A Three-Dimensional Multi-block Newton-Krylov Flow Solver for the Euler Equations

by

Jason C. Nichols

A thesis submitted in conformity with the requirements
for the degree of Masters of Applied Science
Graduate Department of Aerospace Engineering
University of Toronto

Abstract

A Three-Dimensional Multi-block Newton-Krylov Flow Solver for the Euler Equations

Jason C. Nichols
Masters of Applied Science
Graduate Department of Aerospace Engineering
University of Toronto
2004

A three-dimensional multi-block Newton-Krylov flow solver for the Euler equations has been developed for steady aerodynamic flows. The solution is computed through a Jacobian-free inexact-Newton method with an approximate-Newton method for startup. The linear system at each outer iteration is solved using a Generalized Minimal Residual (GMRES) Krylov subspace algorithm. An incomplete lower/upper (ILU) factored preconditioner with reverse Cuthill-Mckee reordering is utilized to increase the efficiency of GMRES.

The parameters in the solver are optimized to provide a balance between speed and robustness. Tests are performed using a variety of flow conditions and grid sizes. Results are compared to experimental values and other established numerical solvers. The solver demonstrates fast convergence and accurate solutions on grids up to one million nodes.

Acknowledgements

The two years I have spent at the University of Toronto Institute for Aerospace Studies (UTIAS) have been an excellent experience for me. Many people have influenced me during my Masters program and I enjoyed working with all of them.

I am grateful to my thesis supervisor, Professor Zingg, for having confidence in me and my abilities. His passion, extensive knowledge, and commitment to research are what make him a leader in his field. His suggestions and insights have always motivated me to work harder and achieve all my goals. His dedication to not only me, but the entire research group is admirable.

I would like to thank all the students in the CFD lab at UTIAS for making sure there was never a dull moment. In particular, Jon Driver, for the theoretical discussions, ideas, and positive reinforcement. He is a valuable colleague and good friend. I am greatly indebted to Todd Chisholm, who on many occasions went out of his way to help me with three-dimensional grid generation. I am very grateful to Scott Northrup, for always making time to help me with numerical methods, programming or Linux issues.

I must also thank my family for always believing in me and giving me the support needed to complete this program. As well, I would like to thank my wife Melissa for providing me with a balance to the purely mathematical and scientific environment of graduate school. She has taught me many valuable lessons about setting high standards of work and always achieving my goals. I drew my strength from her and none of this work would be possible without her support.

Financial assistance from the Natural Sciences and Engineering Research Council (NSERC) and the University of Toronto is also gratefully acknowledged.

Jason Nichols
University of Toronto Institute for Aerospace Studies
July 14, 2004

Contents

List of Tables

List of Figures

List of Symbols

Alphanumeric Symbols

a	speed of sound
A, B, C	inviscid flux Jacobians
b	semi-span of the wing
C_D	coefficient of drag
C_L	coefficient of lift
C_p	coefficient of pressure
$D^{(2\xi)}$	second-difference dissipation (in the ξ-direction)
$D^{(4\xi)}$	fourth-difference dissipation (in the ξ-direction)
E	inviscid flux in x-direction
e	energy
F	inviscid flux in y-direction
G	inviscid flux in z-direction
H	total enthalpy
I	identity matrix
J	metric Jacobian
j, k, m	node coordinates in computational domain
k	level of fill-in in ILU
M	transformation matrix given by Eq. (3.40)
n	normal direction
P	matrix defined by Eq. (3.39)
p	pressure

Q vector of conservative variables

R discretized system of equations; right-hand-side

R_1, R_2, R_3, R_4, R_5 Riemann invariants

R_d drop in residual from first iteration

$R_{d_{tol}}$ parameter used to switch from the approximate-Newton method to the Jacobian-free method

t tangent direction

t time

u x-component of the velocity

U, V, W contravariant velocities

v y-component of the velocity

V_n normal component of the velocity

V_{t_1} first tangential component of the velocity

V_{t_2} second tangential component of the velocity

w z-component of the velocity

x, y, z coordinates in the physical domain

$\mathcal{A}$ Jacobian matrix of R

$\mathcal{A}_1$ first-order Jacobian

$\mathcal{B}$ discretized boundary conditions equations

$\mathcal{M}$ preconditioning matrix

$\mathcal{R}$ set of independent variables used at the boundaries

Greek Symbols

α angle of attack

α, β time step parameters

Δt local time step

Δt_{ref} time step paramter

γ ratio of specific heats

κ_2 second-difference dissipation coefficient

κ_4 fourth-difference dissipation coefficient

ρ density

σ artificial dissipation constant in $\mathcal{A}_1$; Eq. (3.35)

$\sigma^{(\xi)}$ spectral radius (in the ξ-direction)

$\tilde{\eta}$ relative reduction of residual in the nth Newton step

Υ pressure switch

ε scalar used to perturb state quantities in matrix-free GMRES; Eq. (3.51)

ε_2 second-difference dissipation coefficient (in ξ-direction)

ε_4 fourth-difference dissipation coefficient (in ξ-direction)

ε_m machine zero

ξ, η, ζ coordinate in the computational domain

Superscripts

$\bar{}$ dimensional variable

$\hat{}$ transformation to curvilinear coordinates

Subscripts

∞ freestream quantity

Chapter 1

Introduction

1.1 Motivation

Commercial aerospace industries forecast a steady growth of air traffic and replacement of ageing aircraft over the next twenty years [59]. In order to meet these demands as well as satisfy current requirements for reducing development costs, new and more efficient methods of design and optimization must be created [40]. In the years to come, a general three-dimensional full-aircraft optimization program will need to be developed and accurate and efficient solution methods will be critical in its operation. Until recently, solving the flow over a wing in three dimensions has been hindered by large grid sizes, inefficient solution methods and maximum computer clock speeds. New numerical methods and faster computers allow full solutions in three dimensions to be calculated in a reasonable amount of time [43].

Computational Fluid Dynamics (CFD) has greatly altered the path of aerospace research and design. Early aircraft design meant conservative approaches combined with time-consuming experimental testing. Current design methods include CFD as a critical component that allows the designer to make modifications and evaluate the new conditions without having to test experimentally [22, 38].

A fast, accurate and robust algorithm is necessary to solve three-dimensional aerodynamic flows for use in an optimization program. This research is aimed at designing a program that can be used as a base so that viscous effects and turbulence can be added and then used to build a full three-dimensional optimizer. The flow solver must be able to handle a variety of flight conditions with various geometries and configurations. Several strategies are presented in an attempt to balance the robustness and the speed of the flow solver.

1.2 Background

Computational fluid dynamics is used to find the solution to partial differential equations (PDE) on a computational grid, which can be either a structured or unstructured mesh. Both options have advantages and disadvantages when used in CFD. An unstructured flow solver has the ability to solve any arbitrary geometry and some examples of unstructured solvers can be found in [16, 26, 29, 42, 60]. As the name indicates, the grid nodes have no organized pattern and an important advantage to unstructured solvers is that the grid can be fit to any object. Other advantages include less complicated grid generation and easier adaptability to the flow solutions.

A structured grid consists of evenly distributed nodes that surround the geometry in an organized pattern. These grids typically have lower overhead and storage requirements, as well as higher accuracy on a given grid. The main drawback of a structured grid is the difficulty in creating a grid around arbitrary geometries. Single-block structured meshes work very well for simple aerodynamic configurations such as wings, but cannot be used in more complex geometries such as multi-element or wing-body combinations. When multiple surfaces exist and a structured mesh is to be used, a multi-block approach is taken. Using multiple blocks of structured grids combines the accuracy and storage advantages of structured solvers with the versatility of an unstructured mesh. Some examples of structured flow solvers can be found in [7, 17, 33, 37, 48, 55, 58].

Another division of CFD flow solvers can be found between implicit and explicit time-marching methods. Explicit methods are simpler to implement, require less storage space and are computational cheaper per iteration (compared to implicit methods). Explicit methods require smaller stability bounds than an implicit method and therefore require smaller time steps. This requires the program to use many iterations to converge. Improvements and additions to the basic explicit methods such as multigrid [21] have dramatically accelerated the convergence of these algorithms.

Implicit methods provide greater stability limits and thus larger time steps when compared to explicit methods [1, 16, 45]. Implicit methods require far less iterations to converge, but the cost of each iteration is higher. This is because implicit methods require the solution of a linear system of equations at each time step. A widely used and efficient implicit solver is ARC2D developed by Pulliam and Steger [48, 56]. This program uses an implicit approximate-factorization scheme originally developed by Beam and Warming [3]. Approximate-factorization and the ARC2D code are well proven techniques and are commonly used to benchmark other implicit solvers. This method has proven to be very robust, but has been found to be slower

compared to other implicit methods such as Newton's method.

As an implicit scheme, Newton's method is appealing due to its property of quadratic convergence. As mentioned earlier, a large linear system of equations must be solved at each time step, and the method to do this is critical. It is prohibitively expensive in both computational time and storage to directly solve the system, so other alternatives have been developed. An approximate-Newton method uses a simplified Jacobian, which provides the approximate linearization. An inexact-Newton method uses an iterative solver to solve the linear system. Popular iterative solvers for the inexact-Newton method are Krylov subspace methods. The Krylov method selected for this work is the Generalized Minimal Residual (GMRES) method by Saad and Schultz [53]. A Krylov subspace method uses iteration to solve linear systems by searching for the solution within a Krylov subspace [49]. It was found that using an inexact Newton's method provides a competitive alternative to using an explicit multi-grid method [34, 44]. It was also found that GMRES is much more effective when using a preconditioner [2, 15, 49]. In particular, the preconditioner is decomposed using an incomplete lower-upper (ILU) factorization routine. This routine allows the amount of fill in the factored preconditioner to be changed. Higher levels of fill make the precondtioner more accurate, but also require more storage and computational time.

Another advancement in using a Krylov solver is that GMRES only requires the matrix-vector products and does not require the Jacobian explicitly. The matrix-vector product can be approximated by a first-order forward difference of the residual vector and by doing this, the flow Jacobian does not have to be stored. The Jacobian-free method was found to be faster than the Jacobian-present method as well [25, 34]. To reduce computer storage requirements, the compressed sparse row (csr) format provided in Saad's SPARSKIT [51] is used to store the preconditioner and ILU matrices.

Flow solvers such as OPTIMA [39] and PROBE [45] make use of approximate factorization to startup and then switch to a Newton-Krylov method when the residual is reduced below a certain threshold. The advantage to approximate factorization is that it is very stable for most conditions, but the disadvantage is that this method is relatively slow compared to an inexact Newton method. The present work will not rely on approximate factorization to startup; instead a variable time step (Δt) will be applied. A series of increasing time steps based on the norm of the residual has been proven successful by authors such as Venkatakrishnan and Mavriplis [60] and Chisholm and Zingg [8].

1.3 Objectives

The primary objective of this research is to create a multi-block three-dimensional Newton-Krylov flow solver for inviscid flow using structured grids. The flow solver as presented in this work is known as TYPHOON. The project has several areas that were researched and can be broken down into the following sections:

- Multi-block setup for 3D
- Implicit numerical method
- Newton-Krylov startup procedure
- Flow solver accuracy and performance

The setup of the 3D multi-block environment is critical in the operation of the program. Block orientation and boundary condition setup were evaluated and are explained. The feasibility of a Newton-Krylov method has been well established, but the crux of the method is in the startup. A balance between the speed versus robustness of the algorithm is a challenging issue and a universal set of parameters is nearly impossible to find and depends largely on the startup procedure used. This area is explored and discussed. The accuracy of the program is determined by comparing to experimental data as well as other proven numerical algorithms. The comparison of results from an inviscid flow solver (TYPHOON) to experimental data is somewhat difficult due to viscous effects, but a general trend can be evaluated. Lastly, the memory requirements of TYPHOON are presented and compared to other solvers.

1.4 Thesis outline

This thesis is divided into six chapters. Following the introduction, the inviscid Euler equations and boundary conditions are presented. Chapter 3 is a description of the algorithm including 3D multi-block setup and the numerical method used to solve the equations. The next chapter analyzes the parameters used in the algorithm and selects the optimal combination. Chapter 5 presents the cases tested using TYPHOON and shows the results and validation, and the final chapter has the concluding remarks as well as recommendations for future research.

Chapter 2

Governing Equations

This chapter presents the governing equations for three-dimensional inviscid flow. The first section describes the equations in a Cartesian coordinate system, and the second section outlines the equations after a transformation to a curvilinear coordinate system. The boundary equations and normal/tangential velocities are presented in the last section of this chapter.

2.1 The Euler equations

The equations that define true aerodynamic flow are the Navier-Stokes equations, but a simplified set of equations called the Euler equations can be formed by neglecting the viscous terms. The Euler equations are a combination of the conservation of mass, momentum and energy equations. The conservative form of the steady three-dimensional Euler equations is:

$$\frac{\partial Q}{\partial t} + \frac{\partial E}{\partial x} + \frac{\partial F}{\partial y} + \frac{\partial G}{\partial z} = 0 \tag{2.1}$$

The non-dimensional conservative variables and inviscid fluxes are given by:

$$Q = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ e \end{pmatrix} \quad E = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(e+p) \end{pmatrix} \quad F = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ v(e+p) \end{pmatrix} \quad G = \begin{pmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ w(e+p) \end{pmatrix} \tag{2.2}$$

where p is the pressure, e is the energy, and they are related by the equation of state for a

perfect gas.

$$p = (\gamma - 1)\left(e - \frac{1}{2}\rho(u^2 + v^2 + w^2)\right) \tag{2.3}$$

Note that Eqs. 2.1–2.3 are written in a non-dimensional form. The non-dimensional variables are obtained by using the following scaling parameters

$$x = \frac{\bar{x}}{b},\quad y = \frac{\bar{y}}{b},\quad z = \frac{\bar{z}}{b},\quad \rho = \frac{\bar{\rho}}{\rho_\infty},\quad u = \frac{\bar{u}}{a_\infty},\quad v = \frac{\bar{v}}{a_\infty},\quad w = \frac{\bar{w}}{a_\infty},\quad e = \frac{\bar{e}}{\rho_\infty a_\infty^2},\quad t = \frac{\bar{t}\,a_\infty}{b} \tag{2.4}$$

where the bar symbol denotes dimensional variables, ∞ refers to freestream quantities and b is the semi-span length of the wing. The ratio of specific heats, γ, is 1.4 for air under the conditions of interest here.

2.2 Curvilinear coordinate transformation

The objective in designing a three-dimensional structured grid is to use coordinate mappings of the governing equations to bring all body surfaces onto coordinate surfaces. A general transformation is used to map the curvilinear grid into a computational domain where the spacing is uniform and equal to one. Since the curvilinear spacing is equal to one, a grid node at any location will be given as (j, k, m). Therefore, in the transformed plane, standard unweighted spatial discretization can be used. Following the 3D coordinate transformations used by Pulliam and Steger [50], the Cartesian components are transformed into curvilinear space as follows:

$$\tau = t, \qquad \xi = \xi(x, y, z), \qquad \eta = \eta(x, y, z), \qquad \zeta = \zeta(x, y, z)$$

The Euler equations rewritten in generalized curvilinear coordinates are given by

$$\frac{\partial \widehat{Q}}{\partial \tau} + \frac{\partial \widehat{E}}{\partial \xi} + \frac{\partial \widehat{F}}{\partial \eta} + \frac{\partial \widehat{G}}{\partial \zeta} = 0 \tag{2.5}$$

where

$$\widehat{Q} = J^{-1}Q \tag{2.6}$$

and

$$\widehat{E} = J^{-1}\begin{pmatrix} \rho U \\ \rho u U + \xi_x p \\ \rho v U + \xi_y p \\ \rho w U + \xi_z p \\ (e+p)U - \xi_t p \end{pmatrix} \quad \widehat{F} = J^{-1}\begin{pmatrix} \rho V \\ \rho u V + \eta_x p \\ \rho v V + \eta_y p \\ \rho w V + \eta_z p \\ (e+p)V - \eta_t p \end{pmatrix} \quad \widehat{G} = J^{-1}\begin{pmatrix} \rho W \\ \rho u W + \zeta_x p \\ \rho v W + \zeta_y p \\ \rho w W + \zeta_z p \\ (e+p)W - \zeta_t p \end{pmatrix} \tag{2.7}$$

The contravariant velocity components are

$$\begin{aligned} U &= \xi_t + \xi_x u + \xi_y v + \xi_z w \\ V &= \eta_t + \eta_x u + \eta_y v + \eta_z w \\ W &= \zeta_t + \zeta_x u + \zeta_y v + \zeta_z w \end{aligned} \tag{2.8}$$

and the metric Jacobian of the transformation is

$$J^{-1} = (x_\xi y_\eta z_\zeta + x_\zeta y_\xi z_\eta + x_\eta y_\zeta z_\xi - x_\xi y_\zeta z_\eta - x_\eta y_\xi z_\zeta - x_\zeta y_\eta z_\xi) \tag{2.9}$$

2.2.1 Grid metrics

Transforming the grid points from physical to computational space involves calculation of nine metric terms. The metric terms are found from chain rule expansions of (x_ξ, y_ξ, z_ξ, etc.) and solved for (ξ_x, ξ_y, ξ_z, etc.) to give

$$\begin{aligned} \xi_x &= J\,(y_\eta z_\zeta - y_\zeta z_\eta) & \eta_x &= J\,(z_\xi y_\zeta - y_\xi z_\zeta) & \zeta_x &= J\,(y_\xi z_\eta - z_\xi y_\eta) \\ \xi_y &= J\,(z_\eta x_\zeta - x_\eta z_\zeta) & \eta_y &= J\,(x_\xi z_\zeta - x_\zeta z_\xi) & \zeta_y &= J\,(x_\eta z_\xi - x_\xi z_\eta) \\ \xi_z &= J\,(x_\eta y_\zeta - y_\eta x_\zeta) & \eta_z &= J\,(y_\xi x_\zeta - x_\xi y_\zeta) & \zeta_z &= J\,(x_\xi y_\eta - y_\xi x_\eta) \end{aligned} \tag{2.10}$$

In 2D, centered differencing can be used to calculate the metrics, and the metric invariants will automatically be satisfied. If centered differencing is used to calculate the metrics in three-dimensions, the metric invariants will not be satisfied. As indicated by Pulliam [48] and Steger [50], averaging the central differences of the grid values (x_ξ, x_η, x_ζ etc.) and then calculating the metrics and metric Jacobian will satisfy the metric invariants. For example, ξ_x would be calculated as:

$$\xi_x = J\,[(\mu_\zeta \delta_\eta y)(\mu_\eta \delta_\zeta z) - (\mu_\eta \delta_\zeta y)(\mu_\zeta \delta_\eta z)] \tag{2.11}$$

where δ is the central difference operator and μ is an average operator.

$$\mu_\eta x_k = \frac{1}{2}\,(x_{k+1} + x_{k-1}) \tag{2.12}$$

2.3 Boundary conditions

The boundaries consist of the body surface and the far-field regions of the physical space. In the multi-block setup, each block has six faces, and each face will be designated either a body surface, far-field or block interface. Figure 2.1 shows an H-H grid around a wing and Figure 2.2 shows the surface grid from the top view (a) and front view (b). This grid is divided up into 12 blocks and is shown as a schematic in Figure 2.3. Calculation of a normal and two tangential

