number of iterations. However, the method has the property to minimize the residual. McHugh and Knoll [66] show the GMRES method performs better than the Lanczo-based methods when a matrix-free approach is used. The GMRES method is widely used in CFD [7, 67].

Approximations are used to reduce computational cost of the Newton-Krylov method. Approximate Newton methods replace the matrix of the linear system by an approximate Jacobian, which can be constructed from a low order spatial discretization. This reduces the size of the matrix with a lower memory usage. Venkatakrishnan and Mavriplis [58] use an approximate Jacobian constructed by linearizing only the first-order artificial dissipation. Furthermore, the laminar viscosities are treated as constants in the Jacobian, and the turbulence model is treated explicitly. Newton convergence cannot be obtained because of the approximations introduced. Inexact Newton methods solve the linear system partially to some nonzero tolerance. This reduces computational cost of expensive linear solves. Dembo et al. [68] show that the order of nonlinear convergence of inexact Newton methods is related to the linear convergence tolerance. Eisenstat and Walker [69] developed a formulation of the linear tolerance which results in a fast local convergence to the final solution and avoids over-solving of the linear system. A smaller tolerance, or a more accurate linear solve, is used when the solution is closer to nonlinear convergence. They show that solving the linear system below the tolerance is unnecessary. Orkwis [63] compared standard Newton, approximate Newton, and inexact Newton approaches. The standard approach has a better convergence rate than the approximate and the inexact approaches in terms of nonlinear iterations, but the inexact approach is faster in terms

of computational time.

A matrix-free approach can be used for Krylov methods, such as GMRES, that require Jacobian-vector products but not explicit Jacobians. The Jacobian-vector product can be approximated by a finite-difference formula using Frechét derivatives. The matrix-free approach reduces memory usage because Jacobian storage is unnecessary. Usually, an approximate Jacobian with a lower storage is still required for preconditioning purposes. Moreover, Newton convergence can be obtained using the matrix-free approach because the finite-difference Jacobian is a complete linearization of the flow equations. The matrix-free approach was introduced by Brown and Saad [70] and is now widely used [7, 66, 71]. Nielsen et al. [11], Knoll et al. [67], and Chisholm and Zingg [72] studied different step sizes in the finite-difference formula. Blanco and Zingg [73] compared standard Newton, approximate Newton, and matrix-free Newton approaches. The latter is found to have the lowest computational cost. Pueyo and Zingg [60] show the matrix-free Newton-Krylov method converges faster than an approximate Newton algorithm. Comparison between matrix-free and standard Newton approaches is also studied by McHugh and Knoll [66], and Knoll et al. [67].

Convergence of Krylov methods can be improved by preconditioning. Usually the choice of the preconditioner is more important than the choice of the iterative method [74]. Preconditioning transforms the linear system to a better conditioned one so that it can be solved in less iterations. The block-diagonal preconditioner is used by Venkatakrishnan and Mavriplis [58], Anderson et al. [22], and Johan et al. [71]. This preconditioner has a low cost and requires only storage of a block-diagonal matrix. The lower-upper symmetric-Gauss-Seidel method of Yoon and Jameson [75] is used as a preconditioner by Ajmani et al. [59] and Luo et al. [17]. Luo et al. [76, 77] developed a preconditioner using this method which does not require storage of a matrix. Ajmani et al. [59] show the preconditioner is as effective as the more accurate incomplete lower-upper factorization preconditioner for an implicit scheme using a constant time step. However, this is not necessarily true in general when large time steps are used, where the latter preconditioner is expected to be more effective.

Incomplete lower-upper factorization (ILU) provides more effective precondition-

ing for Newton's method. The factorization can be based on a complete or an approximate Jacobian. The latter reduces storage and may improve effectiveness of the preconditioner. Pueyo and Zingg [60] found the approximate Jacobian preconditioner to produce faster convergence. This is also suggested by Geuzaine [78]. Smith et al. [49] use an approximate viscous Jacobian for preconditioning. ILU preconditioners have higher memory usage and computational cost than the block-diagonal and the lower-upper symmetric-Gauss-Seidel approaches. Memory usage can be controlled by limiting the non-zeros in the factors. For the ILU approach with zero fill, ILU(0), elements are dropped in the factors so that they have the same nonzero pattern as the matrix for preconditioning. This approach has a relatively low storage. It is used by Anderson et al. [22], Nielsen et al. [11], and Smith et al. [49]. Barth and Linton [1] use a parallel ILU(0) preconditioner. Venkatakrishnan and Mavriplis [58] compared ILU(0) with the block-diagonal and the symmetric successive over-relaxation preconditioners. The ILU(0) approach is found to provide the fastest convergence. The level-of-fill approach, ILU(p), can be used to improve the accuracy of the ILU(0) approach by allowing fill-in in the factors controlled by a parameter p. This leads to a higher storage and computational cost with a potential reduction in the number of iterations required to solve the linear system. This approach is used by Forsyth and Jiang [64], Blanco and Zingg [73], Pueyo and Zingg [60], and Geuzaine [78]. These researchers show the use of some fill in the factorization is beneficial to convergence. Pueyo and Zingg [60] studied an alternative ILU approach based on a threshold strategy, which is found to be more expensive than the level-of-fill approach. The threshold approach is also studied by Souläimani et al. [79]. Orkwis [63] and Pueyo and Zingg [60] studied different block-fill patterns in the ILU preconditioner.

ILU is originally developed for M-type matrices (defined in Appendix B), where the existence of the factors and a form of stability of the factorization can be proven [80]. ILU can breakdown when applied to non-symmetric indefinite matrices. Chow and Saad [80] describe several causes of ILU breakdown including small and zero pivots, inaccuracy of factorization, and instability of triangular solves. Three methods are proposed for diagnosis. This includes a condition estimate of the factors, the size of the smallest pivot, and the size of the largest element in the factors. Some sug-

gestions are provided in attempt to circumvent the problems. This includes pivoting, reordering of unknowns, scaling, perturbation of diagonal elements, and preserving a symmetric structure in the factors. Chapman et al. [74] studied a number of ILU preconditioners for indefinite systems, including scalar and block versions of the level-of-fill approach, the threshold approach, and an approach based on threshold dropping with diagonal compensation. No one preconditioner is found to be the best for all matrices. The preconditioners are found to perform similarly for a similar amount of fill-in. They suggest the use of block-diagonal preconditioning and increasing the absolute value of the diagonal elements to improve conditioning of the factors. They comment that block preconditioners can have less computational work than scalar versions, if block inversions are performed efficiently.

The ordering of the unknowns affects performance of ILU preconditioners. Blanco and Zingg [73] and Pueyo and Zingg [60] show the reverse Cuthill-McKee approach [81] provides a better convergence than some other approaches. This approach is used by Forsyth and Jiang [64], Anderson et al. [22], and Nielsen et al. [11]. Some other reordering techniques can also be found in the works of George and Liu [82] and Gibbs et al. [83]. Liu and Sherman [84] compared the original and the reverse Cuthill-McKee algorithms. Knoll et al. [67] and Pueyo and Zingg [60] studied reusing the preconditioner to alleviate the cost of preconditioning. Knoll et al. [67] compared reusing the preconditioner and reusing both the preconditioner and the Jacobian. The former is found to provide a better performance. Multigrid preconditioners are studied by Knoll and Rider [85], Knoll et al. [86], and Mavriplis [35]. The objective is to maintain the performance of preconditioning when the grid is refined. Geuzaine et al. [87] demonstrate improved convergence using a multigrid preconditioner for an inviscid flow.

Startup strategies improve robustness of Newton's method during startup. Damped Newton updates are used by McHugh and Knoll [66]. Johan et al. [71] use a back-tracking line-search algorithm to improve the solution update. Implicit schemes with small time steps are used as a startup strategy in many studies, including Anderson et al. [22], Geuzaine [78], Smith et al. [49], Johan et al. [71], and Forsyth and Jiang [64]. Small time steps improve the stability of the nonlinear iterations and

the conditioning of the linear system, leading to a robust method. Chisholm and Zingg [88] developed a time step strategy that provides a good convergence of the Spalart-Allmaras turbulence model [89] during startup. Mesh sequencing is studied by Chisholm and Zingg [88], Geuzaine [78], and Nielsen et al. [11] to provide a better initial solution at a lower cost. Alternatively, Barth and Linton [1] use a low order spatial discretization to provide an improved initial solution.

The Newton-Krylov method is applied to many CFD applications [7], including three-dimensional flows [1, 11, 22]. The method is shown to be efficient for aerodynamic flows [60, 72]. To conclude this section, two references are suggested for further details about the method. The review paper by Knoll and Keyes [7] provides an overview of the matrix-free Newton-Krylov method. The review paper by Saad and van der Vorst [62] provides a comprehensive description about iterative methods for linear systems.

1.3 Objectives

As summarized in the literature review, several researchers have successfully demonstrated effective implicit solution techniques based on unstructured meshes for turbulent aerodynamic flows. However, there still exists a need for faster, more reliable solvers, and the Newton-Krylov approach is among the most promising means to achieve this. The objective of this thesis is to develop an improved Newton-Krylov solver for turbulent flows on hybrid unstructured meshes, thereby advancing the state of the art in this important area. Closely related research includes Geuzaine [78], Barth and Linton [1], Anderson et al. [22], and Luo et al. [17]. The current study is based on the Newton-Krylov framework developed by Pueyo and Zingg [60] and Chisholm and Zingg [88], which is shown to be promising for structured-grid applications. Manzano et al. [90] extended the method to three-dimensional inviscid flows on unstructured grids. The current study extends the work of Manzano et al. [90] to turbulent flows. Improvements are achieved by investigation of various aspects of the Newton-Krylov method, including preconditioning, startup strategy, selection of solver parameters, and treatment of viscous terms in the preconditioner.

Chapter 2

Governing Equations

This chapter summarizes the governing equations of the numerical algorithm. Section 2.1 describes the Navier-Stokes equations for the analysis of viscous compressible flows. Section 2.2 describes turbulence modeling. The boundary conditions are discussed in Section 2.3.

2.1 Navier-Stokes Equations

The governing equations are the Navier-Stokes equations for viscous compressible flows. This set of equations describes the conservation of mass, momentum, and energy of the fluid flow. In non-dimensional integral form, the equations are written as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega} Q\mathrm{d}V + \int_{\partial\Omega} \mathbf{F}\cdot\hat{\mathbf{n}}\mathrm{d}S = \int_{\partial\Omega} \mathbf{G}\cdot\hat{\mathbf{n}}\mathrm{d}S \tag{2.1}$$

where Ω is an arbitrary control volume, and $\partial\Omega$ is the boundary of the control volume. Here, Q is the set of conservative flow variables:

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix} \tag{2.2}$$

with density ρ, momentum components ρu, ρv, ρw, and total energy ρE, $\mathbf{F}$ is the inviscid flux tensor, $\mathbf{G}$ is the flux tensor associated with viscosity and heat conduction, and $\hat{\mathbf{n}}$ is the unit normal pointing outward of the boundary surface. Here, we scale the dimensional variables by

$$x = \frac{\widetilde{x}}{c}, \quad y = \frac{\widetilde{y}}{c}, \quad z = \frac{\widetilde{z}}{c}, \quad t = \frac{\widetilde{t}\widetilde{a}_\infty}{c} \tag{2.3}$$

$$\rho = \frac{\widetilde{\rho}}{\widetilde{\rho}_\infty}, \quad u = \frac{\widetilde{u}}{\widetilde{a}_\infty}, \quad v = \frac{\widetilde{v}}{\widetilde{a}_\infty}, \quad w = \frac{\widetilde{w}}{\widetilde{a}_\infty}, \quad E = \frac{\widetilde{E}}{\widetilde{a}_\infty^2} \tag{2.4}$$

where ∞ refers to free-stream quantities, $\sim$ denotes dimensional quantities, c denotes a reference length, and a is the speed of sound. The choice of the reference length is arbitrary. It can be the chord length in two dimensions, or the mean aerodynamic chord in three dimensions. For ideal fluids, the speed of sound is $a = \sqrt{\gamma p/\rho}$. The ratio of specific heats, γ, is taken as 1.4 for air. Pressure, p, is related to the conservative flow variables by the equation of state for a perfect gas, as follows:

$$p = (\gamma - 1)\left(\rho E - \frac{1}{2}\rho\left(u^2 + v^2 + w^2\right)\right) \tag{2.5}$$

Referring to (2.1), the convective and viscous flux tensors are:

$$\mathbf{F} = \begin{bmatrix} \rho\vec{\mathbf{V}} \\ \rho u\vec{\mathbf{V}} + p\hat{\mathbf{i}} \\ \rho v\vec{\mathbf{V}} + p\hat{\mathbf{j}} \\ \rho w\vec{\mathbf{V}} + p\hat{\mathbf{k}} \\ \vec{\mathbf{V}}(\rho E + p) \end{bmatrix} \tag{2.6}$$

and

$$\mathbf{G} = \frac{1}{\mathcal{R}e_a}\begin{bmatrix} \vec{0} \\ \tau_{xx}\hat{\mathbf{i}} + \tau_{yx}\hat{\mathbf{j}} + \tau_{zx}\hat{\mathbf{k}} \\ \tau_{xy}\hat{\mathbf{i}} + \tau_{yy}\hat{\mathbf{j}} + \tau_{zy}\hat{\mathbf{k}} \\ \tau_{xz}\hat{\mathbf{i}} + \tau_{yz}\hat{\mathbf{j}} + \tau_{zz}\hat{\mathbf{k}} \\ \boldsymbol{\tau}\cdot\vec{\mathbf{V}} + k\nabla T \end{bmatrix} \tag{2.7}$$

respectively. Here, $\vec{\mathbf{V}} = u\hat{\mathbf{i}} + v\hat{\mathbf{j}} + w\hat{\mathbf{k}}$ is the velocity vector, and $\mathcal{R}e_a$ is a Reynolds number scaling. The term $\boldsymbol{\tau}\cdot\vec{\mathbf{V}}$ denotes the work done on the fluid due to viscosity.

It is given by:

$$\begin{aligned} \boldsymbol{\tau} \cdot \vec{\mathbf{V}} &= (u\tau_{xx} + v\tau_{xy} + w\tau_{xz})\,\hat{\mathbf{i}} \\ &+ (u\tau_{yx} + v\tau_{yy} + w\tau_{yz})\,\hat{\mathbf{j}} \\ &+ (u\tau_{zx} + v\tau_{zy} + w\tau_{zz})\,\hat{\mathbf{k}} \end{aligned} \tag{2.8}$$

Fourier's law of heat conduction is used to calculate the heat flux vector:

$$\mathbf{q} = -k\nabla T = -\frac{\mu}{\mathcal{P}r\,(\gamma - 1)}\nabla a^2 \tag{2.9}$$

where $\mathbf{q}$ is the heat flux vector, k is the thermal conductivity, $\mu = \widetilde{\mu}/\widetilde{\mu}_\infty$ is the non-dimensional dynamic viscosity, and $\mathcal{P}r$ is the Prandtl number. The dynamic viscosity is given by Sutherland's law:

$$\frac{\widetilde{\mu}}{\widetilde{\mu}_\infty} = \left(\frac{\widetilde{T}_\infty + 110}{\widetilde{T} + 110}\right)\left(\frac{\widetilde{T}}{\widetilde{T}_\infty}\right)^{3/2} \tag{2.10}$$

The Prandtl number is defined by

$$\mathcal{P}r = \frac{\widetilde{c_p}\widetilde{\mu}}{\widetilde{k}} \tag{2.11}$$

where c_p is the specific heat at constant pressure. The Prandtl number is proportional to the ratio of energy dissipation due to viscosity to the energy transported by thermal conduction. Note that (2.10) is only approximate and does not hold at high temperatures [91]. The Prandtl number is set to $\mathcal{P}r = 0.71$. This is a valid approximation for air over a temperature range up to $\widetilde{T}_\infty \sim 600$K [91]. The Reynolds number scaling is defined by:

$$\mathcal{R}e_a = \frac{\widetilde{\rho}_\infty c\widetilde{a}_\infty}{\widetilde{\mu}_\infty} \tag{2.12}$$

based on the free-stream speed of sound. It is different from the conventional definition based on the free-stream velocity:

$$\mathcal{R}e = \frac{\widetilde{\rho}_\infty c\widetilde{V}_\infty}{\widetilde{\mu}_\infty} \tag{2.13}$$

where $V = |\vec{\mathbf{V}}|$ is the magnitude of the velocity vector. The two Reynolds numbers are related by:

$$\mathcal{R}e = M_\infty \mathcal{R}e_a \tag{2.14}$$

where

$$M_\infty = \frac{\widetilde{V}_\infty}{\widetilde{a}_\infty} \tag{2.15}$$

is the free-stream Mach number. Assuming a Newtonian fluid, the viscous stresses are written using the Stokes relation, as follows:

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3}\mu \left(\nabla \cdot \vec{\mathbf{V}} \right) \delta_{ij} \tag{2.16}$$

where δ_{ij} is the Kronecker Delta. The Euler equations for the analysis of inviscid flows can be obtained from the Navier-Stokes equations by setting the right-hand side of (2.1) to zero.

2.2 Turbulence Modeling

The Favre-averaged Navier-Stokes equations are solved for compressible turbulent flows. The flow solution is separated into a sum of mean and fluctuating components. Only the mean-flow solution is found. This is adequate for many engineering problems with an affordable computational cost. Favre averaging introduces several additional terms to the Navier-Stokes equations. This includes the Reynolds stress tensor in the momentum equations, as well as the turbulent heat-flux vector and the turbulent kinetic energy in the energy equation.

The Reynolds stress tensor can be approximated to have the same form as the viscous stress tensor following the Boussinesq approximation. The Reynolds stress tensor is written as:

$$\tau_{ij}^R = \mu_t \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3}\mu_t \left(\nabla \cdot \vec{\mathbf{V}} \right) \delta_{ij} - \frac{2}{3}\kappa_t \delta_{ij} \tag{2.17}$$

where μ_t is the turbulent dynamic eddy viscosity, and κ_t is the turbulent kinetic energy. The trace of the Reynolds stress tensor is -2κ_t. The eddy viscosity μ_t can be several orders of magnitude larger than the laminar viscosity μ. The flow quantities in (2.17) and for the rest of this section represent mean flow values.

The turbulent heat-flux vector can be approximated to be proportional to the mean temperature gradient through a turbulent Prandtl number. This is written as:

$$\mathbf{q}_t = -k_t \nabla T = -\frac{\mu_t}{\mathcal{P}r_t \left(\gamma - 1\right)} \nabla a^2 \tag{2.18}$$

where $\mathbf{q}_t$ is the turbulent heat-flux vector, and $\mathcal{P}r_t$ is the turbulent Prandtl number. A typical value of $\mathcal{P}r_t = 0.90$ is used for aerodynamic flows.

The eddy viscosity and the turbulent kinetic energy are modeled using a turbulence model. In this study, the Spalart-Allmaras model [89] is used. In its derivation, the turbulent kinetic energy is treated as zero. This is a good approximation for non-hypersonic high-Reynolds-number flows [92, 93]. As a result, the last term in (2.17) is not included in the implementation. Godin et al. [94] compared the Spalart-Allmaras model with the Menter two-equation model. The latter is more accurate for separated flows, while the former is more accurate in attached flows and wakes, including merging boundary layers and wakes. The Spalart-Allmaras model is somewhat more robust, and it requires only one additional equation to be solved. The model is suitable for unstructured grid applications.

In non-dimensional conservative form, the Spalart-Allmaras model is written as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega \rho\tilde{\nu}\mathrm{d}V + \int_{\partial\Omega} \rho\tilde{\nu}\vec{\mathbf{V}}\cdot\hat{\mathbf{n}}\mathrm{d}S \tag{2.19}$$

$$= \frac{1}{\mathcal{R}e_a}\frac{1}{\sigma}\int_{\partial\Omega}\rho(\nu+\tilde{\nu})\nabla\tilde{\nu}\cdot\hat{\mathbf{n}}\mathrm{d}S \tag{2.20}$$

$$-\frac{1}{\mathcal{R}e_a}\frac{1}{\sigma}\int_\Omega(\nu+\tilde{\nu})\nabla\tilde{\nu}\cdot\nabla\rho\mathrm{d}V + \frac{1}{\mathcal{R}e_a}\frac{c_{b2}}{\sigma}\int_\Omega\rho\nabla\tilde{\nu}\cdot\nabla\tilde{\nu}\mathrm{d}V \tag{2.21}$$

$$+\frac{1}{\mathcal{R}e_a}c_{b1}\int_\Omega\rho\left(1-f_{t2}\right)\tilde{S}\tilde{\nu}\mathrm{d}V \tag{2.22}$$

$$-\frac{1}{\mathcal{R}e_a}\int_\Omega\rho\left(c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2}\right)\left(\frac{\tilde{\nu}}{d}\right)^2\mathrm{d}V \tag{2.23}$$

$$+\mathcal{R}e_a\int_\Omega\rho f_{t1}(\Delta U)^2\mathrm{d}V \tag{2.24}$$

The model is written in conservative form to apply a finite-volume method following Walsh [95].

The terms on the right-hand side of the equation are the production term, (2.22), the diffusion terms, (2.20) and (2.21), the destruction term, (2.23), and the trip term, (2.24), respectively. The kinematic eddy viscosity ν_t is calculated from the working variable $\tilde{\nu}$, using:

$$\nu_t = \tilde{\nu}f_{v1}, \quad f_{v1} = \frac{\chi^3}{\chi^3+c_{v1}^3}, \quad \chi = \frac{\tilde{\nu}}{\nu} \tag{2.25}$$

and ν denotes the kinematic viscosity. The eddy viscosity μ_t in (2.17) is obtained from the kinematic eddy viscosity using the relation $\mu_t = \rho\nu_t$.

The vorticity-like term $\tilde{S}$ in the production term is calculated using:

$$\tilde{S} = S\mathcal{R}e_a + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v2}, \quad f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}} \tag{2.26}$$

where S is the magnitude of the vorticity, d is the distance to the closest wall, and κ is the von Kármán constant. The magnitude of the vorticity is calculated using:

$$S = |\nabla \times \vec{\mathbf{V}}| = \left[\left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z} \right)^2 + \left(\frac{\partial u}{\partial z} - \frac{\partial w}{\partial x} \right)^2 + \left(\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \right)^2 \right]^{1/2} \tag{2.27}$$

The destruction term governs the dissipation of the eddy viscosity due to blocking effects from the wall. It contains a function f_w that models near-wall effects. The function is calculated by:

$$f_w = g \left(\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6} \right)^{1/6} \tag{2.28}$$

$$g = r + c_{w2}(r^6 - r), \quad r = \frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2} \tag{2.29}$$

The model includes a trip term that models laminar-to-turbulent flow transition. Transition locations are not predicted and are specified by the user. The trip term includes two functions that are given by:

$$f_{t1} = c_{t1} g_t \exp\left[-c_{t2} \frac{\omega_t^2}{\Delta U^2} (d^2 + g_t^2 d_t^2) \right] \tag{2.30}$$

$$f_{t2} = c_{t3} \exp(-c_{t4}\chi^2) \tag{2.31}$$

$$g_t = \min\left(0.1, \ \frac{\Delta U}{\omega_t \Delta x} \right) \tag{2.32}$$

where ΔU is the norm of the velocity difference between a field point and the trip, ω_t is the magnitude of the vorticity at the trip, d_t is the distance to the closest trip, and Δx is the grid spacing at the trip. The flow can be assumed to be fully-turbulent by setting f_{t1} and f_{t2} to zero. This assumes transition occurs at the leading edge. Closure coefficients are given by:

$$c_{b1} = 0.1355, \quad \sigma = 2/3, \quad c_{b2} = 0.622, \tag{2.33}$$

$$\kappa = 0.41, \quad c_{w1} = c_{b1}/\kappa^2 + (1 + c_{b2})/\sigma, \tag{2.34}$$

$$c_{w2} = 0.3, \quad c_{w3} = 2, \quad c_{v1} = 7.1, \tag{2.35}$$

$$c_{t1} = 1, \quad c_{t2} = 2, \quad c_{t3} = 1.2, \quad c_{t4} = 0.5 \tag{2.36}$$

Note that c_{t3} and c_{t4} are updated with values from newer versions of the model [96]. The wall boundary condition is $\tilde{\nu} = 0$. A value of $\nu_\infty/10$ is used as the free-stream condition for $\tilde{\nu}$, where ν_∞ is the kinematic viscosity in the free stream.

Ashford [97] proposed a modification to $\tilde{S}$ in the production term:

$$\tilde{S} = S\mathcal{R}e_a f_{v3} + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v2}, \quad f_{v2} = \left(1 + \frac{\chi}{c_{v2}}\right)^{-3} \tag{2.37}$$

$$f_{v3} = \frac{(1 + \chi f_{v1})\,(1 - f_{v2})}{\chi} \tag{2.38}$$

with $c_{v2} = 5$. The modification is widely used [25] and is found to produce better numerical properties [88]. It is adopted in the current work.

The turbulence model is solved in a form fully coupled with the mean-flow equations. As a note, a loosely coupled turbulence model is studied by Blanco and Zingg [98]. Combining (2.1) with (2.19) to (2.24), the set of fully-coupled equations is re-written as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega Q \mathrm{d}V + \int_{\partial\Omega} \mathbf{F}\cdot\hat{\mathbf{n}}\mathrm{d}S = \int_{\partial\Omega} \mathbf{G}\cdot\hat{\mathbf{n}}\mathrm{d}S + \int_\Omega P\mathrm{d}V \tag{2.39}$$

with

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \\ \rho\tilde{\nu} \end{bmatrix}, \quad P = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ P_{turb} \end{bmatrix} \tag{2.40}$$

where Q is the coupled mean-flow solution vector, and P is the source term, which contains entries only from the turbulence model. The term P_{turb} is given by:

$$P_{turb} = -\frac{1}{\mathcal{R}e_a}\frac{1}{\sigma}(\nu + \tilde{\nu})\nabla\tilde{\nu}\cdot\nabla\rho$$

$$+\frac{\rho}{\mathcal{R}e_a}\frac{c_{b2}}{\sigma}\nabla\tilde{\nu}\cdot\nabla\tilde{\nu}$$

$$+\frac{\rho}{\mathcal{R}e_a}c_{b1}\left(1-f_{t2}\right)\tilde{S}\tilde{\nu}$$

$$-\frac{\rho}{\mathcal{R}e_a}\left(c_{w1}f_w-\frac{c_{b1}}{\kappa^2}f_{t2}\right)\left(\frac{\tilde{\nu}}{d}\right)^2$$

$$+\rho\mathcal{R}e_a f_{t1}(\Delta U)^2 \tag{2.41}$$

The coupled inviscid and viscous flux tensors become:

$$\mathbf{F}=\begin{bmatrix}\rho\vec{\mathbf{V}}\\ \rho u\vec{\mathbf{V}}+p\hat{\mathbf{i}}\\ \rho v\vec{\mathbf{V}}+p\hat{\mathbf{j}}\\ \rho w\vec{\mathbf{V}}+p\hat{\mathbf{k}}\\ \vec{\mathbf{V}}(\rho E+p)\\ \rho\tilde{\nu}\vec{\mathbf{V}}\end{bmatrix} \tag{2.42}$$

and

$$\mathbf{G}=\frac{1}{\mathcal{R}e_a}\begin{bmatrix}\vec{0}\\ \tau_{xx}\hat{\mathbf{i}}+\tau_{yx}\hat{\mathbf{j}}+\tau_{zx}\hat{\mathbf{k}}\\ \tau_{xy}\hat{\mathbf{i}}+\tau_{yy}\hat{\mathbf{j}}+\tau_{zy}\hat{\mathbf{k}}\\ \tau_{xz}\hat{\mathbf{i}}+\tau_{yz}\hat{\mathbf{j}}+\tau_{zz}\hat{\mathbf{k}}\\ \boldsymbol{\tau}\cdot\vec{\mathbf{V}}+(k+k_t)\nabla T\\ \frac{1}{\sigma}\rho(\nu+\tilde{\nu})\nabla\tilde{\nu}\end{bmatrix} \tag{2.43}$$

respectively. The viscous stresses are given by:

$$\tau_{ij}=(\mu+\mu_t)\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)-\frac{2}{3}(\mu+\mu_t)\left(\nabla\cdot\vec{\mathbf{V}}\right)\delta_{ij} \tag{2.44}$$

which include contributions from the laminar viscous stresses and the Reynolds stresses. The heat-flux vector becomes:

$$\mathbf{q}=-(k+k_t)\nabla T=-\left(\frac{\mu}{\mathcal{P}r}+\frac{\mu_t}{\mathcal{P}r_t}\right)\frac{1}{(\gamma-1)}\nabla a^2 \tag{2.45}$$

2.3 Boundary Conditions

Boundary conditions describe the flow behavior at boundary regions. Physical conditions describe the physical processes, while additional conditions are needed to provide sufficient conditions for the numerical scheme. The latter can be implemented by extrapolation of the flow variables from the interior. For external aerodynamic flows, boundary conditions are applied at body surfaces, far field, and symmetry planes.

2.3.1 Body Surface - Inviscid Flow

Flow tangency is enforced at inviscid body surfaces. Writing the velocity at the surface as $\vec{\mathbf{V}} = V_n\ \hat{\mathbf{n}} + V_{t1}\ \hat{\mathbf{t}}_1 + V_{t2}\ \hat{\mathbf{t}}_2$, the tangency condition is written as:

$$V_n = 0 \tag{2.46}$$

There are two tangential directions, $\hat{\mathbf{t}}_1$, $\hat{\mathbf{t}}_2$, on a surface in three dimensions, which are assumed to be perpendicular to each other. The second physical condition states that the total enthalpy remains constant everywhere in the flow. The enthalpy H at the surface can be determined from the free-stream enthalpy as:

$$H = E + \frac{p}{\rho} = H_\infty \tag{2.47}$$

This condition is valid for steady, adiabatic, inviscid flows. For a calorically perfect gas, the total temperature is also constant everywhere in the flow.

Three numerical conditions are applied. The tangential velocities, V_{t1}, V_{t2}, and pressure, p, are extrapolated from the interior.

2.3.2 Body Surface - Viscous Flow

The no-slip condition is applied at viscous body surfaces:

$$u = 0, \quad v = 0, \quad w = 0 \tag{2.48}$$

According to boundary-layer theory, the pressure gradient normal to the surface, $\partial p/\partial n$, has a magnitude of the order of the boundary-layer thickness. As a result,

this term is considered small when compared to the other terms in the boundary-layer equations. We can write:

$$\frac{\partial p}{\partial n} \approx 0 \tag{2.49}$$

as discussed in reference [91]. Physically, this states that pressure is constant through the boundary layer in the direction normal to the surface. Note that (2.49) is not valid for large hypersonic Mach numbers. A condition for temperature can be specified as a von Neumann condition:

$$k\frac{\partial T}{\partial n} = q \tag{2.50}$$

where k is the thermal conductivity, and q is the fixed heat flux at the body surface. In particular, if there is no heat transfer across the surface, i.e. $q = 0$, then the adiabatic condition results. This is written as:

$$\frac{\partial T}{\partial n} = 0 \tag{2.51}$$

A density derivative condition can be written from (2.49) with the adiabatic condition and the equation of state for a perfect gas: $p = \rho RT$, where R is the specific gas constant. The condition is written as:

$$\frac{\partial \rho}{\partial n} = 0 \tag{2.52}$$

2.3.3 Far-Field Boundary

Far-field boundary conditions are applied using a characteristic approach. Local one-dimensional Riemann invariants normal to the boundary surface are used. This approach has the property of being non-reflecting, thus allowing disturbances in the solution to be propagated outward through the outer boundaries. The Riemann invariants are:

$$\begin{aligned} R_1 &= V_n - \frac{2a}{\gamma - 1} \\ R_2 &= V_n + \frac{2a}{\gamma - 1} \\ R_3 &= p/\rho^\gamma \end{aligned} \tag{2.53}$$

	Subsonic	Supersonic
Inflow	Physical: R_1, $R_{3,4,5,6}$ Numerical: R_2	Physical: R_1, R_2, $R_{3,4,5,6}$ Numerical: none
Outflow	Physical: R_1 Numerical: R_2, $R_{3,4,5,6}$	Physical: none Numerical: R_1, R_2, $R_{3,4,5,6}$

Table 2.1: A summary of far-field Riemann boundary conditions.

where V_n is the local velocity normal to the surface, and R_3 is a function of entropy. The Riemann invariants are associated with three characteristic speeds:

$$\lambda_1 = V_n - a$$

$$\lambda_2 = V_n + a$$

$$\lambda_3 = V_n \tag{2.54}$$

respectively. Additional invariant quantities are used to provide sufficient equations to solve for the boundary unknowns. These quantities are associated with the normal velocity, V_n, as a characteristic speed since they are convected with the flow. They are given by:

$$R_4 = V_{t1}$$

$$R_5 = V_{t2} \tag{2.55}$$

where V_{t1}, V_{t2} are tangential velocities. The turbulence variable is also used as an invariant quantity for turbulent flows:

$$R_6 = \tilde{\nu} \tag{2.56}$$

Depending on the sign of the characteristic speeds, the invariant quantities are either extrapolated from the interior or set to free-stream values. For subsonic inflow, $V_n \in$ (-a, 0), so that $\lambda_1 < 0$ and $\lambda_2 > 0$. Therefore, λ_1 is entering the domain, while λ_2 is leaving the domain. Similarly, for subsonic outflow, $V_n \in (0, a)$, so that $\lambda_1 < 0$ and $\lambda_2 > 0$. Therefore, λ_1 is also entering the domain, while λ_2 is leaving the domain. As a result, R_1 is set to free-stream values and R_2 is extrapolated from the interior

for both cases. Moreover, for inflow, V_n is entering the domain, so $R_{3,4,5,6}$ are set to free-stream values. For outflow, V_n is leaving the domain, so $R_{3,4,5,6}$ are extrapolated from the interior. The combinations are summarized in Table 2.1.