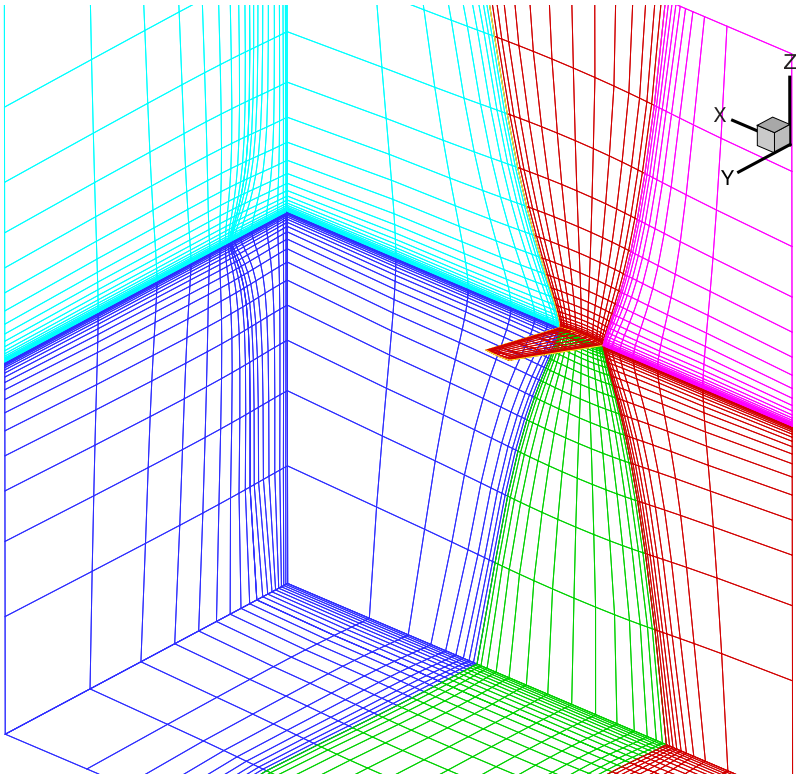
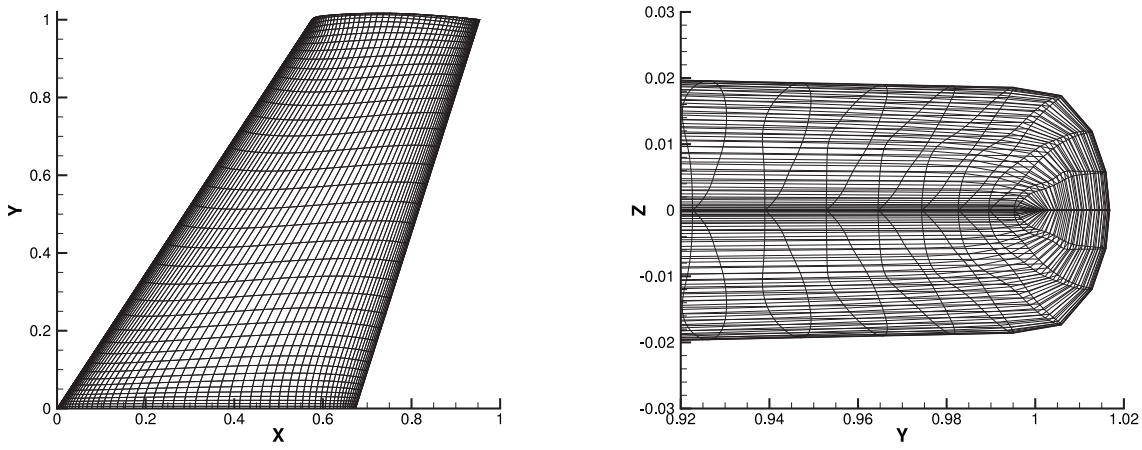


Figure 2.1: 3D 12-block wing grid (28,000 nodes)



(a) Top view of wing grid

(b) Front view of wing tip

Figure 2.2: ONERA M6 wing grid

velocities is necessary for use in the boundary conditions and is outlined in Vinokur [61]. These velocities combined with Riemann invariants and inviscid flow assumptions form the basis of the boundary conditions.

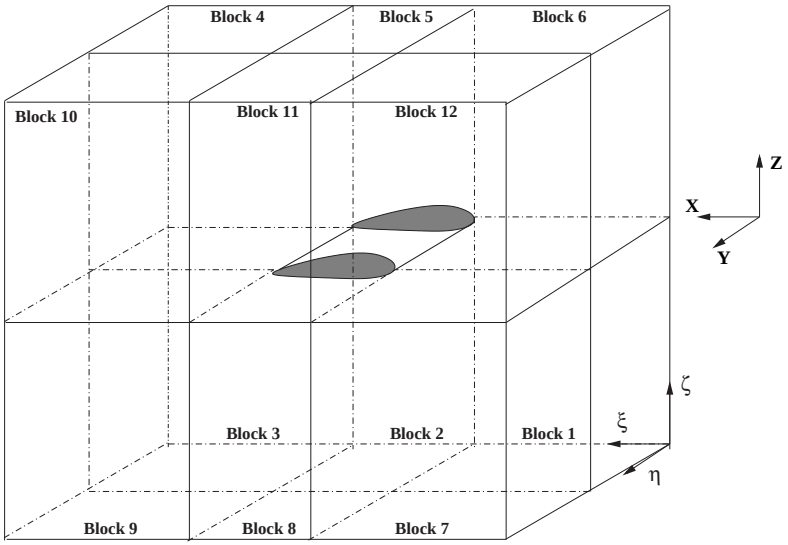


Figure 2.3: 3D 12-block wing schematic

2.3.1 Normal and tangential velocity components

Any given point in space will have a normal and two tangential components to a given plane, which are used in determining inflow or outflow at the computational boundaries. In three dimensions, the two tangential components form a plane which lies on the surface of any given point. The normal (V_n) and tangential (V_{t_1},V_{t_2}) velocities are calculated in terms of the Cartesian velocities (u, v, w) and the grid metrics. V_n, V_{t_1} and V_{t_2} will be different for each plane (ξ, η, ζ). A mix of contravariant and covariant components make up the normal and tangential velocities:

- Contravariant components (tangent to ξ, η, ζ axes)
- Covariant components (normal to ξ, η, ζ axes)

For example, if $\widehat{R}$ is a velocity vector it would be defined by:

$$\widehat{R} = \underbrace{u\hat{i} + v\hat{j} + w\hat{k}}_{Cartesian} = \underbrace{C_1\hat{e}_1 + C_2\hat{e}_2 + C_3\hat{e}_3}_{Contravariant} = \underbrace{c_1\widehat{E}_1 + c_2\widehat{E}_2 + c_3\widehat{E}_3}_{Covariant}$$

where

$$\hat{e}_1 = \frac{x_\xi\hat{i} + y_\xi\hat{j} + z_\xi\hat{k}}{\sqrt{x_\xi^2 + y_\xi^2 + z_\xi^2}} \qquad \hat{e}_2 = \frac{x_\eta\hat{i} + y_\eta\hat{j} + z_\eta\hat{k}}{\sqrt{x_\eta^2 + y_\eta^2 + z_\eta^2}} \qquad \hat{e}_3 = \frac{x_\zeta\hat{i} + y_\zeta\hat{j} + z_\zeta\hat{k}}{\sqrt{x_\zeta^2 + y_\zeta^2 + z_\zeta^2}}$$

$$\widehat{E}_1 = \frac{\xi_x\hat{i} + \xi_y\hat{j} + \xi_z\hat{k}}{\sqrt{\xi_x^2 + \xi_y^2 + \xi_z^2}} \qquad \widehat{E}_2 = \frac{\eta_x\hat{i} + \eta_y\hat{j} + \eta_z\hat{k}}{\sqrt{\eta_x^2 + \eta_y^2 + \eta_z^2}} \qquad \widehat{E}_3 = \frac{\zeta_x\hat{i} + \zeta_y\hat{j} + \zeta_z\hat{k}}{\sqrt{\zeta_x^2 + \zeta_y^2 + \zeta_z^2}}$$

boundary	inflow						outflow					
	V_n	R_1	R_2	R_3	R_4	R_5	V_n	R_1	R_2	R_3	R_4	R_5
$\psi = 1$	> 0	∞	int	∞	∞	∞	< 0	∞	int	int	int	int
$\psi = \psi_{max}$	< 0	int	∞	∞	∞	∞	> 0	int	∞	int	int	int

ψ=j, k or m, ∞: free-stream values, int: extrapolated from interior

Table 2.1: Subsonic inflow and outflow far-field boundary conditions

On the body surface ($\zeta = constant$) the velocities are defined by:

$$u\hat{i} + v\hat{j} + w\hat{k} = V_{t_1}\hat{e}_1 + V_{t_2}\hat{e}_2 + V_n\widehat{E}_3$$

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \underbrace{\begin{bmatrix} \frac{x_\xi}{\sqrt{x_\xi^2 + y_\xi^2 + z_\xi^2}} & \frac{x_\eta}{\sqrt{x_\eta^2 + y_\eta^2 + z_\eta^2}} & \frac{\zeta_x}{\sqrt{\zeta_x^2 + \zeta_y^2 + \zeta_z^2}} \\ \frac{y_\xi}{\sqrt{x_\xi^2 + y_\xi^2 + z_\xi^2}} & \frac{y_\eta}{\sqrt{x_\eta^2 + y_\eta^2 + z_\eta^2}} & \frac{\zeta_y}{\sqrt{\zeta_x^2 + \zeta_y^2 + \zeta_z^2}} \\ \frac{z_\xi}{\sqrt{x_\xi^2 + y_\xi^2 + z_\xi^2}} & \frac{z_\eta}{\sqrt{x_\eta^2 + y_\eta^2 + z_\eta^2}} & \frac{\zeta_z}{\sqrt{\zeta_x^2 + \zeta_y^2 + \zeta_z^2}} \end{bmatrix}}_{\mathcal{B}} \begin{pmatrix} V_{t_1} \\ V_{t_2} \\ V_n \end{pmatrix} \tag{2.13}$$

Values for V_n, V_{t_1} and V_{t_2} can be determined by substituting in for x_ξ, x_η etc., using the curvilinear metrics (eg. $x_\xi = J(\eta_y\zeta_z - \zeta_y\eta_z)$) and inverting the $\mathcal{B}$ matrix. A similar approach is taken to calculate the normal and tangent velocities on the ξ and η planes as well.

2.3.2 Body surface

Five conditions must be specified at each boundary surface. Since a solution to the inviscid Euler equations is desired, a flow tangency (or slip) condition is applied at a solid body surface. All five conditions are summarized below; further details are given in Section 3.2.3.

- Normal velocity is set to zero ($V_n = 0$)
- Tangential velocities (V_{t_1}, V_{t_2}) are extrapolated from the interior
- Pressure is extrapolated from the interior
- Stagnation enthalpy ($\frac{e+p}{\rho}$), is set to the freestream value (H_∞)

2.3.3 Far-field boundaries

Locally one-dimensional Riemann invariants combined with V_{t_1}, V_{t_2} and the entropy are used at the far-field boundaries [4, 20]. These are given by:

$$R_1 = V_n - \frac{2a}{\gamma - 1} \qquad R_2 = V_n + \frac{2a}{\gamma - 1} \qquad R_3 = \frac{\rho^\gamma}{p} \qquad R_4 = V_{t_1} \qquad R_5 = V_{t_2} \tag{2.14}$$

They are either extrapolated from the interior or set to the freestream values depending on the direction of the flow (or the sign of the normal velocity V_n). Table 2.1 details the logic at the inflow or outflow. Details of far-field boundary condition implementation are given in Section 3.2.3.

Chapter 3

Algorithm Description

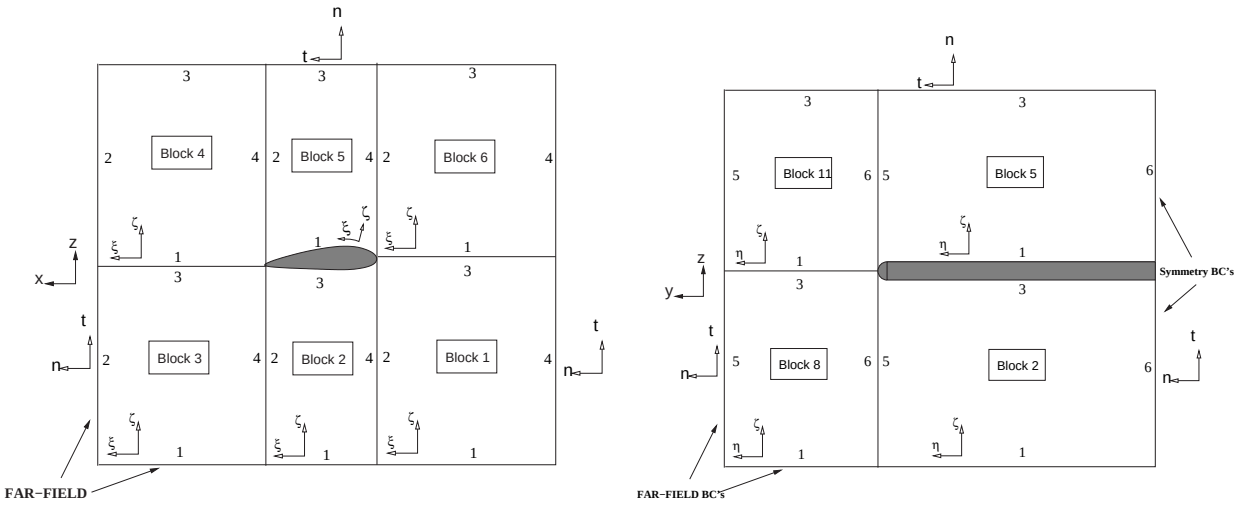
TYPHOON is designed to be a three-dimensional flow solver for the Euler equations on structured grids. The algorithm solves the governing equations using a central-differenced preconditioned Newton-Krylov method with artificial dissipation. This chapter discusses the spatial discretization, time-marching method, startup and multi-block capabilites of the TYPHOON flow solver.

3.1 Three-dimensional multi-block

The structured grid is divided into blocks that are interconnected by the block faces. The setup is illustrated in Figures 2.1 and 2.3. If a slice of Figure 2.3 is taken on the $\eta = 1$ plane the result is shown in Figure 3.1(a) and a slice on the $\xi = 1$ plane is shown in Figure 3.1(b). An important note is that the normal vector always points in the direction of the curvilinear coordinate at that point. For example, the normal on Block 3 side 2 in Figure 3.1(a) points in the direction of positive ξ.

3.1.1 Block orientation

TYPHOON reads the grid file and a connectivity file that specifies the orientation of the blocks, block faces and boundary conditions. The blocks can be numbered in any order but the block faces have a particular setup according to the coordinate axis (ξ, η, ζ). The convention for block face numbering is shown in Figure 3.2 as well as in Table 3.1. A similar setup is used in the 2D flow solver TORNADO [30, 41, 62].



(a) Side view (Root plane)

(b) Front view (Leading edge plane)

Figure 3.1: 2D slices of 3D 12-block wing schematic

Block face	Coordinate
1	$\zeta = \zeta_{min}$
2	$\xi = \xi_{max}$
3	$\zeta = \zeta_{max}$
4	$\xi = \xi_{min}$
5	$\eta = \eta_{max}$
6	$\eta = \eta_{min}$

Table 3.1: 3D block orientation table

3.1.2 Halo points

The face where two blocks meet is known as a block interface. The block faces share physical space but exist as two separate computational planes of points. Calculations of the equations are completed inside each block separately and block to block communication is carried out using temporary or halo nodes that hold information from the neighbouring block. This method of information transfer has proven sucessful by authors such as De Rango [11], De Rango and Zingg [12], Epstein and Seror [14] and Kim and Lee [24]. Figure 3.3 shows a two-dimensional slice of a block and illustrates how halo points are used. The equations are not solved on the halo points, they are only used to provide the current block with information about the neighbouring block.

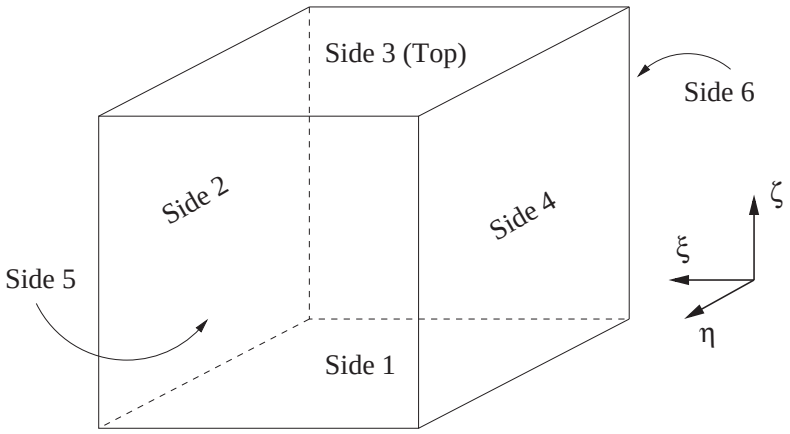


Figure 3.2: 3D block orientation

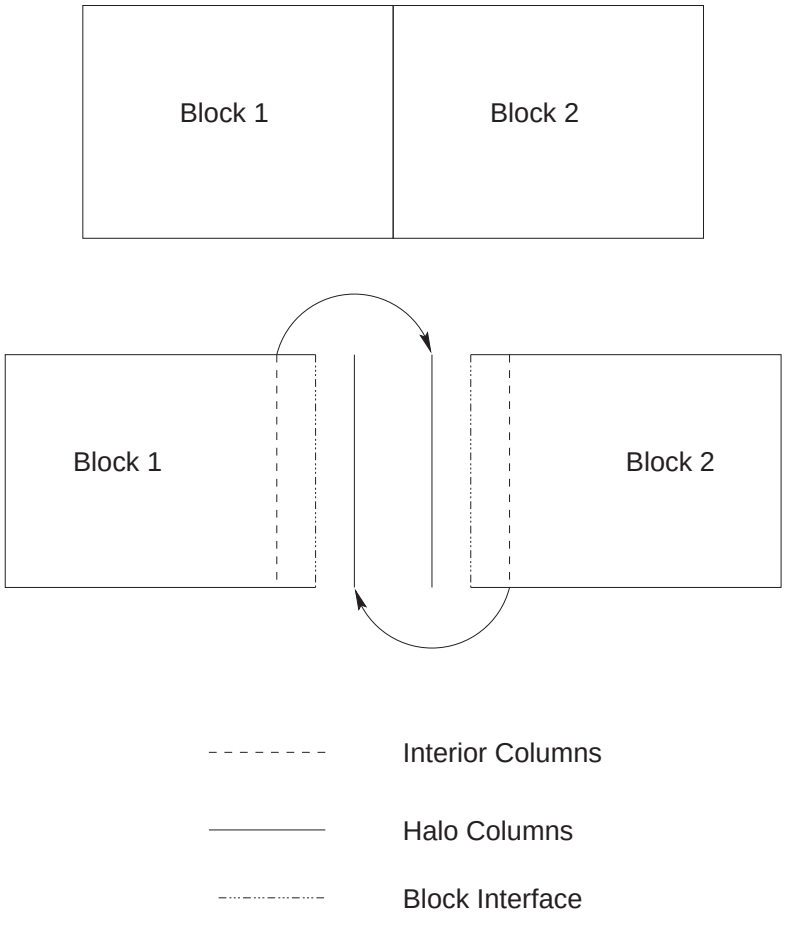


Figure 3.3: 2D slice of block and halo points

3.1.3 Block-to-block grid continuity

In the multi-block arrangement in TYPHOON, continuity of the grid lines between blocks is important. Points in two different blocks that lie on a block interface must share a common physical coordinate (point continuity). As well, by ensuring orthogonality of the grid and equal spacing of points on either side of a block interface, slope continuity can be maintained. Solvers have been designed to handle grids without slope or point continuity. This involves interpolating data between blocks, which creates extra overhead for the program as well as the possibility of de-stabilizing of the solution process due to problems with block continuity. TYPHOON requires both slope and point continuity in the input grids. Grid generation plays a critical role in solving the equations and can drastically affect the convergence of the solver.

3.2 Spatial discretization

The partial differential equations given in Chapter 2 are converted to ordinary differential equations by discretization in space. The area surrounding the wing is divided up into a grid and the equations are solved at each node in the grid. Since the computational grid has equal spacing, the variable $\widehat{Q}$ at any node (j,k,m) is given by:

$$\widehat{Q}_{j,k,m} = \widehat{Q}(j\Delta\xi, k\Delta\eta, m\Delta\zeta) = \widehat{Q}(j,k,m) \tag{3.1}$$

3.2.1 Finite-difference

Spatial discretization in the system is performed using second order centered-differences combined with second and fourth-order dissipation. When centered differencing is applied to each term of Eq. (2.5), the following equations are generated.

$$\frac{\partial \widehat{E}}{\partial \xi} \approx \frac{\widehat{E}_{j+1,k,m} - \widehat{E}_{j-1,k,m}}{2} - \nabla_\xi A_{\mathrm{D}_{j+\frac{1}{2}}} \tag{3.2}$$

$$\frac{\partial \widehat{F}}{\partial \eta} \approx \frac{\widehat{F}_{j,k+1,m} - \widehat{F}_{j,k-1,m}}{2} - \nabla_\eta A_{\mathrm{D}_{k+\frac{1}{2}}} \tag{3.3}$$

$$\frac{\partial \widehat{G}}{\partial \zeta} \approx \frac{\widehat{G}_{j,k,m+1} - \widehat{G}_{j,k,m-1}}{2} - \nabla_\zeta A_{\mathrm{D}_{m+\frac{1}{2}}} \tag{3.4}$$

where A_{D} is the artificial dissipation, and ∇ is the backward difference operator.

3.2.2 Artificial dissipation

Central difference schemes experience high frequency oscillations which need to be damped in order to converge. Oscillations in the vicinity of shocks also need to be eliminated and this is accomplished by adding artificial dissipation to the system. Several methods of dissipation are available, but the second and fourth-order scalar dissipation method by Jameson [23] was selected for this algorithm.

The description of the dissipation terms is presented using the ξ direction terms but similar expressions for the η and ζ directions exist. Following Eq. (3.2), the dissipation term is written as:

$$A_{\mathrm{D}_{j+\frac{1}{2}}} = D^{(2\xi)}_{j+\frac{1}{2},k,m} + D^{(4\xi)}_{j+\frac{1}{2},k,m} \tag{3.5}$$

$$D^{(2\xi)}_{j+\frac{1}{2},k,m} = d^{(2)}_{j+\frac{1}{2},k,m}\, \nabla_\xi \left(J_{j,k,m} \widehat{Q}_{j,k,m} \right) \tag{3.6}$$

$$D^{(4\xi)}_{j+\frac{1}{2},k,m} = d^{(4)}_{j+\frac{1}{2},k,m}\, \nabla_\xi \Delta_\xi \nabla_\xi \left(J_{j,k,m} \widehat{Q}_{j,k,m} \right) \tag{3.7}$$

The second-difference dissipation coefficients are:

$$d^{(2)}_{j+\frac{1}{2},k,m} = 2 \left(\epsilon\, \sigma^{(\xi)}\, J^{-1} \right)_{j+\frac{1}{2},k,m} \tag{3.8}$$

where $\sigma^{(\xi)}$ is the spectral radius of the flux jacobian $\frac{\partial \widehat{E}}{\partial \widehat{Q}}$

$$\sigma^{(\xi)} = |U| + a\sqrt{\xi_x^2 + \xi_y^2 + \xi_z^2} \tag{3.9}$$

and ϵ is given by

$$\epsilon_{j,k,m} = \kappa_2 \left[0.5 \Upsilon^*_{j,k,m} + 0.25 \left(\Upsilon^*_{j+1,k,m} + \Upsilon^*_{j-1,k,m} \right) \right] \tag{3.10}$$

$$\Upsilon^*_{j,k,m} = \max \left(\Upsilon_{j+1,k,m}, \Upsilon_{j,k,m}, \Upsilon_{j-1,k,m} \right) \tag{3.11}$$

$$\Upsilon_{j,k,m} = \frac{|p_{j+1,k,m} - 2p_{j,k,m} + p_{j-1,k,m}|}{|p_{j+1,k,m} + 2p_{j,k,m} + p_{j-1,k,m}|} \tag{3.12}$$

Υ is a sensor designed to activate near shocks (large second difference in pressure), and Υ^* is used to smooth out the values of Υ. Analogous terms appear in the η and ζ direction; however, following Nemec [39], Eq. (3.11) is not used in the ζ direction.

The fourth-difference dissipation coefficients are:

$$d^{(4)}_{j+\frac{1}{2},k,m} = \max\left[0, 2\,\kappa_4\left(\sigma^{(\xi)}\,J^{-1}\right)_{j+\frac{1}{2},k,m} - d^{(2)}_{j+\frac{1}{2},k,m}\right] \tag{3.13}$$

Eq. (3.13) is designed to switch off the fourth-difference dissipation near the shock. The operators Δ_ξ and ∇_ξ represent first-order forward and backward difference operators as shown below:

$$\begin{aligned}\Delta_\xi\,(\cdot) &= (\cdot)_{j+1,k,m} - (\cdot)_{j,k,m}\\ \nabla_\xi\,(\cdot) &= (\cdot)_{j,k,m} - (\cdot)_{j-1,k,m}\end{aligned} \tag{3.14}$$

Typical values of κ_2 and κ_4 are 1.0 and 0.02, respectively. Values at half nodes are calculated using arithmetic averages given by

$$(\cdot)_{j+\frac{1}{2},k,m} = \frac{(\cdot)_{j+1,k,m} + (\cdot)_{j,k,m}}{2} \tag{3.15}$$

3.2.3 Boundary conditions

Surface boundaries

The boundary equations for the body surface are explained in Section 2.3.2. Since the flow is inviscid, first-order extrapolation of V_{t_1}, V_{t_2} and pressure (p) is used. Physical values of normal and tangential velocity (not scaled by the metric Jacobian J^{-1}) are used. For illustrative purposes the surface boundary conditions at the $m = 1$ plane are given by the following:

$$(V_n)_{j,k,1} = 0 \tag{3.16}$$

$$(V_{t_1})_{j,k,1} - 2(V_{t_1})_{j,k,2} + (V_{t_1})_{j,k,3} = 0 \tag{3.17}$$

$$(V_{t_2})_{j,k,1} - 2(V_{t_2})_{j,k,2} + (V_{t_2})_{j,k,3} = 0 \tag{3.18}$$

$$p_{j,k,1} - 2p_{j,k,2} + p_{j,k,3} = 0 \tag{3.19}$$

$$\left(\tfrac{\gamma}{\gamma-1}p + \tfrac{1}{2}\rho(u^2+v^2+w^2) - \rho H_\infty\right)_{j,k,1} = 0 \tag{3.20}$$

Far-field boundaries

For an outflow condition, a zeroth-order extrapolation is used in calculating the far-field boundary conditions. An inflow condition is calculated by setting the far-field nodes to the free stream values of the invariant. As shown in Table 2.1 the following equations are used on the $m = m_{max}$

far-field boundary:

$$R_1: \quad \left(V_n - \tfrac{2}{\gamma-1}\sqrt{\tfrac{\gamma p}{\rho}}\right)_{j,k,m_{max}} - \left(V_n - \tfrac{2}{\gamma-1}\sqrt{\tfrac{\gamma p}{\rho}}\right)_{\infty} = 0 \tag{3.21}$$

$$R_2: \quad \left(V_n + \tfrac{2}{\gamma-1}\sqrt{\tfrac{\gamma p}{\rho}}\right)_{j,k,m_{max}} - \left(V_n + \tfrac{2}{\gamma-1}\sqrt{\tfrac{\gamma p}{\rho}}\right)_{j,k,m_{max}-1} = 0 \tag{3.22}$$

The three remaining equations depend on an inflow or outflow through the boundary face. For an inflow condition:

$$R_3: \quad \left(\tfrac{\rho^\gamma}{p}\right)_{j,k,m_{max}} - \left(\tfrac{\rho^\gamma}{p}\right)_{\infty} = 0 \tag{3.23}$$

$$R_4: \quad (V_{t_1})_{j,k,m_{max}} - (V_{t_1})_{\infty} = 0 \tag{3.24}$$

$$R_5: \quad (V_{t_2})_{j,k,m_{max}} - (V_{t_2})_{\infty} = 0 \tag{3.25}$$

and for an outflow condition, Eqs. (3.23), (3.24) and (3.25) are replaced by:

$$R_3: \quad \left(\tfrac{\rho^\gamma}{p}\right)_{j,k,m_{max}} - \left(\tfrac{\rho^\gamma}{p}\right)_{j,k,m_{max}-1} = 0 \tag{3.26}$$

$$R_4: \quad (V_{t_1})_{j,k,m_{max}} - (V_{t_1})_{j,k,m_{max}-1} = 0 \tag{3.27}$$

$$R_5: \quad (V_{t_2})_{j,k,m_{max}} - (V_{t_2})_{j,k,m_{max}-1} = 0 \tag{3.28}$$

3.2.4 Block interfaces

Coincident points in the grid occupy the same point in physical space, but have different locations in the computational space. Block interfaces in TYPHOON are coincident, but do not require any special treatment since the block faces are contained in the interior scheme. The nodes on these planes will follow Eqs. (3.2) to (3.4) for discretization. This method of handling block interfaces requires the use of the halo nodes for communication between blocks, but compared to a block face averaging technique the interface condition provides a more accurate solution.

3.3 Time marching and Newton's method

After spatial discretization of the curvilinear Euler equations shown in Eq. (2.5), the resulting system has the form

$$\frac{d\widehat{Q}}{dt} + R(\widehat{Q}) = 0 \tag{3.29}$$

If implicit Euler time-marching is applied the following equation results

$$\widehat{Q}^{n+1} - \widehat{Q}^n + \Delta t \left(\widehat{E}_\xi^{n+1} + \widehat{F}_\eta^{n+1} + \widehat{G}_\zeta^{n+1}\right) = 0 \tag{3.30}$$

As presented in [48] the goal of the solver is to solve Eq. (3.30) for $\widehat{Q}^{n+1}$ using $\widehat{Q}^n$. Local time linearizations are necessary because the flux vectors $\widehat{E}$, $\widehat{F}$ and $\widehat{G}$ are nonlinear functions of $\widehat{Q}$. Using a Taylor series expansion about $\widehat{Q}^n$ [28] and neglecting second-order and higher terms yield

$$\begin{aligned} \widehat{E}^{n+1} &\approx \widehat{E}^n + \widehat{A}^n \Delta\widehat{Q}^n \\ \widehat{F}^{n+1} &\approx \widehat{F}^n + \widehat{B}^n \Delta\widehat{Q}^n \\ \widehat{G}^{n+1} &\approx \widehat{G}^n + \widehat{C}^n \Delta\widehat{Q}^n \end{aligned} \tag{3.31}$$

where $\widehat{A} = \frac{\partial \widehat{E}}{\partial \widehat{Q}}$, $\widehat{B} = \frac{\partial \widehat{F}}{\partial \widehat{Q}}$ and $\widehat{C} = \frac{\partial \widehat{G}}{\partial \widehat{Q}}$. The flux Jacobian matrices for $\widehat{A}$, $\widehat{B}$ and $\widehat{C}$ are given by

$$\begin{bmatrix} 0 & \kappa_x & \kappa_y & \kappa_z & 0 \\ \kappa_x\phi^2 - u\theta & \theta - (\gamma-2)\kappa_x u & \kappa_y u - (\gamma-1)\kappa_x v & \kappa_z u - (\gamma-1)\kappa_x w & (\gamma-1)\kappa_x \\ \kappa_y\phi^2 - v\theta & \kappa_x v - (\gamma-1)\kappa_y u & \theta - (\gamma-2)\kappa_y v & \kappa_z v - (\gamma-1)\kappa_y w & (\gamma-1)\kappa_y \\ \kappa_z\phi^2 - w\theta & \kappa_x w - (\gamma-1)\kappa_z u & \kappa_y v - (\gamma-1)\kappa_z v & \theta - (\gamma-2)\kappa_z w & (\gamma-1)\kappa_z \\ \theta\left(\phi^2 - \psi\right) & \kappa_x\psi - (\gamma-1)u\theta & \kappa_y\psi - (\gamma-1)v\theta & \kappa_z\psi - (\gamma-1)w\theta & \gamma\theta \end{bmatrix} \tag{3.32}$$

where $\kappa = \xi$ for $\widehat{A}$, $\kappa = \eta$ for $\widehat{B}$ and $\kappa = \zeta$ for $\widehat{C}$. The other entries used in the matrix are

$$\begin{aligned} \psi &= \gamma\frac{e}{\rho} - \phi^2 \\ \theta &= \kappa_x u + \kappa_y v + \kappa_z w \\ \phi^2 &= \frac{1}{2}(\gamma-1)\left(u^2 + v^2 + w^2\right) \end{aligned}$$

If Eq. (3.31) is applied to Eq. (3.30) we get

$$\left[\frac{I}{\Delta t} + \underbrace{\partial_\xi \widehat{A} + \partial_\eta \widehat{B} + \partial_\zeta \widehat{C}}_{\mathcal{A}}\right]^{(n)} \Delta\widehat{Q}^{(n)} = -\left[\underbrace{\partial_\xi \widehat{E} + \partial_\eta \widehat{F} + \partial_\zeta \widehat{G}}_{R}\right]^{(n)} \tag{3.33}$$

which can be rewritten as

$$\left[\frac{I}{\Delta t} + \mathcal{A}\right]^{(n)} \Delta\widehat{Q}^{(n)} = -R^{(n)} \tag{3.34}$$

where $\mathcal{A}$ is the flow Jacobian and Δt is the time step, which is explained in Section 3.6. It can be noted that the implicit Euler time-marching method with local time-linearization yields Newton's method as $\Delta t \to \infty$.

In three dimensions, $\mathcal{A}$ requires thirteen 5 x 5 blocks per row in the matrix. This stencil is expensive to compute and very large to store. If a first-order approximation to $\mathcal{A}$ is used, only seven 5 x 5 blocks per row in the matrix are needed. Such a method is called an approximate-Newton method with the first-order Jacobian designated $\mathcal{A}_1$. As well, the first-order approximation to

the Jacobian is better conditioned and more diagonally dominant compared to the exact Jacobian. In the first-order Jacobian $\mathcal{A}_1$, the fourth-order terms are combined with the second-order terms by the following equation

$$\varepsilon_2^{(l)} = \varepsilon_2^{(r)} + \sigma\varepsilon_4^{(r)} \tag{3.35}$$

where the superscript r denotes values on the right-hand side and l on the left. An optimal value of the constant σ will be determined through numerical experiments. The non-linear Newton iterations Eq. (3.30) will be referred to the "outer iterations" and the linear iterations Eq. (3.34) as the "inner iterations".

3.3.1 Linearization of the boundary conditions

The boundary conditions must be treated implicitly to achieve a Newton convergence rate. The following sections parallel work presented in 2D by Pueyo [45]. The boundary condition equations were presented in Section 2.3 and can be written as

$$\mathcal{B}(\mathcal{R}) = 0 \tag{3.36}$$

where $\mathcal{R}$ is the set of variables chosen to write the equations.

$$\mathcal{R} = \begin{bmatrix} \rho \\ u \\ v \\ w \\ p \end{bmatrix} \tag{3.37}$$

If Newton's linearization is applied to Eq. (3.36), we obtain

$$P\Delta\mathcal{R} = -\mathcal{B}^{(n)} \tag{3.38}$$

where

$$P = \left(\frac{\partial\mathcal{B}}{\partial\mathcal{R}}\right)^{(n)} \tag{3.39}$$

Since the unknowns in the global system are $\Delta\widehat{Q}$, we have to change variables in Eq. (3.38). The Jacobian matrix $M = \frac{\partial Q}{\partial\mathcal{R}}$ is the transformation between the variables ΔQ and $\Delta\mathcal{R}$. It is

defined by

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ u & \rho & 0 & 0 & 0 \\ v & 0 & \rho & 0 & 0 \\ w & 0 & 0 & \rho & 0 \\ \frac{u^2+v^2+w^2}{2} & \rho u & \rho v & \rho v & \frac{1}{\gamma-1} \end{bmatrix} \tag{3.40}$$

Therefore, Eq. (3.38) becomes

$$PM^{-1}J\Delta\widehat{Q} = -\mathcal{B}^{(n)} \tag{3.41}$$

where

$$M^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -\frac{u}{\rho} & \frac{1}{\rho} & 0 & 0 & 0 \\ -\frac{v}{\rho} & 0 & \frac{1}{\rho} & 0 & 0 \\ -\frac{w}{\rho} & 0 & 0 & \frac{1}{\rho} & 0 \\ (\gamma-1)\frac{u^2+v^2+w^2}{2} & -(\gamma-1)u & -(\gamma-1)v & -(\gamma-1)w & \gamma-1 \end{bmatrix} \tag{3.42}$$

Body surface

For inviscid flow at a body surface, a tangent or slip condition is applied to the surface nodes. A first-order extrapolation is used. For illustrative purposes the body conditions are written for a surface on the $m = 1$ plane, which corresponds to side 1 of the computational cube (top surface of the wing).

$$\left[PM^{-1}J\Delta\widehat{Q}\right]_{j,k,1} - 2\left[PM^{-1}J\Delta\widehat{Q}\right]_{j,k,2} + \left[PM^{-1}J\Delta\widehat{Q}\right]_{j,k,3} = -\mathcal{B}^{(n)} \tag{3.43}$$

where

$$P_{j,k,1} = \begin{bmatrix} 0 & \frac{\partial V_n}{\partial u} & \frac{\partial V_n}{\partial v} & \frac{\partial V_n}{\partial w} & 0 \\ 0 & \frac{\partial V_{t_1}}{\partial u} & \frac{\partial V_{t_1}}{\partial v} & \frac{\partial V_{t_1}}{\partial w} & 0 \\ 0 & \frac{\partial V_{t_2}}{\partial u} & \frac{\partial V_{t_2}}{\partial v} & \frac{\partial V_{t_2}}{\partial w} & 0 \\ 0 & 0 & 0 & 0 & 1 \\ \frac{1}{2}(u^2+v^2+w^2) - H_\infty & \rho u & \rho v & \rho w & \gamma(\gamma-1)^{-1} \end{bmatrix}_{j,k,1} \tag{3.44}$$

$$P_{j,k,2} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{\partial V_{t_1}}{\partial u} & \frac{\partial V_{t_1}}{\partial v} & \frac{\partial V_{t_1}}{\partial w} & 0 \\ 0 & \frac{\partial V_{t_2}}{\partial u} & \frac{\partial V_{t_2}}{\partial v} & \frac{\partial V_{t_2}}{\partial w} & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{j,k,2} \tag{3.45}$$

$$P_{j,k,3} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{\partial V_{t_1}}{\partial u} & \frac{\partial V_{t_1}}{\partial v} & \frac{\partial V_{t_1}}{\partial w} & 0 \\ 0 & \frac{\partial V_{t_2}}{\partial u} & \frac{\partial V_{t_2}}{\partial v} & \frac{\partial V_{t_2}}{\partial w} & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{j,k,3} \tag{3.46}$$

Additional details on the calculation of the contravariant velocity derivative terms like $\frac{\partial V_n}{\partial u}$ can be found in Appendix A.