2.3.4 Symmetry Condition

The number of unknowns in the domain can be reduced if the flow field is known to be symmetric. Examples include a symmetric airfoil at zero angle of attack and a wing-body configuration at zero side-slip and rolling angles. Boundary conditions can be applied at the symmetric plane to obtain a symmetric solution across the plane. Flow tangency, $V_n = 0$, is applied so that the flow direction is parallel to the symmetry plane. In addition, zero gradients normal to the surface, i.e. $\partial Q/\partial n = 0$, are specified so that the solution is unchanged across the symmetry plane.

Chapter 3

Numerical Algorithm

This chapter describes the numerical method of solving the governing equations on a computational grid. The finite-volume method is used to solve the integral form of the equations on unstructured grids. Following a semi-discrete approach, the spatial terms in the equations are first approximated. The resulting system of ordinary differential equations with only time derivatives can then be solved using various time-marching methods. In particular, for steady-state problems, the discrete equations reduce to a system of nonlinear algebraic equations. Efficient methods can be used to solve these equations to reduce computational time.

3.1 Spatial Discretization

The spatial discretization follows the finite-volume approach of Mavriplis and Venkatakrishnan [28] for hybrid unstructured grids. It is a vertex-based approach; the flow variables are stored at grid vertices. For a given grid G, a dual grid G_{Dual} is constructed using the median dual (centroidal-median dual) approach to represent the control volumes around the vertices. Once the dual grid is constructed, the flow equations are solved for each control volume. A description of the dual grid construction is found in the work by Barth and Jespersen [99]. A dual grid has the following properties:

1. Each vertex of G_{Dual} is associated with a cell of G

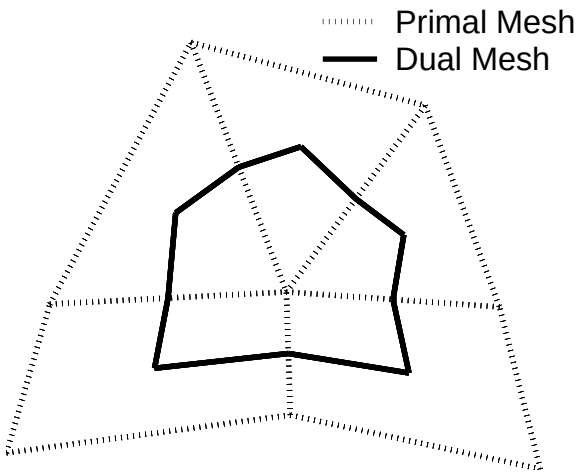

Figure 3.1: Illustration of a median dual control volume in two dimensions. This extends to three dimensions, where dual edges become dual faces.

2. Each edge of G is associated with an edge of G_{Dual}

3. If an edge separates two cells c_i and c_j of G, then the associated dual edge connects two vertices of G_{Dual} associated with c_i and c_j.

Note that edges of G are assumed to be straight, but edges of G_{Dual} need not be. An example of a dual-grid control volume is depicted in Figure 3.1. This approach extends to three dimensions, where each edge of G is associated with a face of G_{Dual}, which connects vertices of G_{Dual} associated with cells of G. The dual faces are not flat in general, and are composed of a number of flat sub-faces.

A typical control volume is depicted in Figure 3.2. In the figure, i and k are control volumes, and ik is the face with neighboring cells i and k. Here, $\vec{\mathbf{n}}_{ik}$ is an equivalent area-weighted normal of face ik. It is obtained by a sum of the sub-face area-weighted normals:

$$\vec{\mathbf{n}}_{ik} = \vec{\mathbf{n}}_{ik1} + \cdots + \vec{\mathbf{n}}_{ikn} \tag{3.1}$$

where $ik1$, ..., ikn are the sub-faces of face ik. Once the control volumes are constructed, the governing equations are discretized on the control volumes. Define $\mathcal{Q}_i$ to be the cell-averaged solution, the volume integral of the flow variables can be written

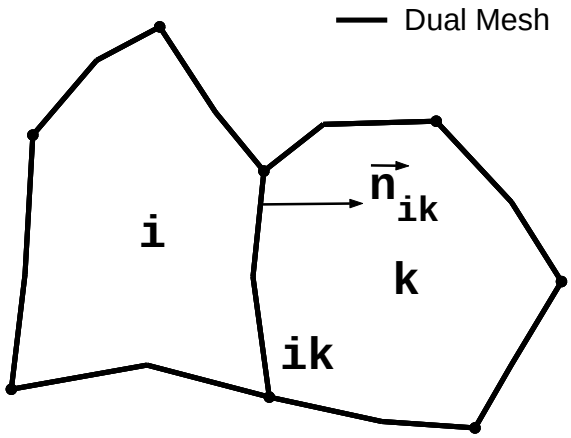

Figure 3.2: A typical control volume

as:

$$\left(\int_{\Omega} Q \mathrm{d}V\right)_i = \Omega_i \mathcal{Q}_i \tag{3.2}$$

where Ω_i is the cell volume. The first term in (2.39) becomes:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\int_{\Omega} Q \mathrm{d}V\right)_i = \frac{\mathrm{d}}{\mathrm{d}t}\left(\Omega_i \mathcal{Q}_i\right) = \Omega_i \frac{\mathrm{d}\mathcal{Q}_i}{\mathrm{d}t} \tag{3.3}$$

for stationary control volumes. The inviscid term can be written as:

$$\left(\int_{\partial\Omega} \mathbf{F}\cdot\hat{\mathbf{n}}\mathrm{d}S\right)_i \simeq \sum_{ik}\left(\mathbf{F}_{ik1}\cdot\vec{\mathbf{n}}_{ik1} + \cdots + \mathbf{F}_{ikn}\cdot\vec{\mathbf{n}}_{ikn}\right) \tag{3.4}$$

where the sum is performed over all faces of cell i. Here, $\mathbf{F}_{ik1}, \ldots, \mathbf{F}_{ikn}$ are the fluxes on the sub-faces. In particular, if the sub-face fluxes are equal, we can write:

$$\mathbf{F}_{ik} = \mathbf{F}_{ik1} = \cdots = \mathbf{F}_{ikn} \tag{3.5}$$

Then, the inviscid term can be written as:

$$\left(\int_{\partial\Omega} \mathbf{F}\cdot\hat{\mathbf{n}}\mathrm{d}S\right)_i \simeq \sum_{ik} \mathbf{F}_{ik}\cdot\vec{\mathbf{n}}_{ik} \tag{3.6}$$

where $\vec{\mathbf{n}}_{ik}$ is the equivalent normal as defined in (3.1). The inviscid flux, $\mathbf{F}_{ik}$, can be approximated by averaging from the two neighboring cells:

$$\mathbf{F}_{ik}\cdot\vec{\mathbf{n}}_{ik} \simeq \frac{\mathbf{F}(\mathcal{Q}_i)+\mathbf{F}(\mathcal{Q}_k)}{2}\cdot\vec{\mathbf{n}}_{ik} + D_{ik} \tag{3.7}$$

where $\mathbf{F}(Q)$ is calculated using (2.42), which is only a function of the flow variables, and D_{ik} is the numerical dissipation operator which is discussed later. Excluding numerical dissipation, the resulting discretization has a compact stencil involving only neighboring cells. It is second-order accurate on regular grids, and it has a five-point stencil on regular quadrilateral grids.

Similarly, the viscous term is written as:

$$\left(\int_{\partial\Omega} \mathbf{G}\cdot\hat{\mathbf{n}}\mathrm{d}S\right)_i \simeq \sum_{ik} \mathbf{G}_{ik}\cdot\vec{\mathbf{n}}_{ik} \tag{3.8}$$

where $\mathbf{G}_{ik}$ is the viscous flux on a face. The viscous flux is a function of the flow variables and their gradients, i.e. $\mathbf{G} = \mathbf{G}(Q, \nabla Q)$ as given in (2.43). For example, the viscous stresses consist of velocity derivatives, and the heat flux vector is a function of the temperature gradient. One approach to approximate the viscous flux is from the flow variables and gradients at cell faces:

$$\mathbf{G}_{ik} \simeq \mathbf{G}\left(\mathcal{Q}_{ik}, \nabla\mathcal{Q}_{ik}\right) \tag{3.9}$$

where $\mathcal{Q}_{ik}$ and $\nabla\mathcal{Q}_{ik}$ are approximated by averaging from neighboring cells:

$$\mathcal{Q}_{ik} \simeq \frac{\mathcal{Q}_i + \mathcal{Q}_k}{2} \tag{3.10}$$

$$\nabla\mathcal{Q}_{ik} \simeq \frac{\nabla\mathcal{Q}_i + \nabla\mathcal{Q}_k}{2} \tag{3.11}$$

To reduce floating point operations, gradients of ρ, u, v, w, T are used in the implementation [100]. The gradient at a cell, $\nabla\mathcal{Q}_i$, can be computed using the following integral [99]:

$$\int_\Omega \nabla Q\mathrm{d}V = \int_{\partial\Omega} Q\hat{\mathbf{n}}\mathrm{d}S \tag{3.12}$$

When discretized, it becomes:

$$\Omega_i\nabla\mathcal{Q}_i \simeq \sum_{ik} \mathcal{Q}_{ik}\vec{\mathbf{n}}_{ik} \tag{3.13}$$

where $\mathcal{Q}_{ik}$ is approximated using (3.10). This approach has a non-compact viscous stencil involving next-to-nearest neighboring terms. It is referred to as the simple

averaging approach for the rest of the study. It has a 13-point stencil on regular quadrilateral grids.

Using (3.3), (3.6) and (3.8), the governing equations, (2.39), can be written in a spatially discrete form as:

$$\Omega_i \frac{\mathrm{d}\mathcal{Q}_i}{\mathrm{d}t} + \sum_{ik} \mathbf{F}_{ik} \cdot \vec{\mathbf{n}}_{ik} = \sum_{ik} \mathbf{G}_{ik} \cdot \vec{\mathbf{n}}_{ik} + \Omega_i \mathcal{P}_i \tag{3.14}$$

3.1.1 Alternative Viscous Formulations

In this section, alternative viscous formulations based on various calculations of the face gradients, $\nabla \mathcal{Q}_{ik}$ in (3.11), are described. These formulations have a smaller stencil than the simple averaging approach.

Approximate Difference

The first approach approximates the directional gradient normal to the face by finite differences from neighboring cells:

$$\nabla \mathcal{Q}_{ik} \cdot \hat{\mathbf{n}}_{ik} \simeq \left(\frac{\mathcal{Q}_k - \mathcal{Q}_i}{l_{ik}} \, \hat{\mathbf{l}}_{ik} \right) \cdot \hat{\mathbf{n}}_{ik} \tag{3.15}$$

This approach is used by Smith et al. [49] and Mavriplis [35]. Here, $\hat{\mathbf{l}}_{ik}$ is the unit vector from the centroid of cell i to the centroid of cell k, l_{ik} is the distance between the centroids of cells i and k, and $\hat{\mathbf{n}}_{ik}$ is the unit normal of face ik. This approach has a compact stencil involving only neighboring cells. It has a five-point stencil on regular quadrilateral grids. However, the approximation is only exact on regular grids. The method is inaccurate when the line joining the centroids of cells i and k is not in the normal direction of face ik. This occurs on irregular grids, especially on stretched triangular grids.

Diamond Path

The second approach computes the face gradient by integration over a diamond shaped control volume. This approach is studied by Coirier [50]. The gradient is

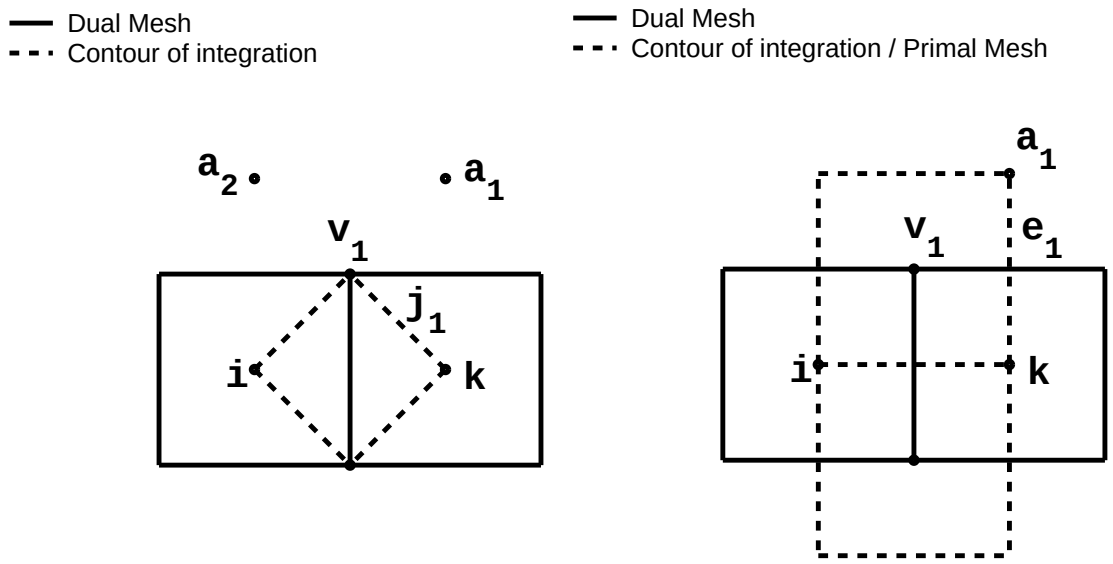

Figure 3.3: Calculation of the face gradients by integration over (a) a diamond path, (b) a primal-grid cell.

given by:

$$\nabla \mathcal{Q}_{ik} \simeq \frac{1}{\Omega_\psi} \sum_j \mathcal{Q}_j \vec{\mathbf{n}}_j \tag{3.16}$$

where Ω_ψ is the volume of the integrating region. The region is illustrated in Figure 3.3(a) for a square grid. The region is defined by the closed volume region bounded by the centroids of cells i and k and the vertices of face ik. This extends to three dimensions. Here, $j = 1, \ldots, N_j$ are the faces bounding the integrating region. The flow variables on these faces can be obtained by averaging from the face vertices:

$$\mathcal{Q}_j = \frac{1}{N_p} \sum_p \mathcal{Q}_p \tag{3.17}$$

where $p = 1, \ldots, N_p$ are the vertices of face j. The vertices can be dual-grid vertices or primal-grid vertices. If p is a primal-grid vertex (e.g. point k in Figure 3.3(a)), then $\mathcal{Q}_p$ is available from the corresponding dual cell. On the other hand, if p is a dual-grid vertex (e.g. point v_1), then $\mathcal{Q}_p$ can be obtained by averaging from the neighboring dual cells. A simple averaging procedure can be used:

$$\mathcal{Q}_p = \frac{1}{N_h} \sum_h \mathcal{Q}_h \tag{3.18}$$

where $h = 1, \ldots, N_h$ are the neighboring dual cells of the dual vertex p. This approach is used in the current work due to a lower cost. On the other hand, the weighting function of Holmes and Connell [46], which may be potentially more accurate, can be used. The search for the neighboring cells h can be implemented through the primal grid. Let C be the corresponding primal-grid cell of a dual vertex p (e.g. point v_1). Let V_h be the corresponding vertices of h on the primal grid (i.e. points i, k, a_1, a_2). Then V_h are vertices of C.

The diamond path approach has a larger stencil than the approximate difference approach on quadrilateral grids. It has a nine-point stencil on regular quadrilateral grids. Interestingly, it has the same stencil as the approximate difference approach on triangular grids.

Primal Grid

The third approach computes the face gradient by integration over control volumes on the primal grid [50]. This is illustrated in Figure 3.3(b), again for a square grid. The gradient is obtained by averaging from face vertices:

$$\nabla \mathcal{Q}_{ik} = \frac{1}{N_v} \sum_{v} \nabla \mathcal{Q}_v \tag{3.19}$$

where $v = 1, \ldots, N_v$ are the vertices of face ik. Each v has a corresponding cell on the primal grid, which is denoted as C_v. The gradient $\nabla \mathcal{Q}_v$ can be obtained by integration over the primal-grid cell C_v:

$$\nabla \mathcal{Q}_v \simeq \frac{1}{\Omega_v} \sum_{e} \mathcal{Q}_e \vec{\mathbf{n}}_e \tag{3.20}$$

where $e = 1, \ldots, N_e$ are faces of cell C_v on the primal grid, and Ω_v is the volume of the cell. The variables $\mathcal{Q}_e$ can be obtained by averaging from the face vertices:

$$\mathcal{Q}_e = \frac{1}{N_q} \sum_{q} \mathcal{Q}_q \tag{3.21}$$

where $q = 1, \ldots, N_q$ are vertices of face e on the primal grid. Each q has a corresponding cell on the dual grid, where the flow variables can be obtained. The primal grid approach has the same stencil as the diamond path approach.

3.1.2 Artificial Dissipation

Artificial dissipation is added to the spatial discretization to resolve shocks and to provide numerical stability. The dissipation scheme follows the approach of Jameson and Mavriplis [31] using undivided Laplacian and biharmonic operators. The matrix dissipation scheme of Swanson and Turkel [39] is used to improve accuracy. The dissipation operator in (3.7) is given by:

$$D_{ik} = -\frac{1}{2}|A_{ik}| \left[\varepsilon_{ik}^{(2)}(\mathcal{Q}_k - \mathcal{Q}_i) - \varepsilon_{ik}^{(4)}(L_k - L_i)\right] \tag{3.22}$$

$$L_i = \sum_k (\mathcal{Q}_k - \mathcal{Q}_i) \tag{3.23}$$

where

$$\varepsilon_i^{(2)} = \sum_k \kappa_2 \frac{|p_k - p_i|}{p_k + p_i} \tag{3.24}$$

and

$$\varepsilon_i^{(4)} = \max\left(0, \kappa_4 - \varepsilon_i^{(2)}\right) \tag{3.25}$$

where ε_{ik} is computed by averaging from the two neighboring cells i and k. Two parameters κ_2 and κ_4 control the addition of second- and fourth-difference dissipation. A pressure switch selects the second-difference operator in the presence of shocks, while the fourth-difference operator is used in areas of smooth flow. The resulting scheme is second-order accurate except in the vicinity of shocks, where it becomes locally first-order accurate. The Laplacian operator is denoted as L, and A is the inviscid flux Jacobian on a face:

$$A(Q, \vec{\mathbf{n}}) = \frac{\partial \mathbf{F}}{\partial Q} \cdot \vec{\mathbf{n}} \tag{3.26}$$

On face ik, it is evaluated as:

$$A_{ik} = \left.\frac{\partial \mathbf{F}}{\partial Q}\right|_{Q=\mathcal{Q}_{ik}} \cdot \vec{\mathbf{n}}_{ik} \tag{3.27}$$

where $\mathcal{Q}_{ik}$ can be calculated using Roe's average or a simple average from the two neighbouring cells. The absolute value of the inviscid Jacobian is given by:

$$|A| = X|\Lambda|X^{-1}, \quad |\Lambda| = \mathrm{diag}(\lambda_1, \ldots, \lambda_5) \tag{3.28}$$

where Λ and X are the eigenvalues and eigenvectors of the Jacobian respectively, and λ_i are the diagonal entries of $|\Lambda|$, given by:

$$\lambda_{1,2,3} = \max(|U|,\ V_l\sigma)$$

$$\lambda_4 = \max(|U + an|,\ V_n\sigma)$$

$$\lambda_5 = \max(|U - an|,\ V_n\sigma) \tag{3.29}$$

with

$$U = \vec{\mathbf{V}} \cdot \vec{\mathbf{n}}, \quad n = |\vec{\mathbf{n}}| \tag{3.30}$$

Here, a is the speed of sound, and $\sigma = |U| + an$ is the spectral radius. The eigenvectors are given in Appendix A. The Jacobian is modified to avoid vanishing eigenvalues using two parameters V_l and V_n. This occurs near stagnation points and sonic points. A scalar dissipation scheme is recovered with $V_l = V_n = 1$, while $V_l = V_n = 0$ leads to a pure matrix-dissipation scheme.

The turbulence model is discretized using a first-order scheme as suggested by Spalart and Allmaras [89]. This is implemented by setting $\varepsilon^{(2)} = 1$ and $\varepsilon^{(4)} = 0$ in (3.22) with $|A_{ik}| = |\vec{\mathbf{V}}_{ik} \cdot \vec{\mathbf{n}}_{ik}|$.

3.1.3 Boundary Conditions

This section describes the implementation of the boundary conditions as described in Section 2.3. The implementation follows the approach of Lassaline [100] and Barth and Jespersen [99]. A typical boundary cell is depicted in Figure 3.4. In the figure, i denotes a boundary cell, and b represents a boundary face with unit outward normal $\hat{\mathbf{n}}_b$.

The flow variables are extrapolated to boundary faces using a zeroth-order extrapolation:

$$\mathcal{Q}_e = \mathcal{Q}_i \tag{3.31}$$

where the subscript e denotes extrapolated quantities at boundary faces. Appropriate boundary conditions are applied to the extrapolated variables to obtain the solution

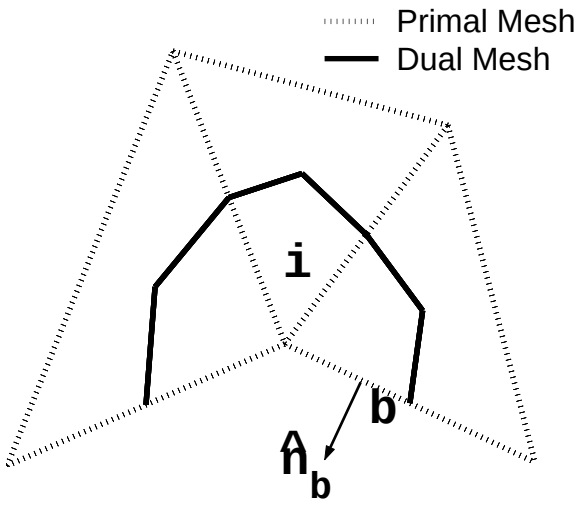

Figure 3.4: A typical boundary cell

at boundary faces. The resulting boundary solution is written as:

$$\mathcal{Q}_b = \mathcal{Q}_b(\mathcal{Q}_e) \tag{3.32}$$

which is a function of the extrapolated solution. Here, the subscript b denotes boundary values after applying boundary conditions. The boundary solution, $\mathcal{Q}_b$, is used to compute the inviscid and viscous terms at boundary cells. The inviscid term at a boundary cell is written as:

$$\left(\int_{\partial\Omega} \mathbf{F}\cdot\hat{\mathbf{n}}\mathrm{d}S\right)_i \simeq \sum_{ik} \mathbf{F}_{ik}\cdot\vec{\mathbf{n}}_{ik} + \sum_{b} \mathbf{F}_b\cdot\vec{\mathbf{n}}_b \tag{3.33}$$

where $\mathbf{F}_b$ is obtained from the boundary solution, i.e. $\mathbf{F}_b = \mathbf{F}(\mathcal{Q}_b)$.

For viscous flows, boundary conditions are applied to the gradients as well. At a boundary cell, $\nabla\mathcal{Q}_i$ is computed using:

$$\Omega_i\nabla\mathcal{Q}_i \simeq \sum_{ik} \mathcal{Q}_{ik}\vec{\mathbf{n}}_{ik} + \sum_{b} \mathcal{Q}_b\vec{\mathbf{n}}_b \tag{3.34}$$

using the boundary solution $\mathcal{Q}_b$. The gradients are extrapolated to boundary faces using a zeroth-order extrapolation:

$$\nabla\mathcal{Q}_e = \nabla\mathcal{Q}_i \tag{3.35}$$

Boundary conditions are applied to the extrapolated gradients to obtain the gradients at boundary faces:

$$\nabla \mathcal{Q}_b = \nabla \mathcal{Q}_b(\nabla \mathcal{Q}_e) \tag{3.36}$$

The viscous term at a boundary cell is written as:

$$\left(\int_{\partial\Omega} \mathbf{G} \cdot \hat{\mathbf{n}} \mathrm{d}S \right)_i \simeq \sum_{ik} \mathbf{G}_{ik} \cdot \vec{\mathbf{n}}_{ik} + \sum_b \mathbf{G}_b \cdot \vec{\mathbf{n}}_b \tag{3.37}$$

where $\mathbf{G}_b$ is obtained from the boundary solution and gradients, i.e. $\mathbf{G}_b = \mathbf{G}(\mathcal{Q}_b, \nabla \mathcal{Q}_b)$.

Body Surface - Inviscid Flow

Flow tangency is applied to obtain the velocities, u, v, w, at inviscid boundaries so that the normal velocity is zero and the tangential velocities are equal to extrapolated values. This is written as:

$$V_{n,b} = 0, \quad V_{t1,b} = V_{t1,e}, \quad V_{t2,b} = V_{t2,e} \tag{3.38}$$

where V_n, V_{t1}, V_{t2} are related to u, v, w using:

$$\vec{\mathbf{V}} = V_n \hat{\mathbf{n}} + V_{t1} \hat{\mathbf{t}}_1 + V_{t2} \hat{\mathbf{t}}_2 = u\hat{\mathbf{i}} + v\hat{\mathbf{j}} + w\hat{\mathbf{k}} \tag{3.39}$$

Here, $\hat{\mathbf{n}} = \hat{\mathbf{n}}_b$ is the unit outward normal at the boundary face, and $\hat{\mathbf{t}}_1$, $\hat{\mathbf{t}}_2$ are tangential directions. Using (3.38) and (3.39) to solve for u_b, v_b, and w_b, the velocity boundary conditions are given by:

$$\begin{aligned} u_b &= u_e - V_{n,e} n_x \\ v_b &= v_e - V_{n,e} n_y \\ w_b &= w_e - V_{n,e} n_z \end{aligned} \tag{3.40}$$

In the implementation, the velocities u_e, v_e, w_e are extrapolated and the conditions (3.40) are applied to compute u_b, v_b, w_b. The term $V_{n,e}$ is obtained using:

$$V_{n,e} = \vec{\mathbf{V}}_e \cdot \hat{\mathbf{n}} = u_e n_x + v_e n_y + w_e n_z \tag{3.41}$$

where n_x, n_y, n_z are components of $\hat{\mathbf{n}}$. Pressure is extrapolated from the interior:

$$p_b = p_e \tag{3.42}$$

Density is obtained using the enthalpy condition:

$$H_\infty = H_b = E_b + \frac{p_b}{\rho_b} \tag{3.43}$$

Rearranging, we get:

$$\rho_b = \frac{\gamma p_b}{(\gamma - 1)\left(H_\infty - \frac{1}{2}(u_b^2 + v_b^2 + w_b^2)\right)} \tag{3.44}$$

Since $\vec{\mathbf{V}} \cdot \hat{\mathbf{n}} = 0$, the inviscid body flux is given by:

$$\mathbf{F}_b \cdot \vec{\mathbf{n}}_b = \begin{bmatrix} 0 \\ pn_x \\ pn_y \\ pn_z \\ 0 \end{bmatrix} \tag{3.45}$$

Body Surface - Viscous Flow

The no-slip condition is applied to obtain the velocities at viscous surfaces:

$$u_b = 0, \quad v_b = 0, \quad w_b = 0 \tag{3.46}$$

Since the velocity is zero, total energy is adjusted to remove excessive kinetic energy. This keeps pressure the same as the extrapolated value. i.e.:

$$p_b = p_e \tag{3.47}$$

and

$$E_b = E_e - \frac{1}{2}\left(u_e^2 + v_e^2 + w_e^2\right) \tag{3.48}$$

Density is extrapolated from the interior:

$$\rho_b = \rho_e \tag{3.49}$$

The adiabatic condition is written as:

$$\nabla T_b \cdot \hat{\mathbf{n}} = 0 \tag{3.50}$$

The density gradient can be obtained using (2.52), and the velocity gradients are extrapolated from the interior. Since the velocity is zero, the inviscid body flux $\mathbf{F}_b \cdot \vec{\mathbf{n}}_b$ is also given by (3.45). The viscous body flux is given by:

$$\mathbf{G}_b \cdot \vec{\mathbf{n}}_b = \frac{1}{\mathcal{R}e_a} \begin{bmatrix} 0 \\ \tau_{xx}n_x + \tau_{yx}n_y + \tau_{zx}n_z \\ \tau_{xy}n_x + \tau_{yy}n_y + \tau_{zy}n_z \\ \tau_{xz}n_x + \tau_{yz}n_y + \tau_{zz}n_z \\ 0 \end{bmatrix} \tag{3.51}$$

The viscous energy flux is zero due to the adiabatic and the no-slip conditions. For turbulent flows, the turbulence variable is specified as zero on body surfaces. The condition is applied to boundary cells instead of boundary faces. This is found to improve stability of the method. We write:

$$\tilde{\nu}_i = 0 \tag{3.52}$$

for all boundary cells i at a body surface.

Far-Field Boundary

Far-field boundary conditions are obtained as described in Section 2.3.3. Invariant quantities are computed from either extrapolated or free-stream solutions based on the direction of local wave propagation. The invariants are used to determine the boundary face solution. They are given by:

$$R_1 = \begin{cases} V_{n,\infty} - \dfrac{2a_\infty}{\gamma - 1} & \text{if } V_{n,e} - a_e < 0 \\ V_{n,e} - \dfrac{2a_e}{\gamma - 1} & \text{otherwise} \end{cases} \tag{3.53}$$

$$R_2 = \begin{cases} V_{n,e} + \dfrac{2a_e}{\gamma - 1} & \text{if } V_{n,e} + a_e > 0 \\ V_{n,\infty} + \dfrac{2a_\infty}{\gamma - 1} & \text{otherwise} \end{cases} \tag{3.54}$$

$$R_3 = \begin{cases} p_e/\rho_e^\gamma & \text{if } V_{n,e} > 0 \\ \\ p_\infty/\rho_\infty^\gamma & \text{otherwise} \end{cases} \tag{3.55}$$

$$R_4 = \begin{cases} V_{t1,e} & \text{if } V_{n,e} > 0 \\ \\ V_{t1,\infty} & \text{otherwise} \end{cases} \tag{3.56}$$

$$R_5 = \begin{cases} V_{t2,e} & \text{if } V_{n,e} > 0 \\ \\ V_{t2,\infty} & \text{otherwise} \end{cases} \tag{3.57}$$

The turbulence variable is also used as an invariant quantity for turbulent flows:

$$R_6 = \begin{cases} \tilde{\nu}_e & \text{if } V_{n,e} > 0 \\ \\ \tilde{\nu}_\infty & \text{otherwise} \end{cases} \tag{3.58}$$

The flow variables at boundary faces are calculated from the invariant quantities as follows:

$$V_{n,b} = (R_1 + R_2)/2 \tag{3.59}$$

$$a_b = (\gamma - 1)(R_2 - R_1)/4 \tag{3.60}$$

$$\rho_b = \left(\frac{a_b}{\gamma R_3}\right)^{1/(\gamma-1)} \tag{3.61}$$

$$p_b = \rho_b a_b^2/\gamma \tag{3.62}$$

$$V_{t1,b} = R_4 \tag{3.63}$$

$$V_{t2,b} = R_5 \tag{3.64}$$

The velocities u_b, v_b, w_b are solved from the normal and tangential velocities, $V_{n,b}$, $V_{t1,b}$, and $V_{t2,b}$, using (3.39). The velocity conditions become:

$$\begin{aligned} u_b &= V_{n,b}n_x + V_{t1,b}t_{1,x} + V_{t2,b}t_{2,x} \\ v_b &= V_{n,b}n_y + V_{t1,b}t_{1,y} + V_{t2,b}t_{2,y} \\ w_b &= V_{n,b}n_z + V_{t1,b}t_{1,z} + V_{t2,b}t_{2,z} \end{aligned} \tag{3.65}$$

For turbulent flows, we also have:

$$\tilde{\nu}_b = R_6 \tag{3.66}$$

Symmetry

At symmetry boundaries, flow tangency is applied as given in (3.40). Pressure and density are extrapolated from the interior. In addition, the gradients normal to the symmetry plane are specified as zero:

$$\nabla \mathcal{Q}_b \cdot \hat{\mathbf{n}} = 0 \tag{3.67}$$

3.2 The Newton-Krylov Method

3.2.1 Newton Iterations

The governing equations after spatial discretization can be written as a system of ordinary differential equations:

$$\mathbf{\Omega} \frac{\mathrm{d}\mathcal{Q}}{\mathrm{d}t} + \mathcal{R}(\mathcal{Q}) = 0 \tag{3.68}$$

where $\mathbf{\Omega}$ is a block diagonal matrix of cell volumes, $\mathcal{Q}$ is a set of spatially discrete conservative flow variables, and $\mathcal{R}$ is a set of discrete flow equations. The matrix $\mathbf{\Omega}$ is given by:

$$\mathbf{\Omega} = \mathrm{diag}(\Omega_1 I_B,\ \Omega_2 I_B,\ \ldots,\ \Omega_N I_B) \tag{3.69}$$

where I_B is a $B \times B$ identity matrix block, and B is the block size. For example, the block size is six in three dimensions using a one-equation turbulence model, as shown in (2.40). The terms $\mathcal{Q}$ and $\mathcal{R}$ are given by:

$$\mathcal{Q} = \begin{bmatrix} \mathcal{Q}_1 \\ \mathcal{Q}_2 \\ \vdots \\ \mathcal{Q}_N \end{bmatrix}, \quad \mathcal{R} = \begin{bmatrix} \mathcal{R}_1 \\ \mathcal{R}_2 \\ \vdots \\ \mathcal{R}_N \end{bmatrix} \tag{3.70}$$

where $\mathcal{Q}_i$, $\mathcal{R}_i$ are vectors of length B, and N is the grid size. The terms $\mathcal{R}_i$ are given by (3.14):

$$\mathcal{R}_i = \sum_{ik} \mathbf{F}_{ik} \cdot \vec{\mathbf{n}}_{ik} - \sum_{ik} \mathbf{G}_{ik} \cdot \vec{\mathbf{n}}_{ik} - \Omega_i \mathcal{P}_i \tag{3.71}$$

Boundary conditions are included in $\mathcal{R}(\mathcal{Q})$ and are handled in a fully-implicit manner. A steady-state solution can be obtained by setting $\mathrm{d}\mathcal{Q}/\mathrm{d}t$ to zero. The resulting equations become a set of nonlinear algebraic equations:

$$\mathcal{R}(\mathcal{Q}) = 0 \tag{3.72}$$

Newton's method can be used to solve the steady-state equations. This method attempts to zero the residual in the next iteration, $\mathcal{R}^{n+1}$. Performing a Taylor series expansion of $\mathcal{R}^{n+1}$ about $\mathcal{Q}^n$, we can write:

$$\begin{aligned} \mathcal{R}^{n+1} &= \mathcal{R}(\mathcal{Q}^{n+1}) = \mathcal{R}(\mathcal{Q}^n + \Delta\mathcal{Q}^n) \\ &= \mathcal{R}(\mathcal{Q}^n) + \left.\frac{\partial \mathcal{R}}{\partial \mathcal{Q}}\right|_{\mathcal{Q}^n} \Delta\mathcal{Q}^n + \dots \end{aligned} \tag{3.73}$$

where $\partial\mathcal{R}/\partial\mathcal{Q}$ is an $N \times N$ block Jacobian matrix:

$$\frac{\partial \mathcal{R}}{\partial \mathcal{Q}} = \begin{bmatrix} \dfrac{\partial \mathcal{R}_1}{\partial \mathcal{Q}_1} & \dfrac{\partial \mathcal{R}_1}{\partial \mathcal{Q}_2} & \cdots & \dfrac{\partial \mathcal{R}_1}{\partial \mathcal{Q}_N} \\ \dfrac{\partial \mathcal{R}_2}{\partial \mathcal{Q}_1} & \dfrac{\partial \mathcal{R}_2}{\partial \mathcal{Q}_2} & \cdots & \dfrac{\partial \mathcal{R}_2}{\partial \mathcal{Q}_N} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial \mathcal{R}_N}{\partial \mathcal{Q}_1} & \dfrac{\partial \mathcal{R}_N}{\partial \mathcal{Q}_2} & \cdots & \dfrac{\partial \mathcal{R}_N}{\partial \mathcal{Q}_N} \end{bmatrix} \tag{3.74}$$

A first-order approximation of $\mathcal{R}^{n+1}$ can be written from (3.73) as:

$$\mathcal{R}^{n+1} \simeq \mathcal{R}(\mathcal{Q}^n) + \left(\frac{\partial \mathcal{R}}{\partial \mathcal{Q}}\right)^n \Delta\mathcal{Q}^n \tag{3.75}$$

Hence, $\mathcal{R}^{n+1} = 0$ can be written as a linear system of the following form:

$$\left(\frac{\partial \mathcal{R}}{\partial \mathcal{Q}}\right)^n \Delta\mathcal{Q}^n = -\mathcal{R}(\mathcal{Q}^n) \tag{3.76}$$

This is the linear system to be solved for Newton's method. Given $\mathcal{Q}^n$, we can compute $\mathcal{R}(\mathcal{Q}^n)$ and $(\partial\mathcal{R}/\partial\mathcal{Q})^n$. The system can then be solved for the update $\Delta\mathcal{Q}^n$. The solution at the next iteration is given by:

$$\mathcal{Q}^{n+1} = \mathcal{Q}^n + \Delta\mathcal{Q}^n \tag{3.77}$$

Starting from an initial guess $\mathcal{Q}^0$, the procedure is repeated until the solution satisfies some convergence tolerance.

A quadratic convergence rate can be obtained using Newton's method, where the number of precision digits of a solution is doubled every iteration. However, the method may not converge when the solution is far from the final result. This is because the linear system, (3.76), is a reasonable approximation to $\mathcal{R}^{n+1} = 0$ only for small $\Delta\mathcal{Q}^n$, when the solution is close to the final solution. Alternatively, the implicit-Euler method can be used, which is more robust than Newton's method. Applying a time integration to the unsteady equations, (3.68), and evaluating $\mathcal{R}(\mathcal{Q})$ at step $n+1$, we can write:

$$\mathbf{\Omega}\frac{\mathcal{Q}^{n+1} - \mathcal{Q}^n}{\Delta t} + \mathcal{R}(\mathcal{Q}^{n+1}) = 0 \tag{3.78}$$

where Δt is the time step. Expanding $\mathcal{R}(\mathcal{Q}^{n+1})$ using (3.75), equation (3.78) can be rewritten as:

$$\left[\ \Phi^n + \left(\frac{\partial\mathcal{R}}{\partial\mathcal{Q}}\right)^n\ \right]\ \Delta\mathcal{Q}^n = -\mathcal{R}(\mathcal{Q}^n) \tag{3.79}$$

This is the linear system to be solved for the implicit Euler method. Here, Φ^n is a block diagonal matrix of cell volumes divided by time steps:

$$\Phi^n = \frac{1}{\Delta t}\mathbf{\Omega} = \mathrm{diag}\left(\frac{\Omega_1}{\Delta t}I_B,\ \frac{\Omega_2}{\Delta t}I_B,\ \ldots,\ \frac{\Omega_N}{\Delta t}I_B\right) \tag{3.80}$$

In general, Δt can vary among implicit Euler iterations, and different time steps can be used locally. This is written as:

$$\Phi^n = \mathrm{diag}\left(\frac{\Omega_1}{\Delta t_1^n}I_B,\ \frac{\Omega_2}{\Delta t_2^n}I_B,\ \ldots,\ \frac{\Omega_N}{\Delta t_N^n}I_B\right) \tag{3.81}$$

The implicit Euler method is more robust than Newton's method, but it only has a linear convergence rate for a fixed time step. However, when the time step is increased

towards infinity, the term Φ^n vanishes in (3.79), and the method approaches Newton's method. In the current study, the implicit-Euler approach is used to provide a robust startup strategy. Newton-type convergence is obtained using large time steps near convergence.

3.2.2 The Linear System

The linear system that arises in the implicit Euler iterations can be written as $\mathcal{A}x = b$, where:

$$\mathcal{A} = \mathcal{A}(\mathcal{Q}^n) = \Phi^n + \left(\frac{\partial \mathcal{R}}{\partial \mathcal{Q}}\right)^n \tag{3.82}$$

with $x = \Delta\mathcal{Q}^n$ and b = -$\mathcal{R}(\mathcal{Q}^n)$. The linear system is sparse and non-symmetric. In addition, the matrix is indefinite due to the hyperbolic nature of the Navier-Stokes equations. Krylov subspace methods can be used to solve this class of problems. In particular, the generalized minimum residual method, GMRES, developed by Saad and Schultz [101] is found to be effective for aerodynamic applications. A summary of the method is given in this section. Further details can be found in the book by Saad [61]. The method is implemented using the PETSc software [102].

The GMRES method belongs to the class of projection methods for iterative solutions of linear systems. A projection method solves a linear system, $\mathcal{A}x = b$, by finding an approximate solution, x_m, from an affine subspace $x_0 + \mathcal{K}_m$ of dimension m, by imposing the Petrov-Galerkin conditions:

$$b - \mathcal{A}x_m \perp \mathcal{L}_m \tag{3.83}$$

where $\mathcal{K}$ is the search subspace, $\mathcal{L}$ is the subspace of constraints of dimension m, and x_0 is an arbitrary initial guess to the solution. The Petrov-Galerkin conditions constrain the residual vector $r_m = b$ - $\mathcal{A}x_m$ to be orthogonal to m linearly independent vectors that span the subspace $\mathcal{L}$. In particular, if $\mathcal{L} = \mathcal{K}$, the Petrov-Galerkin conditions are called the Galerkin conditions. A Krylov subspace method is a projection method where the subspace $\mathcal{K}$ is chosen to be the Krylov subspace. By doing so, the approximations obtained from the method can be written in terms of a polynomial of $\mathcal{A}$. The GMRES method is a Krylov subspace method with the subspace $\mathcal{L}$

chosen to be the subspace $\mathcal{AK}$. The method has the property to minimize the norm of the residual over all vectors in the Krylov subspace. It can be shown that the Petrov-Galerkin conditions in (3.83) are satisfied if and only if the L_2-norm of the residual vector, $||b - \mathcal{A}x||_2$, is minimized over the Krylov subspace. The proof of this optimality property is given in [61].

GMRES consists of three steps. First, we define the first Krylov vector to be $v_1 = r_0/||r_0||_2$, where r_0 is the initial residual vector. Then, we build an orthogonal basis of the Krylov subspace,

$$\mathcal{K}_m(\mathcal{A}, v_1) = \text{span}\left\{v_1, \mathcal{A}v_1, \mathcal{A}^2 v_1, \ldots, \mathcal{A}^{m-1} v_1\right\} \tag{3.84}$$

based on the first Krylov vector. This can be implemented by applying the Arnoldi's procedure:

$$\begin{aligned}
&\text{For } j = 1, 2, \ldots, m \\
&\quad h_{ij} = (\mathcal{A}v_j, v_i) \quad \text{for } i = 1, 2, \ldots, j \\
&\quad w_j = \mathcal{A}v_j - \sum_{i=1}^{j} h_{ij} v_i \\
&\quad h_{j+1,j} = ||w_j||_2 \\
&\quad v_{j+1} = w_j / h_{j+1,j}
\end{aligned} \tag{3.85}$$

where the bracket denotes an inner vector product. The procedure generates a set of vectors, v_1, v_2, ..., v_m, which forms an orthonormal basis of the Krylov subspace and spans the subspace. The vectors can be written in the following form:

$$v_j = q_{j-1}(\mathcal{A}) v_1 \tag{3.86}$$

where q_{j-1} is some polynomial of degree j-1. The quantities h_{ij} in the procedure can be related to the vectors v_j, by:

$$\mathcal{A}V_m = V_{m+1} \bar{H}_m \tag{3.87}$$

where $V_m = [v_1 \; v_2 \; \ldots \; v_m]$ is an $N \times m$ matrix, and $\bar{H}_m$ is an $(m+1) \times m$ Hessenberg matrix whose non-zero entries are given by h_{ij}.

Once the orthogonal vectors are defined, the third step is to compute the approximate solution. This can be achieved by exploiting the optimality property. Any vector x in $x_0 + \mathcal{K}_m$ can be written as $x = x_0 + V_m y$, where y is a vector of length m. Using the optimality property, if we can find the values of the vector y such that the residual norm, $J(y) = ||b - \mathcal{A}x||_2$, is minimized, then the solution satisfies the Petrov-Galerkin conditions. We can write:

$$\begin{aligned} b - \mathcal{A}x &= b - \mathcal{A}\left(x_0 + V_m y\right) \\ &= r_0 - \mathcal{A}V_m y \\ &= \beta v_1 - V_{m+1}\bar{H}_m y \\ &= V_{m+1}\left(\beta e_1 - \bar{H}_m y\right) \end{aligned} \tag{3.88}$$

using (3.87) with the relations $r_0 = \beta v_1$ and $v_1 = V_{m+1}e_1$, where e_1 denotes a column vector of zeros with the first element equals to one. Since the column vectors of V_{m+1} are orthonormal, we can write:

$$\begin{aligned} J(y) &= ||b - \mathcal{A}x||_2 \\ &= ||V_{m+1}\left(\beta e_1 - \bar{H}_m y\right)||_2 \\ &= ||\beta e_1 - \bar{H}_m y||_2 \end{aligned} \tag{3.89}$$

using (3.88). The approximate solution which satisfies the Petrov-Galerkin conditions is given by $x_m = x_0 + V_m y_m$, where y_m minimizes the function $J(y)$ in (3.89). The GMRES algorithm is summarized in Figure 3.5. The norm $J(y)$ is written in the form shown in (3.89) so that the minimization problem can be implemented efficiently. The least-squares problem, $\min||\beta e_1 - \bar{H}_m y||$ in line 9 of Figure 3.5, can be solved efficiently by transforming the Hessenberg matrix $\bar{H}_m$ into upper triangular form using plane rotations.

The GMRES method computes a new search vector every iteration. The vector is added to the Krylov subspace to progressively improve the solution. However, more search directions incur higher memory and computational costs. The computational work of the Arnoldi's procedure in (3.85) increases quadratically while storage increases linearly. For large problems, this limits the maximum number of iterations

1. $r_0 = b$ - $\mathcal{A}x_0$, $\beta = ||r_0||_2$, and $v_1 = r_0/\beta$
2. For $j = 1, 2, \ldots, m$
3. $\quad w_j = \mathcal{A}v_j$
4. $\quad$ For $i = 1, \ldots, j$
5. $\quad\quad h_{ij} = (w_j, v_i)$
6. $\quad\quad w_j \leftarrow w_j$ - $h_{ij}v_i$
7. $\quad h_{j+1,j} = ||w_j||_2$
8. $\quad v_{j+1} = w_j$ / $h_{j+1,j}$
9. Compute the minimizer y_m of $||\beta e_1 - \bar{H}_m y||_2$ and $x = x_0 + V_m y_m$

Figure 3.5: The GMRES algorithm

that can be used. The restarted version of the algorithm with a fixed m can be used, where the algorithm is restarted from the most recent solution. However, for indefinite matrices, convergence of the restarted algorithm may stagnate. The GMRES algorithm is guaranteed to converge in at most N steps, but it is impractical for large problems. It is desirable to keep m small. This can be achieved effectively by the use of preconditioning.

3.2.3 Inexact Newton Methods

Complete solving of the linear system (3.76) is found to be unnecessary to obtain a fast local convergence rate of Newton's method to the final solution [69]. This is because the system is only an approximation to zero the residual at the next iteration. An inexact Newton method can be used by not solving the system completely. This can significantly reduce computational work by avoiding over-solving of the system. The concept of over-solving is illustrated in the work by Eisenstat and Walker [69], where the reduction of the linear residual below a certain tolerance does not reduce the residual of the nonlinear equation $\mathcal{R}(\mathcal{Q})$. As a result, the computational effort of solving below the required tolerance is unnecessary. The linear system can be solved

until:

$$||b - \mathcal{A}x|| = ||\mathcal{R}(\mathcal{Q}^n) + \mathcal{A}(\mathcal{Q}^n)\Delta\mathcal{Q}^n|| \leq \eta_n||\mathcal{R}(\mathcal{Q}^n)|| \tag{3.90}$$

with a tolerance parameter η_n. When a constant value of $\eta_n > 0$ is used, the inexact Newton method has a linear convergence rate. The linear system, (3.76), becomes a better approximation to $\mathcal{R}^{n+1} = 0$ when the solution is close to the final result. A variable value of η_n can be used to obtain a fast local convergence of Newton's method as developed by Eisenstat and Walker [69]. The approach is written as given by Geuzaine [92]:

$$\eta_n = \begin{cases} \eta_{max} & n = 0 \\ \min(\eta_{max}, \eta_n^A) & n > 0 \text{ and } \gamma\eta_{n-1}^{\alpha} \leq \delta \\ \min(\eta_{max}, \max(\eta_n^A, \gamma\eta_{n-1}^{\alpha})) & n > 0 \text{ and } \gamma\eta_{n-1}^{\alpha} > \delta \end{cases} \tag{3.91}$$

with

$$\eta_n^A = \gamma \left(\frac{||\mathcal{R}(\mathcal{Q}^n)||}{||\mathcal{R}(\mathcal{Q}^{n-1})||} \right)^{\alpha} \quad \text{with } \alpha \in (1, 2] \text{ , } \gamma \in [0, 1] \tag{3.92}$$

Suggested parameters are [92]:

$$\gamma = 0.9, \quad \alpha = 2, \quad \eta_{max} = 0.05, \quad \delta = 0.1$$

3.2.4 Matrix-Free Approach

A matrix-free approach can be used for Krylov methods that require Jacobian matrix-vector products but not explicit Jacobian matrices. The GMRES method is one of these approaches. For example, step 3 of Figure 3.5 only requires the vector $\mathcal{A}v_j$. The matrix-vector product can be approximated using finite differences from Fréchet derivatives:

$$\mathcal{A}v \simeq \frac{\mathcal{R}(\mathcal{Q} + \epsilon v) - \mathcal{R}(\mathcal{Q})}{\epsilon} + \Phi v \tag{3.93}$$

where ϵ is the step size of the method. A first-order forward difference can be used, which involves one nonlinear residual calculation. The matrix-free approach allows quadratic convergence of Newton's method because the matrix of the linear system is a complete linearization of the residual vector. Memory usage is reduced because a matrix is not required. Usually an approximate Jacobian with lower memory usage is

still required for preconditioning purposes. The matrix-free approach is also preferable over the matrix-explicit Jacobian due to potential difficulties in linearization when constructing the Jacobian matrix. For example, the spectral radius in (3.29), the switch in (3.25), and the trip term of the turbulence model are non-differentiable. A typical value of the step size is:

$$\epsilon\bar{v} = \sqrt{\epsilon_m} \tag{3.94}$$

where $\bar{v}$ is the root mean square of v, and ϵ_m is the machine accuracy. In this work, a stepsize of:

$$\epsilon||v||_2 = \sqrt{10^{-10}} \tag{3.95}$$

is used following recent results from Chisholm and Zingg [72].

3.2.5 Preconditioning

Preconditioning transforms the linear system to one which has the same solution, but is easier to solve by an iterative solver. It can reduce the number of iterations required to solve the system. Two variants of preconditioning can be used with GMRES. Left preconditioning transforms the system to:

$$\mathcal{M}^{-1}\mathcal{A}x = \mathcal{M}^{-1}b \tag{3.96}$$

where $\mathcal{M}$ is a preconditioner, while for right preconditioning:

$$\mathcal{A}\mathcal{M}^{-1}u = b, \quad u = \mathcal{M}x \tag{3.97}$$

The right-preconditioned residual is the same as the original residual, but the left-preconditioned residual is not. It can be shown that left-preconditioned GMRES minimizes the norm of $\mathcal{M}^{-1}r$, while the right-preconditioned version minimizes the original residual r. Right preconditioning is chosen in the current work because the original residual is available every step, which is beneficial for checking convergence. The GMRES method with right preconditioning is given in Figure 3.6. There are two modifications from the original method of Figure 3.5. Line 3 of the algorithm is replaced by:

$$w_j = \mathcal{A}\mathcal{M}^{-1}v_j \tag{3.98}$$

1. $r_0 = b$ - $\mathcal{A}x_0$, $\beta = ||r_0||_2$, and $v_1 = r_0/\beta$
2. For $j = 1, 2, \ldots, m$
3. $\quad w_j = \mathcal{A}\mathcal{M}^{-1}v_j$
4. $\quad$ For $i = 1, \ldots, j$
5. $\quad\quad h_{ij} = (w_j, v_i)$
6. $\quad\quad w_j \leftarrow w_j$ - $h_{ij}v_i$
7. $\quad h_{j+1,j} = ||w_j||_2$
8. $\quad v_{j+1} = w_j$ / $h_{j+1,j}$
9. Compute the minimizer y_m of $||\beta e_1 - \bar{H}_m y||_2$ and $x = x_0 + \mathcal{M}^{-1}V_m y_m$

Figure 3.6: The right-preconditioned GMRES algorithm

where Arnoldi's procedure builds an orthogonal basis of the right-preconditioned Krylov subspace:

$$\text{span}\left\{v_1, \mathcal{A}\mathcal{M}^{-1}v_1, (\mathcal{A}\mathcal{M}^{-1})^2 v_1, \ldots, (\mathcal{A}\mathcal{M}^{-1})^{m-1}v_1\right\} \tag{3.99}$$

A similar modification is applied to line 9 with:

$$x = x_0 + \mathcal{M}^{-1}V_m y_m \tag{3.100}$$

Equation (3.98) can be implemented with:

$$w_j = \mathcal{A}z_j \tag{3.101}$$

and

$$z_j = \mathcal{M}^{-1}v_j \tag{3.102}$$

For a matrix-free approach, this can be written as:

$$\mathcal{A}\mathcal{M}^{-1}v \simeq \frac{\mathcal{R}(\mathcal{Q} + \epsilon\mathcal{M}^{-1}v) - \mathcal{R}(\mathcal{Q})}{\epsilon} + \Phi\mathcal{M}^{-1}v \tag{3.103}$$

The preconditioner is chosen so that the matrix $\mathcal{A}\mathcal{M}^{-1}$ has a better condition number than the original matrix $\mathcal{A}$. This improves convergence of an iterative solver.

A definition of the conditioner number is given in Appendix B. As illustrated by Pueyo [103], conditioning of the linear system can be improved by a clustering of the eigenvalues of $\mathcal{A}\mathcal{M}^{-1}$ around unity. An effective preconditioner can be chosen so that $\mathcal{M}^{-1}$ approximates $\mathcal{A}^{-1}$ while $\mathcal{M}^{-1}$ is efficient to compute. A preconditioner is effective if the preconditioned system can be solved in a small number of iterations.

Incomplete Factorization

An incomplete lower-upper factorization, ILU, computes a sparse lower triangular matrix $\mathcal{L}$, and a sparse upper triangular matrix $\mathcal{U}$, for a given sparse matrix $\widetilde{\mathcal{A}}$, so that:

$$\mathcal{L}\mathcal{U} \simeq \widetilde{\mathcal{A}} \tag{3.104}$$

where equality of (3.104) represents a complete factorization. The preconditioner can be written as:

$$\mathcal{M} = \mathcal{L}\mathcal{U} \tag{3.105}$$

Once the factors are constructed, the preconditioner in (3.102) can be applied with:

$$\mathcal{M}^{-1} = \mathcal{U}^{-1}\mathcal{L}^{-1} \tag{3.106}$$

which is an inexpensive procedure involving back-solves of triangular matrices.

It is logical to choose $\widetilde{\mathcal{A}}$ to be a close approximation to the original matrix $\mathcal{A}$ so that $\mathcal{M}^{-1}$ approximates $\mathcal{A}^{-1}$ in some sense. However, as shown by Pueyo and Zingg [60], a more diagonally dominant matrix can provide a more effective preconditioner. Moreover, an approximate Jacobian matrix with less nonzero elements can reduce memory usage.

Approximate Jacobian

An approximate Jacobian is used as the matrix $\widetilde{\mathcal{A}}$ for preconditioning. The matrix is constructed by a complete linearization of the residual vector $\mathcal{R}$ with modifications in the artificial dissipation and the viscous terms. The resulting Jacobian has contributions from nearest neighbors only. The modifications affect convergence of the inner (GMRES) iterations, but not solution accuracy. Artificial dissipation in the

preconditioner includes only second-difference dissipation, following the approach of Pueyo and Zingg [60]. The dissipation coefficients are given by:

$$\begin{aligned} \varepsilon_p^{(2)} &= \varepsilon^{(2)} + \sigma\varepsilon^{(4)} \\ \varepsilon_p^{(4)} &= 0 \end{aligned} \tag{3.107}$$

where σ is a parameter which controls the diagonal dominance of the matrix, $\varepsilon^{(2)}$ and $\varepsilon^{(4)}$ are the dissipation coefficients in the residual $\mathcal{R}$ as given in (3.22), and the subscript p denotes values of the preconditioner. The matrix dissipation parameters, V_l, V_n in the matrix $\widetilde{\mathcal{A}}$, are the same as those used on the right-hand side.

Approximate viscous calculations are used to construct the matrix $\widetilde{\mathcal{A}}$. For example, the simple averaging approach in (3.11), and the diamond-path and primal-grid approaches in Section 3.1.1, have a non-compact stencil involving next-to-nearest neighboring terms. These terms are dropped in the matrix $\widetilde{\mathcal{A}}$ to maintain a compact stencil. The inclusion of these terms leads to an increase in the number of nonzeros in the matrix, which increases cost and memory usage of preconditioning, especially in three dimensions.

The ordering of unknowns in $\widetilde{\mathcal{A}}$ has an effect on the effectiveness of incomplete factorization. In particular, the reverse Cuthill-McKee reordering technique [81] is found to be effective. This approach is used to reorder the unknowns of the matrix before applying incomplete factorization.

Level-of-fill and Threshold Approaches

Once $\widetilde{\mathcal{A}}$ is constructed, $\mathcal{M}$ can be formed using an incomplete factorization (ILU) of $\widetilde{\mathcal{A}}$. This method is used because $\widetilde{\mathcal{A}}$ is non-symmetric in general. The zero-fill approach, ILU(0), computes the sparse lower and upper factors, $\mathcal{L}$ and $\mathcal{U}$, so that $\mathcal{L}$ has the same nonzero pattern as the lower part of $\widetilde{\mathcal{A}}$, while $\mathcal{U}$ has the same nonzero pattern as the upper part of $\widetilde{\mathcal{A}}$. This approach has the same memory usage as the original sparse matrix, but the factors are not necessarily accurate. There can be elements in the complete factors in the zero pattern of $\widetilde{\mathcal{A}}$, which are neglected in the ILU(0) approach. A more accurate preconditioner can be constructed using a level-of-fill approach, ILU(p), by allowing fill-in in the zero pattern of $\widetilde{\mathcal{A}}$. The parameter p

1. Set $U = A$, and $L =$ the identity matrix I
2. For $i = 2, \ldots, N$
3. For $k = 1, \ldots, i-1$
4. $L_{i,k} = U_{i,k}/U_{k,k}$
5. For $j = 1, \ldots, N$
6. $U_{i,j} \leftarrow U_{i,j}$ - $L_{i,k} \times U_{k,j}$
7. Update $lev_{i,j} = \min\{lev_{i,j}, lev_{i,k} + lev_{k,j} + 1\}$
8. Replace elements in rows L_i and U_i with $lev_{i,j} > p$ by zero

Figure 3.7: An incomplete LU factorization using a level-of-fill approach, ILU(p)

controls the amount of fill-in. The approach is summarized in Figure 3.7. A level of fill is assigned to each element in the factors. An element is kept if it has a level of fill not exceeding p. Larger values of p allow more elements in the factors, which produce more accurate preconditioners with more memory usage. The method reduces to the ILU(0) approach for $p = 0$. For large p, the method approaches a complete factorization. The level of fill is defined by:

$$lev_{i,j} = \min\{lev_{i,j}, lev_{i,k} + lev_{k,j} + 1\} \tag{3.108}$$

recursively as given in line 7 of the algorithm. The initial levels are set as:

$$lev_{i,j} = \begin{cases} 0 & \text{if } a_{i,j} \neq 0, \text{ or } i = j \\ \\ \infty & \text{otherwise} \end{cases} \tag{3.109}$$

For M-type matrices as defined in Appendix B, the higher the level of fill, the smaller the elements. Therefore, dropping elements above a certain level p may produce an accurate factorization. However, this is not necessary true for indefinite matrices.

There are drawbacks of the level-of-fill approach. The amount of fill-in and computational work are not predictable for $p>0$. The level of fill for indefinite matrices may not be a good indicator of the size of the elements to be dropped. The algorithm

1. Set $U = A$, and $L =$ the identity matrix I
2. For $i = 2, \ldots, N$
3. Compute the norm of the i-th row of A: $\phi_i = \left(\sum_{j=1}^{N} a_{i,j}^2 \right)^{1/2}$
4. Let $\tau_i = \tau \times \phi_i$ to be the drop tolerance for row i
5. For $k = 1, \ldots, i-1$
6. $L_{i,k} = U_{i,k}/U_{k,k}$
7. For $j = 1, \ldots, N$
8. $U_{i,j} \leftarrow U_{i,j}$ - $L_{i,k} \times U_{k,j}$
9. For $k = 1, \ldots, i-1$
10. If $|L_{i,k}| < \tau_i$, set $L_{i,k}$ to zero
11. For $k = i+1, \ldots, N$
12. If $|U_{i,k}| < \tau_i$, set $U_{i,k}$ to zero
13. Keep only the largest $P_{l,i} + P$ elements in row L_i
14. Keep only the largest $P_{u,i} + P$ elements in row U_i

Figure 3.8: An incomplete LU factorization using a threshold strategy, ILUT(P, τ)

may drop large elements and result in an inaccurate factorization. An alternative approach based on a threshold strategy attemps to remedy these issues. The approach is given in Figure 3.8. The threshold approach, ILUT(P,τ), is based on dropping elements in the factorization according to their magnitudes rather than their locations. The drop tolerance τ determines the elements to be neglected in the factorization, as shown in lines 10 and 12 of the algorithm. The parameter P controls how many elements are to be kept. The definitions of the parameters are the same as those used in the work of Pueyo [103]. Let $P_{l,i}$ and $P_{u,i}$ be the number of nonzero elements in the lower and upper parts of row i, respectively, of the original matrix $\widetilde{\mathcal{A}}$, then the largest $P_{l,i}$+P elements are kept in row i of $\mathcal{L}$, and the largest $P_{u,i}$+P elements are kept in the same row of $\mathcal{U}$. Unlike the level-of-fill approach, the memory usage of the threshold approach is more predictable.