Far-field boundary

The far-field boundary conditions use a zeroth-order extrapolation of either the interior variables or the free-stream depending on an inflow or outflow across the boundary. The equations shown here apply to the $m = m_{max}$ plane, but are applicable to all the other far-field boundaries.

$$\left[PM^{-1}J\Delta\widehat{Q}\right]_{j,k,m_{max}} - \left[PM^{-1}J\Delta\widehat{Q}\right]_{j,k,m_{max}-1} = -\mathcal{B}^{(n)} \tag{3.47}$$

where the P matrices are defined by

$$P_{j,k,m_{max}} = \begin{bmatrix} \frac{a}{(\gamma-1)\rho} & \frac{\partial V_n}{\partial u} & \frac{\partial V_n}{\partial v} & \frac{\partial V_n}{\partial w} & \frac{-\gamma}{(\gamma-1)a\rho} \\ \frac{-a}{(\gamma-1)\rho} & \frac{\partial V_n}{\partial u} & \frac{\partial V_n}{\partial v} & \frac{\partial V_n}{\partial w} & \frac{\gamma}{(\gamma-1)a\rho} \\ \frac{\gamma\rho^{\gamma-1}}{p} & 0 & 0 & 0 & \frac{-\rho^\gamma}{p^2} \\ 0 & \frac{\partial V_{t_1}}{\partial u} & \frac{\partial V_{t_1}}{\partial v} & \frac{\partial V_{t_1}}{\partial w} & 0 \\ 0 & \frac{\partial V_{t_2}}{\partial u} & \frac{\partial V_{t_2}}{\partial v} & \frac{\partial V_{t_2}}{\partial w} & 0 \end{bmatrix}_{j,k,m_{max}} \tag{3.48}$$

The last three rows of $P_{j,k,m_{max}-1}$ depend on whether it is an inflow condition or an outflow condition. For an inflow condition,

$$P_{j,k,m_{max}-1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ \frac{-a}{(\gamma-1)\rho} & \frac{\partial V_n}{\partial u} & \frac{\partial V_n}{\partial v} & \frac{\partial V_n}{\partial w} & \frac{\gamma}{(\gamma-1)a\rho} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{j,k,m_{max}-1} \tag{3.49}$$

and for an outflow condition, the last three rows are given by the last three rows of (3.48), but evaluated at $(j, k, m_{max} - 1)$.

Boundary condition scaling

Since the boundary conditions are computed using physical values (not scaled by the metric Jacobian), they are not of the same magnitude as the interior equations. Having large differences in magnitude of the entries in the matrix causes some of the eigenvalues of the Jacobian to become very large, thus increasing the stiffness of the matrix. This in turn can lead to poor convergence of the linear solver, which may not converge at all [39, 45].

Another potential issue with the boundary conditions is the possibility of having a zero on the diagonal of the matrix $PM^{-1}J$. This situation must be avoided since no zeros can exist on the main diagonal when the matrix is decomposed using ILU [9]. The solution is to perform row swapping on the boundaries to move the zero elements off the diagonal. Since each node contains a 5×5 block of numbers, a row with a non-zero entry is found and the two are swapped. This process is illustrated below:

$$\begin{bmatrix} X & X & 0 & X & 0 \\ X & X & 0 & X & X \\ X & 0 & 0 & X & X \\ X & X & 0 & X & 0 \\ X & 0 & X & 0 & X \end{bmatrix} \Longrightarrow \begin{bmatrix} X & X & 0 & X & 0 \\ X & X & 0 & X & X \\ X & 0 & X & 0 & X \\ X & X & 0 & X & 0 \\ X & 0 & 0 & X & X \end{bmatrix}$$

In this example, row 3 has been swapped with row 5 to remove the zero-diagonal entry. Following the row swapping, the matrix (and later the right-hand-side) is scaled by the largest element on the diagonal of that row. This method ensures that there will be no zero-elements

on the diagonal of the boundary blocks and that the equations (interior and boundary) have similar magnitude.

3.3.2 Inexact-Newton method

Once the equations are discretized and the time-marching method applied, the resulting system shown in Eq. (3.34) has the form $Ax = b$. There are many methods available to solve a problem of this type. Solving this problem directly involves inverting a very large and sparse matrix, which would be prohibitively expensive computationally. Several methods exist that solve an approximate system iteratively, which is appealing due to smaller memory and computational requirements. It has been found to be faster to solve the linear system a few orders of magnitude rather than exactly [35]. This leads to a method known as the inexact Newton method shown in Eq. (3.50).

$$||R(Q^n) + \frac{\partial R}{\partial Q}\Delta Q|| \leq \tilde{\eta}||R(Q^n)|| \tag{3.50}$$

This equation allows the selection of the convergence tolerance ($\tilde{\eta}$). The full Newton method is recovered if $\tilde{\eta} = 0$ is selected. A value of $\tilde{\eta}$ that is too small will force *oversolving* of the system. This means that the linear system is solved past what is needed to have an effective update to the outer iteration. Conversely, if an $\tilde{\eta}$ that is too large is selected there is a possibility of *undersolving* the system. This could lead to the outer iterations stalling because an accurate enough update to the linear problem was not found. A range of vaules of $\tilde{\eta}$ are tested and the results shown in Section 4.2.3.

3.4 Solving the linear problem

An alternative to direct solving is to use an iterative solver to solve the linear problem. For aerodynamic applications it has been found by many authors [1, 7, 34, 39, 44, 45] that a Krylov subspace method works very well and is typically faster than other iterative solvers. The Generalized Minimal Residual (GMRES) method, developed by Saad and Schultz [53] is a Krylov subspace method that iteratively solves a problem in the form of $Ax = b$. A detailed explanation of GMRES can be found in Appendix B. Two methods of solving the equations using GMRES were investigated. An approximate-Newton method which uses a first-order approximation of the Jacobian is compared to a Jacobian-free method.

3.4.1 Jacobian-free GMRES

An interesting property of GMRES is that it only requires a matrix-vector product and does not need the actual Jacobian matrix. An approximation to the matrix-vector product can be made by taking the forward difference of the residual vector and adding a term associated with the finite timestep:

$$\left[\mathcal{A} + \frac{I}{\Delta t}\right] v \simeq \frac{R(\widehat{Q} + \varepsilon v) - R(\widehat{Q})}{\varepsilon} + \frac{v}{\Delta t} \tag{3.51}$$

where ε is a small scalar used to perturb the state quantities $\widehat{Q}$ in the direction of v. The value of ε is a balance between an inaccurate approximation (ε too large) and introducing roundoff error (ε too small). The value of ε is based on work by Nielsen [42] and is calculated using the following equation

$$\varepsilon \simeq \frac{\sqrt{\varepsilon_m}}{\bar{v}} \tag{3.52}$$

where $\bar{v}$ is the root mean square of v, and ε_m is the value of "machine zero" for the hardware being used. The Jacobian-free method uses a derivative of the residual vector to approximate the matrix-vector product. The accuracy of this approximation is found to be better than that of the approximate-Newton method and therefore yields better convergence rates.

3.5 Preconditioner

The performance of iterative solvers can be increased significantly by preconditioning the linear system of equations. Following preconditioning, the eigenvalue spectrum will be clustered near unity, and the efficency and performance of GMRES will be greatly improved [45]. The preconditioning matrix will be designated $\mathcal{M}$ and is calculated using an approximation to the flow Jacobian. The preconditioner can be applied from both sides of the equation, but when applied from the right side the residual of the linear system is not affected by the preconditioner. Saad [52] indicates that selecting right or left preconditioning will not have a significant impact on the convergence of GMRES unless $\mathcal{M}$ is ill-conditioned. An ill-conditioned $\mathcal{M}$ will slow or even stagnate the GMRES convergence. Therefore, $\mathcal{M}$ is applied to the right side, such that Eq. (3.34) becomes

$$\left[\frac{I}{\Delta t} + \mathcal{A}\right] \mathcal{M}^{-1}\mathcal{M}\,\Delta\widehat{Q} = -R \tag{3.53}$$

The preconditioner used in this work is based on an incomplete factorization of a first-order approximation of the flow Jacobian ($\mathcal{A}_1$), which uses Eq. (3.35) to reduce the stencil size of the matrix. This way the preconditioner will be much more diagonally dominant and easier to solve while still retaining adequate accuracy for the linear solver.

3.5.1 Incomplete lower/upper factorization

The need for a preconditioner has been established and now a method of inverting the matrix is needed. Rather than performing a direct inversion (which could turn a sparse matrix into a dense one) an incomplete lower/upper (ILU) factorization is performed. The matrix $\mathcal{M}$ is decomposed using a level of fill k such that

$$\mathcal{M} = \tilde{L} \cdot \tilde{U} \approx L \cdot U = \mathcal{A}_1 \tag{3.54}$$

where L and U represent the true lower/upper factors and $\tilde{L}$,$\tilde{U}$ are the incomplete factors. An ILU factorization still provides a lower and upper triangular matrix, but allows the user to select a fill level k. The incomplete part of the ILU process drops elements of the factored matrix that exceed certain criteria based on the level of fill k. Higher values of fill provide a more accurate representation of the original matrix, but also increase the amount of calculation time and storage needed. After ILU factorization, the preconditioner must still be a reasonably good approximation of $\mathcal{A}_1$. Basically, we require a matrix that is easier to invert than $\mathcal{A}_1$, while its inverse remains a good approximation to $\mathcal{A}_1^{-1}$.

3.5.2 Nodal ordering

The ordering of the nodes in the system has a significant impact on the calculation time and storage space of the ILU factorization as well as the convergence rate of the linear solver. In a complete LU factorization, elements that are far from the diagonal prior to decomposition will contribute significant non-zero elements after LU factorization. The goal is to order the nodes in such a way that there is a minimum bandwidth in the matrix prior to factorization [13]. Several ordering and re-ordering methods were examined and are presented in the following sections.

Natural ordering

The simplest method of ordering the nodes is natural ordering. This method orders the nodes from beginning to end of each block with no regard for block interfaces or boundaries. As an example Figure 3.4 was ordered using natural ordering. Each entry in the matrix represents a 5×5 block of numbers. Starting at the first element of each block $(1, 1, 1)$, the nodes were ordered as follows: $\zeta \rightarrow \xi \rightarrow \eta$. The result is a matrix with a very large bandwidth since nodes that are close to each other in physical space (like on block boundaries) could be located far apart in the matrix (for example in Figure 2.3, the nodes in block 12 that touch block 1). Because of the

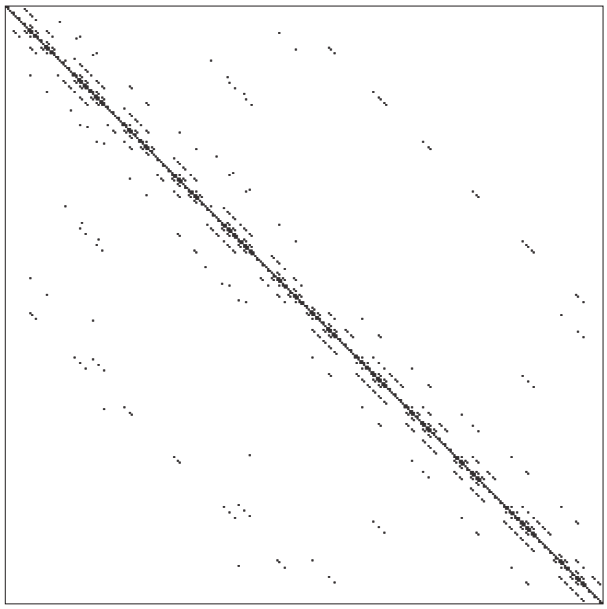


Figure 3.4: Natural ordering for the preconditioning matrix before ILU factorization

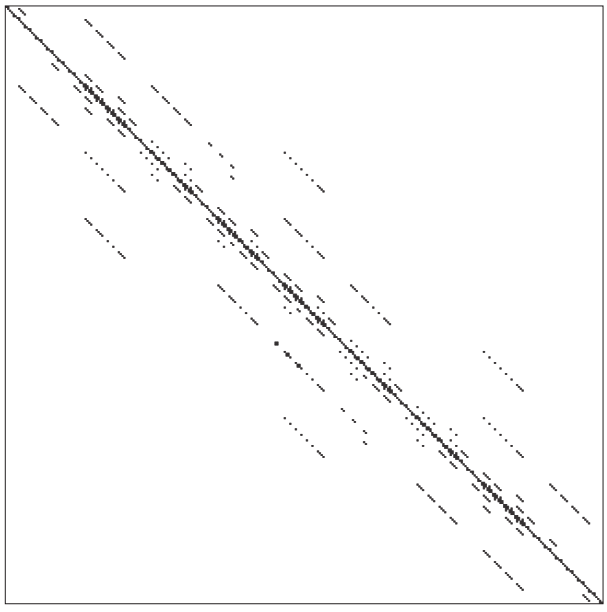


Figure 3.5: Reverse upstream ordering for the preconditioning matrix before ILU factorization

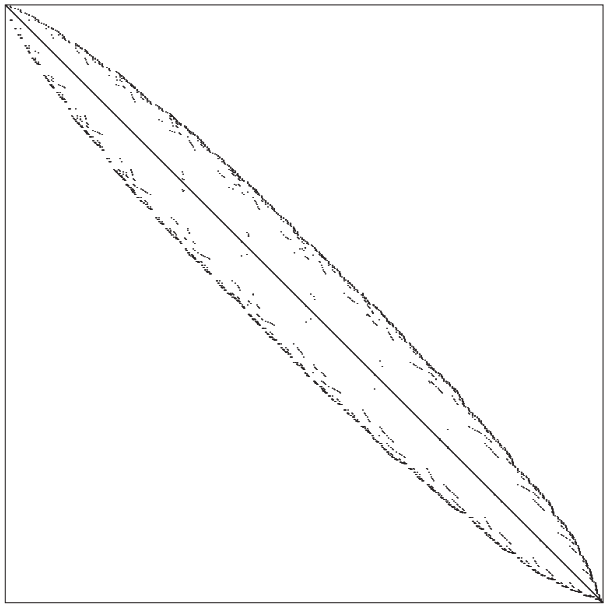


Figure 3.6: Reverse Cuthill-McKee re-ordering for the preconditioning matrix

large bandwidth, a large number of entries will be discarded after ILU factorization, leading to a less accurate result.

Reverse upstream ordering

An alternative to natural ordering is a reverse upstream ordering scheme orginally outlined for 2D by Pueyo [45, 47] and Nemec [39]. This method was modified for 3D and applied to the 12-block grid (Figure 2.3) with the resulting matrix shown in Figure 3.5. This method orders the nodes in ξ planes starting at the rear of the wing and moving to the front. The nodes are ordered through the block interfaces which allows nodes physically close to each other to remain close together in the matrix. As shown, this matrix now has a much smaller bandwidth compared to the natural ordering scheme and will be much more accurate when ILU is applied since fewer entries will be dropped.

Reverse Cuthill-McKee re-ordering

After the initial ordering has been set, a re-ordering scheme can be applied to the matrix. The Reverse Cuthill-McKee (RCM) [10] re-ordering scheme has been found to provide an efficient and consistent preconditioner for use with ILU [5, 46, 60]. RCM is based on a symmetric re-ordering of the nodes to meet certain minimum bandwidth criteria and depends on the initial

node. Figure 3.6 shows the matrix with reverse upstream ordering and RCM applied to it. The result is a matrix with a minimum bandwith, which was found to work very well with ILU.

3.6 Algorithm startup

Algorithm startup is the most critical phase when solving the equations using Newton's method. Speed is a dominant factor in designing a program, but equally important is robustness. By definition Newton's method works extremely well when the initial guess is near to the solution, but can diverge otherwise. This algorithm uses a variable time step to modify Newton's method (Implicit Euler), which is an approach used successfully by many authors, such as Venkatakrishnan and Mavriplis [60], Orkwis [44], Chisholm and Zingg [6, 8] and Manzano et al. [31].

TYPHOON uses a variable time step in conjunction with an approximate-Newton method and Jacobian-free GMRES in the solution process. The advantages of both the approximate-Newton and Jacobian-free strategies are combined into one hybrid method. The robustness in startup of the approximate-Newton method is combined with the rapid later stages of convergence of the Jacobian-free method. The two-phase startup is outlined in the following sections.

3.6.1 Local time step

The Δt term in Eq. (3.33) is used to stabilize the system during startup and is also known as pseudo-transient timestepping. This time term strengthens the diagonal of the preconditioner thereby increasing the efficiency of the linear solver. Based on Pulliam [48] the time step is scaled to the size of each cell thereby making it a local time step:

$$\Delta t^{(n)}_{j,k,m} = \frac{\Delta t^{(n)}_{ref}}{1 + \sqrt{J_{j,k,m}}} \tag{3.55}$$

Δt_{ref} is the reference time step and is increased as the solution proceeds toward convergence.

3.6.2 Phase 1: Approximate-Newton

This phase uses a first-order approximation to the flow Jacobian in place of the true Jacobian. The system is less accurate, but requires less computation than the true Jacobian. If the first-order Jacobian is substituted into the system in Eq. (3.34) the result is:

$$\left[\frac{I}{\Delta t} + \mathcal{A}_1\right]^{(n)} \Delta\widehat{Q}^{(n)} = -R^{(n)} \tag{3.56}$$

This system is more diagonally dominant and easier to solve than one using the true Jacobian. This strategy is also found to be more robust than the Jacobian-free method during startup. Since this system is solving an approximate system, the convergence rate of the outer iterations will be limited and is typically slower than that of the Jacobian-free methods.

The time step used in the first phase is a simple geometric formula used in the flow solver HURRICANE [26, 27], which is given by:

$$\Delta t^0_{ref} = A; \qquad \Delta t^n_{ref} = B\Delta t^{n-1}_{ref} \tag{3.57}$$

where A and B are constants with common values of $A=1.0$ and $B=1.7$. The algorithm begins using this preset time step sequence combined with the approximate-Newton method.

3.6.3 Phase 2: Jacobian-free GMRES

The second phase of the startup procedure involves a variable time step based on the residual and Jacobian-free GMRES. The formula used to calculate the reference time step is called the Switched Evolution Relaxation method by Mulder and van Leer [36].

$$\Delta t^n_{ref} = \max\left[\frac{\alpha}{{R_d}^{\beta}}, \Delta t_{min}\right] \tag{3.58}$$

where α and β are constants and Δt_{min} is set to 50. Typically, $\alpha=1.0$ and $\beta=1.3$. A relative residual $R_d^{(n)}$ is used in calculating the change in residual from when the code begins as shown in Eq. (3.59).

$$R_d^{(n)} = \frac{||R^{(n)}||}{||R^0||} \tag{3.59}$$

where $||R^0||$ is the starting residual.

The solution is converged approximately two orders of magnitude ($R_d = 0.01$) using the approximate-Newton method. The Jacobian-free method is then used to converge the solution to machine zero. A setback with the Jacobian-free method is that in the early iterations this method has difficulties with the approximation, especially in flows with shocks. Due to the presence of large non-linearities, even with small time steps this method has the potential to diverge. It is desirable to reduce the amount of time spent using the approximate-Newton method since the convergence is slower, but if sufficient convergence is not attained, the Jacobian-free method could fail. The convergence criterion used to switch to the Jacobian-free method is outlined in Section 4.2.4.

Chapter 4

Algorithm Optimization

This chapter outlines several tests performed on TYPHOON in order to optimize the key parameters used in solving the equations. Since nonlinear equations are being solved, the parameters must be coupled; therefore, finding a truly optimum set of parameters is an iterative process. In most cases, a clear optimum can be found by assuming that the parameters are independent of each other.

4.1 Test cases

Two test cases were used in the parametric study. The geometry is an ONERA M6 wing with a curved wing-tip as shown in Figure 2.2. The multi-block grid has twelve blocks and approximately 100,000 nodes. Table 4.1 contains the details of the test grid.

Grid	Grid Size	Far Field (semi-spans)	Off-wall Spacing ($\times 10^{-3}$)	Leading Edge Clustering ($\times 10^{-3}$)	Root Spacing ($\times 10^{-3}$)	Tip Spacing ($\times 10^{-3}$)	Nodes on Wing Surface
A	99,840	5	2.0	2.0	10.0	10.0	1,200

Table 4.1: ONERA M6 grid used in parameterization study

4.2 Parametric studies

This section presents four independent tests of the following parameters used in TYPHOON:

- ILU fill parameter (k)
- Preconditioner parameter (σ)
- Inexact Newton parameter ($\tilde{\eta}$)
- Approximate-Newton convergence parameter ($R_{d_{tol}}$)

All parameters examined in this analysis are tested using subsonic and transonic flight conditions. The subsonic conditions are $M = 0.5$ with an angle of attack $\alpha = 3°$, and the transonic conditions are given by $M = 0.84$ and $\alpha = 3.06°$.

4.2.1 ILU fill parameter (k)

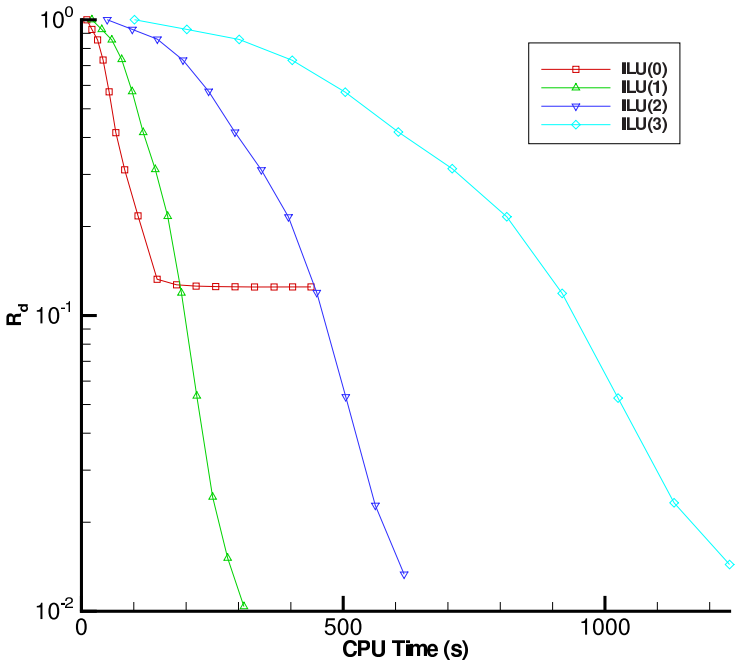
The fill level (k) used in the incomplete lower/upper factorization of the preconditioner is used by both the approximate-Newton startup as well as the Jacobian-free method. The analysis of the fill level is divided into approximate-Newton and Jacobian-free subsections.

Approximate-Newton

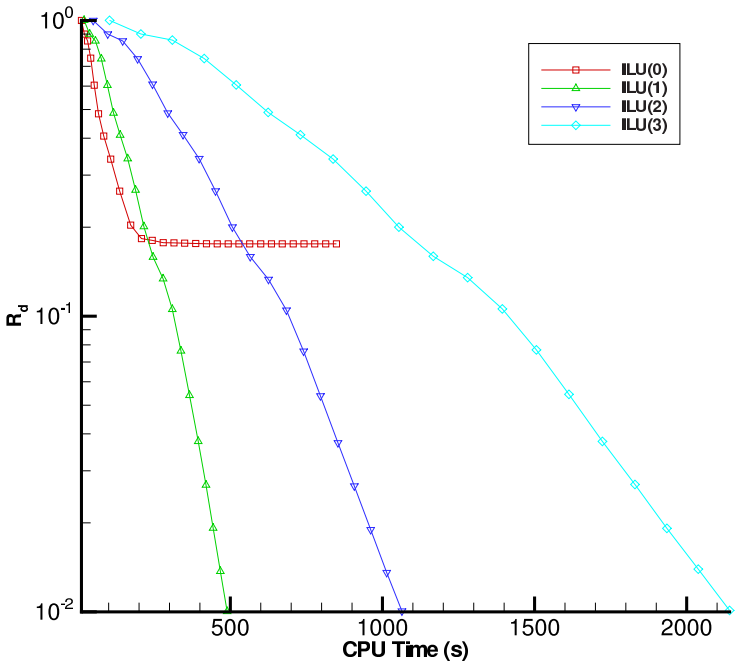
The effect of the fill level on the approximate-Newton startup was tested by converging the solution two orders of magnitude ($R_d = 0.01$) using various fill levels for both subsonic and transonic cases. The results of the test are shown in Figure 4.1 where (a) and (b) represent the subsonic and transonic cases respectively. The two graphs clearly show that ILU(1) requires the least amount of time to converge two orders of magnitude. As well, in both cases a fill level of zero is found to stagnate since the preconditioner is not accurate enough with a fill level of zero. Fill levels higher than one require more computation time and slow the startup using approximate-Newton.

Jacobian-free

A similar test was conducted on the Jacobian-free phase of the program. In this test, the solution is converged two orders of magnitude using the approximate-Newton method and ILU(1). The Jacobian-free test begins at this point, making it independent of the startup. The effect of level of fill on the convergence rate of the Jacobian-free method is shown in Figure 4.2. It can be seen that a fill level of one achieves the fastest convergence. The results from this set of tests demonstrate that a fill level of one ($k = 1$) is the best balance between a sufficiently accurate preconditioner and the computational time required to factor it.

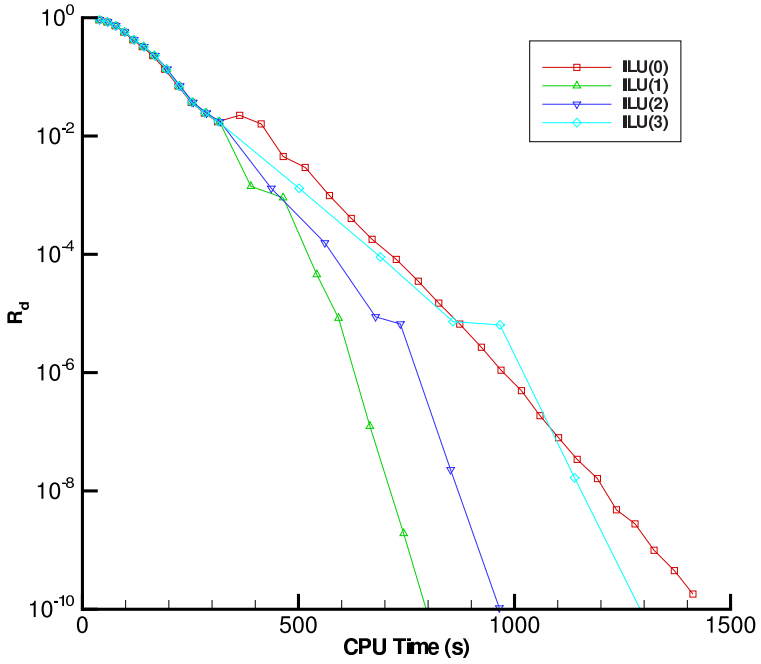


(a) Subsonic case: $M = 0.5$ and $\alpha = 3^\circ$

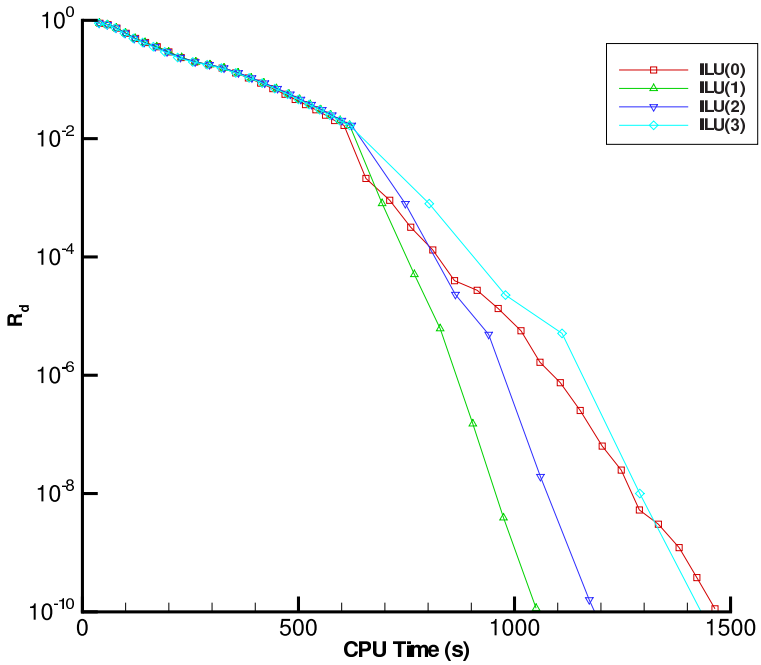


(b) Transonic case: $M = 0.84$ and $\alpha = 3.06^\circ$

Figure 4.1: ILU fill level (k) vs. relative residual (R_d) in the approximate-Newton phase



(a) Subsonic case: $M = 0.5$ and $\alpha = 3^{\circ}$



(b) Transonic case: $M = 0.84$ and $\alpha = 3.06^{\circ}$

Figure 4.2: ILU fill level (k) vs. relative residual (R_d) in the Jacobian-free phase

4.2.2 Preconditioner parameter (σ)

The preconditioner used in TYPHOON is a first-order approximation, as described in Section 3.5. The formation of the preconditioner uses a parameter σ to combine the fourth-difference with the second-difference dissipation. The effect of σ is tested in this section. Figure 4.3 shows the results of the σ tests . As in the previous tests, the results are separated into the approximate-Newton and Jacobian-free phases. Both cases show that lower values of σ require fewer inner GMRES iterations and therefore less CPU time overall. Values of $\sigma \approx 3$ are the minimum in all cases, but it was found that serious stability issues arise when values of $\sigma \leq 3$ are used. It is recommended that $\sigma = 5$ be used as a good balance between speed and robustness.

4.2.3 Inexact-Newton parameter ($\tilde{\eta}$)

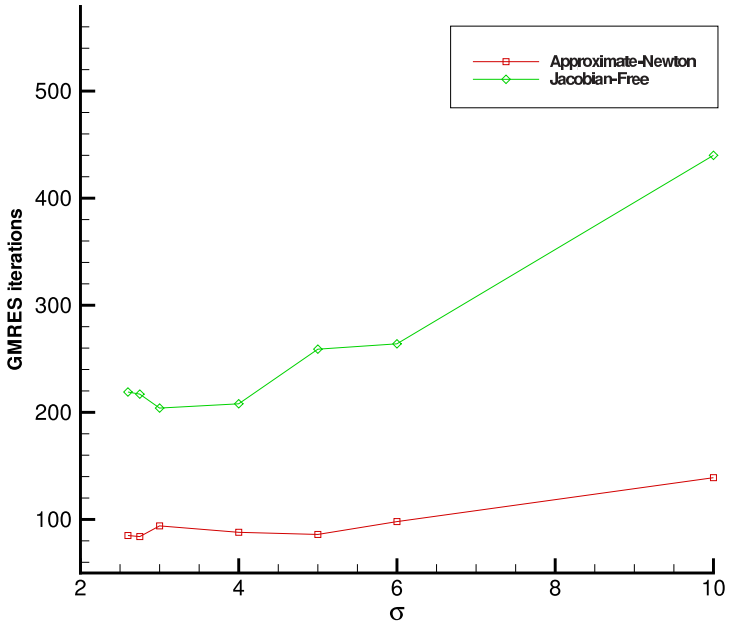
The inexact-Newton parameter ($\tilde{\eta}$) is used to set the convergence tolerance of the inner GMRES iterations, which is explained in Section 3.3.2. This parameter will be analyzed independently, since it is used in both the approximate-Newton and Jacobian-free phases of the program.

Approximate-Newton

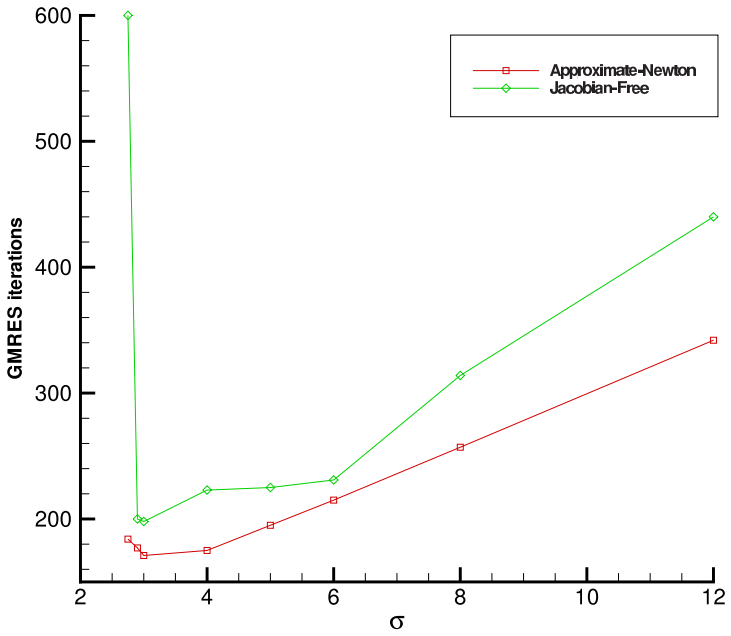
The parameter $\tilde{\eta}_{AN}$ controls the convergence of GMRES during startup with the approximate-Newton (AN) method. The solution is converged two orders of magnitude ($R_d = 0.01$) using a range of values of $\tilde{\eta}_{AN}$. Figure 4.4 shows the results of the tests conducted on the subsonic and transonic cases during the approximate-Newton phase. Minimum values of 0.3 and 0.5 were found for the subsonic and transonic cases respectively. As in previous tests, the minimum values lie very near unstable regions. A value of $\tilde{\eta}_{AN} = 0.1$ is recommended because it is found to be stable in all conditions and grid sizes. The number of search directions in GMRES is limited to 40 with no restarts during both approximate-Newton and Jacobian-free phases. The average number of inner iterations required per outer iteration for the approximate-Newton method is approximately 10.

Jacobian-free

Following startup, the algorithm switches to the Jacobian-free (JF) method which converges the solution to machine zero. The inexact-Newton parameter still applies to this phase and is tested in this section. The results are shown in Figure 4.5 for both test cases. It is clearly



(a) Subsonic case: $M = 0.5$ and $\alpha = 3^\circ$



(b) Transonic case: $M = 0.84$ and $\alpha = 3.06^\circ$

Figure 4.3: Total number of GMRES iterations vs. σ

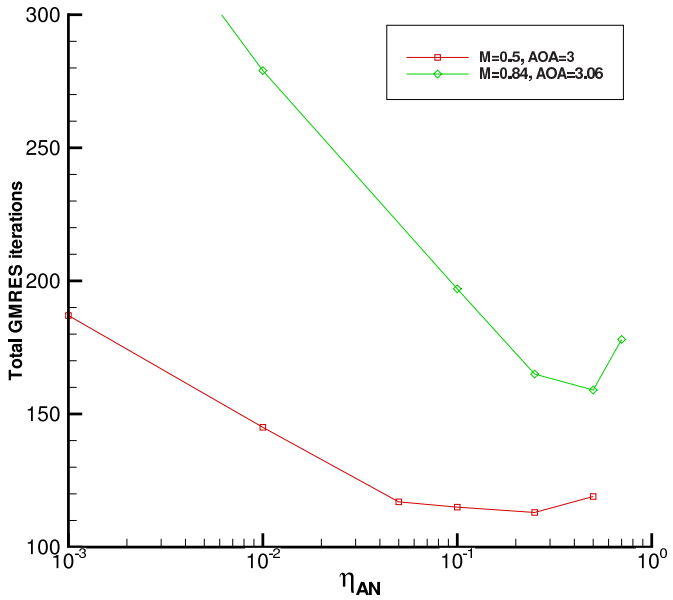


Figure 4.4: $\tilde{\eta}_{AN}$ vs. total number of GMRES iterations required to converge two orders of magnitude during the approximate-Newton phase

shown that a value of $\tilde{\eta}_{JF} = 0.01$ minimizes the number of required inner GMRES iterations for both cases. It is important to note that in difficult cases with small off-wall spacings, a lower value of $\tilde{\eta}_{JF} = 0.001$ may be required for convergence.

4.2.4 Approximate-Newton convergence parameter ($R_{d_{tol}}$)

The parameter $R_{d_{tol}}$ controls the switch from approximate-Newton to Jacobian-free. The program must be converged sufficiently before switching to Jacobian-free operation, or the solution could diverge. Figure 4.6 shows the results of the $R_{d_{tol}}$ parameter tests. Values of $R_{d_{tol}} = 0.02$ and $R_{d_{tol}} = 0.015$ were found to be minimums for the subsonic and transonic cases respectively. For flows with shocks, values of $R_{d_{tol}} \geq 0.1$ should be avoided since the solution could diverge. Difficult cases should use lower values of $R_{d_{tol}}$ to ensure that the solution is converged enough for the Jacobian-free method to take over.