Since the matrix $\widetilde{\mathcal{A}}$ has a block structure such as that given in (3.74), block versions of factorizations can be used. The block approach has less factorization operations, since the matrix has a smaller number of nonzero blocks than nonzero elements, but it requires inversions of matrix blocks during factorization. A block version of ILU(p) is used in the current work, while a scalar version of ILUT(P,τ) is also included.

3.2.6 Time-Stepping Strategy

This section describes the choice of the time step, Δt_i^n in (3.81), of the implicit Euler method. Local time steps are used to accelerate convergence. A time-stepping strategy which combines the use of small time steps during startup and switches to large time steps near convergence is used to provide a robust and efficient algorithm with a Newton-type convergence. The advantage of small time steps to robustness is twofold. They improve stability of the nonlinear iterations and the conditioning of the linear system.

Different time steps are used in the mean-flow equations and in the turbulence model. For the mean-flow equations, the local time step following Pulliam [104] is used:

$$\Delta t_{flow,i} = \frac{\Delta t_{ref}}{1 + \sqrt{\Omega_i^{-1}}} \tag{3.110}$$

where Ω_i is the local cell volume. A reference time step Δt_{ref} is used to control the time step globally. The switched evolution relaxation approach from Mulder and van Leer [53] can be used. This approach increases Δt_{ref} in inverse proportion to the residual norm, thus approaching Newton's method when the residual converges to zero. An alternative approach, which is found to be more robust, is used in the current work. It is written as:

$$\Delta t_{ref}^n = \begin{cases} \Delta t_1 & n \leq n_1 \\ \alpha & n = n_1 + 1 \\ \beta \times \Delta t_{ref}^{n-1} & n \text{ is a multiple of } f \\ \Delta t_{ref}^{n-1} & \text{otherwise} \end{cases} \tag{3.111}$$

where $n_1 \geq 0$, $\beta \geq 1$, and $f \geq 1$. A small time step Δt_1 is used in the first n_1 iterations. If n_1=0, then the small time step is unused. After that, the time step is switched to

a larger value α. The time step is multiplied by a factor β every f iterations. Typical values of the parameters are:

$$n_1 = 2, \quad \Delta t_1 = 0.05, \quad \alpha = 2, \quad \beta = 2, \quad f = 5 \tag{3.112}$$

The turbulence model requires small time steps during startup. Otherwise, large solution updates occur which cause the solution to become unstable. Small time steps slow down the convergence rate. Spalart and Allmaras [89] suggest the use of an M-type Jacobian matrix to prevent negative values of the turbulent solution. This improves robustness at a penalty on the convergence rate. Chisholm and Zingg [88] suggest an alternative approach using a local time step. The approach attempts to prevent large updates by locally reducing the time step. It allows larger time steps to be used elsewhere in the domain. Moreover, a matrix-free approach can be used due to the lack of modification in the Jacobian matrix. This provides an effective startup strategy. The time step is written as:

$$\Delta t_{turb,i} = \begin{cases} \Delta t_{flow,i} & \text{if } |\delta_{e,i}| < \delta_{m,i} \\ \\ |\Delta t_{limit,i}| & \text{otherwise} \end{cases} \tag{3.113}$$

where $\delta_{e,i}$ is an estimate of the local solution update, and $\delta_{m,i} = r\tilde{\nu}_i$ is the maximum allowable change specified by a parameter r. A typical value of r is 0.3, which allows a 30 percent change. When the estimate exceeds the allowable value, the time step is reduced to $\Delta t_{limit,i}$. Otherwise, the mean-flow time step is used. The estimate is determined by solving the following equation:

$$J_{66,i}\ \delta_{e,i} = -\mathcal{R}_{6,i} \tag{3.114}$$

which is a scalar equation analogous to (3.76) of Newton's method applied to a local turbulence equation. The quantities $J_{66,i}$ and $\mathcal{R}_{6,i}$ are obtained from:

$$\widetilde{\mathcal{A}}_{i,i} = \begin{bmatrix} J_{11,i} & J_{12,i} & \dots & J_{16,i} \\ J_{21,i} & J_{22,i} & \dots & J_{26,i} \\ \vdots & \vdots & \ddots & \vdots \\ J_{61,i} & J_{62,i} & \dots & J_{66,i} \end{bmatrix} \tag{3.115}$$

and

$$\mathcal{R}_i = \begin{bmatrix} \mathcal{R}_{1,i} \\ \mathcal{R}_{2,i} \\ \vdots \\ \mathcal{R}_{6,i} \end{bmatrix} \tag{3.116}$$

where $\widetilde{\mathcal{A}}_{i,i}$ is the (i,i) block of $\widetilde{\mathcal{A}}$, which is a 6×6 block when using a one-equation turbulence model in three dimensions. The term $J_{66,i}$ is the last entry in the block, and $\mathcal{R}_{6,i}$ is the last entry of $\mathcal{R}_i$. The last (6-th) row of $\widetilde{\mathcal{A}}_{i,i}$ and $\mathcal{R}_i$ corresponds to the turbulence model. Note that $J_{66,i}$ is obtained from the approximate matrix $\widetilde{\mathcal{A}}$.

The limiting time step is determined so that it reduces the estimate to the allowable value, i.e. $|\delta_{e,i}| = \delta_{m,i}$. This can be calculated by solving the following equation for $\Delta t_{limit,i}$:

$$\left(\frac{\Omega_i}{\Delta t_{limit,i}} + J_{66,i} \right) \delta_{m,i} = -\mathcal{R}_{6,i} \tag{3.117}$$

which is analogous to (3.79) of the implicit Euler method. Details about the local time step can be found in the original work by Chisholm and Zingg [72].

A safeguarding mechanism is included to prevent nonphysical solution updates caused by too large a time step. The solution update is checked every nonlinear iteration. If nonphysical flow quantities, such as negative pressure or density, are encountered, then the recent solution update is rejected and the time step of the next iteration is halved. This is written as:

$$\mathcal{Q}^{n+1} = \begin{cases} \mathcal{Q}^n & \text{if } \mathcal{Q}^{n+1} \text{ nonphysical} \\ \\ \mathcal{Q}^n + \Delta\mathcal{Q}^n & \text{otherwise} \end{cases} \tag{3.118}$$

$$\Delta t_{ref}^{n+1} = \begin{cases} \dfrac{1}{2} \times \Delta t_{ref}^n & \text{if } \mathcal{Q}^{n+1} \text{ nonphysical} \\ \\ \Delta t_{ref}^{n+1} & \text{as given by (3.111), otherwise} \end{cases} \tag{3.119}$$

A similar approach is used by Smith et al. [49]. The turbulence model is designed for nonnegative values of $\tilde{\nu}$ only. The turbulent solution update is checked every

iteration, and negative values of $\tilde{\nu}$ are clipped to zero. When this happens, the mean-flow solution is not rejected and the timestep is not halved. This is written as:

$$\tilde{\nu}^{n+1} = \begin{cases} 0 & \text{if } \tilde{\nu}^{n+1}<0 \\ \\ \tilde{\nu}^{n+1} & \text{otherwise} \end{cases} \tag{3.120}$$

Clipping may prevent the turbulence model from converging fully on some grids. This is because the converged solution would contain some negative values of $\tilde{\nu}$, which are prevented by clipping. An approach which attempts to improve positivity of the turbulent solution is given in Appendix C. Other approaches based on the use of logarithmic variables to ensure positivity of the turbulent solution can be found in the works by Ilinca and Pelletier [105] and Soulaimani et al. [106].

Chapter 4

Results and Discussions

4.1 Test Cases

A total of eight cases are studied. A summary is given in Table 4.1. The grids are summarized in Table 4.2. All turbulent cases are assumed to be fully-turbulent; laminar-to-turbulent transition is assumed to occur at the leading edge. The first three cases are inviscid, laminar, and turbulent flows, respectively, over the NACA 0012 airfoil. Case 3 is studied on triangular and quadrilateral grids. The quadrilateral grid is depicted in Figure D.1 in the appendix. Case 4 is a turbulent flow over the RAE 2822 airfoil. The case is studied on quadrilateral, hybrid, and triangular grids. The hybrid grid has quadrilaterals, while the triangular grid has stretched triangular cells, in the viscous region. The three grids have off-wall spacings of 2.5×10^{-6}, 10^{-5}, and 1.2×10^{-6}, respectively. Case 5 is a turbulent flow over the multi-element airfoil AGARD 303 A2. The triangular grid for this case is depicted in Figure D.2.

Case 6 is an inviscid flow over the ONERA M6 wing [107]. This geometry has been studied by a number of researchers, as shown in Table 1.1. A total of six grids are studied. Grids 6a to 6c are tetrahedral grids with increasing densities up to one million nodes. Grid 6a is similar to the one used in the work of Manzano et al. [90]. Grids 6d to 6f are hexahedral grids used in the work of Nichols and Zingg [108]. Case 7 is a turbulent flow over the wing. The case is studied on three hybrid grids with increasing densities up to over a million nodes. The grids have prisms in the viscous

Case	Geometry	M_∞	α°	$\mathcal{R}e$
1	NACA 0012	0.63	2.0	Inviscid
2	NACA 0012	0.8	5.0	500
3	NACA 0012	0.3	6.0	9.0×10^6
4	RAE 2822	0.729	2.31	6.5×10^6
5	AGARD 303 A2	0.197	20.18	3.52×10^6
6	ONERA M6	0.8395	3.06	Inviscid
7	ONERA M6	0.8395	3.06	11.7×10^6
8	DLR F6	0.75	0.52	3.0×10^6

Table 4.1: A summary of test cases

region, and tetrahedra elsewhere in the domain. Grid 7c is depicted in Figure D.3. Case 8 is a turbulent flow over the DLR F6 wing-body configuration [109] used in the second Drag Prediction Workshop [6]. Four grids are used for Case 8. Grids 8a and 8b are hybrid grids with a half million nodes and over one million nodes, respectively. Grid 8b is depicted in Figure D.4. Grid 8c is the tetrahedral grid with 1,121,301 nodes from the workshop [6]. Grid 8d is a hexahedral grid. It is generated by removing every other grid plane from a three-million-node grid from the workshop [29]. Grid 8d has roughly a half million nodes.

4.1.1 Grid Generation

The ICEM CFD grid generator is used to generate grids 6a to 6c, 7a to 7c, 8a, and 8b. In grids 7b, 7c, 8a, and 8b, a density volume region is included above the wing surface to provide a better resolution of the shock wave. This is depicted in Figures D.3 and D.4. For the viscous grids 7a, 7b, and 8a with up to roughly a half million nodes, the prism layers are generated by extruding 15 layers of prism elements from the body surface mesh using a growth ratio of 1.5. For grids 7c and 8b with over a million nodes, the prism layers are generated by extruding 26 layers of prism elements from the surface mesh. The first 15 layers from the body have a growth ratio of 1.2, and the remaining 11 layers have a growth ratio of 1.5. These parameters are comparable to those described in [6]. The off-wall spacing is 2.5×10^{-6} mean

Grid	Geometry	Size	Type
1	NACA 0012	14,193	Triangular
2	NACA 0012	14,575	Triangular
3a	NACA 0012	14,174	Triangular
3b	NACA 0012	27,580	Quadrilateral
4a	RAE 2822	14,620	Quadrilateral
4b	RAE 2822	13,494	Hybrid
4c	RAE 2822	15,900	Triangular
5	AGARD 303 A2	51,779	Triangular
6a	ONERA M6	237,767	Tetrahedral
6b	ONERA M6	490,451	Tetrahedral
6c	ONERA M6	1,017,061	Tetrahedral
6d	ONERA M6	234,534	Hexahedral
6e	ONERA M6	477,080	Hexahedral
6f	ONERA M6	971,160	Hexahedral
7a	ONERA M6	179,758	Hybrid
7b	ONERA M6	480,775	Hybrid
7c	ONERA M6	1,124,270	Hybrid
8a	DLR F6	584,543	Hybrid
8b	DLR F6	1,293,250	Hybrid
8c	DLR F6	1,121,301	Tetrahedral
8d	DLR F6	442,393	Hexahedral

Table 4.2: A summary of grids

aerodynamic chords for the wing grids, and is 2×10^{-6} mean aerodynamic chords for the wing-body grids. The far-field boundary is specified at 15 mean aerodynamic chords from the wing. It is located at 12 times the length of the fuselage from the wing-body configuration. The grids are not expected to be sufficiently fine to achieve a low numerical error in drag.

4.2 Numerical Study

4.2.1 Units for Comparing Efficiency

A typical convergence history is depicted in Figure D.5. The history records the computational work required to reduce the nonlinear residual to a certain tolerance.

The norm of the coupled residual, $||\mathcal{R}(\mathcal{Q})||_2$, including contributions from the mean-flow equations and the turbulence model, is plotted against the CPU time in terms of right-hand side (RHS) evaluations, or equivalent function evaluations (denoted as CPU-f.e.). The CPU time required for one RHS evaluation is used as a unit to measure computational work following the approach of Pueyo and Zingg [60]. It is the CPU time required to perform one calculation of the residual vector, $\mathcal{R}(\mathcal{Q})$, given a flow vector $\mathcal{Q}$. This includes the calculations of the inviscid, viscous, and source terms for every cell, including boundary conditions. The cost depends on the hardware and the grid, and is usually a linear function of the grid size. The unit provides a convenient measure to compare efficiency of algorithms on various platforms and grids. Note that the CPU time in terms of RHS evaluations (CPU-f.e.) is not a measure of the number of RHS evaluations. Rather it is the total CPU time divided by the CPU time required for one RHS evaluation.

4.2.2 Size of the Krylov Subspace

The size of the Krylov subspace as described in Section 3.2.2 is studied. The resulting solver is denoted as GMRES(m), where m is the size of the subspace. Figure D.5 depicts the convergence using different values of m. The study is performed for Case 7a, which is a short-hand notation for Case 7 on grid 7a. This notation is used for the rest of the document. A linear system tolerance of $\eta = 10^{-2}$ is used.

In the figure, GMRES(∞) denotes the case without specifying a maximum value of m. Thus every linear solve is converged to the tolerance η. The largest number of GMRES iterations is 70 for this case. On the other hand, when m is specified, the linear solver is terminated when the number of GMRES iterations exceeds m. As a result, the reduction of the linear residual may not satisfy the tolerance, but this can reduce computational time. Convergence to machine zero is not obtained using $m = 10$ for this case, where convergence stalls. For $m \geq 15$, the residual converges fully to machine zero, and the total time required to converge increases with m. Restarting slows down the convergence rate. Using m=15 with one restart is slower than using m=30. Thus using a larger Krylov subspace is more effective

Case	Size of Krylov subspace	o-it	$\sum$i-it	CPU-f.e.
7a	GMRES(15)	108	1,113	3,359
7a	GMRES(15), 1 restart	100	1,577	3,855
7a	GMRES(30)	94	1,375	3,438
7a	GMRES(50)	92	1,668	3,777
7a	GMRES(∞)	92	1,853	4,063
7b	GMRES(30)	132	2,211	5,164
7b	GMRES(50)	125	2,546	5,565
8a	GMRES(30)	143	2,552	6,086
8a	GMRES(50)	123	2,698	5,755

Table 4.3: Convergence statistics of a GMRES parametric study.

than restarting the GMRES algorithm, although restarting has a lower memory usage. Figures D.6 and D.7 depict the results for Cases 7b and 8a, respectively. The solution is not fully converged using m=15 for these cases.

The number of outer (o-it) and inner (i-it) iterations using various values of m is summarized in Table 4.3, for the cases that converge to machine zero. Outer iterations refer to the nonlinear iterations, and inner iterations refer to the linear (GMRES) iterations. Using a larger m or restarts increases the number of inner iterations, while a small value of m may affect the number of outer iterations. The use of m=15 is optimal for speed for Case 7a, while using m=50 is faster than using m=30 for Case 8a. The computational cost (CPU-f.e.) is not proportional to the number of inner iterations (i-it). It also depends on the number of outer iterations. This is because the cost of preconditioning every outer iteration is a major cost in three dimensions. Based on these results, GMRES(50) with no restart is selected for the rest of the study. It is noted that using a smaller value of m can be faster for some cases.

4.2.3 Linear Tolerance

The linear system tolerance η, as described in Section 3.2.3, is studied. Figure D.8 depicts the convergence using different values of η for Case 7b over a wing. The

variable η formula developed by Eisenstat and Walker [69], as given in (3.91) and (3.92), is also included in the study. GMRES(∞) is used with the variable η approach, while GMRES(50) is used otherwise.

The use of η=0.5 has the fastest convergence, followed by η=10^{-2}, η=10^{-1}, and η=10^{-3}. Convergence using η=10^{-1} and 10^{-2} is similar, but η=10^{-1} requires fewer inner iterations as shown in Table 4.4. The CPU time is not proportional to the number of inner iterations because the cost of preconditioning every outer iteration is a major contribution to the total cost in three dimensions. Thus the CPU time also depends on the number of outer iterations. On the other hand, the cost of preconditioning is less significant in two dimensions, where the CPU time is expected to be proportional to the number of inner iterations only. The variable η approach has a slower convergence in terms of CPU time. A value of η_{max}=0.01 is used. The parameter γ is selected to be 0.01 to demonstrate the effect of reducing η, which is reduced from 0.01 to 0.0005 in the study. The other parameters are the same as given in (3.91) and (3.92). The maximum number of GMRES iterations is 140 to solve the system to the required tolerance. The total number of inner iterations for η=10^{-3} and the variable η approach are similar, as given in Table 4.4. However, the latter approach has a slower convergence in terms of CPU time due to the use of a larger Krylov subspace. It is beneficial to impose a maximum number of inner iterations, as shown in the previous section.

Figure D.9 depicts a study of η for Case 8a over a wing-body configuration. For this case, η=10^{-2} has the fastest convergence, followed by η=10^{-3}, η=10^{-1}, and η=0.5. The effect of η seems to be case dependent, while η=10^{-2} performs fairly well for both cases. The number of outer and inner iterations using various values of η is summarized in Table 4.4. Using a smaller linear tolerance increases the number of inner iterations and reduces the number of outer iterations. A linear system tolerance of η=10^{-2} is selected for the rest of the study.

4.2.4 Preconditioning

Table 4.5 tabulates the results of the study of various preconditioners. The pre-

Case	η	o-it	$\sum$i-it	CPU-f.e.
7b	0.5	143	1,308	4,257
7b	10^{-1}	155	2,152	5,594
7b	10^{-2}	125	2,546	5,565
7b	10^{-3}	134	3,522	7,021
7b	variable	124	3,489	8,085
8a	0.5	370	2,088	9,270
8a	10^{-1}	204	2,126	6,386
8a	10^{-2}	123	2,698	5,755
8a	10^{-3}	121	3,138	6,284

Table 4.4: Convergence statistics of a linear system tolerance study.

conditioners are compared based on their effectiveness to reduce the linear residual by two orders of magnitude. The comparisons are performed for Case 7a using the linear system when the nonlinear residual is 10^{-4}.

The matrix $\widetilde{\mathcal{A}}$ is constructed as described in Section 3.2.5. The viscous term is calculated using a simple averaging approach as given in (3.11). Matrices with and without next-to-nearest neighboring terms are considered. The matrix neglecting these terms is denoted as a distance-1 (D1) matrix. This approach is used unless otherwise stated. The matrix including these terms is denoted as a distance-2 (D2) matrix.

There are three considerations when choosing a preconditioner. They are effectiveness, cost, and storage. Effectiveness is evaluated by the number of GMRES iterations to solve the system. With a more effective preconditioner, fewer iterations are required. Cost of preconditioning includes formation of the matrix $\widetilde{\mathcal{A}}$, factorization to obtain the factors $\mathcal{L}$ and $\mathcal{U}$, and applying the preconditioner with the operations $\mathcal{U}^{-1}$ and $\mathcal{L}^{-1}$, which are triangular back-solves. Storage is required for the matrix $\widetilde{\mathcal{A}}$ and the factors, which can be significant, especially in three dimensions. In particular, storage of the factors can be done in-place, reusing memory of $\widetilde{\mathcal{A}}$, thus saving memory usage. This is not done in the current work due to implementation of the numerical software.*

*The in-place option in PETSc only supports ILU(0).

Preconditioner	nnzB/N	CPU-form	i-it	CPU-total
ILU(0)	10.9	10	33	236
ILU(1)	21.8	29	24	202
ILU(2)	41.4	96	12	197
ILU(3)	69.7	269	10	375
ILU(4)	106.8	638	9	759
D2-ILU(0)	45.4	139	16	350
D2-ILU(1)	134.0	1,124	10	1,362
D2-ILU(2)	294.5	5,537	8	5,856
ILUT(20,10^{-3})	14.2	548	42	918
ILUT(80,10^{-3})	25.1	1,176	21	1,395
ILUT(160,10^{-3})	36.0	2,065	15	2,262
ILUT(20,10^{-5})	15.3	2,802	42	3,199
ILUT(80,10^{-5})	31.6	7,365	20	7,616
ILUT(160,10^{-5})	50.1	14,971	14	15,201

Table 4.5: Memory, CPU cost and effectiveness to reduce the inner residual by two orders of magnitude for different preconditioners. CPU time units are in seconds.

The level-of-fill approach ILU(p) is studied with fill parameters from 0 to 4. The block version of ILU(p) is used, where block inversions are implemented with unrolling [73]. The preconditioner becomes more effective for larger p. For this case, the number of GMRES iterations, denoted as i-it in Table 4.5, is reduced from 33 to 9 when p is increased from 0 to 4. The computational work to factorize the matrix, denoted as CPU-form in the table, increases nonlinearly with the fill parameter. The total cost, denoted as CPU-total, includes factorization cost and the cost of the triangular back-solves. The former cost increases with the fill parameter, while the latter cost increases with the number of GMRES iterations. Therefore an optimal fill parameter can be found, in this case p=2. The cost of constructing the matrix $\widetilde{\mathcal{A}}$ is excluded in the table, since it is unaffected by p. The storage of the factors, denoted by nnzB/N, also increases nonlinearly with the fill parameter. In general, memory usage is unpredictable for the level-of-fill approach. The value nnzB/N is the average number of nonzero blocks per block row in the factors. For ILU(0), the number is the same as that in the matrix $\widetilde{\mathcal{A}}$. The current case is a three-dimensional case using a one-equation turbulence model, so the block size is six. The number of nonzero blocks

Grid	nnz Dist-1	nnz Dist-2
2D triangular	7	19
3D tetrahedral	14	55

Table 4.6: A summary of dimensionality and grid type on the sparsity of the matrix used for preconditioning [1].

per row in $\widetilde{\mathcal{A}}$ is 10.9 for the current case on a hybrid grid, which is between the value of 14[†] on a tetrahedral grid and 7 on a regular hexahedral grid. In two dimensions, the number is 7 on a regular triangular grid and is 5 on a regular quadrilateral grid. The values are discussed in the work of Barth and Linton [1] and are summarized in Table 4.6. Considering both costs and memory usage, the ILU(1) preconditioner is a better choice than ILU(2), although the latter is optimal for speed.

D2-ILU denotes preconditioning with a distance-2 matrix with next-to-nearest neighboring terms. D2-ILU(p) with p=0, 1, and 2 are studied. D2-ILU(0) has the same storage as the distance-2 matrix. In the current case, the matrix has 45.4 nonzero blocks per row, which is less than the value of 55 on a pure tetrahedral grid, as given in Table 4.6. The distance-2 matrix has roughly four times more nonzero blocks than the distance-1 matrix. The D2-ILU(p) preconditioner has a large increase in memory usage and factorization costs when p is increased. For a similar level of effectiveness, D2-ILU(p) preconditioners are more expensive and require more memory usage than the ILU(p) approach. An example is the comparison between ILU(3) and D2-ILU(1) for the current case. The distance-2 approach is not as detrimental in two dimensions.

The threshold approach ILUT(P,τ) is also studied. Values of P=20, 80, and 160, and τ values of 10^{-3} and 10^{-5} are used. A scalar version of ILUT is used. For example, P=20 allows an additional 20 scalar elements to be added in each row of $\mathcal{L}$ and $\mathcal{U}$ respectively. For the current case, the block size is six, so an additional $20 \div 6 \simeq 3.3$ nonzero blocks are allowed to fill in each row in each of the factors $\mathcal{L}$ and $\mathcal{U}$. Therefore, the storage per row for P=20 will have a maximum of 10.9 + 3.3

[†]The exact number is 13 on a regular tetrahedral grid. The number can be higher for general tetrahedral grids.

+ 3.3 = 17.5 nonzero blocks per row. For comparison, the storage of ILUT(20,10^{-3}) is 14.2 from the table, and that for ILUT(20,10^{-5}) is 15.3. Similarly, the maximum allowed storage per row for P=80 and P=160 are 37.5 and 64.2 nonzero blocks per row respectively. Unlike the level-of-fill approach, memory usage of the threshold approach is predictable. The factorization cost of the threshold approach is much higher than the level-of-fill approach for the same level of effectiveness, but with a similar memory usage. An example is the comparison of ILUT(80,10^{-3}) and ILU(1) for the current case. There are three possible reasons why the threshold approach is more expensive: 1. More entries are computed since they cannot be dropped by their locations in the factors. 2. To determine the dropping entries based on a threshold incurs extra cost. 3. There are more factorization operations using a scalar version of factorization.

The ILUT preconditioner is more effective with a higher P. The use of a smaller drop tolerance of 10^{-5} over 10^{-3} is only slightly more effective at a much higher factorization cost. The level-of-fill approach ILU(p) with p=1 is used for the rest of the study.

4.2.5 Approximate Jacobian

The parameter σ in (3.107) is studied. The parameter controls the amount of second-difference dissipation in the matrix $\widetilde{\mathcal{A}}$ and hence the diagonal dominance of the matrix. On the other hand, the larger the value of σ, the larger the difference between $\widetilde{\mathcal{A}}$ and the complete Jacobian $\mathcal{A}$. Hence too large a value of σ can affect the performance of the preconditioner. The study is depicted in Figure D.10 for Cases 7a and 7b over a wing, and Case 8a over a wing-body configuration. Matrix dissipation is used in the study, where parameters of V_l=V_n=0.25 are used in the matrix $\widetilde{\mathcal{A}}$ and on the right-hand side. The total number of GMRES iterations to converge the mean-flow residual to 10^{-10} using different values of σ from 2 to 14 is plotted. Convergence to machine zero may not be obtainable using σ<2. This is probably due to insufficient diagonal dominance of the matrix. The optimal values of σ are 10, 8, and 12, respectively, for Cases 7a, 7b, and 8a. A value of σ=10 for

matrix dissipation is selected for the rest of the study.

4.2.6 Viscous Preconditioning

Various viscous formulations are studied to construct the matrix $\widetilde{\mathcal{A}}$ for preconditioning. This includes the simple-averaging, approximate-difference, diamond-path, and primal-grid approaches, as described in Section 3.1. Except for the approximate-difference approach, these approaches have a non-compact stencil involving next-to-nearest neighboring terms. These terms are dropped in the matrix $\widetilde{\mathcal{A}}$, as described in Section 3.2.5. This is referred to as a distance-1 (D1) approach. On the other hand, distance-2 preconditioning leads to prohibitive cost of factorization and memory usage in three dimensions, as pointed out in Section 4.2.4.

Figure D.11 depicts the convergence histories using various viscous formulations in $\widetilde{\mathcal{A}}$ for preconditioning. The study is performed for Case 7a. The simple-averaging approach is used on the right-hand side. Thus these cases converge to the same solution. The simple-averaging-D1, approximate-difference, diamond-path-D1, and primal-grid-D1 approaches have similar convergence. These approaches have a faster convergence than the diamond-path and the primal-grid approaches when next-to-nearest neighbors are included. The simple-averaging approach with next-to-nearest neighbors is not included in the study because it is too expensive. The cost and storage for this approach can be estimated from Table 4.5.

The number of outer and inner iterations is summarized in Table 4.7. The cost of preconditioning can be measured in terms of the cost per inner iteration. This is denoted as CPU-f.e./$\sum$i-it in the table, where CPU-f.e. denotes equivalent function evaluations. The D1 preconditioners (simple-averaging-D1, diamond-path-D1, primal-grid-D1) and the approximate-difference approach have a cost of roughly 2.4 units, while the diamond-path and the primal-grid approaches have a higher cost of 3.7 and 3.5 units, respectively. These two approaches have more nonzero elements in the matrix $\widetilde{\mathcal{A}}$. They have 18.4 nonzero blocks per row for the current case. On the other hand, the D1 preconditioners and the approximate-difference approach have 10.9 nonzero blocks per row, as given in Section 4.2.4. On a regular quadrilateral

Preconditioner	RHS	o-it	$\sum$i-it	CPU-f.e.	CPU-f.e./$\sum$i-it
Simple Avg. D1	Simple Avg.	92	1,668	3,777	2.3
Approx Diff.	Simple Avg.	92	1,590	3,591	2.3
Diamond Path D1	Simple Avg.	92	1,592	3,845	2.4
Primal Grid D1	Simple Avg.	95	1,597	3,851	2.4
Diamond Path	Simple Avg.	92	1,504	5,495	3.7
Primal Grid	Simple Avg.	97	1,704	6,017	3.5
Approx Diff.	Approx Diff.	92	1,593	3,884	2.4
Diamond Path D1	Diamond Path	92	1,588	5,285	3.3
Primal Grid D1	Primal Grid	92	1,582	5,178	3.3

Table 4.7: Convergence statistics using different viscous calculations. CPU-f.e. based on the simple averaging approach.

grid, the number is 9 for the diamond-path and the primal-grid approaches, and is 5 for the D1 preconditioners and the approximate-difference approach. It can be concluded from Figure D.11 that the D1 preconditioners and the approximate-difference approach have a lower cost.

The above results are computed using the various viscous formulations only in the matrix $\widetilde{\mathcal{A}}$ for preconditioning. Next we consider using these formulations on the right-hand side as well. This affects both accuracy and convergence. The next-to-nearest neighboring terms are kept on the right-hand side. These terms are dropped in the matrix $\widetilde{\mathcal{A}}$ using a D1 approach. The results are depicted in Figure D.12. The simple-averaging and the approximate-difference approaches have a faster convergence than the diamond-path and the primal-grid approaches. This is due to more expensive right-hand side calculations using the latter two approaches. The simple-averaging right-hand side formulation as given in (3.11) is inexpensive since the cell gradients $\nabla\mathcal{Q}_i$ are pre-calculated. These terms are required to compute the turbulent source terms $\mathcal{P}_i$.

The computed lift and drag coefficients are tabulated in Table 4.8 for the various viscous right-hand side formulations. For this particular case and mesh, all four approaches give very similar force coefficients. In principle, the diamond-path and the primal-grid approaches are more accurate than the simple-averaging approach. This is not indicated in the table and would require further grid refinement studies to be

RHS	C_L	C_D
Simple Avg.	0.2640	0.01466
Approx Diff.	0.2649	0.01462
Diamond Path	0.2654	0.01466
Primal Grid	0.2648	0.01461

Table 4.8: Lift and drag coefficients using different viscous calculations on the right-hand side.

verified. On the other hand, the approximate-difference approach can be inaccurate on stretched triangular grids. For the remaining studies, the simple-averaging approach is used on the right-hand side. The simple averaging-D1 approach is used in $\widetilde{\mathcal{A}}$ for preconditioning.

4.2.7 Freezing of the Preconditioner

Freezing of the preconditioner can be used to alleviate the high cost of forming the preconditioner. This option is studied for Case 7a. The results are depicted in Figure D.13. The preconditioner is updated every iteration until the mean-flow residual is reduced to 10^{-4}. At this point, the preconditioner is frozen and is reused for the subsequent iterations. This produces a slower convergence as shown in the figure. Freezing earlier is not recommended because the changes in the preconditioner can be large when the solution is far from convergence. Freezing after the residual is reduced to 10^{-8} has a slightly slower convergence than computing the preconditioner every outer iteration. On the other hand, the preconditioner can be frozen for a certain number of iterations and recalculated regularly. Updating the preconditioner every 3, 5, and 10 iterations, after the residual is reduced to 10^{-4}, is found to produce a faster convergence than updating the preconditioner every iteration. In particular, updating every 5 iterations is optimal for speed for the current case. This may be correlated to the fact that the time step Δt_{ref} in (3.111) is modified every 5 iterations using a parameter of f=5.