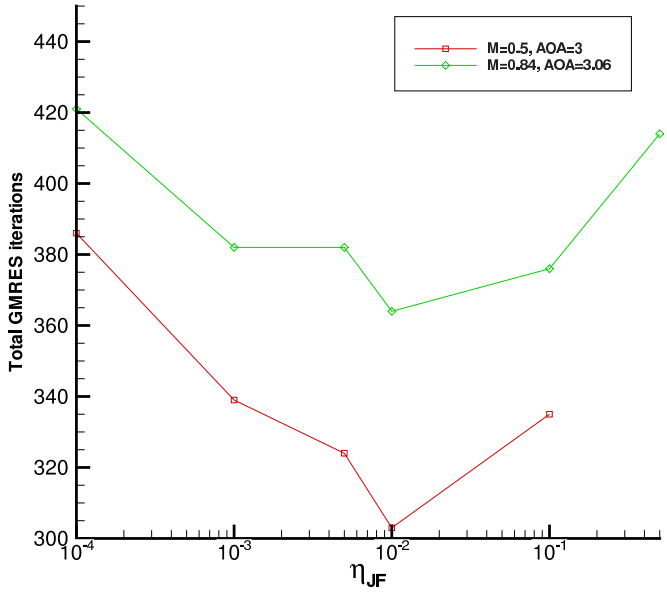


Figure 4.5: $\tilde{\eta}_{JF}$ vs. total number of GMRES iterations required to converge to machine zero during the Jacobian-free phase

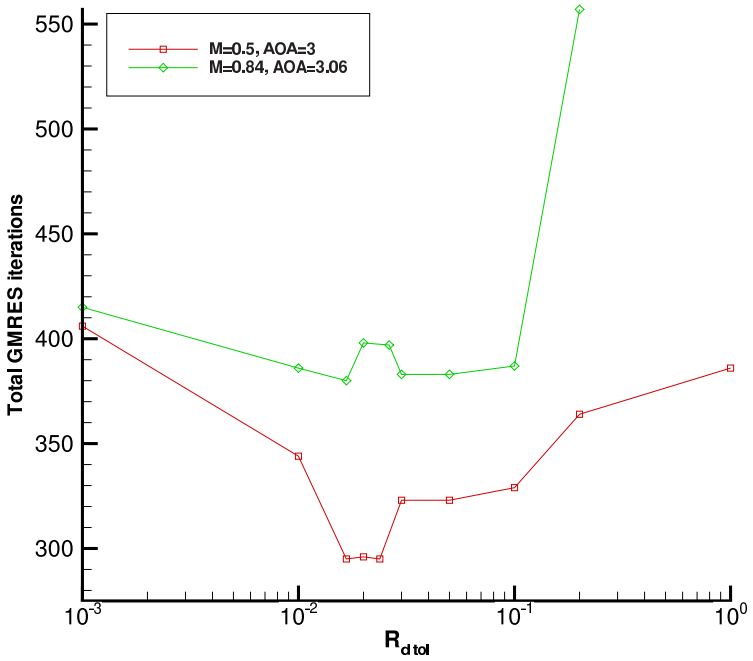


Figure 4.6: Values of $R_{d_{tol}}$ vs. total number of GMRES iterations

4.3 Optimized algorithm

Following the parameter study, the optimized parameters are summarized below:

- ILU(1) preconditioner based on the first-order Jacobian with $\sigma = 5$
- Reverse Cuthill-McKee reordering based on a reverse upstream initial ordering
- Approximate-Newton startup convergence tolerance: $R_{d_{tol}} = 0.02$
- Approximate-Newton inner tolerance: $\tilde{\eta}_{AN} = 0.1$
- Jacobian-free inner tolerance: $\tilde{\eta}_{JF} = 0.01$
- GMRES limited to 40 search directions and no restart

The following is a more conservative set of parameters for use on difficult cases:

- Approximate-Newton startup convergence tolerance: $R_{d_{tol}} = 0.001$
- Jacobian-free inner tolerance: $\tilde{\eta}_{JF} = 0.001$
- 60 search directions in GMRES

Chapter 5

Results and Validation

The first section of this chapter outlines the grids and flow conditions for the test cases used. This is followed by a comparison of results from TYPHOON to experimental data as well as other solvers such as OPTIMA [39] and HURRICANE [26, 31]. The results and performance of the TYPHOON flow solver when tested on a variety of grids and flow conditions will also be presented in this chapter. The final sections contain a discussion of TYPHOON's performance and memory requirements.

5.1 Grids and test cases

TYPHOON is tested using a variety of grids around the ONERA M6 wing geometry. Table 5.1 summarizes the details of the grids used in this chapter. All of the grids have a 12 block H-H mesh configuration as shown in Figure 2.1 and were created using an elliptical grid generator: GEM3D [18, 19]. Several test cases containing a wide variety of flow conditions are used to

Grid	Grid Size	Far Field (semi-spans)	Off-wall Spacing ($\times 10^{-3}$)	Leading Edge Clustering ($\times 10^{-3}$)	Root Spacing ($\times 10^{-3}$)	Tip Spacing ($\times 10^{-3}$)	Nodes on Wing Surface
A	99,840	5	2.0	2.0	10.0	10.0	1,200
B	248,000	10	1.0	2.0	10.0	5.0	3,000
C	500,000	10	1.0	1.0	5.0	2.0	3,600
D	1,008,000	20	1.0	1.0	5.0	5.0	4,800

Table 5.1: ONERA M6 grids used to test TYPHOON

Case	Flow Condition	Mach Number (M)	Angle of Attack (α)
1	Subsonic	0.3	8.0
2	Subsonic	0.5	3.0
3	Transonic	0.699	3.06
4	Transonic	0.84	3.06

Table 5.2: Flow conditions for the four inviscid test cases

show the results and performance of TYPHOON. Table 5.2 shows the details of these test cases. All test cases are inviscid and have 0° of roll or yaw. Only the angle of attack (pitch) and Mach number are studied in this work.

5.2 Computational comparison and resources

This section briefly outlines the computational resources used to test TYPHOON as well as the units used for comparison. The unit used to benchmark the performance of TYPHOON that is independent of computer architecture is the number of equivalent function evaluations (FE). A function evaluation is defined as one right-hand-side evaluation or one residual evaluation. The total number of FE's is calculated by dividing the total CPU time by the CPU time needed to calculate one right-hand-side. This normalization technique removes the dependence on the type of processor used. A single function evaluation includes calculation of the fluxes, pressure field, artificial dissipation and boundary conditions.

CPU time is also valuable in defining how much physical time is needed to perform the task and is also used in this work. The tests conducted on TYPHOON were run using the University of Toronto's High Performance Aerospace Computing Facility (HPACF). HPACF is a Beowulf Cluster running Hewlett-Packard ES45 AlphaServers and 1000MHz EV68CB Alpha processors.

5.3 2D Validation – NACA0012

In this section, the results of TYPHOON are compared to two other well established solvers: HURRICANE [26] and OPTIMA [39]. The test case is a NACA0012 airfoil with 11,200 nodes, 20 chords to the far-field and 5×10^{-4} off-wall spacing. TYPHOON is designed to be a three-dimensional flow solver, so the 2D NACA grid was extruded into 6 slices and a symmetry

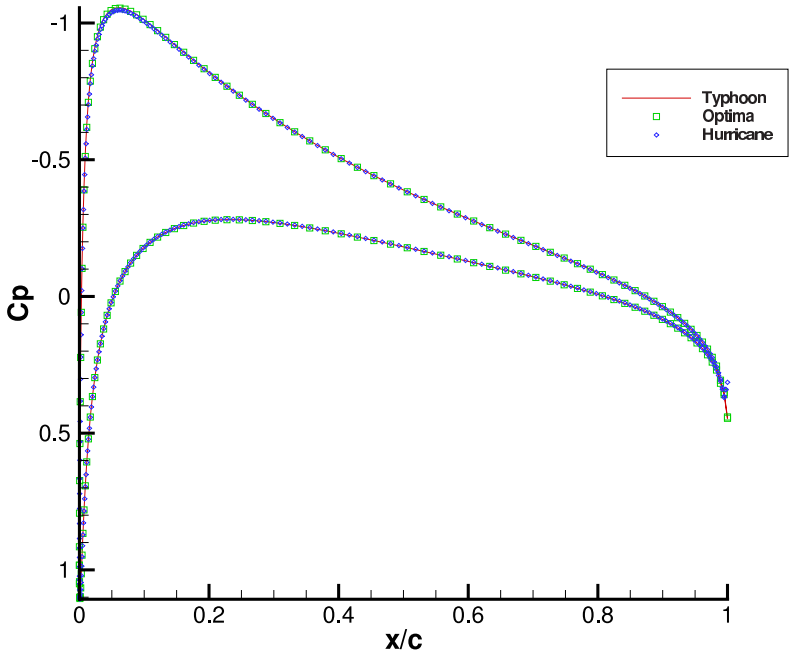
boundary condition applied at the root and the tip. All other conditions were identical when compared to the other solvers. Two test cases were run using this grid: subsonic flow at $M = 0.63$, $\alpha = 2^\circ$ and transonic flow at $M = 0.8$, $\alpha = 1.25^\circ$. The results of the tests are shown in Figures 5.1 and 5.2, which include both (a) a C_p plot and (b) a Mach contour plot. It can be seen that the results of TYPHOON agree very well with HURRICANE and OPTIMA in both test cases. TYPHOON produces excellent shock capture in the transonic case with minimal overshoot.

5.4 3D Validation – ONERA M6

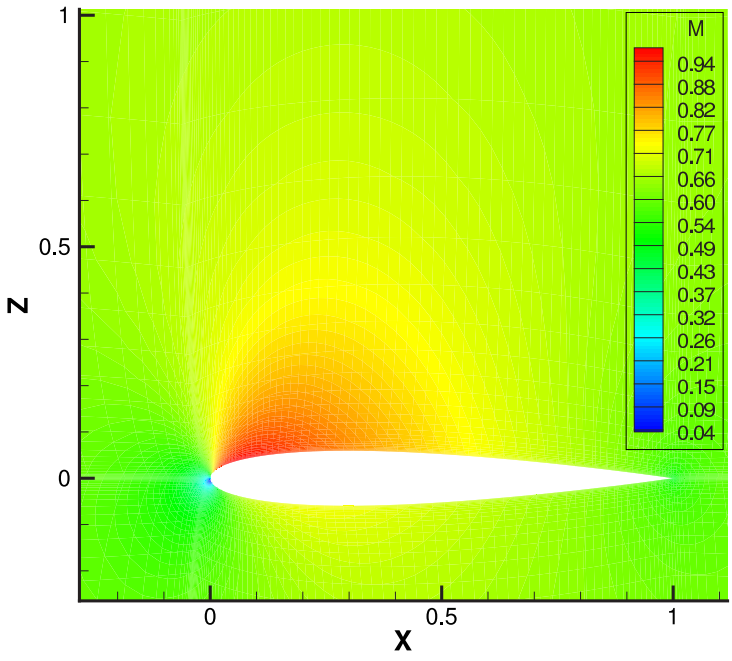
As discussed in Chapter 1, validation with experimental results is difficult due to the presence of viscous and turbulent effects. Nonetheless, a qualitative comparison can be made to ensure general accuracy. This section compares TYPHOON to a classic set of experimental results performed on the ONERA M6 wing by Charpin and Schmitt [54] in 1979. The test case used here is transonic flow at $M = 0.84$, $\alpha = 3.06^\circ$ (Case 4 from Table 5.2). A comparison of TYPHOON results to experimental data for case 3 is shown in Figure C.2 of Appendix C.

Figure 5.3 shows a 3D Mach contour plot of the ONERA wing at the test conditions for grid D. The most distinctive feature of this picture is the pattern of the shock wave on the upper surface of the wing. This type of shock is called a λ shock because of how two shocks at the root eventually join together toward the tip. This shock pattern can be seen more clearly in Figure 5.4 (a). This picture shows the coefficient of pressure (C_p) contours on the top surface Figure 5.4 (a) and on the bottom surface (b) for grid D. To compare the results of TYPHOON to the experiment, it is necessary to take spanwise slices on the wing and compare the coefficient of pressure (C_p). Six slices at spanwise locations of 20%, 44%, 65%, 80%, 90% and 96% are plotted in Figure 5.5. Three grid sizes are overlaid along with the experimental data in this figure. Overall, the results from TYPHOON correspond well to the experimental data. The code follows the leading edge pressure rise very well and captures the shock with minimal overshoot. Figure 5.6 shows Mach contour slices of the wing where the leading-edge shock and mid-chord shock can be seen.

Table 5.3 compares the lift and drag coefficients of HURRICANE, TYPHOON, and a NASA 3D unstructured multi-grid code by Mavriplis [32]. Although there is some scatter in the lift and drag values, the general agreement is good. Coefficients of lift and drag for all grids and cases can be found in Table C.1 in Appendix C.

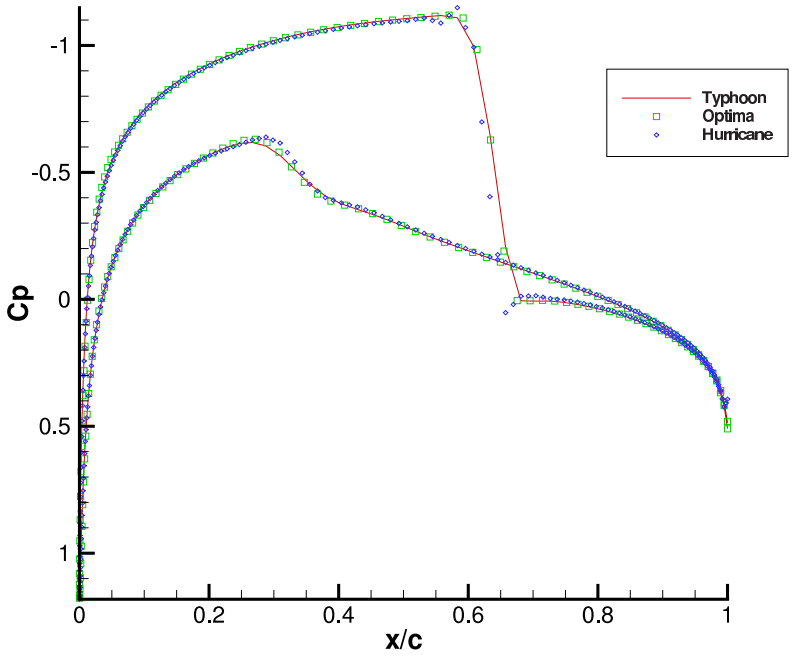


(a) C_p distribution

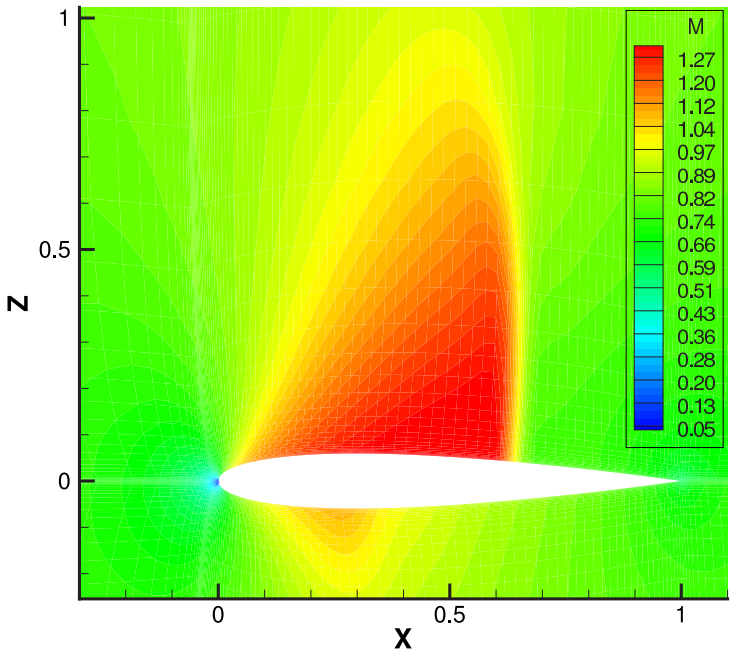


(b) Mach contours

Figure 5.1: TYPHOON subsonic flow over NACA0012 airfoil



(a) C_p distribution



(b) Mach contours

Figure 5.2: TYPHOON transonic flow over NACA0012 airfoil

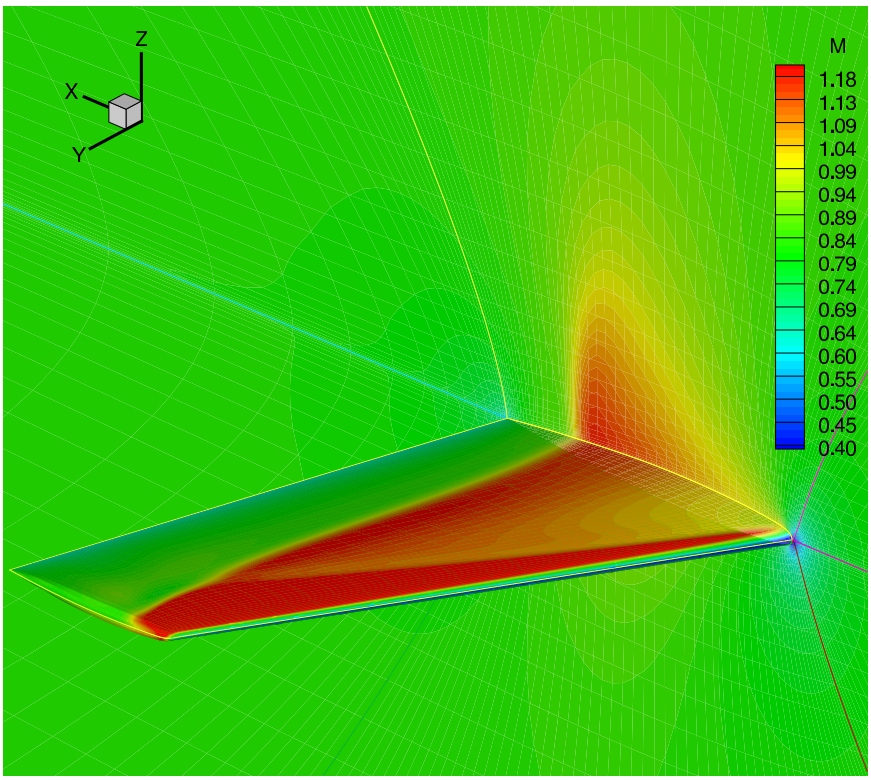
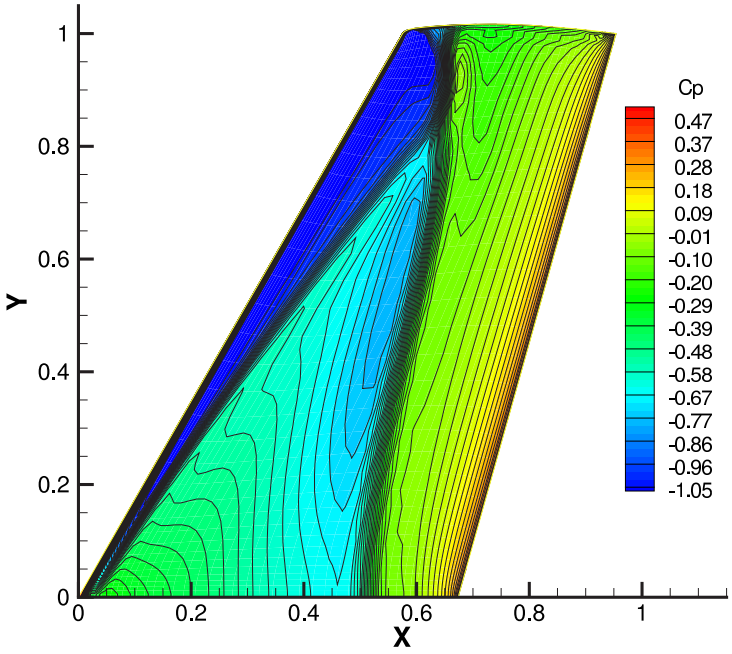