The number of outer and inner iterations is summarized in Table 4.9. Freezing the preconditioner increases the number of inner iterations, but the total CPU time

Preconditioner	o-it	$\sum$i-it	CPU-f.e.
not freezing	92	1,668	3,777
freezing 3	94	1,737	3,567
freezing 5	93	1,737	3,483
freezing 10	95	1,829	3,541

Table 4.9: Convergence statistics of a freezing preconditioner study.

can be reduced. Since the benefit is small, freezing of the preconditioner is not used for the rest of the study.

4.2.8 Scaling of the Linear System

Scaling of the equations is found to improve convergence of a matrix-free inexact-Newton method when a turbulence model is used [110]. Row scaling is motivated by the observation that the mean-flow and the turbulence equations are of different magnitudes. When the linear system is solved inexactly, it is possible that only the turbulence model residual is reduced, since it is usually larger in magnitude, while the mean-flow residual may increase. This affects convergence and robustness. The objective of row scaling is to obtain a set of mean-flow and turbulence equations of similar magnitudes so that they are equally solved by an inexact linear solver.

Column scaling, on the other hand, is motivated by the observation that the mean-flow and the turbulent variables in $\mathcal{Q}$ are of different magnitudes. The mean-flow quantities have an order of magnitude around one, while the turbulence variable can be as large as 10^3. This may affect the accuracy of the matrix-free approach, as explained by Chisholm [110]. The accuracy of the matrix-free approach, as given by (3.93), depends on the choice of the step size ϵ. The step size can be computed using (3.94) or (3.95), which are functions of v. For the first GMRES iteration, v=$\mathcal{M}^{-1}r_0/||r_0||_2$, where r_0 is the initial linear residual. Since $\mathcal{M}\simeq\mathcal{A}$ and r_0=-$\mathcal{R}$, it can be argued that there is a similar difference in magnitude as in $\mathcal{Q}$ between the elements of v that correspond to the mean-flow variables and the elements of v that correspond to the turbulence variables. If ϵ is chosen using (3.94) or (3.95), then

its value will be controlled by the elements of v that correspond to the turbulence variables, since they are larger in magnitude. The resulting ϵ will be smaller than that obtained when the elements of v are of similar magnitude (i.e. when v is well-scaled). Small values of ϵ can introduce round-off errors in the matrix-free formula, which affects convergence of Newton's method. The objective of column scaling is to reduce these round-off errors by scaling the variables so that the elements of $\mathcal{Q}$ (and also v) are of similar magnitudes. The scaled linear system is written as:

$$\widehat{\mathcal{A}}\widehat{x} = \widehat{b} \tag{4.1}$$

where $\widehat{\ }$ denotes scaled quantities, with:

$$\widehat{\mathcal{A}} = \mathcal{D}_r \mathcal{A} \mathcal{D}_c^{-1}, \quad \widehat{x} = \mathcal{D}_c x, \quad \widehat{b} = \mathcal{D}_r b \tag{4.2}$$

where $\mathcal{D}_r$, $\mathcal{D}_c$ are $N \times N$ block diagonal matrices containing scaling factors. They are given by:

$$\begin{aligned} \mathcal{D}_r &= \text{diag}\,(d_r,\ d_r,\ \ldots,\ d_r) \\ \mathcal{D}_c &= \text{diag}\,(d_c,\ d_c,\ \ldots,\ d_c) \end{aligned} \tag{4.3}$$

where d_r, d_c are 6×6 diagonal blocks when using a one-equation turbulence model in three dimensions. The blocks are given by:

$$\begin{aligned} d_r &= \text{diag}\,(1,\ 1,\ 1,\ 1,\ 1,\ s_r) \\ d_c &= \text{diag}\,(1,\ 1,\ 1,\ 1,\ 1,\ s_c) \end{aligned} \tag{4.4}$$

where s_r and s_c are parameters which scale the turbulence equation and the turbulence variables, respectively. The parameter s_r controls row scaling, and s_c controls column scaling. Scaling is applied globally; the same scale is used for every node. A value of s_c=10^{-3} can be selected to scale the turbulence variables, $\tilde{\nu}$, from 10^3 to 1, so that they have similar magnitudes as the mean-flow variables. Different values of s_r can be selected to improve convergence of the algorithm. Optimal value of s_r should balance the magnitudes of the mean-flow and the turbulence equations so that they are equally solved by an inexact linear solver.

Row scaling s_r	o-it	$\sum$i-it	CPU-f.e.
10^{-1}	159	3,685	7,725
10^{-2}	129	2,731	6,172
10^{-3}	125	2,546	5,565
10^{-4}	160	3,482	7,558

Table 4.10: Convergence statistics of a row scaling study for Case 7b. Column scaling: s_c=10^{-3}.

Figure D.14 depicts a study of row scaling using different values of s_r for Case 7b. The number of outer and inner iterations is summarized in Table 4.10. A column scaling of s_c=10^{-3} is used in the study. The use of s_r=10^{-3} has the fastest convergence, followed by s_r=10^{-2}, 10^{-4}, and 10^{-1}. The solution fails to converge for s_r=1. In Figure D.14, it is observable that the use of a larger s_r (e.g. 0.1) has a better initial convergence rate. This is because the initial residual is dominated by the turbulence equation. Using a larger scale s_r increases the magnitude of this equation, which causes the linear solver to focus on solving it, resulting in a faster initial convergence rate. On the other hand, a good overall convergence will require a good balance between solving both the mean-flow and the turbulence equations. It is noted that the time step is unaffected by row scaling in the current study. It is because the time step is calculated after row scaling is applied. Moreover, the time step sequence given by (3.111) is not computed based on the residual. In addition, the equations are unscaled before plotting so that convergence histories using different values of s_r are comparable.

Row and column scalings are studied for Case 7b over a wing and Case 8a over a wing-body configuration. Both cases have roughly a half million nodes. In this study, no scaling refers to s_r=s_c=1, while scaling refers to s_r=s_c=10^{-3}. Figure D.15 depicts the result for Case 7b using η=0.5, 10^{-1}, and 10^{-2}. Scaling improves convergence using the three values of η in terms of CPU time. Note that the ratio of CPU-f.e./$\sum$i-it is not roughly constant. The total cost also depends on the number of outer iterations, since the cost of forming the preconditioner is a major cost in three dimensions. The optimal value of η is increased from 0.1 to 0.5 with the use of scaling.

Case	η	Scaling	o-it	$\sum$i-it	CPU-f.e.
7b	0.5	no scaling	262	1,364	6,419
7b	10^{-1}	no scaling	243	1,463	6,197
7b	10^{-2}	no scaling	210	2,337	6,755
7b	0.5	scaling	143	1,308	4,257
7b	10^{-1}	scaling	155	2,152	5,594
7b	10^{-2}	scaling	125	2,546	5,565
8a	10^{-2}	no scaling	557	3,627	14,675
8a	10^{-3}	no scaling	146	2,981	6,507
8a	10^{-2}	scaling	123	2,698	5,755
8a	10^{-3}	scaling	121	3,138	6,284

Table 4.11: Convergence statistics of a linear system scaling study. No scaling: s_r=s_c=1. Scaling: s_r=s_c=10^{-3}.

As a note, a value of σ=15 is used for η=0.5 and 10^{-1} when scaling is not used due to convergence problems, while the default value of σ=10 is used otherwise. Figure D.16 depicts the result for Case 8a using η=10^{-2} and 10^{-3}. Scaling improves convergence for both cases, but the improvement is larger for η=10^{-2}. Using η=10^{-3} is faster than using η=10^{-2} when scaling is not used, while using η=10^{-2} is faster than using η=10^{-3} when scaling is used.

The number of outer and inner iterations is summarized in Table 4.11. Scaling reduces the number of outer iterations. The number of inner iterations may increase but the overall convergence is faster in terms of CPU time, due to fewer outer iterations. The cost reduction due to fewer outer iterations can be larger than the cost increase caused by more inner iterations. Based on these results, scaling with s_r=s_c=10^{-3} is used for the rest of the study.

4.2.9 Reordering of Unknowns

Reordering of the unknowns in the matrix $\widetilde{\mathcal{A}}$ affects the performance of ILU preconditioning. Starting from the preconditioning operation, $\mathcal{M}z_j$=v_j in (3.102), reordering can be written as:

$$\mathcal{M}' z'_j = v'_j \tag{4.5}$$

where $'$ denotes reordered quantities, with:

$$z'_j = \mathcal{P} z_j, \quad v'_j = \mathcal{P} v_j \tag{4.6}$$

where $\mathcal{P}$ is a permutation matrix. The permutation matrix contains the ordering information for various reordering approaches. The preconditioner is given by:

$$\mathcal{M}' = \mathcal{L}'\mathcal{U}' \tag{4.7}$$

and the factors are given by:

$$\mathcal{L}'\mathcal{U}' \simeq \widetilde{\mathcal{A}}' = \mathcal{P}\widetilde{\mathcal{A}}\mathcal{P}^T \tag{4.8}$$

where the reordered matrix $\widetilde{\mathcal{A}}'$ is used in the factorization. Reordering is applied so that the bandwidth of $\widetilde{\mathcal{A}}'$ is smaller than the bandwidth of $\widetilde{\mathcal{A}}$.

Four reordering approaches available in the numerical software PETSc [102] are studied, namely Nested Dissection [111], One-way Dissection [112], Reverse Cuthill-McKee (RCM) [81], and Quotient Minimum Degree [113]. The reordering approaches are based on an initial ordering, which can affect convergence, as shown by Chisholm and Zingg [88]. There is no natural ordering for unstructured grids in general. The initial ordering comes from the grid generator.

The results are depicted in Figure D.17. Among the various approaches, the Reverse Cuthill-McKee reordering has the fastest convergence, followed by One-way Dissection and the initial ordering. Nested Dissection and Quotient Minimum Degree reorderings are found to produce a slower convergence than the initial ordering of the unknowns. The number of outer and inner iterations is summarized in Table 4.12. With a good reordering, the number of inner iterations can be reduced to improve convergence in terms of CPU time. The Reverse Cuthill-McKee reordering is used for the rest of the study.

Reordering	o-it	$\sum$i-it	CPU-f.e.
no reordering	91	1,892	4,596
Nested Dissection	97	2,577	6,774
One-way Dissection	92	1,664	4,579
Reverse Cuthill-McKee	92	1,668	3,777
Quotient Minimum Degree	92	2,263	6,489

Table 4.12: Convergence statistics of a reordering study.

4.3 Results

4.3.1 Solver Parameters

A summary of the solver parameters is given in this section. Consistent with the original objective of determining a set of solver parameters that works for a range of cases and flow conditions, these parameters are used unless otherwise stated. Matrix dissipation is used with κ_2=2, κ_4=0.1, and V_l=V_n=0.25. A non-restarted version of GMRES with 50 Krylov vectors is used as a linear solver. An inexact-Newton approach with η=10^{-1} is used in two dimensions, and η=10^{-2} is used in three dimensions. Scaling is applied to the linear system: s_r=s_c=10^{-3}. It only affects convergence for turbulent cases. Preconditioning is obtained using ILU(4) in two dimensions and ILU(1) in three dimensions. The incomplete factorization is based on an approximate Jacobian matrix with σ=10. The matrix dissipation parameters in the approximate Jacobian are the same as those on the right-hand side. The simple-averaging approach is used to calculate the viscous term on the right-hand side. The simple-averaging-D1 approach is used in the matrix for preconditioning. The Reverse Cuthill-McKee approach is used to reorder the matrix.

The time step sequence in (3.111) is used with n_1=2 and Δt_1=0.05. Values of α=2, β=2, and f=3 are used in two dimensions, while α=2, β=2, and f=5 are used in three dimensions. A first-order startup stage is included before switching to the matrix-dissipation stage in three dimensions. The same time step sequence is used in both stages. The first-order scheme is defined with $\varepsilon^{(2)}$=1/4, $\varepsilon^{(4)}$=0, and $\varepsilon_p^{(2)}$=$\varepsilon^{(2)}$. When the mean-flow residual is reduced to 10^{-4}, the algorithm switches to the matrix-

dissipation stage. The startup stage is unused in two dimensions. A nonzero initial solution of $\tilde{\nu} = 10\nu_\infty$ is used for the turbulence model, as suggested by Chisholm and Zingg [88].

4.3.2 Convergence Results

Convergence histories for two-dimensional cases are depicted in Figure D.18. Except for Cases 2, 4c and 5, convergence to 10^{-10} can be obtained in under 4,000 RHS evaluations. This is comparable to the results obtained by Geuzaine [78]. The solver parameters as summarized in the previous section are used except for Cases 2 and 4c. Case 2 is a laminar flow, which is solved using a D2-ILU(4) preconditioner with σ=15. The use of a distance-1 preconditioner is found to produce a much slower convergence for this case. Case 4c is a turbulent flow on a triangular grid with stretched cells in the viscous region. The case is solved using a D2-ILU(4) preconditioner, resulting in a slow convergence. The solution for this case would not converge during startup using a distance-1 preconditioner.

The convergence statistics are summarized in Table 4.13. All two-dimensional cases are converged to 10^{-10} in less than 100 outer iterations. The CPU time reported in the table is based on either a single 1 GHz alpha EV68 processor, or a single 1.5 GHz Intel Itanium 2 processor. The former is used in two dimensions while the latter is used in three dimensions. The ratio of CPU-f.e. to $\sum$i-it is much larger for Cases 2 and 4c. This is due to the use of a distance-2 preconditioner. Pressure coefficients for Case 4 are depicted in Figure D.20 on three different grids. The solution on a triangular grid is different from those obtained on the quadrilateral and hybrid grids. The latter two grids produce similar solutions. Pressure coefficients for Case 5 over a multi-element airfoil are depicted in Figure D.21. The solution agrees well with experimental results and with those obtained by Lassaline [100].

Convergence for an inviscid flow over a wing (Case 6) is depicted in Figure D.22. Except for Case 6c on a tetrahedral grid which has roughly one million nodes, all the other cases are converged in fewer than 6,000 RHS evaluations. Case 6c is converged using a different version of the code, which is referred to as the 64-bit code. The

Case	o-it	$\sum$i-it	$\sum$i-it/o-it	CPU-f.e.	CPU time
1	39	352	9.0	1,281	1.48 mins
2	38	274	7.2	5,008	11.5 mins
3a	40	526	13.2	1,944	4.5 mins
3b	47	1,033	22.0	2,822	10.0 mins
4a	94	954	10.2	3,634	6.5 mins
4b	76	794	10.5	2,951	5.3 mins
4c	84	819	9.8	11,846	29.4 mins
5	91	2,598	28.6	6,983	65.8 mins
6a	122	1,703	14.0	4,261	4.9 hours
6b	141	2,273	16.1	5,399	13.0 hours
6c†	169	3,061	18.1	12,993	53.2 hours
6d	82	1,682	20.5	4,293	2.0 hours
6e	83	1,742	21.0	4,639	4.5 hours
6f	86	2,056	23.9	5,372	9.9 hours
7a	92	1,668	18.1	3,777	5.0 hours
7b	125	2,546	20.4	5,565	20.9 hours
7c†	187	4,511	24.1	15,542	130.6 hours
8a	154	4,348	28.2	8,568	38.1 hours
8b†*	99	2,068	20.9	7,812	72.4 hours
8c†	298	9,925	33.3	35,156	360.5 hours
8d*	340	9,779	28.8	18,227	38.1 hours

Table 4.13: Statistics of the Newton-Krylov algorithm. (†) Using 64-bit code. (*) Not fully converged. Note: for cases that are not fully converged, results are shown for convergence to 10^{-6}.

details are given in Appendix E. There is a conversion factor to be considered when comparing between the two versions of the code. The convergence statistics are given in Table 4.13. The three cases on tetrahedral grids 6a to 6c require more outer iterations than the three cases 6d to 6f on hexahedral grids. The number of inner iterations increases when the grid is refined. The scalability of the algorithm with respect to grid size can be studied following the approach of Pueyo [103]. We can write:

$$\text{CPU-f.e.} = bN^a \tag{4.9}$$

where CPU-f.e. is the CPU time in terms of RHS evaluations, and N is the grid size. The coefficient b and the exponent a can be determined from the convergence data

Convergence criterion	CPU-f.e.
0.5% of C_L	3,950
0.1% of C_L	4,270
0.01% of C_L	4,588
0.5% of C_D	3,906
0.1% of C_D	4,230
0.01% of C_D	4,571
Fully converged	5,565

Table 4.14: Convergence data for the lift and drag coefficients.

using a least-squares linear fit of the logarithmic variables of CPU-f.e. and N. This is depicted in Figure D.26. For the ideal case of linear scalability, we have a=0, thus CPU-f.e. is constant. For cases 6a to 6c on tetrahedral grids, we obtain a value of a=0.34, showing that the scalability is nonlinear. For cases 6d to 6f on hexahedral grids, we obtain a value of a=0.16, showing that the scalability is close to linear. As a comparison, Pueyo [103] obtained a value of a=0.325 for a range of cases using his Newton-Krylov algorithm. The scalability of the algorithm with respect to grid size seems to depend on the grid type. The use of hexahedral grids leads to better scalability for the current case. The conversion factor in Appendix E is taken into account in the analysis.

Convergence for a viscous flow over a wing (Case 7) is depicted in Figure D.24. Convergence on grids with increasing densities is obtained in 3,777, 5,565 and 15,542 RHS evaluations respectively. The 64-bit code is used for Case 7c. The convergence statistics are given in Table 4.13. The number of outer and inner iterations increases when the grid is refined. Case 7 has a scalability exponent of a=0.43.

Convergence of lift and drag coefficients for Case 7b is given in Figure D.25. The force coefficients converge asymptotically to their final values as the residual converges to zero. The time required to converge the force coefficients to some specified tolerances is summarized in Table 4.14. It requires about 4,000 RHS evaluations, which is about three quarters of the total time, to converge to within 0.5 percent of the converged lift and drag coefficients, which are 0.262 and 0.0146 respectively.

Figure D.27 depicts the pressure contours over the wing on grid 7c. The pressure coefficients at different wingspan locations are compared to experimental data in Figure D.28. Reasonable agreement is obtained, but the solution is not grid converged. The solution on grid 7c has a stronger shock wave than the solution on grid 7b.

Convergence for a viscous flow over a wing-body configuration (Case 8) is depicted in Figure D.29. Cases 8a and 8b are converged using the solver parameters as given in the previous section. Cases 8c and 8d are converged using a modified startup procedure as summarized in Appendix F. Both Cases 8a and 8c are converged below 10^{-10} in the study, while Case 8a requires about 8,000 RHS evaluations to converge. Residual convergence to 10^{-10} is not obtained for Cases 8b and 8d, where convergence stalls. This may be due to unsteady effects caused by flow separation, which are discussed in the Drag Prediction Workshop [6, 29]. Nevertheless, the lift and drag coefficients are converged, as depicted in Figures D.30 and D.31. The experimental results are C_L=0.5 and C_D=0.0295 as given in [6]. The figures are plotted with a range of $\pm$ 50 percent of the experimental values. Figure D.32 depicts the pressure contours over the wing-body configuration on grid 8a. The pressure coefficients at different wingspan locations are compared to experimental data in Figure D.33.

Chapter 5

Conclusions, Contributions and Recommendations

5.1 Conclusions

A Newton-Krylov algorithm has been developed for solving the compressible Navier-Stokes equations on hybrid unstructured grids. Turbulence is modeled using the one-equation Spalart-Allmaras model, which is solved in a fully coupled form with the mean-flow equations. The spatial discretization is based on a finite-volume matrix dissipation scheme. The discrete steady-state equations are solved using the Newton-Krylov method, where the linear system is solved using a preconditioned matrix-free GMRES approach. An incomplete lower-upper factorization based on an approximate Jacobian is used as the preconditioner after applying the reverse Cuthill-McKee reordering.

Various aspects of the algorithm are studied for improvements in efficiency and reliability. The following can be concluded from the studies:

1. Specifying a maximum Krylov subspace size can reduce computational cost.

2. A linear system tolerance can be selected to avoid over-solving of the linear system to reduce computational cost. The choice of the tolerance should balance between the number of outer and inner iterations. Controlling the number

of outer iterations is important for three-dimensional cases, since the cost of preconditioning every outer iteration is a major contribution to the total cost.

3. Distance-2 ILU preconditioners are expensive in three dimensions due to the increased number of non-zeros in the matrix for preconditioning. Distance-1 preconditioners can be used to reduce cost.

4. The optimal fill parameter for the ILU(p) approach is lower in three dimensions than in two. This is due to the larger number of non-zeros in the preconditioning matrix, which causes high-fill ILU(p) to be more expensive than in two dimensions. The ILU(1) preconditioner is found to be relatively efficient with reasonable effectiveness and memory usage.

5. Four viscous formulations were studied in the preconditioning matrix and on the right-hand side. Approximate compact formulations are considered for viscous preconditioning to reduce cost. Distance-1 preconditioners using four viscous formulations are found to be feasible alternatives and have similar computational costs.

6. Scaling of the equations as suggested by Chisholm [110] is found to improve convergence. Row scaling ensures both the turbulence and the mean-flow residuals are reduced in an inexact linear solve, while column scaling improves accuracy of the matrix-free Jacobian, which improves convergence of Newton's method.

Based on the numerical studies, a set of solver parameters and a startup strategy are selected as summarized in Section 4.3.1. The algorithm and the selected parameters are successfully applied to solve a range of two- and three-dimensional inviscid and turbulent cases on unstructured grids. Residual convergence to 10^{-10} for a number of two-dimensional cases can be obtained in an equivalent CPU time of 4,000 RHS evaluations. This is comparable to the results obtained by Geuzaine [78]. The algorithm is also applied to turbulent flows over the ONERA M6 wing and the DLR F6 wing-body configuration on grids with up to one million nodes. Residual convergence to 10^{-10} for the wing and the wing-body configuration can be obtained in 5,500 and 8,000 RHS evaluations, respectively, on grids with a half million nodes.

5.2 Contributions

1. Development of a Newton-Krylov algorithm for the compressible Navier-Stokes equations on hybrid unstructured grids using the Spalart-Allmaras turbulence model.

2. A study of various approaches to solve the linear system and construct the preconditioner, including a study of approximate viscous formulations for preconditioning. A set of solver parameters and a startup strategy are selected based on the numerical study.

3. Evaluation of the performance of the algorithm.

5.3 Recommendations

Currently, preconditioning is a major cost of the algorithm. Further research is recommended on preconditioning to reduce computational time especially with finer grids. Approximations can be introduced to the preconditioner. For example, the research of Luo et al. [76] using the lower-upper symmetric Gauss Seidel approach can successfully reduce cost and memory usage, but with a trade off in convergence rate when compared to the ILU approach. The challenge here is to reduce computational cost, while maintaining the effectiveness of the ILU approach.

Domain decomposition methods may be a potential area of research, by separating the computing domain into smaller sub-domains, where ILU can be computed more efficiently. Furthermore, these methods can be implemented on parallel computers to further reduce computational time. Schwarz preconditioners have been studied with success [1, 114, 115]. On the other hand, Schur complement methods may be more promising when many processors are used. Pioneering work on Schur complement preconditioners can be found in the works by Barth et al. [116] and Saad and Sosonkina [117].

References

[1] T. J. Barth and S. W. Linton, "An Unstructured Mesh Newton Solver for Compressible Fluid Flow and its Parallel Implementation," *AIAA Paper 95-0221*, 1995.

[2] A. Jameson, "Computational Aerodynamics for Aircraft Design," *Science*, vol. 245, pp. 361–371, 1989.

[3] B. N. Nield, "An Overview of the Boeing 777 High Lift Aerodynamic Design," *Aeronautical Journal*, pp. 361–371, 1995.

[4] T. E. Nelson and D. W. Zingg, "Fifty Years of Aerodynamics: Successes, Challenges, and Opportunities," *CAS Journal*, vol. 50, no. 1, pp. 61–84, 2004.

[5] F. T. Johnson, E. N. Tinoco, and N. J. Yu, "Thirty Years of Development and Application of CFD at Boeing Commercial Airplanes, Seattle," *AIAA Paper 2003-3439*, 2003.

[6] E. M. Lee-Rausch, N. T. Frink, D. J. Mavriplis, R. D. Rausch, and W. E. Milholen, "Transonic Drag Prediction on a DLR-F6 Transport Configuration Using Unstructured Grid Solvers," *AIAA Paper 2004-0554*, 2004.

[7] D. A. Knoll and D. E. Keyes, "Jacobian-Free Newton-Krylov Methods: A Survey of Approaches and Applications," *Journal of Computational Physics*, vol. 193, pp. 357–397, 2004.

[8] T. J. Barth, "A 3-D Upwind Euler Solver for Unstructured Meshes," *AIAA Paper 91-1548*, 1991.

[9] N. T. Frink, P. Parikh, and S. Pirzadeh, "A Fast Upwind Solver for the Euler Equations on Three-Dimensional Unstructured Meshes," *AIAA Paper 91-0102*, 1991.

[10] V. Venkatakrishnan and D. Mavriplis, "Agglomeration Multigrid for the Three-Dimensional Euler Equations," *AIAA Journal*, vol. 33, no. 4, pp. 633–640, 1995.

[11] E. J. Nielsen, W. K. Anderson, R. W. Walters, and D. E. Keyes, "Application of Newton-Krylov Methodology to a Three-Dimensional Unstructured Euler Code," *AIAA Paper 95-1733*, 1995.

[12] N. T. Frink, "Assessment of an Unstructured-Grid Method for Predicting 3-D Turbulent Viscous Flows," *AIAA Paper 96-0292*, 1996.

[13] D. J. Mavriplis, "Directional Agglomeration Multigrid Techniques for High Reynolds Number Viscous Flow Solvers," *AIAA Paper 98-0612*, 1998.

[14] H. Luo, D. Sharov, J. D. Baum, and R. Löhner, "Development and Application of Unstructured-Grid Methodologies for Turbulent Flows," *AIAA Paper 2003-0277*, 2003.

[15] D. J. Mavriplis and D. W. Levy, "Transonic Drag Prediction Using an Unstructured Multigrid Solver," *AIAA Paper 2002-0838*, 2002.

[16] S. Z. Pirzadeh and N. T. Frink, "Assessment of the Unstructured Grid Software TetrUSS for Drag Prediction of the DLR-F4 Configuration," *AIAA Paper 2002-0839*, 2002.

[17] H. Luo, J. D. Baum, and R. Löhner, "High-Reynolds Number Viscous Flow Computations Using an Unstructured-Grid Method," *AIAA Paper 2004-1103*, 2004.

[18] D. J. Mavriplis, "Grid Resolution Study of a Drag Prediction Workshop Configuration Using the NSU3D Unstructured Mesh Solver," *AIAA Paper 2005-4729*, 2005.

[19] V. Venkatakrishnan, "A Perspective on Unstructured Grid Flow Solvers," *AIAA Paper 95-0667*, 1995.

[20] J. Batina, "Implicit Upwind Solution Algorithms for Three-Dimensional Unstructured Meshes," *AIAA Journal*, vol. 31, no. 5, pp. 801–805, 1993.

[21] N. T. Frink, "Recent Progress Toward a Three-Dimensional Unstructured Navier-Stokes Flow Solver," *AIAA Paper 94-0061*, 1994.

[22] W. K. Anderson, R. D. Rausch, and D. L. Bonhaus, "Implicit/Multigrid Algorithms for Incompressible Turbulent Flows on Unstructured Grids," *Journal of Computational Physics*, vol. 128, pp. 391–408, 1996.

[23] D. Mavriplis, "Three-Dimensional Multigrid Reynolds-Averaged Navier-Stokes Solver for Unstructured Meshes," *AIAA Journal*, vol. 33, pp. 445–453, 1995.

[24] D. W. Levy, T. Zickuhr, J. Vassberg, S. Agrawal, R. A. Wahls, S. Pirzadeh, and M. J. Hemsch, "Summary of Data from the First AIAA CFD Drag Prediction Workshop," *AIAA Paper 2002-0841*, 2002.

[25] E. M. Lee-Rausch, P. G. Buning, J. H. Morrison, M. A. Park, S. M. Rivers, C. L. Rumsey, and D. Mavriplis, "CFD Sensitivity Analysis of a Drag Prediction Workshop Wing/Body Transport Configuration," *AIAA Paper 2003-3400*, 2003.

[26] M. J. Hemsch and J. Morrison, "Statistical Analysis of CFD Solutions from 2nd Drag Prediction Workshop (Invited)," *AIAA Paper 2004-0556*, 2004.

[27] N. T. Frink, "Tetrahedral Unstructured Navier-Stokes Method for Turbulent Flows," *AIAA Journal*, vol. 36, no. 11, pp. 1975–1982, 1998.

[28] D. J. Mavriplis and V. Venkatakrishnan, "A Unified Multigrid Solver for the Navier-Stokes Equations on Mixed Element Meshes," *AIAA Paper 95-1666*, 1995.

[29] G. May, E. van der Weide, A. Jameson, Sriram, and L. Martinelli, "Drag Prediction of the DLR-F6 Configuration," *AIAA Paper 2004-0396*, 2004.

[30] D. Mavriplis, "Unstructured Grid Techniques," *Annu. Rev. Fluid Mach.*, vol. 29, pp. 473–514, 1997.

[31] A. Jameson and D. Mavriplis, "Finite Volume Solution of the Two-Dimensional Euler Equations on a Regular Triangular Mesh," *AIAA Journal*, vol. 24, no. 4, pp. 611–618, 1986.

[32] A. Jameson, W. Schmidt, and E. Turkel, "Numerical Solutions of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time-Stepping Schemes," *AIAA Paper 81-1259*, 1981.

[33] D. Mavriplis, "Multigrid Solution of the Two-Dimensional Euler Equations on Unstructured Triangular Meshes," *AIAA Journal*, vol. 26, pp. 824–831, 1988.

[34] D. Mavriplis, "Three-Dimensional Unstructured Multigrid for the Euler Equations," *AIAA Journal*, vol. 30, pp. 1753–1761, 1992.

[35] D. J. Mavriplis, "Multigrid Strategies for Viscous Flow Solvers on Anisotropic Unstructured Meshes," *Journal of Computational Physics*, vol. 145, pp. 141–165, 1998.

[36] P. L. Roe, "Approximate Riemann Solvers, Parameter Vectors and Difference Schemes," *Journal of Computational Physics*, vol. 43, pp. 357–372, 1981.

[37] S. Osher, "Riemann Solvers, the Entropy Condition and Difference Approximations," *SIAM J. Numer. Anal.*, vol. 21, pp. 217–235, 1984.

[38] C. F. Ollivier-Gooch, "Multigrid Acceleration of an Upwind Euler Solver on Unstructured Meshes," *AIAA Journal*, vol. 33, no. 10, pp. 1822–1827, 1995.

[39] R. C. Swanson and E. Turkel, "On Central-Difference and Upwind Schemes," *J. Comp. Phys.*, vol. 101, pp. 292–306, 1992.

[40] P. C. Walsh and D. W. Zingg, "Solution Adaptation of Unstructured Grids for Two-Dimensional Aerodynamic Computations," *AIAA Journal*, vol. 39, no. 5, pp. 831–837, 2001.

[41] M. Aftosmis, D. Gaitonde, and T. Tavares, "Behavior of Linear Reconstruction Techniques on Unstructured Meshes," *AIAA Journal*, vol. 33, no. 11, pp. 2038–2049, 1995.

[42] T. J. Barth, "Aspects of Unstructured Grids and Finite-Element Volume Solvers for the Euler and Navier-Stokes Equations," *von Karman Inst. Lect. Ser., AGARD Publ. R-787*, 1992.

[43] A. Haselbacher, J. McGuirk, and G. Page, "Finite Volume Discretization Aspects for Viscous Flows on Mixed Unstructured Grids," *AIAA Journal*, vol. 37, no. 2, pp. 177–184, 1999.

[44] T. J. Barth, "Numerical Aspects of Computing Viscous High Reynolds Number Flows on Unstructured Meshes," *AIAA Paper 91-0721*, 1991.

[45] W. K. Anderson and D. L. Bonhaus, "Navier-Stokes Computations and Experimental Comparisons for Multielement Airfoil Configurations," *AIAA Paper 93-0645*, 1993.

[46] D. G. Holmes and S. D. Connell, "Solution of the 2D Navier-Stokes Equations on Unstructured Adaptive Grids," *AIAA Paper 89-1932*, 1989.

[47] J. V. Lassaline and D. W. Zingg, "Development of an Agglomeration Multigrid Algorithm with Directional Coarsening," *AIAA Paper 99-3338*, 1999.

[48] A. Haselbacher and J. Blazek, "On the Accurate and Efficient Discretisation of the Navier-Stokes Equations on Mixed Grids," *AIAA Journal*, vol. 38, no. 11, pp. 2094–2102, 2000.

[49] T. M. Smith, R. W. Hooper, C. C. Ober, A. A. Lorber, and J. N. Shadid, "Comparison of Operators for Newton-Krylov Method for Solving Compressible Flows on Unstructured Meshes," *AIAA Paper 2004-0743*, 2004.

[50] W. J. Coirier, *An Adaptively-Refined, Cartesian, Cell-Based Scheme for the Euler and Navier-Stokes Equations*. PhD thesis, University of Michigan, 1994.

[51] V. Venkatakrishnan, "Newton Solution of Inviscid and Viscous Problems," *AIAA Journal*, vol. 27, no. 7, pp. 885–891, 1989.

[52] V. Venkatakrishnan, "Viscous Computations Using a Direct Solver," *Computers and Fluids*, vol. 18, no. 2, pp. 191–204, 1990.

[53] W. A. Mulder and B. van Leer, "Experiments with Implicit Upwind Methods for the Euler Equations," *Journal of Computational Physics*, vol. 59, pp. 232–246, 1985.

[54] R. Beam and R. F. Warming, "An Implicit Finite-Difference Algorithm for Hyperbolic Systems in Conservation Law Form," *Journal of Computational Physics*, vol. 22, pp. 87–110, 1976.

[55] T. H. Pulliam and D. S. Chaussee, "A Diagonal Form of an Implicit Approximate Factorization Algorithm," *Journal of Computational Physics*, vol. 39, p. 347, 1981.

[56] W. K. Anderson and D. L. Bonhaus, "An Implicit Upwind Algorithm for Computing Turbulent Flows on Unstructured Grids," *Computers and Fluids*, vol. 23, no. 1, pp. 1–21, 1994.

[57] J. M. Weiss, J. P. Maruszewski, and W. A. Smith, "Implicit Solution of Preconditioned Navier-Stokes Equations Using Algebraic Multigrid," *AIAA Journal*, vol. 37, no. 1, pp. 29–36, 1999.

[58] V. Venkatakrishnan and D. J. Mavriplis, "Implicit Solvers for Unstructured Meshes," *Journal of Computational Physics*, vol. 105, pp. 83–91, 1992.

[59] K. Ajmani, W.-F. Ng, and M.-S. Liou, "Preconditioned Conjugate Gradient Methods for the Navier-Stokes Equations," *Journal of Computational Physics*, vol. 110, pp. 68–81, 1994.

[60] A. Pueyo and D. W. Zingg, "Efficient Newton-Krylov Solver for Aerodynamic Computations," *AIAA Journal*, vol. 36, no. 11, pp. 1991–1997, 1998.

[61] Y. Saad, *Iterative Methods For Sparse Linear Systems*. International Thomson Publishing Company, Boston, 1996.

[62] Y. Saad and H. A. van der Vorst, "Iterative Solution of Linear Systems in the 20-th Century," *Journal of Computational and Applied Mathematics*, vol. 123, pp. 1–33, 2000.

[63] P. D. Orkwis, "Comparison of Newton's and Quasi-Newton's Method Solvers for the Navier-Stokes Equations," *AIAA Journal*, vol. 31, no. 5, pp. 832–836, 1993.

[64] P. A. Forsyth and H. Jiang, "Nonlinear Iteration Methods for High Speed Laminar Compressible Navier-Stokes Equations," *Computers and Fluids*, vol. 26, no. 3, pp. 249–268, 1997.

[65] P. Geuzaine and J.-A. Essers, "A Newton-Krylov Solver for the Computation of Compressible Laminar and Turbulent Flows on Unstructured Grids," *4th Belgian National Congress of Theoretical and Applied Mechanics*, 1997.

[66] P. R. McHugh and D. A. Knoll, "Comparison of Standard and Matrix-Free Implementationss of Several Newton-Krylov Solvers," *AIAA Journal*, vol. 32, no. 12, pp. 2394–2400, 1994.

[67] D. A. Knoll, P. R. McHugh, and D. E. Keyes, "Newton-Krylov Methods for Low-Mach Number Compressible Combustion," *AIAA Journal*, vol. 34, no. 5, pp. 961–967, 1996.

[68] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, "Inexact Newton Methods," *SIAM J. Numer. Anal.*, vol. 19, no. 2, pp. 400–408, 1982.

[69] S. C. Eisenstat and H. F. Walker, "Choosing the Forcing Terms in an Inexact Newton Method," *SIAM J. Sci. Comput.*, vol. 17, no. 1, pp. 16–32, 1996.

[70] P. N. Brown and Y. Saad, "Hybrid Krylov Methods for Nonlinear Systems of Equations," *SIAM J. Sci. Stat. Comput.*, vol. 11, no. 3, pp. 450–481, 1990.

[71] Z. Johan, T. Hughes, and F. Shakib, "A Globally Convergent Matrix-Free Algorithm for Implicit Time-Marching Schemes Arising in Finite Element Analysis in Fluids," *Computer Methods in Applied Mechanics and Engineering*, vol. 87, pp. 281–304, 1991.

[72] T. Chisholm and D. W. Zingg, "Start-up Issues in a Newton-Krylov Algorithm for Turbulent Aerodynamic Flows," *AIAA Paper 2003-3708*, 2003.

[73] M. Blanco and D. W. Zingg, "Fast Newton-Krylov Method for Unstructured Grids," *AIAA Journal*, vol. 36, no. 4, pp. 607–612, 1998.

[74] A. Chapman, Y. Saad, and L. Wigton, "High-order ILU preconditioners for CFD problems," *International Journal for Numerical Methods in Fluids*, vol. 33, pp. 767–788, 2000.

[75] S. Yoon and A. Jameson, "Lower-Upper Symmetric-Gauss-Seidal Method for the Euler and Navier-Stokes Equations," *AIAA Journal*, vol. 26, no. 9, pp. 1025–1026, 1988.

[76] H. Luo, J. D. Baum, and R. Löhner, "A Fast, Matrix-free Implicit Method for Compressible Flows on Unstructured Grids," *Journal of Computational Physics*, vol. 146, pp. 664–690, 1998.

[77] H. Luo, J. D. Baum, and R. Löhner, "An Accurate, Fast, Matrix-free Implicit Method for Computing Unsteady Flows on Unstructured Grids," *Computers & Fluids*, vol. 30, no. 2, pp. 137–159, 2001.

[78] P. Geuzaine, "Newton-Krylov Strategy for Compressible Turbulent Flows on Unstructured Meshes," *AIAA Journal*, vol. 39, no. 3, pp. 528–531, 2000.

[79] A. Soulaimani, N. Salah, and Y. Saad, "Enhanced GMRES Acceleration Techniques for some CFD problems," *International Journal of Computational Fluid Dynamics*, vol. 16, no. 1, pp. 1–20, 2002.

[80] E. Chow and Y. Saad, "Experimental study of ILU preconditioners for indefinite matrices," *Journal of Computational and Applied Mathematics*, vol. 87, pp. 387–414, 1997.

[81] E. Cuthill and J. McKee, "Reducing the Bandwidth of Sparse Symmetric Matrices," *Proc. ACM National Conference, Association for Computing Machinery, New York*, pp. 157–172, 1969.

[82] A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*. Prentice Hall, 1981.

[83] N. E. Gibbs, W. G. Poole, and P. K. Stockmeyer, "An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix," *SIAM Journal on Numerical Analysis*, vol. 13, no. 2, pp. 236–250, 1976.

[84] J. W. H. Liu and A. H. Sherman, "Comparative Analysis of the Cuthill-McKee and the Reverse Cuthill-McKee Ordering Algorithms for Sparse Matrices," *SIAM Journal on Numerical Analysis*, vol. 13, no. 2, pp. 198–213, 1976.

[85] D. A. Knoll and W. J. Rider, "A Multigrid Preconditioned Newton-Krylov Method," *SIAM Journal on Scientific Computing*, vol. 21, no. 2, pp. 691–710, 1999.

[86] D. A. Knoll, V. A. Mousseau, and W. J. Rider, "On Coarse Grid Operators and Multigrid Preconditioned Newton-Krylov Methods in CFD," *AIAA Paper 99-3340*, 1999.

[87] P. Geuzaine, I. Lepot, F. Meers, and J. A. Essers, "Multilevel Newton-Krylov Algorithms for Computing Compressible Flows on Unstructured Meshes," *AIAA Paper 99-3341*, 1999.

[88] T. Chisholm and D. W. Zingg, "A Newton-Krylov Algorithm for Turbulent Aerodynamic Flows," *AIAA Paper 2003-0071*, 2003.

[89] P. R. Spalart and S. R. Allmaras, "A One-Equation Turbulence Model for Aerodynamic Flows," *AIAA Paper 92-0439*, 1992.

[90] L. M. Manzano, J. V. Lassaline, P. Wong, and D. W. Zingg, "A Newton-Krylov Algorithm for the Euler Equations Using Unstructured Grids," *AIAA Paper 2003-0274*, 2003.

[91] J. D. Anderson, *Fundamentals of Aerodynamics.* McGraw-Hill Inc., 1991.

[92] P. Geuzaine, *An Implicit Upwind Finite Volume Method for Compressible Turbulent Flows on Unstructured Meshes.* PhD thesis, Université de Liège, 1999.

[93] D. C. Wilcox, *Turbulence Modeling for CFD.* DCW Industries, 2002.

[94] P. Godin, D. W. Zingg, and T. E. Nelson, "High-Lift Aerodynamic Computations with One- and Two-Equation Turbulence Models," *AIAA Journal*, vol. 35, no. 2, pp. 237–243, 1997.

[95] P. C. Walsh, *Adaptive Solution of Viscous Aerodynamic Flows Using Unstructured Grids.* PhD thesis, University of Toronto, 1998.

[96] P. R. Spalart and S. R. Allmaras, "A One-Equation Turbulence Model for Aerodynamic Flows," *La Recherche Aérospatiale,* No. 1, pp. 5–21, 1994.

[97] G. A. Ashford, *An Unstructured Grid Generation and Adaptive Solution Technique for High Reynolds Number Compressible Flows.* PhD thesis, University of Michigan, 1996.

[98] M. Blanco and D. W. Zingg, "An Unstructured Newton-Krylov Algorithm with a Loosely-Coupled Turbulence Model for Aerodynamic Flows," *AIAA Paper 2006-691*, 2006.

[99] T. J. Barth and D. C. Jespersen, "The Design and Application of Upwind Schemes on Unstructured Meshes," *AIAA Paper 89-0366*, 1989.

[100] J. V. Lassaline, *A Navier-Stokes Equation Solver Using Agglomerated Multigrid Featuring Directional Coarsening and Line Implicit Smoothing.* PhD thesis, University of Toronto, 2003.

[101] Y. Saad and M. H. Schultz, "GMRES: A Generalized Minimum Residual Algorithm For Solving Nonsymmetric Linear Systems," *SIAM J. Sci. Stat. Computing*, vol. 7, pp. 856–869, 1986.

[102] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc Web page," 2001. http://www.mcs.anl.gov/petsc.

[103] A. Pueyo, *An Efficient Newton-Krylov Method for the Euler and Navier-Stokes Equations*. PhD thesis, University of Toronto, 1998.

[104] T. H. Pulliam, "Efficient Solution Methods for the Navier-Stokes Equations." Lecture Notes for the von Kármán Inst. for Fluid Dynamics Lecture Series: Numerical Techniques for Viscous Flow Computation in Turbomachinery Bladings, Brussels, Belgium, Jan. 1986.

[105] F. Ilinca and D. Pelletier, "Positivity Preservation and Adaptive Solution for the $k-\epsilon$ Model of Turbulence," *AIAA Journal*, vol. 36, no. 1, pp. 44–50, 1998.

[106] A. Soulaimani, Y. Saad, and A. Rebaine, "An Edge Based Stabilized Finite Element Method for Solving Compressible Flows: Formulation and Parallel Implementation," *Computational Methods in Applied Mechanical Engineering*, vol. 190, pp. 6735–6761, 2001.

[107] V. Schmitt and F. Charpin, "Pressure Distributions on the ONERA-M6-Wing at Transonic Mach Numbers." Experimental Data Base for Computer Program Assessment, Report of the Fluid Dynamics Panel Working Group 04, AGARD AR 138, May 1979.

[108] J. C. Nichols and D. W. Zingg, "A Three-Dimensional Multi-Block Newton-Krylov Flow Solver for the Euler Equations," *AIAA Paper 2005-5230*, 2005.

[109] O. Brodersen and A. Stürmer, "Drag Prediction of Engine-Airframe Interference Effects Using Unstructured Navier-Stokes Calculations," *AIAA Paper 2001-2414*, 2001.

[110] T. T. Chisholm, *A Fully Coupled Newton-Krylov Solver with a One-Equation Turbulence Model*. PhD thesis, University of Toronto, 2005.

[111] A. George and J. W. H. Liu, "An Automatic Nested Dissection Algorithm for Irregular Finite Element Problems," *SIAM Journal on Numerical Analysis*, vol. 15, no. 5, pp. 1053–1069, 1978.

[112] A. George, "An Automatic One-Way Dissection Algorithm for Irregular Finite Element Problems," *SIAM Journal on Numerical Analysis*, vol. 17, no. 6, pp. 740–751, 1980.

[113] A. George and J. W. H. Liu, "A Fast Implementation of the Minimum Degree Algorithm Using Quotient Graphs," *ACM Transactions on Mathematical Software*, vol. 6, no. 3, pp. 337–358, 1980.

[114] C. P. T. Groth and S. A. Northrup, "Parallel Implicit Adaptive Mesh Refinement Scheme for Body-Fitted Multi-Block Mesh," *AIAA Paper 2005-5333*, 2005.

[115] X.-C. Cai, W. D. Gropp, D. E. Keyes, R. C. Melvin, and D. P. Young, "Parallel Newton-Krylov-Schwarz Algorithms for the Transonic Full Potential Equation," *SIAM Journal on Scientific Computing*, vol. 19, no. 1, pp. 246–265, 1998.

[116] T. J. Barth, T. F. Chan, and W.-P. Tang, "A Parallel Non-Overlapping Domain-Decomposition Algorithm for Compressible Fluid Flow Problems on Triangulated Domains," *Contemporary Mathematics*, vol. 218, pp. 23–41, 1998.

[117] Y. Saad and M. Sosonkina, "Distributed Schur Complement Techniques for General Sparse Linear Systems," *SIAM Journal on Scientific Computing*, vol. 21, no. 4, pp. 1337–1356, 1999.

[118] S. R. Allmaras, "Multigrid for the 2-D Compressible Navier-Stokes Equations," *AIAA Paper 99-3336*, 1999.

[119] P. Wong and D. W. Zingg, "A Newton-Krylov Algorithm for Turbulent Aerodynamic Flows on Unstructured Grids," *CASI 50th Annual Conference Aerodynamics Symposium*, 2003.

Appendices

Appendix A

Eigensystem of the Inviscid Jacobian

The eigenvalue decomposition of the inviscid flux Jacobian is given by $A = X\Lambda X^{-1}$, as described in Section 3.1.2. In two dimensions, the eigenvalues are given by:

$$\Lambda = \text{diag}(U,\ U,\ U + an,\ U - an) \tag{A.1}$$

where $U = \vec{\mathbf{V}} \cdot \vec{\mathbf{n}}$, $n = |\vec{\mathbf{n}}|$ as given in (3.30), and a is the speed of sound. The eigenvectors are given by [104, 100]:

$$X = \begin{pmatrix} 1 & 0 & \alpha & \alpha \\ u & \tilde{\kappa}_y \rho & \alpha(u + \tilde{\kappa}_x a) & \alpha(u - \tilde{\kappa}_x a) \\ v & -\tilde{\kappa}_x \rho & \alpha(v + \tilde{\kappa}_y a) & \alpha(v - \tilde{\kappa}_y a) \\ \dfrac{\phi^2}{(\gamma - 1)} & \rho(\tilde{\kappa}_y u - \tilde{\kappa}_x v) & \alpha\left(\dfrac{\phi^2 + a^2}{\gamma - 1} + \dfrac{aU}{n}\right) & \alpha\left(\dfrac{\phi^2 + a^2}{\gamma - 1} - \dfrac{aU}{n}\right) \end{pmatrix} \tag{A.2}$$

with

$$\alpha = \rho/(\sqrt{2}a) \tag{A.3}$$

$$\phi^2 = \frac{1}{2}(\gamma - 1)(u^2 + v^2) \tag{A.4}$$

$$\tilde{\kappa}_x = n_x/n, \quad \tilde{\kappa}_y = n_y/n \tag{A.5}$$

In three dimensions, the eigenvalues are:

$$\Lambda = \text{diag}(U,\ U,\ U,\ U + an,\ U - an) \tag{A.6}$$

and the eigenvectors are:

$$X = \begin{pmatrix} \tilde{\kappa}_x & \tilde{\kappa}_y & \tilde{\kappa}_z & \alpha & \alpha \\ \tilde{\kappa}_x u & \tilde{\kappa}_y u - \tilde{\kappa}_z \rho & \tilde{\kappa}_z u + \tilde{\kappa}_y \rho & \alpha(u + \tilde{\kappa}_x a) & \alpha(u - \tilde{\kappa}_x a) \\ \tilde{\kappa}_x v + \tilde{\kappa}_z \rho & \tilde{\kappa}_y v & \tilde{\kappa}_z v - \tilde{\kappa}_x \rho & \alpha(v + \tilde{\kappa}_y a) & \alpha(v - \tilde{\kappa}_y a) \\ \tilde{\kappa}_x w - \tilde{\kappa}_y \rho & \tilde{\kappa}_y w + \tilde{\kappa}_x \rho & \tilde{\kappa}_z w & \alpha(w + \tilde{\kappa}_z a) & \alpha(w - \tilde{\kappa}_z a) \\ \dfrac{\tilde{\kappa}_x \phi^2}{\gamma - 1} + \rho(\tilde{\kappa}_z v - \tilde{\kappa}_y w) & \dfrac{\tilde{\kappa}_y \phi^2}{\gamma - 1} + \rho(\tilde{\kappa}_x w - \tilde{\kappa}_z u) & \dfrac{\tilde{\kappa}_z \phi^2}{\gamma - 1} + \rho(\tilde{\kappa}_y u - \tilde{\kappa}_x v) & \alpha\left(\dfrac{\phi^2 + a^2}{\gamma - 1} + \dfrac{aU}{n}\right) & \alpha\left(\dfrac{\phi^2 + a^2}{\gamma - 1} - \dfrac{aU}{n}\right) \end{pmatrix} \tag{A.7}$$

with

$$\phi^2 = \frac{1}{2}(\gamma - 1)(u^2 + v^2 + w^2) \tag{A.8}$$

and

$$\tilde{\kappa}_z = n_z / n \tag{A.9}$$

Appendix B

Some Useful Definitions

A real square matrix A is positive definite if:

$$x^T Ax > 0, \quad \text{for any vector } x \neq 0 \tag{B.1}$$

If A is symmetric and positive definite, then its eigenvalues are positive.

A square matrix A is diagonally dominant if:

$$|a_{i,i}| \geq \sum_{j \neq i} |a_{i,j}|, \quad \text{for all } i \tag{B.2}$$

If the equalities are strict, then the matrix is strictly diagonally dominant. A strictly diagonally dominant matrix is nonsingular. A symmetric diagonally dominant real matrix with nonnegative diagonal entries is positive semidefinite; the eigenvalues are nonnegative.

An M-type matrix is diagonally dominant with positive diagonal elements and nonpositive off-diagonal elements. The inverse of an M-type matrix contains only nonnegative elements.

The condition number for a square matrix A is defined as:

$$\kappa(A) = ||A||\ ||A^{-1}|| \tag{B.3}$$

Usually a p-norm is used in the definition:

$$||A||_p = \max_{x \neq 0} \frac{||Ax||_p}{||x||_p}, \quad p \geq 1 \tag{B.4}$$

The number measures the relative sensitivity of a solution x of a linear system $Ax = b$, to changes in A and b. If $\kappa(A)$ is relatively small, then the matrix is well-conditioned. If $\kappa(A)$ is large, then the matrix is ill-conditioned. The condition number for a singular matrix is defined as infinity. If the matrix is symmetric and positive definite, then

$$\kappa(A) = \frac{M(A)}{m(A)} \tag{B.5}$$

where $M(A)$ and $m(A)$ are the largest and smallest absolute eigenvalues of A respectively.

Appendix C

Turbulence Model Modifications

Convergence of the turbulence model to machine zero may not be obtainable on some grids if clipping as described in Section 3.2.6 is used. One option is to allow negative values of $\tilde{\nu}$ in the solution [118, 100, 119], but this is not recommended for Newton's method because the method seems to behave differently for negative values of $\tilde{\nu}$. Investigation into the source of the negative values in the converged solution reveals that it is due to numerical oscillation. It usually occurs near sharp corners such as the trailing edge. The problem can be fixed for some cases by refining the grid at these locations. Since the boundary condition of $\tilde{\nu}$ is zero on a body surface, a slight numerical oscillation will cause $\tilde{\nu}$ to become negative. Attempts were made to use a slightly positive body surface condition for $\tilde{\nu}$, but it adversely affects the stability of the turbulence model. The reason for this is unclear.

An approach is suggested in the current work by increasing numerical dissipation of the turbulence model. The dissipation of the turbulence model is given in Section 3.1.2 with $\varepsilon^{(2)} = 1$, $\varepsilon^{(4)} = 0$, and $|A_{ik}| = |\vec{\mathbf{V}}_{ik} \cdot \vec{\mathbf{n}}_{ik}|$. This is modified as:

$$|A_{ik}| = \max\left(|\vec{\mathbf{V}}_{ik} \cdot \vec{\mathbf{n}}_{ik}|,\ \sigma_1|\vec{\mathbf{V}}_{i} \cdot \vec{\mathbf{n}}_{i}|,\ \sigma_1|\vec{\mathbf{V}}_{k} \cdot \vec{\mathbf{n}}_{k}|,\ \sigma_2\right) \tag{C.1}$$

with parameters σ_1 and σ_2. The original scheme is recovered with σ_1=σ_2=0. The use of σ_1=1 is found to improve positivity of the turbulent solution for some cases, while the use of σ_1=1 and σ_2=10^{-6} is found to be sufficient for most cases to obtain a nonnegative solution.

Appendix D

Figures

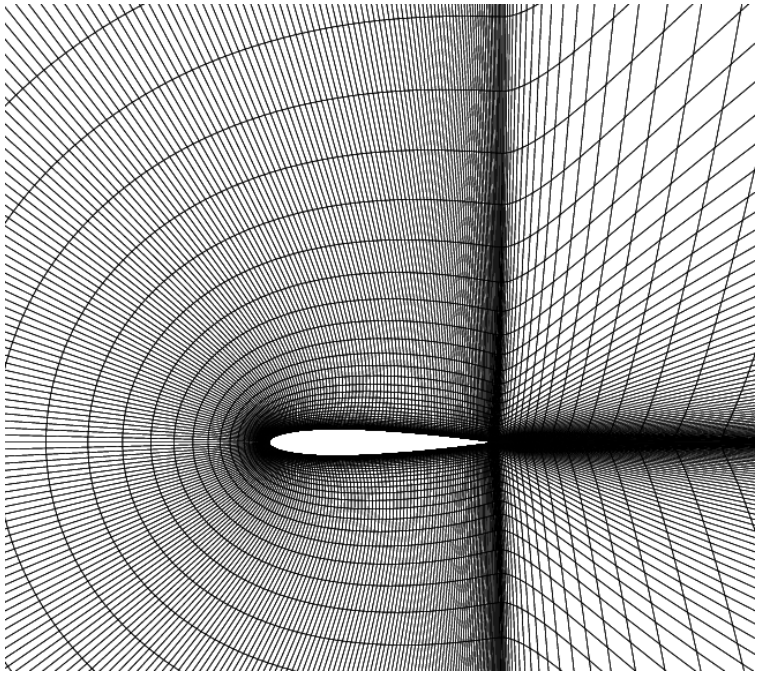

Figure D.1: Case 3b quadrilateral grid

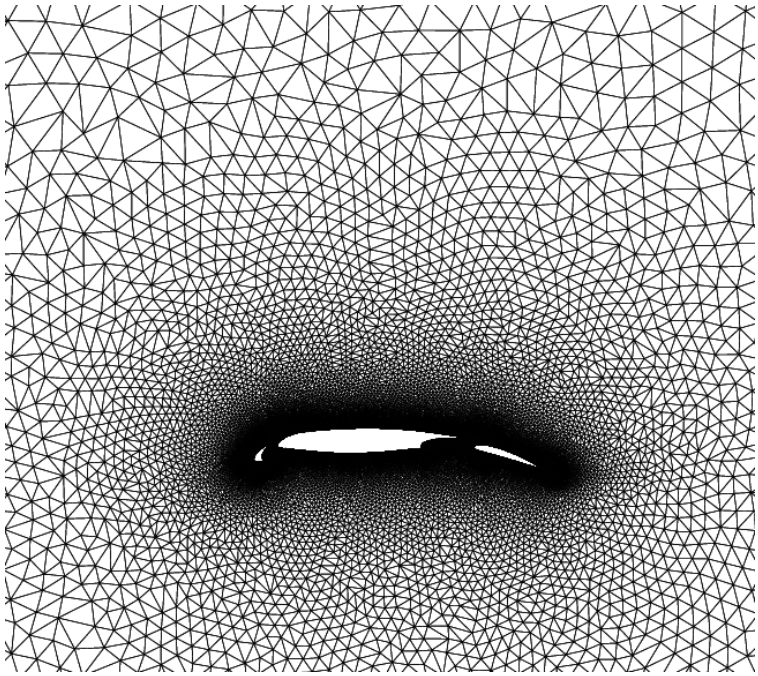

Figure D.2: Case 5 triangular grid

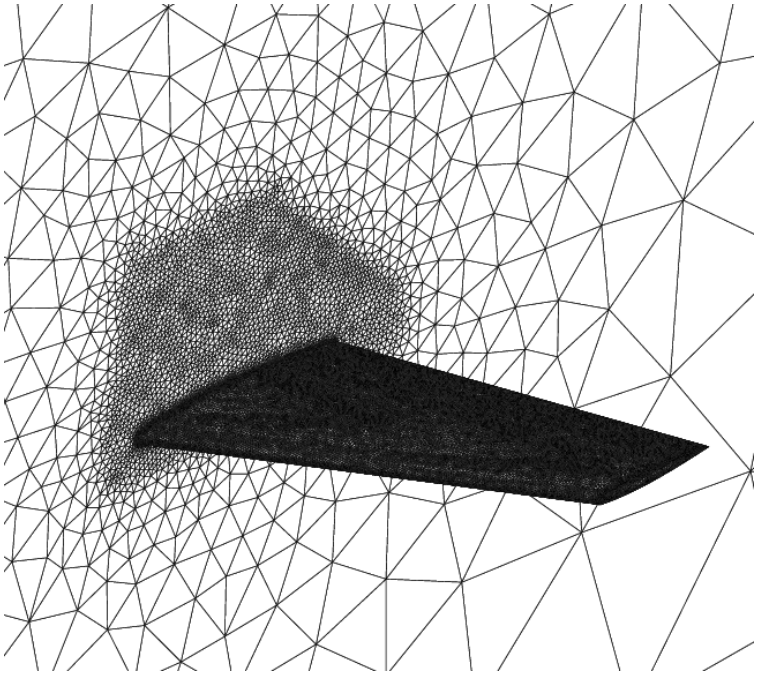

Figure D.3: Case 7c hybrid grid

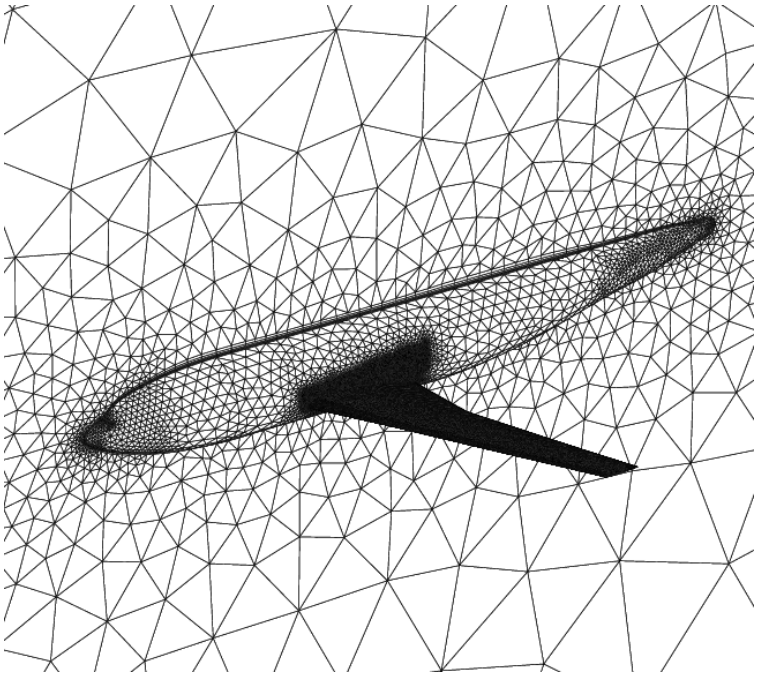

Figure D.4: Case 8b hybrid grid

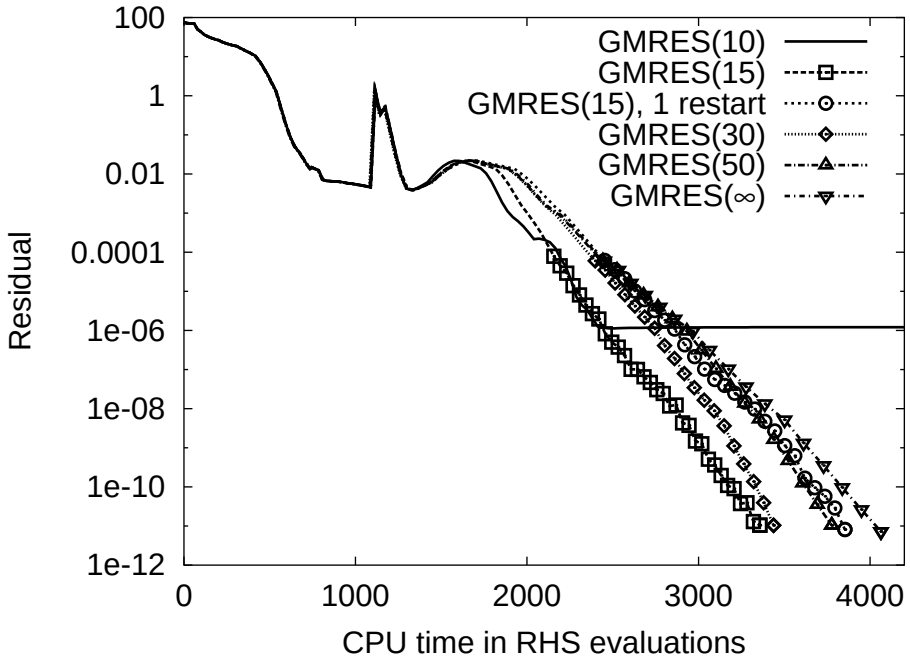

Figure D.5: A GMRES parametric study for Case 7a.