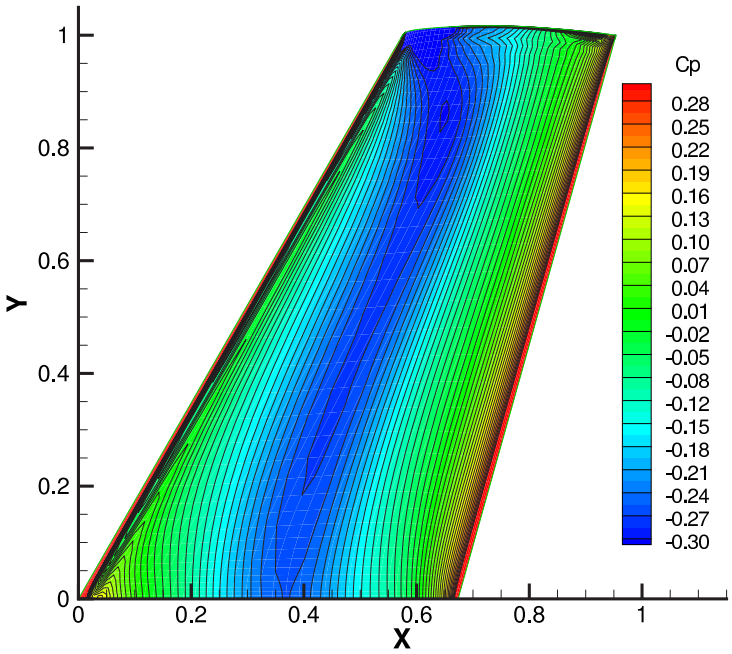
Figure 5.3: 3D Mach contours of ONERA M6 wing at $M = 0.84, \alpha = 3.06°$

	Grid Size	C_L	C_D
TYPHOON	100,000	0.3048	0.0116
	250,000	0.3017	0.0116
	500,000	0.3014	0.0108
	1,000,000	0.3031	0.0107
HURRICANE	235,000	0.2820	0.0162
NASA code	358,000	0.2923	0.0131

Table 5.3: Comparison of lift and drag coefficients for case 4

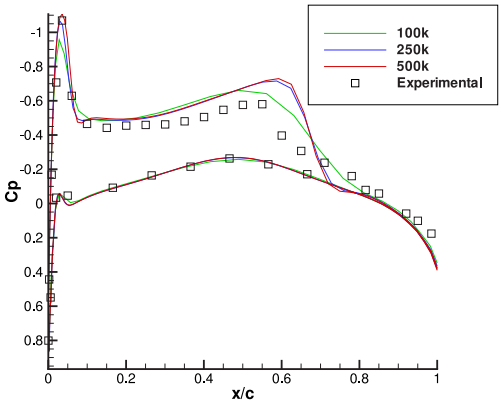


(a) Top surface

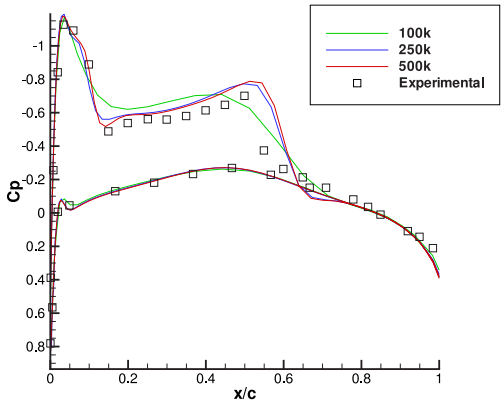


(b) Bottom surface

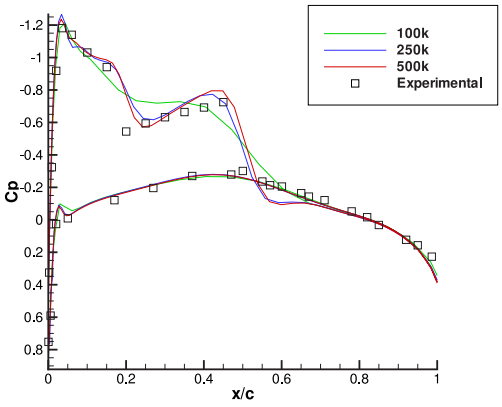
Figure 5.4: Pressure contours of ONERA M6 wing, $M = 0.84, \alpha = 3.06°$



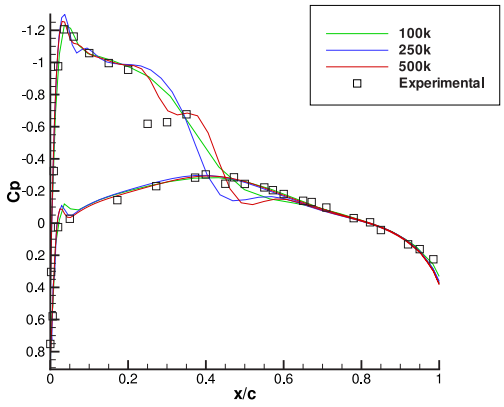
(a) 20% span



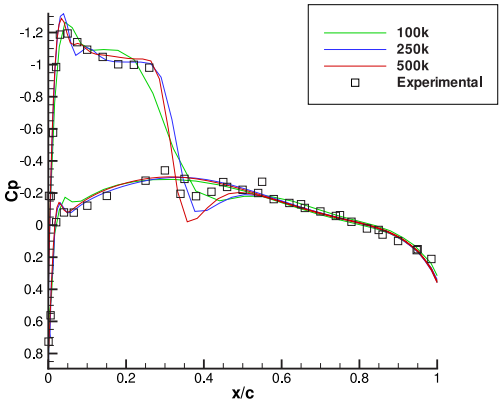
(b) 44% span



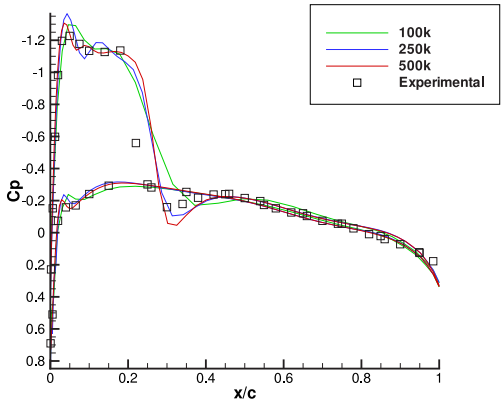
(c) 65% span



(d) 80% span

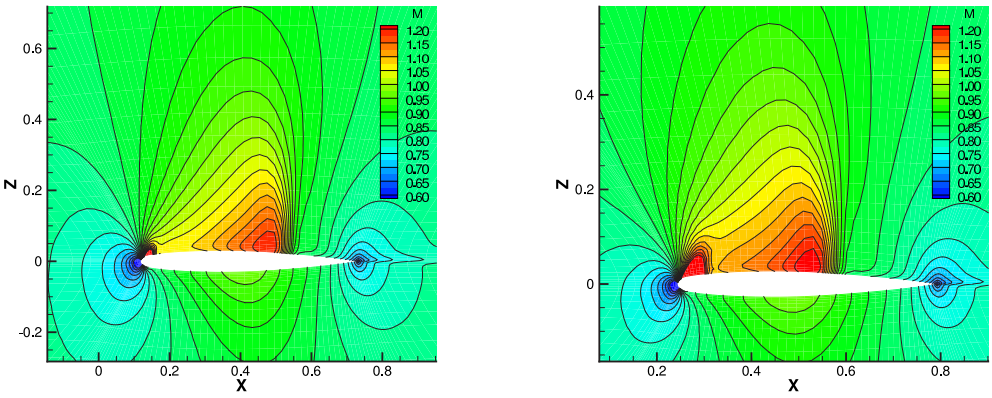


(e) 90% span



(f) 96% span

Figure 5.5: ONERA M6 wing C_p distribution, $M = 0.84, \alpha = 3.06°$



(a) 20% span

(b) 44% span

(c) 65% span

(d) 80% span

(e) 90% span

(f) 96% span

Figure 5.6: ONERA M6 wing 2D slices and Mach contours, $M = 0.84, \alpha = 3.06°$

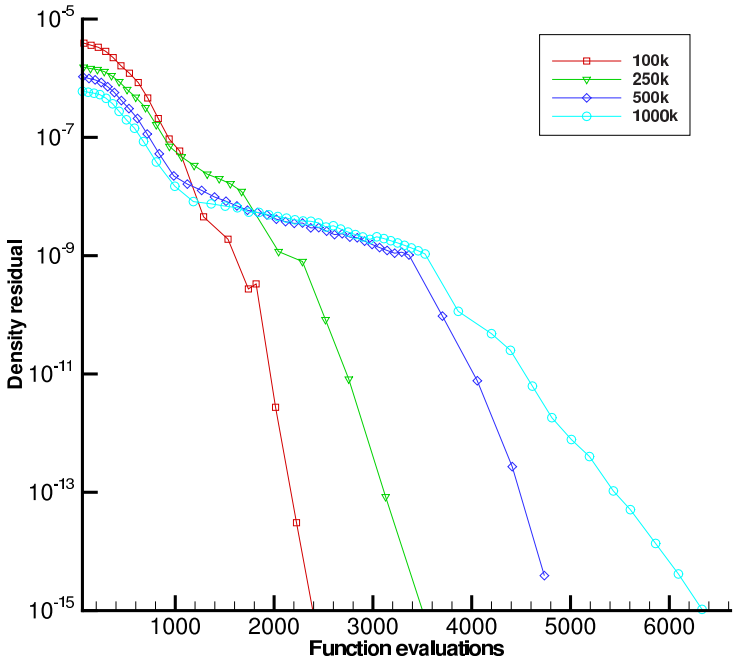
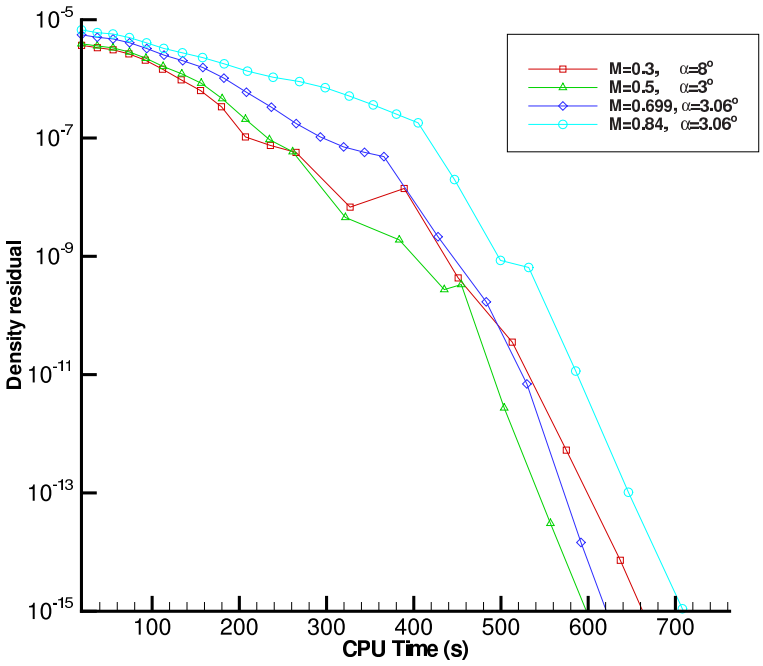


Figure 5.7: Residual vs. function evaluations for $M = 0.5$, $\alpha = 3°$

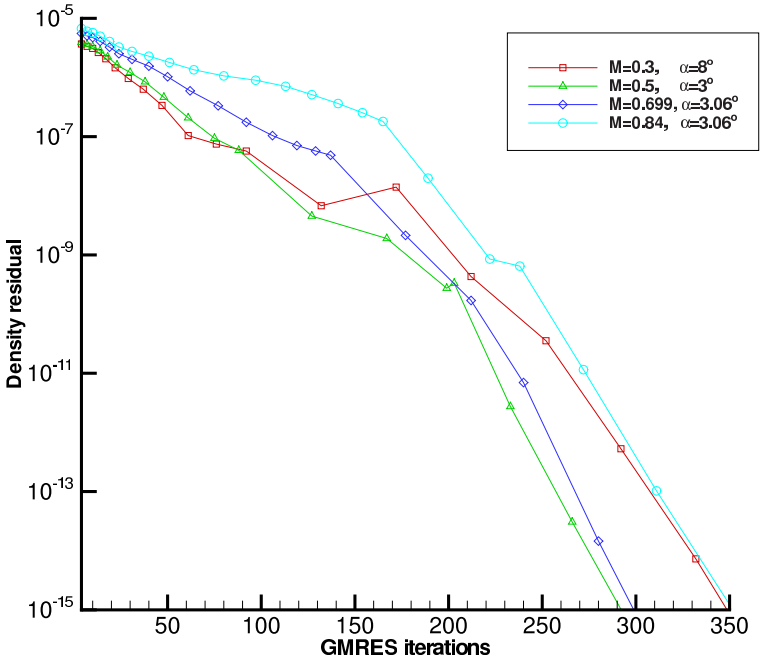
5.5 Algorithm performance

The four cases on the four ONERA M6 wing grids were run using TYPHOON. First, TYPHOON is compared across all the grids using the subsonic case 2. Figure 5.7 shows the CPU time in function evaluations and allows comparison of the wide range of grid sizes on a similar scale. For the 100,000 node grid, TYPHOON requires about 2,400 function evaluations to converge to machine zero, while the 1,000,000 node grid requires about 6,300. The 100k and 250k node cases require two orders of magnitude convergence using approximate-Newton, while the 500k and 1000k node grid require three orders. The results of the other cases are grouped into plots of each grid size and include density residual as a function of CPU time (a) and inner iterations (b). Figures 5.8 to 5.11 show the convergence histories for grids A to D respectively. Plots of the residual vs. function evaluations for all the cases can be found in Appendix C.

One of the goals of the flow solver is to calculate the coefficients of lift and drag, and another measure of code performance is how quickly these values are attained. Figure 5.12 shows the convergence history for the lift and drag coefficients on the 100k and 1000k grids. Table 5.4 shows the convergence times to given criteria for the 100k and 1000k grids. On the 100k grid, TYPHOON produces results for the coefficient of lift to four decimal places in 6.5 minutes. For