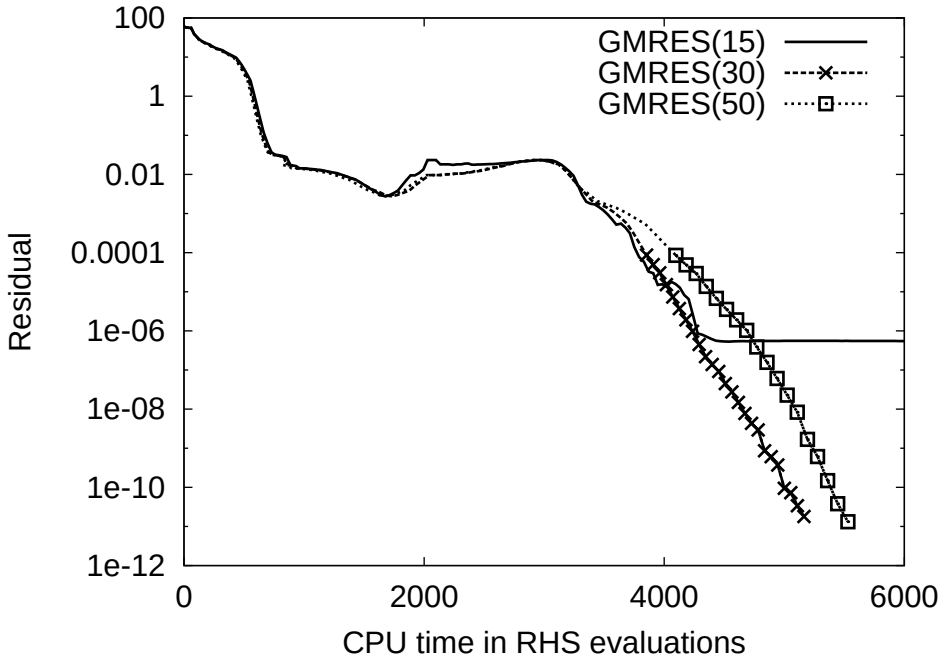

Figure D.6: A GMRES parametric study for Case 7b.

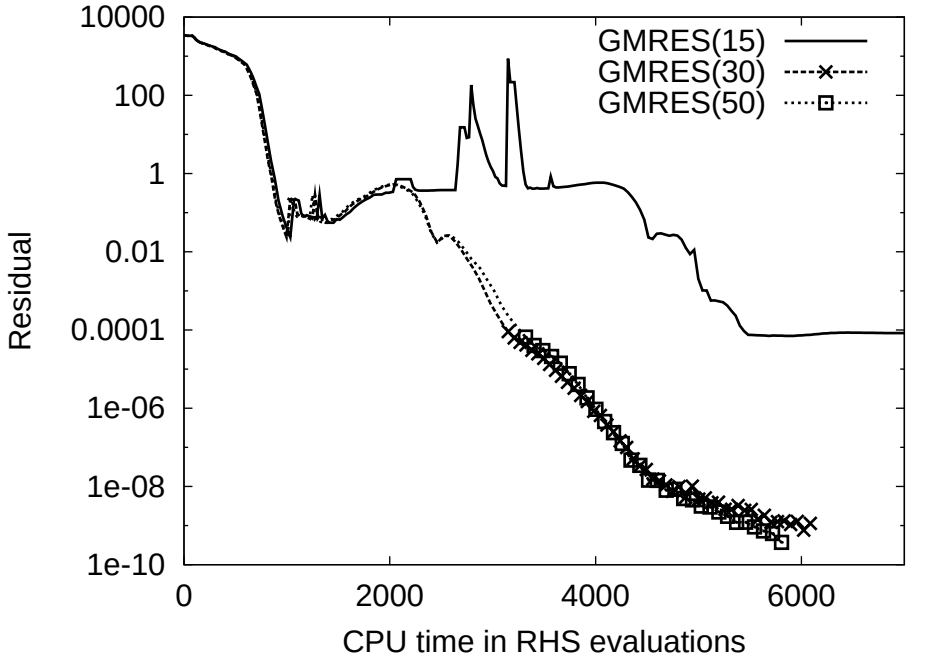

Figure D.7: A GMRES parametric study for Case 8a.

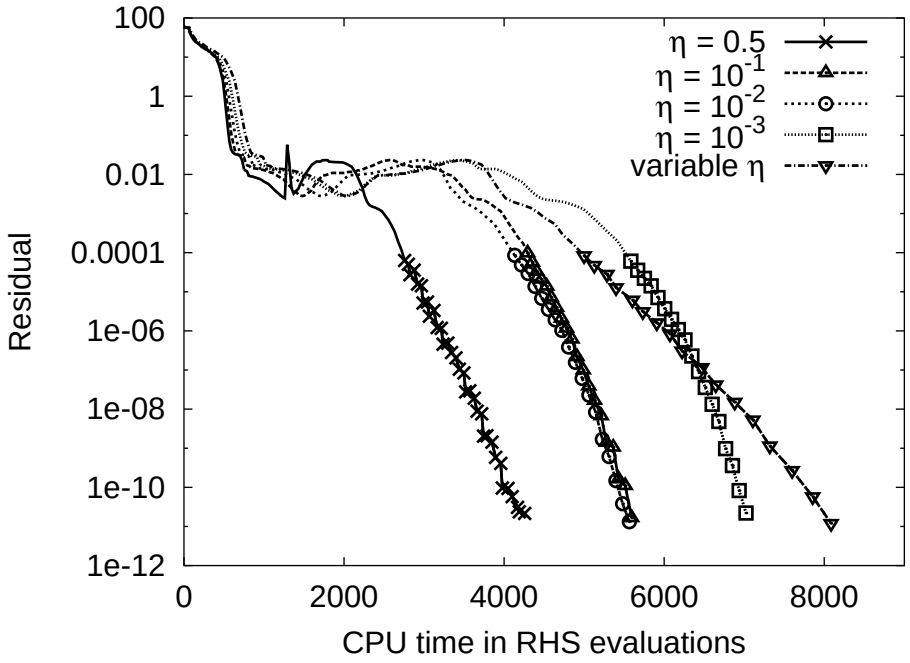

Figure D.8: A linear system tolerance study for Case 7b. Symbol spacing: 1 nonlinear iteration.

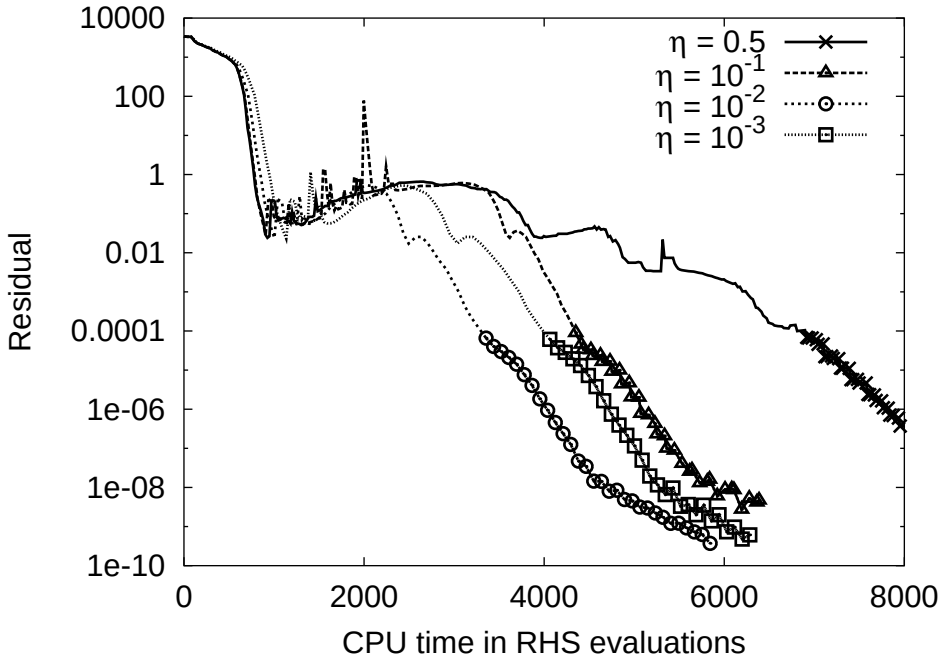

Figure D.9: A linear system tolerance study for Case 8a.

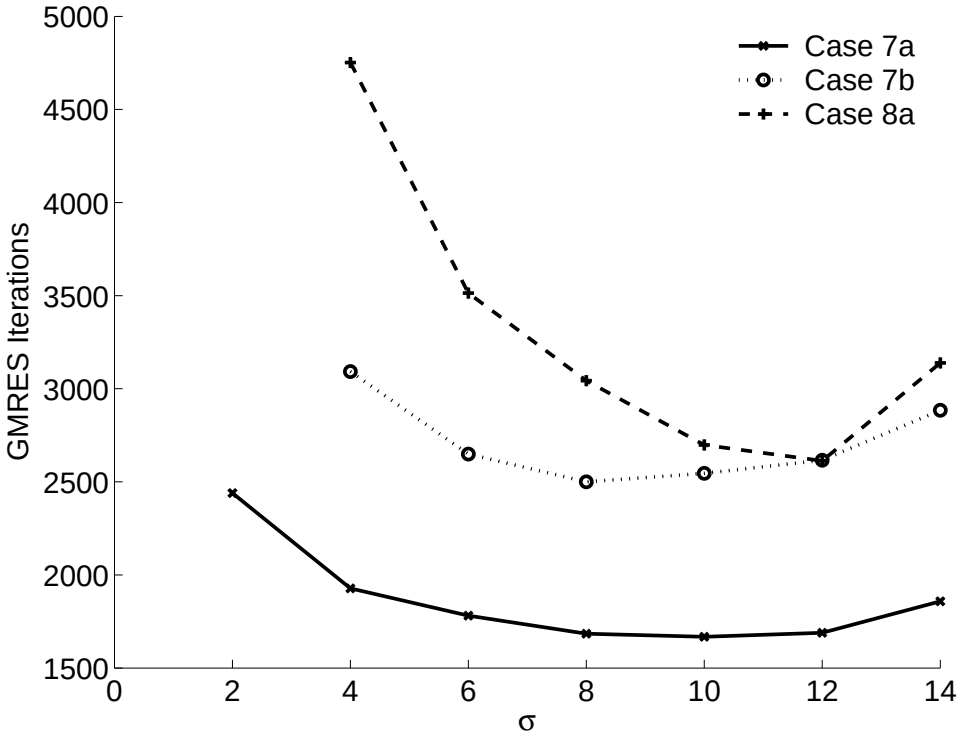

Figure D.10: Total number of GMRES iterations required to converge for different values of σ.

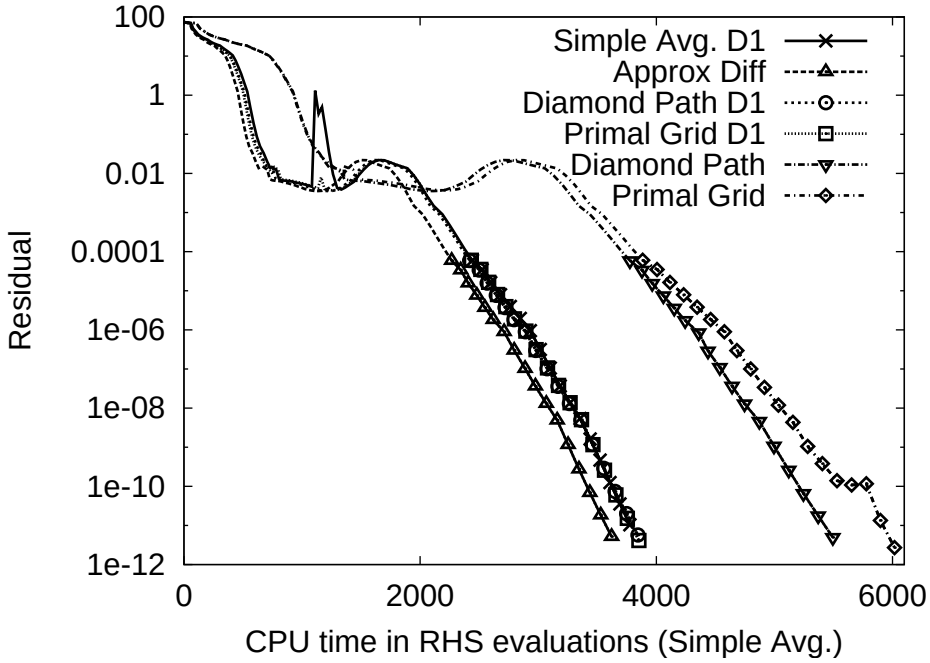

Figure D.11: A study of viscous preconditioning for Case 7a. The simple averaging approach is used on the right-hand side.

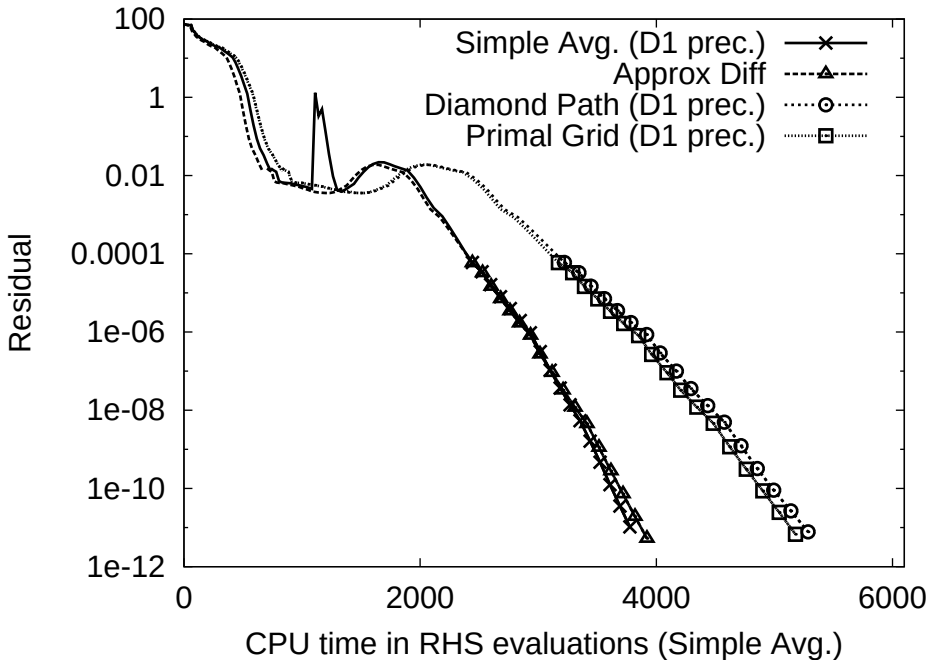

Figure D.12: A study of various viscous calculations in the preconditioner and on the right-hand side for Case 7a.

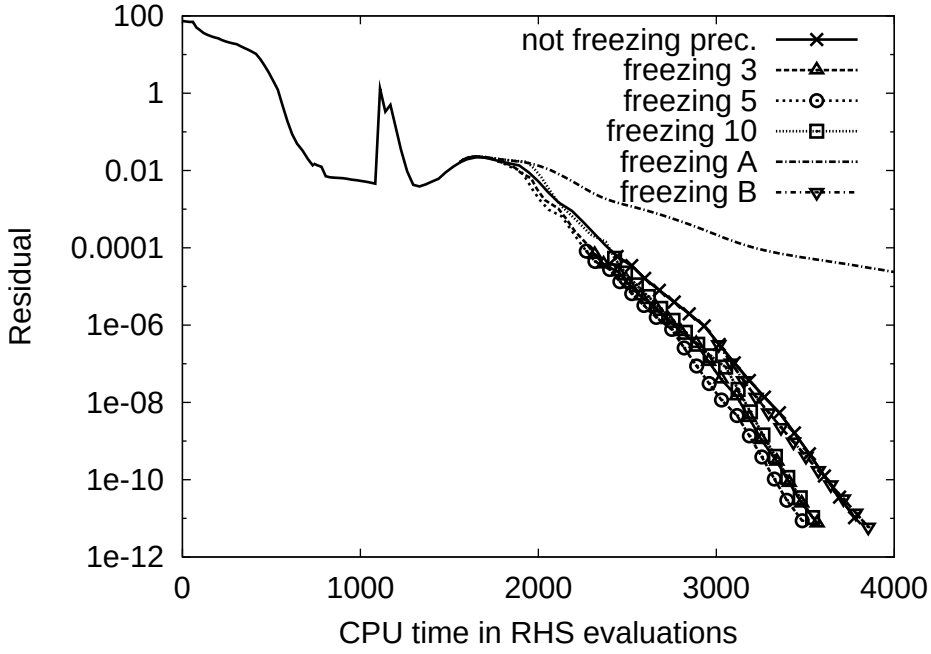

Figure D.13: A study of freezing of the preconditioner for Case 7a. A: freeze after mean-flow residual reduces to 10^{-4}. B: freeze after mean-flow residual reduces to 10^{-8}.

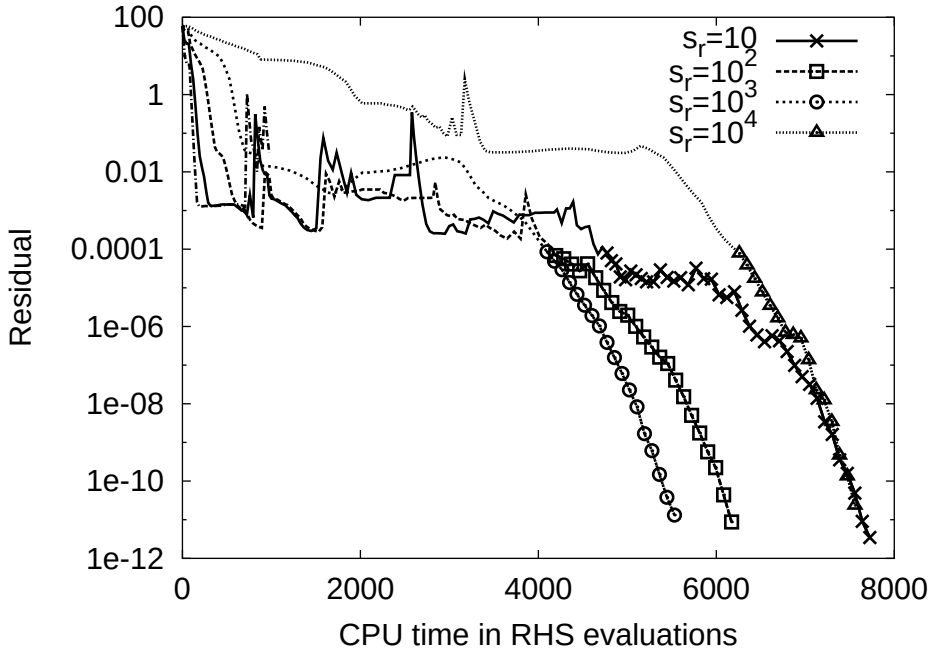

Figure D.14: A study of row scaling, s_r, for Case 7b. Column scaling: s_c=10^{-3}.

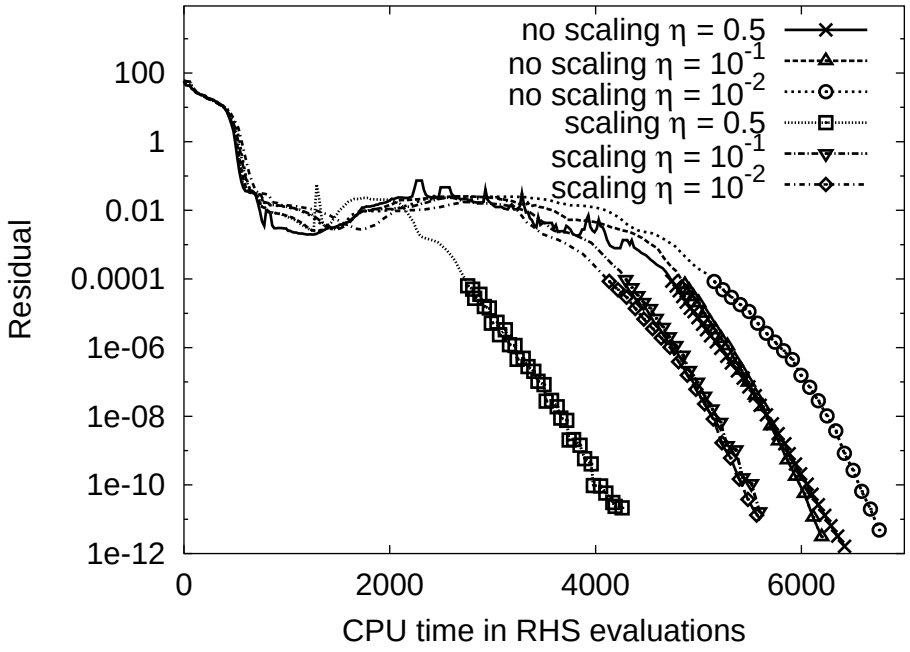

Figure D.15: A study of row and column scaling for Case 7b. No scaling: s_r=s_c=1. Scaling: s_r=s_c=10^{-3}.

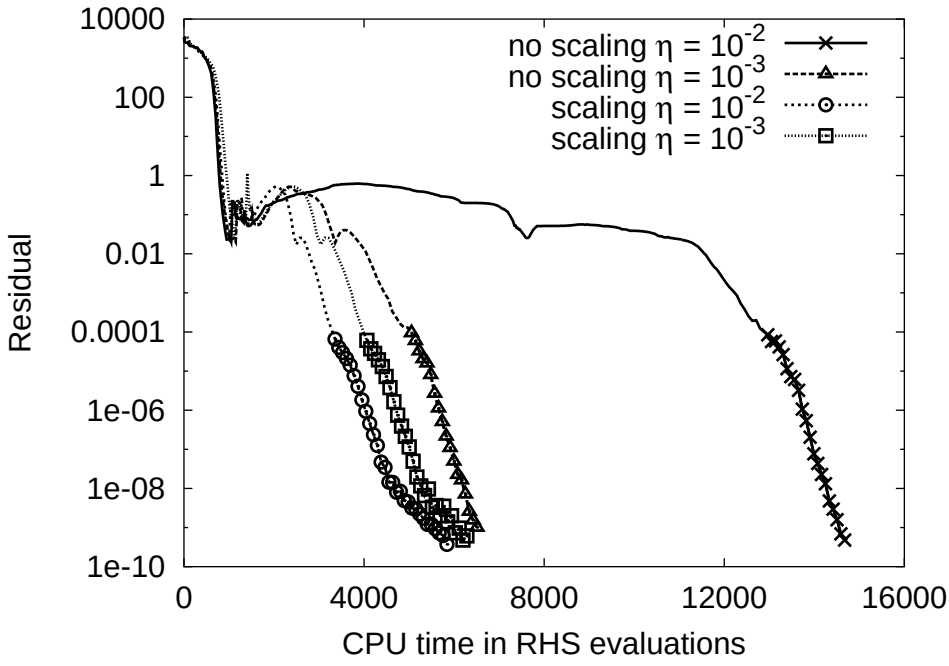

Figure D.16: A study of row and column scaling for Case 8a. No scaling: s_r=s_c=1. Scaling: s_r=s_c=10^{-3}.

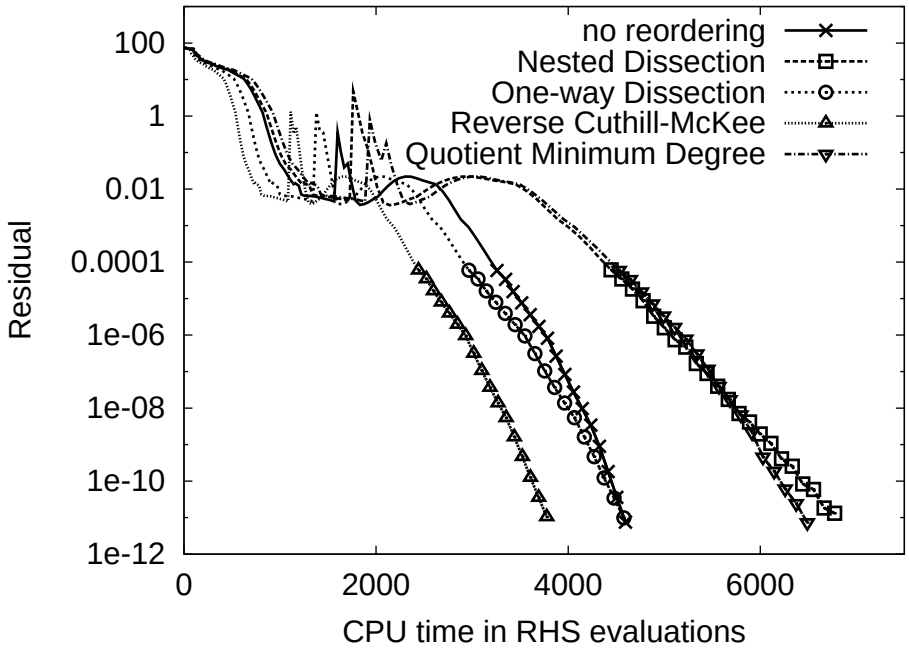

Figure D.17: A study of various orderings for Case 7a.

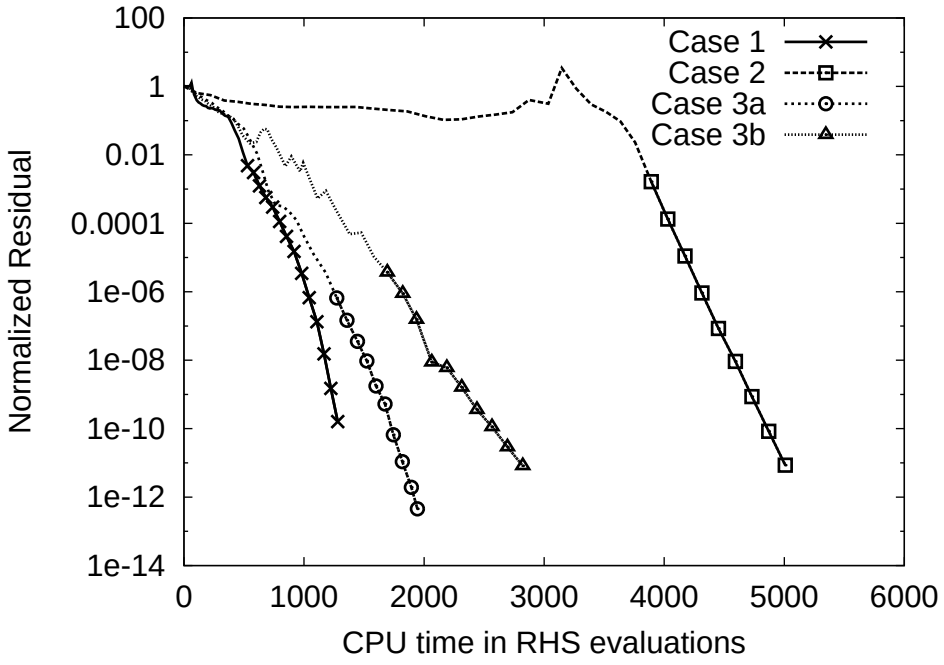

Figure D.18: Convergence for two-dimensional cases (Cases 1 to 3).

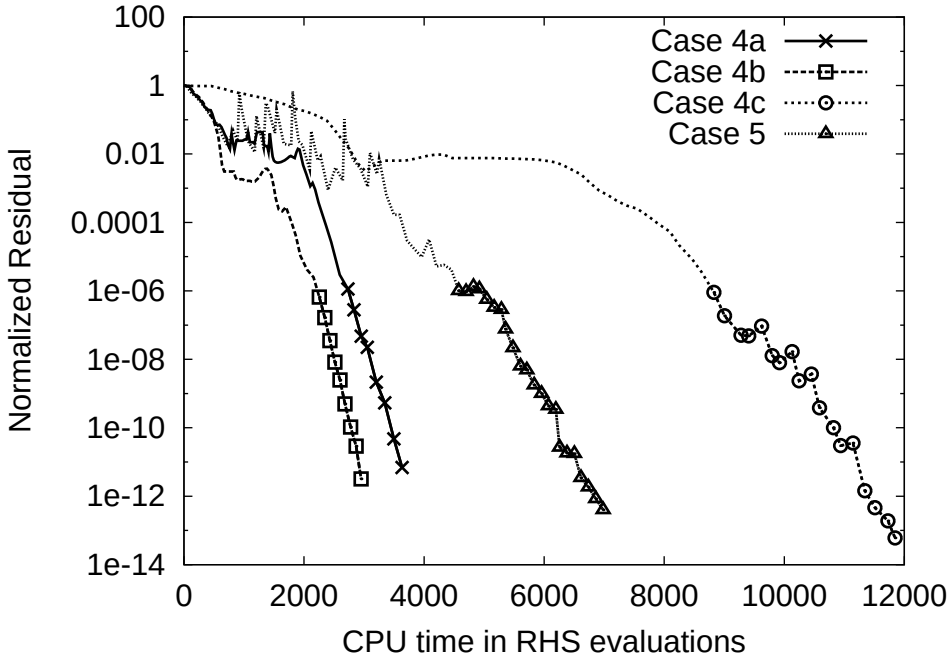

Figure D.19: Convergence for two-dimensional cases (Cases 4 and 5).

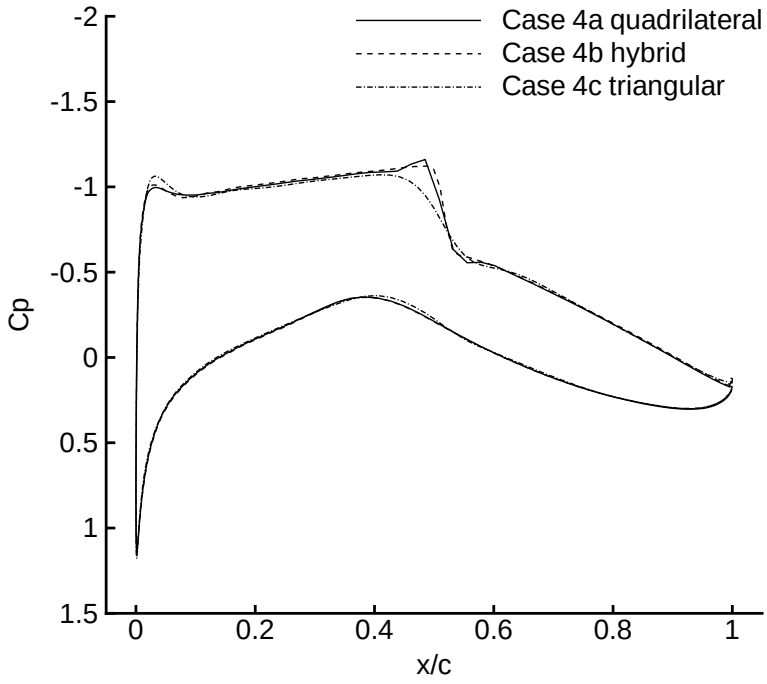

Figure D.20: Case 4 pressure coefficients.

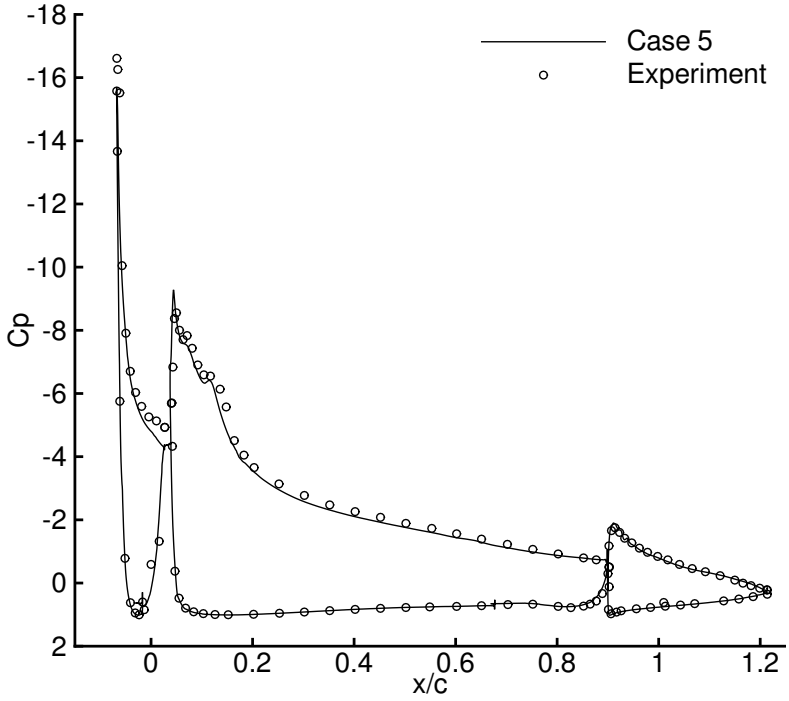

Figure D.21: Case 5 pressure coefficients.

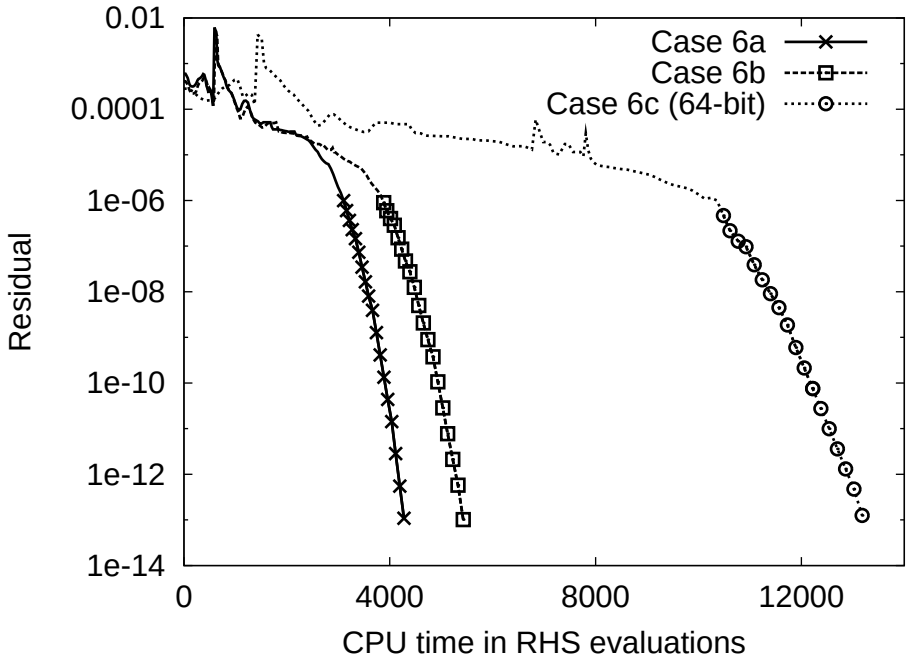

Figure D.22: Convergence for Case 6 on tetrahedral grids.

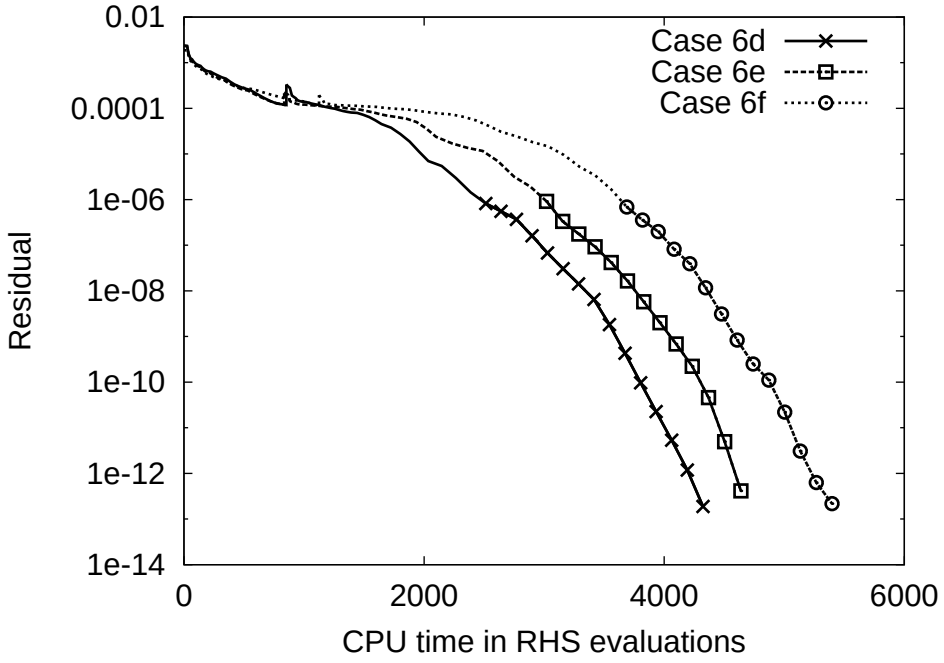

Figure D.23: Convergence for Case 6 on hexahedral grids.

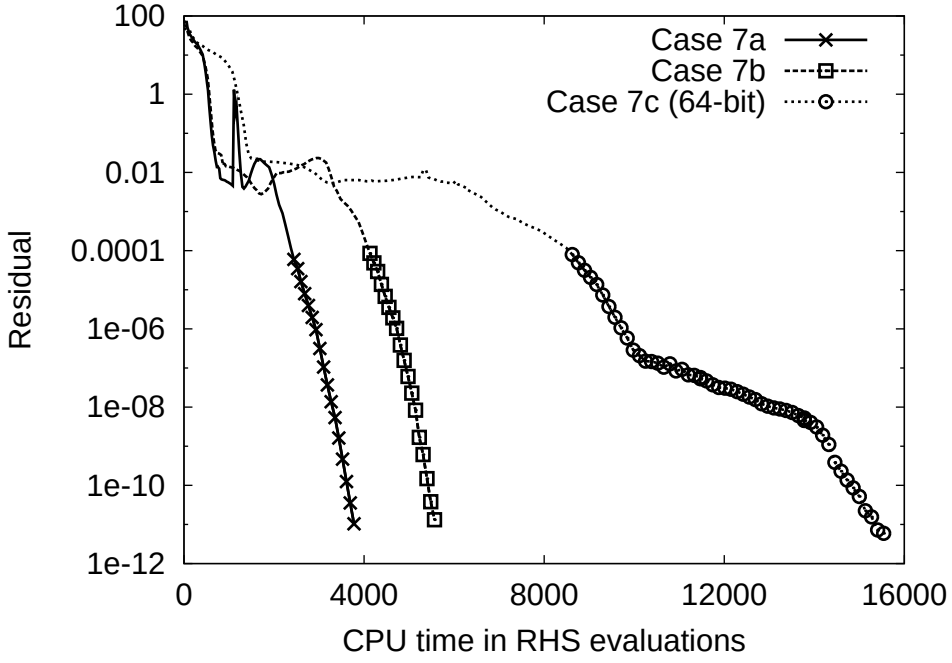

Figure D.24: Convergence for Case 7.

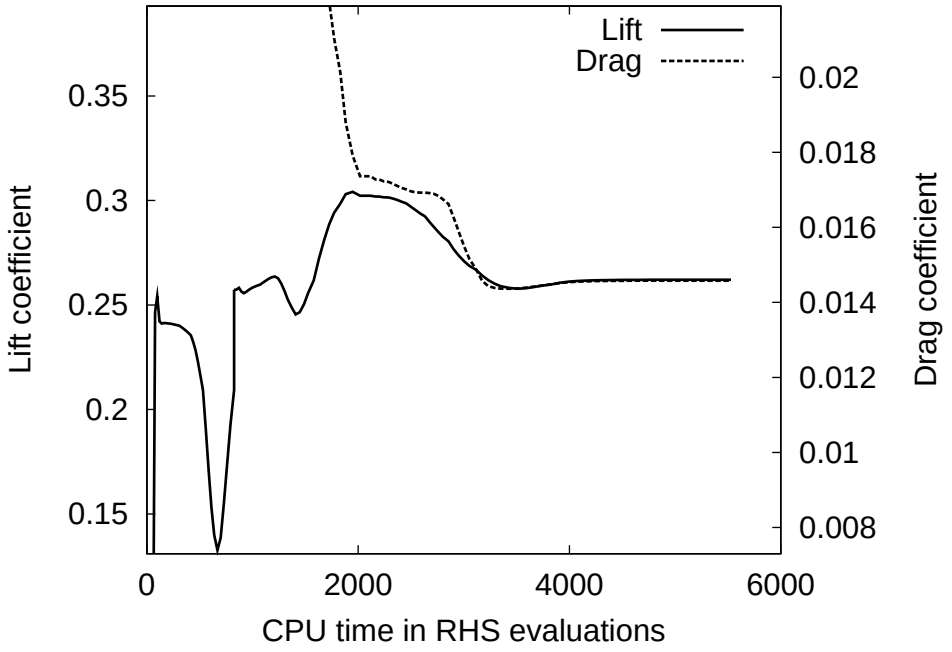

Figure D.25: Convergence of lift and drag coefficients for Case 7b.

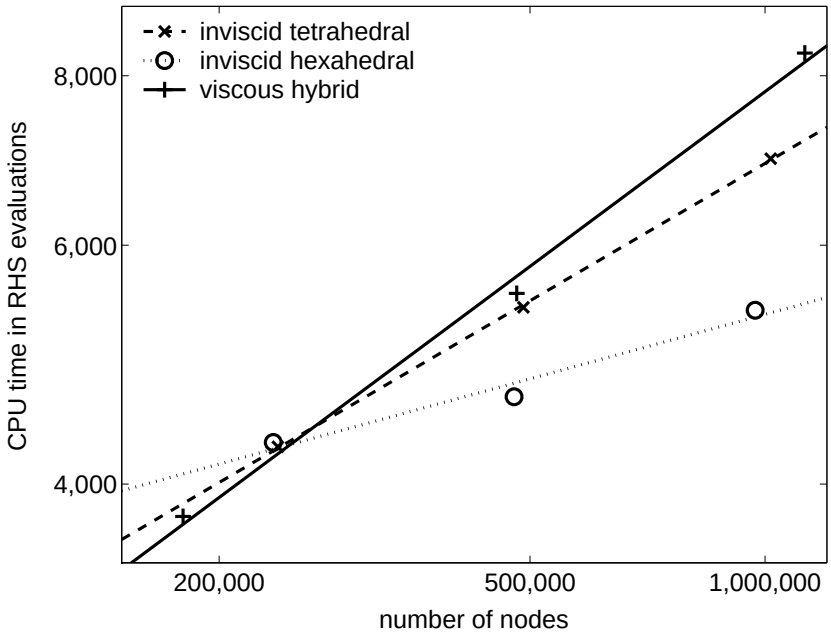

Figure D.26: Scalability of the algorithm.

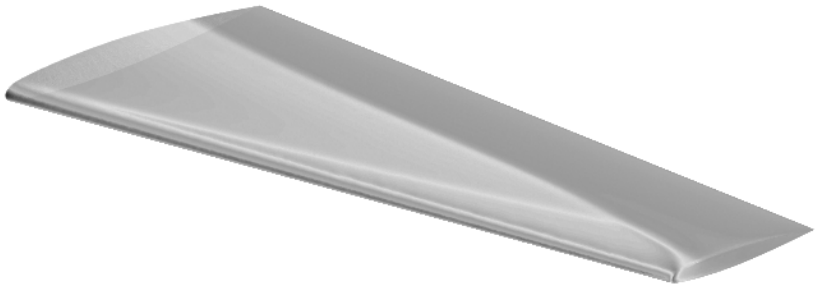

Figure D.27: Pressure contours over the ONERA M6 wing at $M_\infty = 0.8395$, $\alpha = 3.06°$, and $Re = 11.7 \times 10^6$.

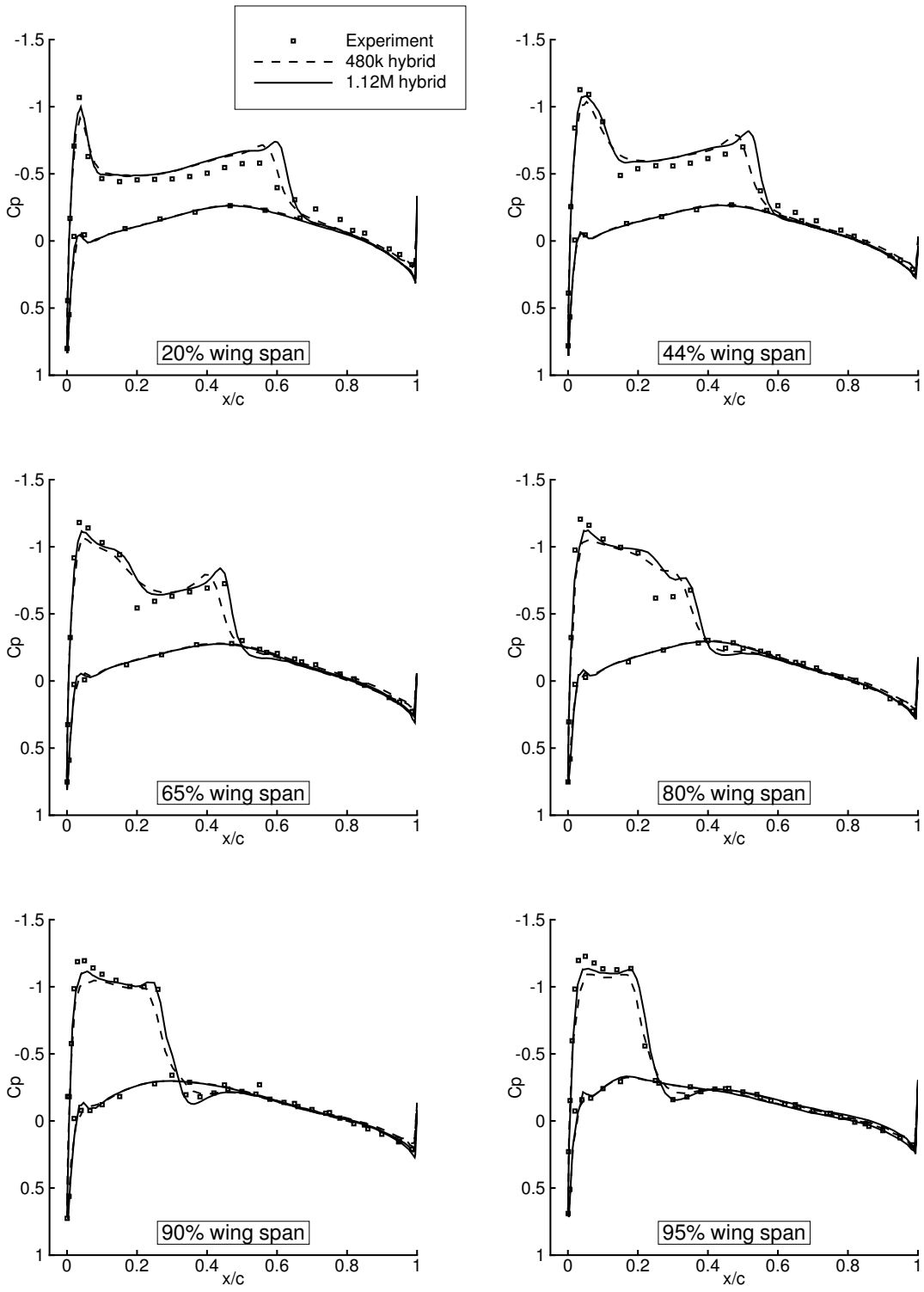

Figure D.28: Comparison with experimental pressure coefficients (Schmitt and Charpin, 1979) at different wing span locations for Cases 7b and 7c.

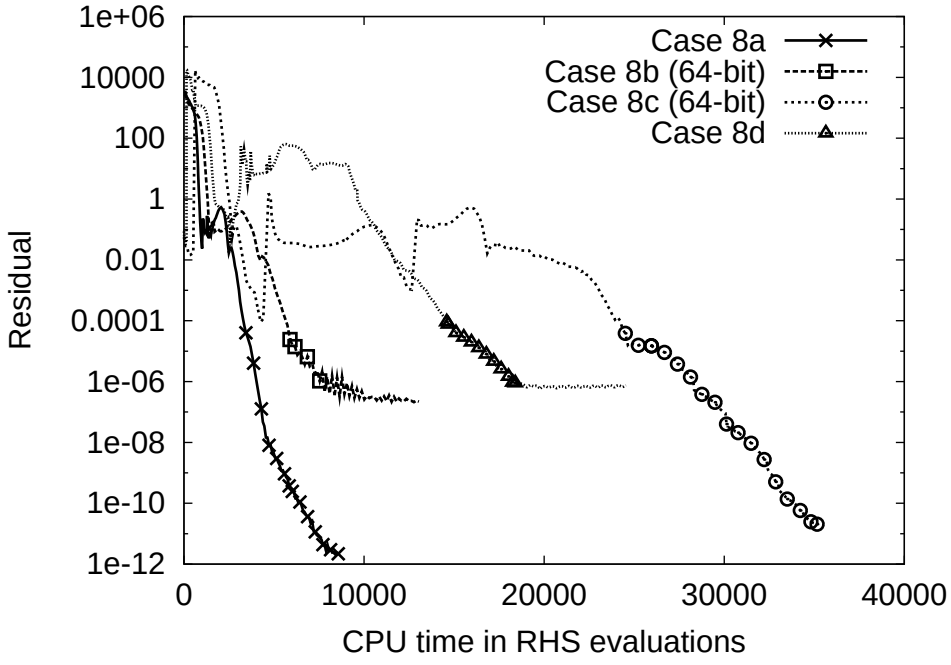

Figure D.29: Convergence for Case 8 over a wing-body configuration. Symbol spacing: 5 iterations.

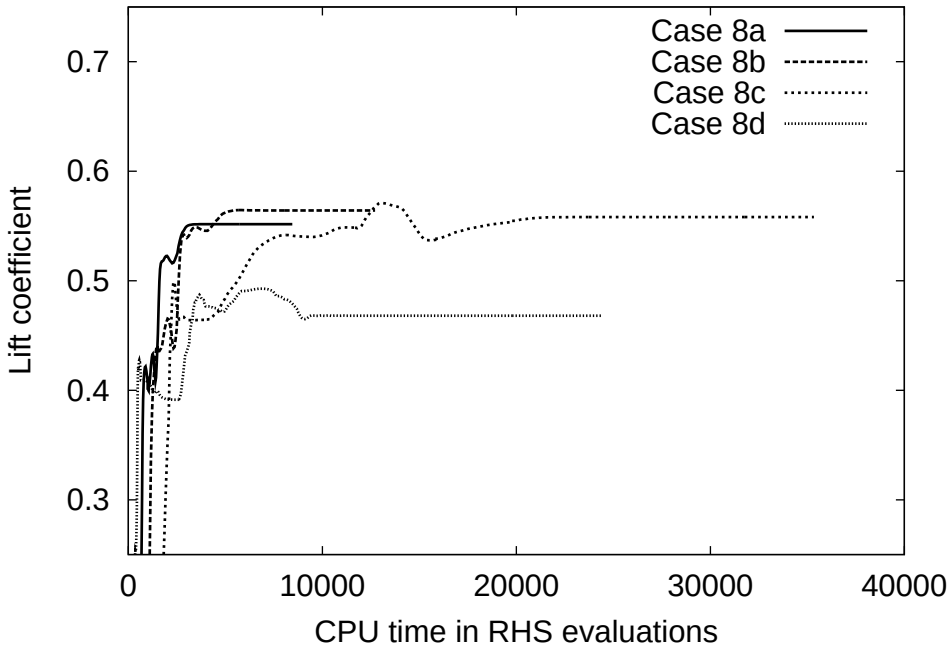

Figure D.30: Case 8 lift convergence.

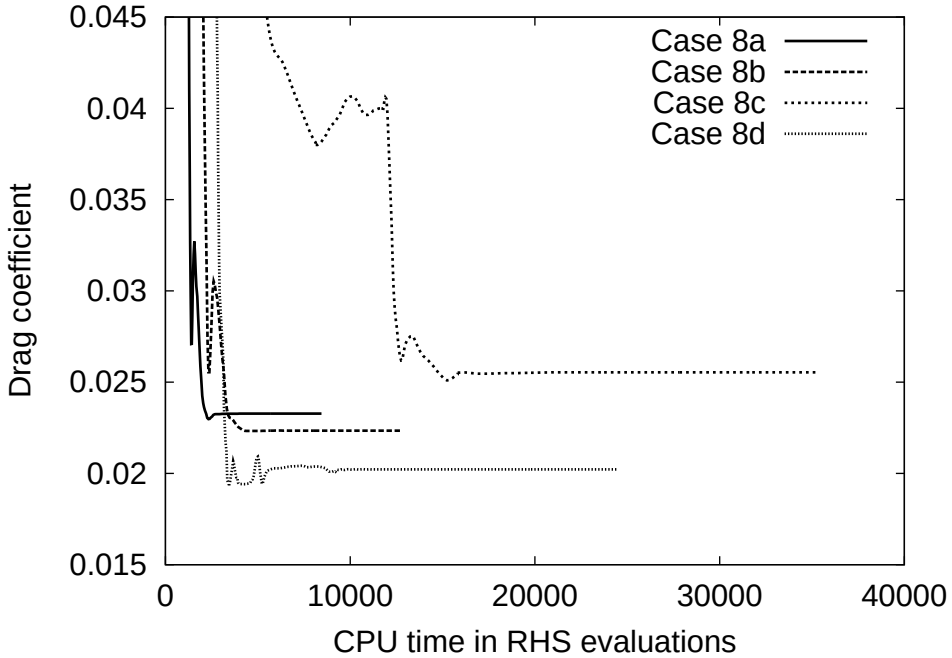

Figure D.31: Case 8 drag convergence.

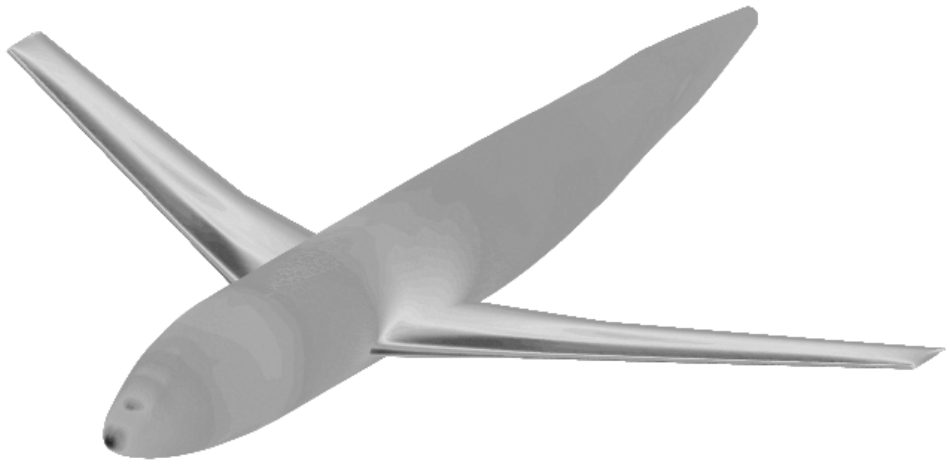

Figure D.32: Pressure contours over the DLR-F6 wing-body configuration at $M_\infty = 0.75$, $\alpha = 0.52°$, and $Re = 3 \times 10^6$.

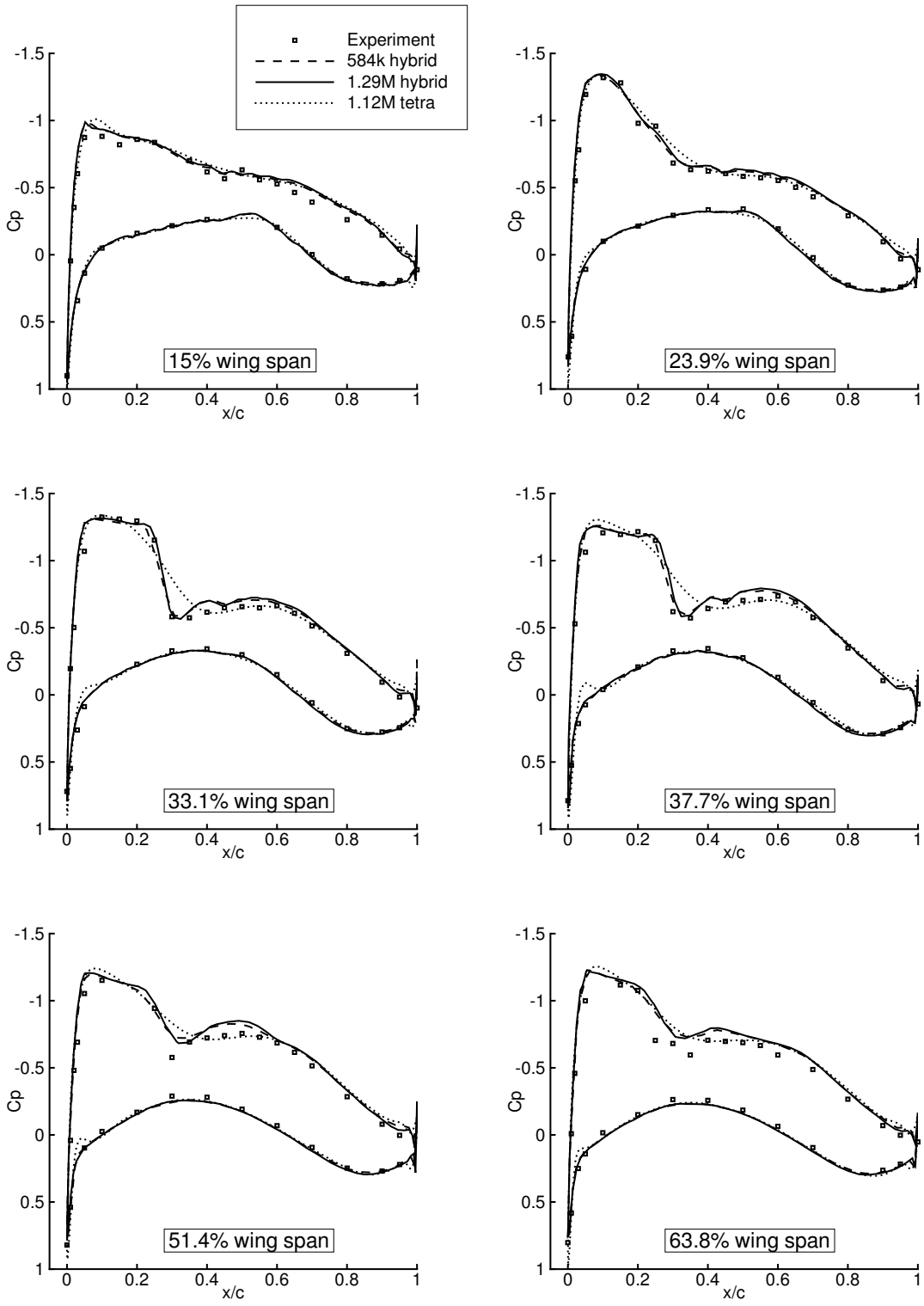

Figure D.33: Case 8a, 8b, 8c pressure coefficients. Note that the tetrahedral grid (grid 8c) does not have increased density in the shock wave region.

Appendix E

Results Interpretation Note

There are two versions of the code in the current work. The original code is denoted as the 32-bit code, which uses 32-bit integers to store memory locations. For large matrices with over two million unknowns, the numerical software PETSc [102], can no longer handle the memory locations within a matrix using these integers. PETSc provides an option to use 64-bit long integers to handle these memory locations. The resulting code is denoted as the 64-bit code.

Figure E.1 depicts the convergence using the two versions of the code. The test is run on Case 7b. The 32-bit code requires 5,565 CPU-f.e. (20.9 hours) to converge, while the 64-bit code requires 10,317 CPU-f.e. (38.9 hours) to converge. There is a factor of 1.86 increase in both CPU time and CPU-f.e. when using the latter code. This conversion factor should be taken into account when comparing convergence obtained from the two versions of the code. The cost of one RHS calculation, and the number of outer and inner iterations, are the same using the two versions of the code. It can be deduced that the increase in cost using the 64-bit code is from matrix operations used for preconditioning.

The four cases 6c, 7c, 8b, and 8c are computed using the 64-bit code in the study. The cases have grids with over one million nodes. They are marked with a † in Table 4.13. The rest of the cases are run with the 32-bit code.

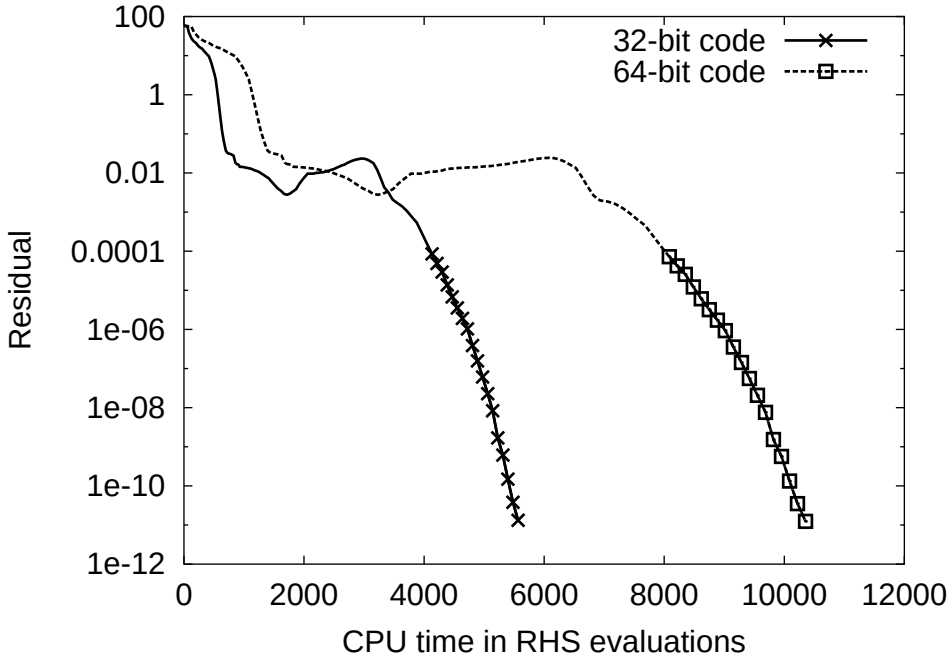

Figure E.1: A comparison of convergence using two versions of the code for interpretation of results.

Appendix F

Startup Procedure

The default startup procedure in three dimensions is:

1. Start with a first-order scheme: $\varepsilon^{(2)}$=1/4, $\varepsilon^{(4)}$=0, $\varepsilon_p^{(2)}$=$\varepsilon^{(2)}$. Converge mean-flow residual to 10^{-4}.

2. Switch to a matrix dissipation scheme: κ_2=2, κ_4=0.1, V_l=V_n=0.25.

The startup procedure for Case 8c is:

1. Start with a first-order scheme: $\varepsilon^{(2)}$=1, $\varepsilon^{(4)}$=0, $\varepsilon_p^{(2)}$=1. Switch off the turbulence model. Run 10 iterations.

2. Switch to a first-order scheme: $\varepsilon^{(2)}$=1/4, $\varepsilon^{(4)}$=0, $\varepsilon_p^{(2)}$=1/2. Switch on the turbulence model. Switch off scaling: s_r=s_c=1. Converge mean-flow residual to 10^{-4}. Switching off scaling allows larger time steps in this stage. Focusing on solving the turbulence model during this startup stage seems to provide a more stable solution development for this particular case.

3. Switch on scaling: s_r=s_c=10^{-3}. Re-converge mean-flow residual to 10^{-4}.

4. Switch to a scalar dissipation scheme: κ_2=2, κ_4=0.4, σ=10. Impose a maximum time step of Δt_{ref}=16. Run 50 iterations.

5. Switch off the maximum time step. Run 30 iterations.

6. Switch to a matrix dissipation scheme: κ_2=2, κ_4=0.1, V_l=V_n=0.25. Use σ=10, V_l=V_n=0.6 in the preconditioner. Impose a maximum time step of Δt_{ref}=256. Run 70 iterations.

7. Use V_l=V_n=0.25 in the preconditioner. Switch off the maximum time step.

The startup procedure for Case 8d is:

1. Start with a first-order scheme: $\varepsilon^{(2)}$=1/4, $\varepsilon^{(4)}$=0, $\varepsilon_p^{(2)}$=$\varepsilon^{(2)}$. Switch off the turbulence model. Run 10 iterations.

2. Switch on the turbulence model. Converge mean-flow residual to 10^{-4}.

3. Switch to a matrix dissipation scheme: κ_2=2, κ_4=0.2, V_l=V_n=0.25. Increased κ_4 and σ=8 are used. Using σ=8 is faster than using the default value of σ=10 for this case.