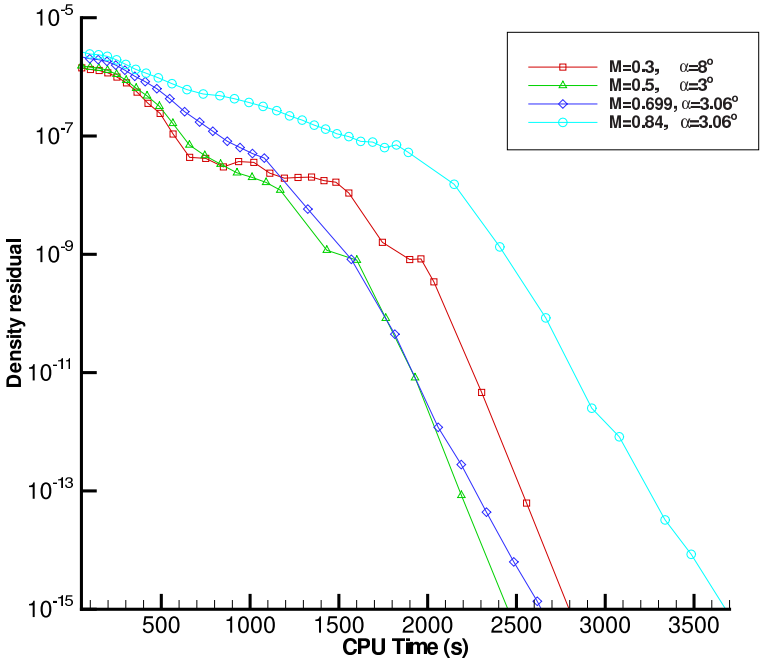


(a) Density residual vs. CPU time

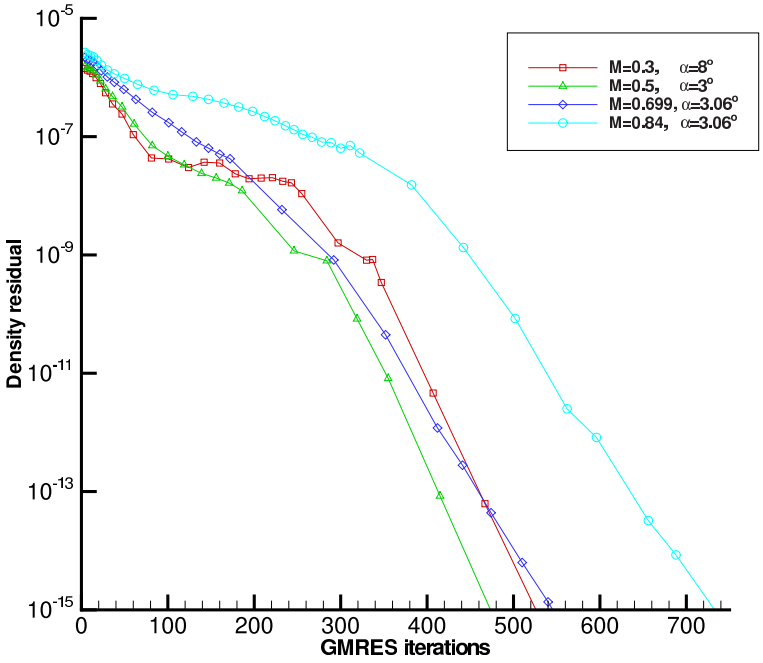


(b) Density residual vs. inner iterations

Figure 5.8: TYPHOON convergence histories for 100,000 nodes

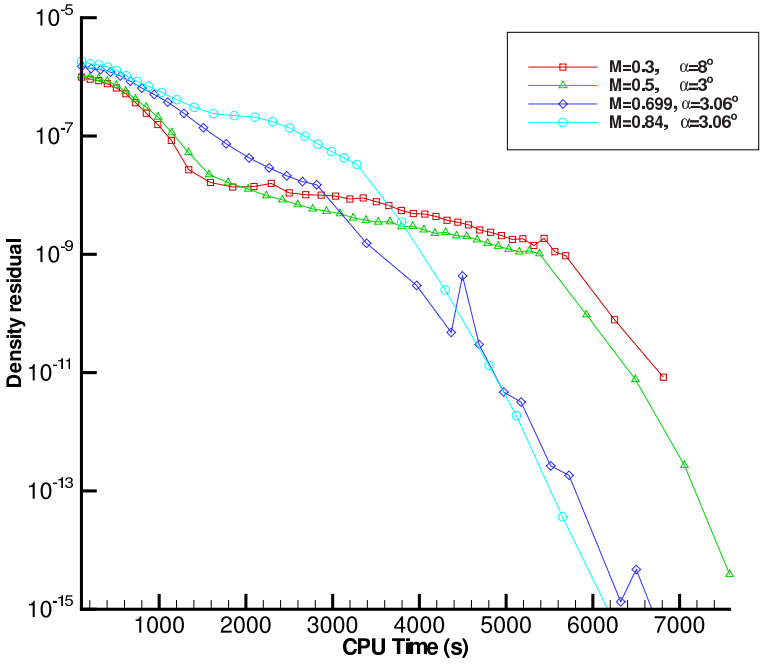


(a) Density residual vs. CPU time

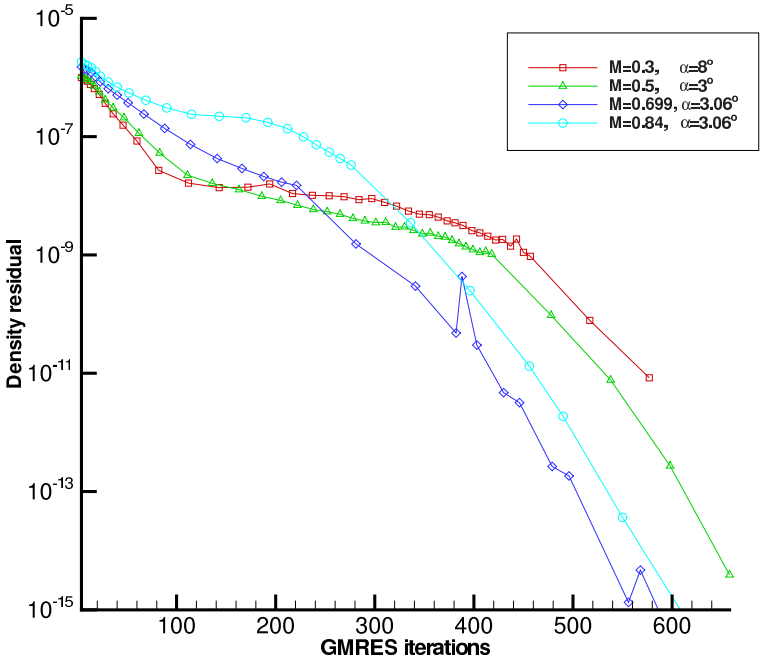


(b) Density residual vs. inner iterations

Figure 5.9: TYPHOON convergence histories for 250,000 nodes

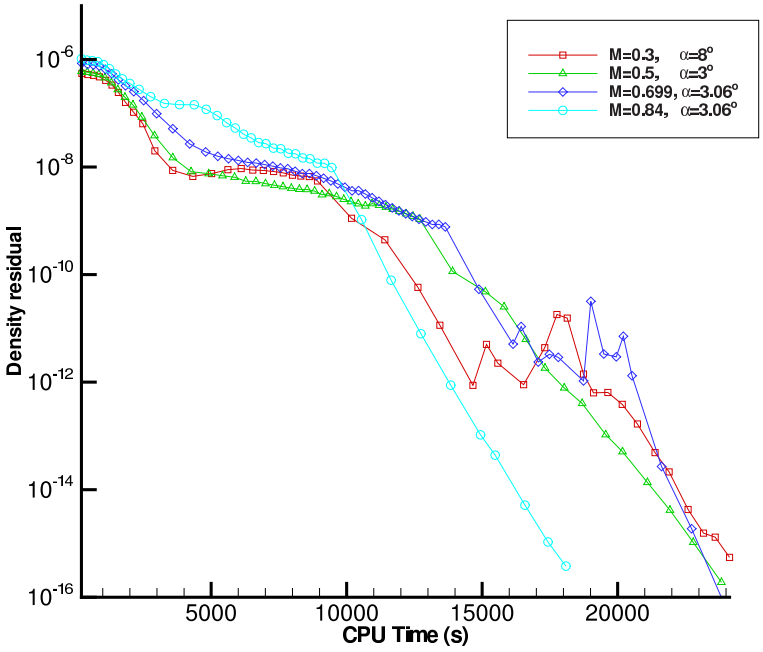


(a) Density residual vs. CPU time

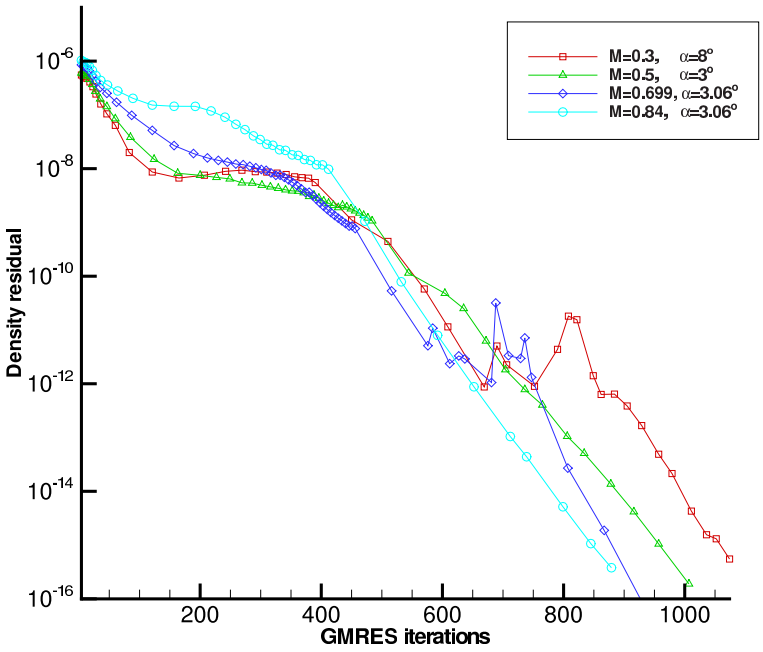


(b) Density residual vs. inner iterations

Figure 5.10: TYPHOON convergence histories for 500,000 nodes



(a) Density residual vs. CPU time



(b) Density residual vs. inner iterations

Figure 5.11: TYPHOON convergence histories for 1,000,000 nodes

Convergence criterion	TYPHOON grid size	
	100,000	1,000,000
0.5% of C_L	351s	5400s
0.1% of C_L	377s	6305s
0.01% of C_L	392s	7673s
0.5% of C_D	420s	5900s
0.1% of C_D	477s	7100s
0.01% of C_D	496s	7525s

Table 5.4: Convergence data for the coefficient of lift and drag for case 4

the same target, TYPHOON will solve the one million node grid in 127 minutes. Additional results including convergence histories, coefficient of pressure figures, and 3D contour plots can be found in Appendix C.

5.6 Grid scaling

An important aspect of a flow solver is the correlation between the CPU time and the problem size. From the tests conducted using grids A through D as well as a 50,000 and 75,000 node grid the CPU time in function evaluations for convergence to machine zero is plotted as a function of the grid size in Figure 5.13. A similar analysis was performed by Pueyo [45] on the PROBE solver and ARC2D. The results from Pueyo's tests along with the TYPHOON data are plotted. The equation of a line of best fit on the log-log plot and can be represented by:

$$\omega = \kappa N^{\beta} \tag{5.1}$$

where N is the grid size, ω is the CPU time in function evaluations, and κ and β are constants. The ideal case for a flow solver is to have the function evaluations remain constant with the grid size ($\beta = 0$). For ARC2D: $\beta = 0.73$, PROBE: $\beta = 0.325$ and TYPHOON: $\beta = 0.288$. Therefore, the scalability of the 3D multi-block solver TYPHOON is found to be very good.

5.7 Newton-Krylov solver statistics

Table 5.5 shows the details of TYPHOON's inner and outer iterations and performance for grids A and D. The average number of inner iterations remains relatively constant at 20 and independent of grid size. The number of outer iterations is higher for larger grids due to the

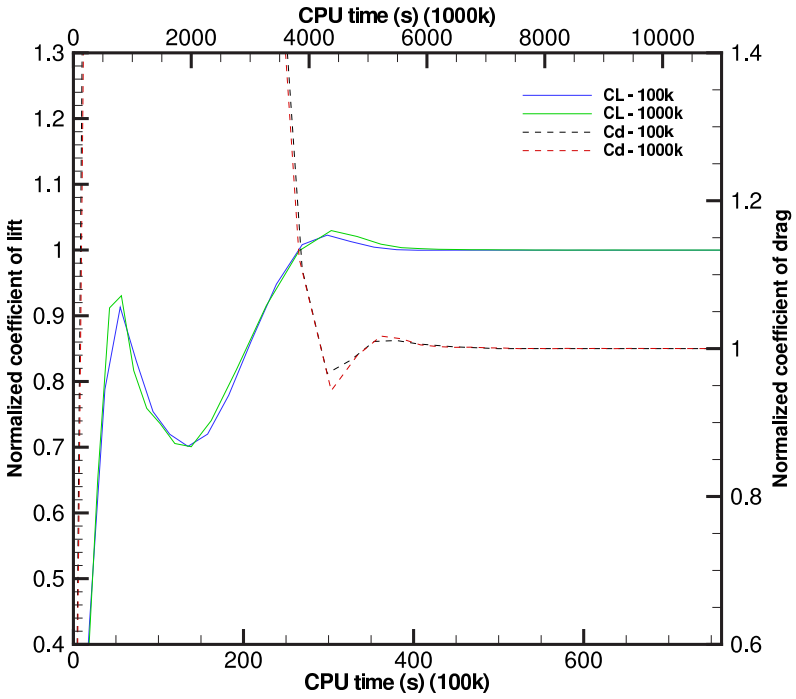


Figure 5.12: Normalized coefficients of lift and drag as a function of CPU time for case 4

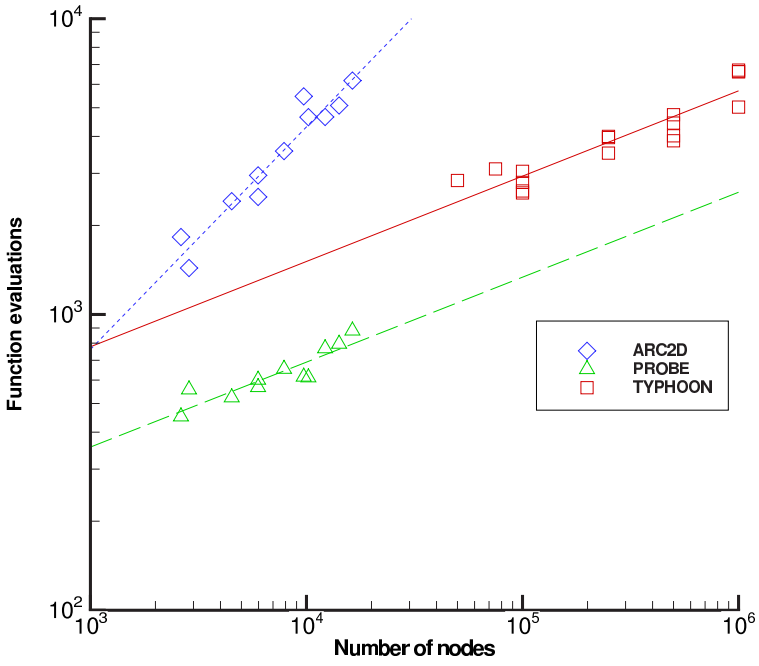


Figure 5.13: CPU time in function evaluations required to converge as a function of the grid size for TYPHOON, ARC2D and PROBE

Grid	Case	Approximate-Newton		Jacobian-Free		Total Outer Iterations (o-it)	Total Inner Iterations (i-it)	$\frac{i-it}{o-it}$	FE
		Outer Iters	Inner Iters	Outer Iters	Inner Iters				
A	1	12	92	8	280	20	372	18.6	2796
A	2	12	88	8	229	20	317	15.9	2576
A	3	16	137	6	183	22	320	14.5	2616
A	4	17	165	8	220	25	385	15.4	3048
D	1	24	390	22	684	46	1074	23.3	6703
D	2	40	484	13	523	53	1007	19.0	6619
D	3	46	456	15	471	61	927	15.2	6619
D	4	31	412	9	467	40	879	22.0	5025

Table 5.5: Newton-Krylov statistics

need for longer runs in the approximate-Newton startup. During Jacobian-free operation, fewer outer iterations are required but more inner iterations are needed. Grid A uses GMRES(40) while grid D uses GMRES(60).

5.8 Memory requirements

The memory allocation is critical when using implicit solution methods such as in TYPHOON. Implicit methods require significantly more memory than their explicit counterparts since a large linear system must be solved. Several methods are employed in TYPHOON to reduce the amount of memory needed to solve the equations implicitly.

- Jacobian-free GMRES is used, which approximates the flow Jacobian by using a forward difference of the residual vector
- The preconditioner for GMRES is based on a first-order approximation to the flow Jacobian
- Compressed sparse row (CSR) format from Saad's SPARSKIT [51] is used, which represents large sparse matrices by smaller single-dimensional arrays

The three dominant contributions to memory use in TYPHOON are the approximate Jacobian ($\mathcal{A}_1$), the ILU factors and the GMRES subspace. Combined, these three items require 93%

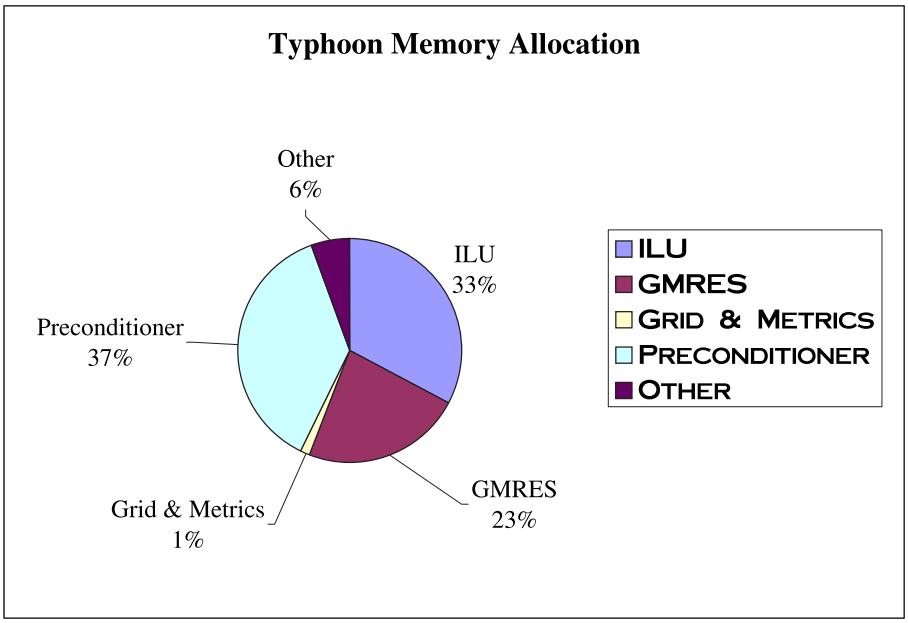


Figure 5.14: TYPHOON memory allocation

Solver	2D or 3D	Structured or Unstructured	Memory (Kb/node)
TYPHOON	3D	Structured	13
OPTIMA-MB	2D	Structured	28
HURRICANE	3D	Unstructured	34

Table 5.6: Flow solver memory comparison

of the total memory used in TYPHOON. A breakdown of the memory use in TYPHOON is shown in Figure 5.14. When in operation with ILU(1) and GMRES(40), TYPHOON requires approximately 13Kb/node of memory. Table 5.6 shows a comparison of two other flow solvers to TYPHOON. Careful allocation of memory and minimal use of unused work arrays produce a three-dimensional multi-block solver that has reasonably low memory use.

Chapter 6

Conclusions and Recommendations

6.1 Conclusions

A three-dimensional multi-block Newton-Krylov flow solver for the Euler equations has been developed for steady aerodynamic flows. Multi-block structured grids are transformed into curvilinear coordinates thus permitting unweighted spatial differences. Second-order spatial discretization is used in combination with the scalar second and fourth-difference dissipation of Jameson et al. [23]. First-order implicit Euler time-marching is applied using a variable timestep that is proportional to the residual. As the residual approaches zero, the time-step approaches infinity and Newton's method is recovered. A Krylov subspace method (GMRES) is used to solve the linear system that arises at each Newton iteration. This linear system is preconditioned using an incomplete lower/upper (ILU) factorization routine, which increases the effectiveness of the linear solver. The preconditioner is based on a first-order approximation to the flow Jacobian that uses a modified stencil to reduce the amount of storage and computation. A combination of approximate-Newton and Jacobian-free solution methods are combined to form a robust and fast algorithm capable of handling a variety of flow conditions on various grids.

A study has been conducted to optimize the parameters used in the flow solver. The main focus of the study is to find a balance between speed and robustness for the algorithm. The approximate-Newton startup is found to be very robust, which makes it ideal for algorithm startup in the region where a Jacobian-free method is not stable. The objective is to switch to the Jacobian-free method early enough to benefit from the increased convergence rate, but late enough so that the solution is in the Newton convergence region. Difficult cases such as those with strong shocks or small cell volumes have the potential to stiffen the equations being solved. In such cases, more conservative time-step parameters and further convergence using

approximate-Newton is necessary.

Tests were conducted on grids ranging in size from 100,000 nodes to 1,000,000 nodes and flow conditions from $M = 0.3$, $\alpha = 8°$ to $M = 0.84$, $\alpha = 3.06°$. The flow solver converges to machine zero in 2,400 function evaluations (10 minutes) for the 100,000 node grid and 6,300 (6.3 hours) for the one million node grid on a single processor. The scalability of TYPHOON is found to be good with the CPU cost proportional to the grid size to the power of 1.288.

6.2 Recommendations

The program developed provides a substantial base for the development of a full aircraft optimization routine. The objective was to construct a program to solve inviscid flow on multi-block structured grids. Following this, viscous and turbulent components can be added to TYPHOON. Some additions and possible research areas that would increase the effectiveness and versatility of the solver are presented here.

The addition of matrix dissipation by Swanson and Turkel [57] would increase the accuracy of the solver. Care must be taken when implementing this type of dissipation because it has been found to degrade the preconditioner and could cause the solution to diverge.

The block interfaces in TYPHOON are treated as interior nodes and not as special cases. Since the faces that join blocks together share physical space and use identical computational stencils, some calculation time can be saved by implementing a master-slave condition at this interface.

It would also be beneficial to investigate the use of hybrid C-H or C-O grids in TYPHOON. Currently, the code uses an H-H type mesh and suffers from highly skewed cells at the leading edge when small spacings are used. The use of a C-mesh in the chordwise direction would combine the benefits of the curved mesh around the leading edge (large curvature) and the advantages of an H mesh at the sharp trailing edge. Also, in a structured grid environment, blunt surfaces are challenging to grid, but an O-grid has the ability to project the grid lines onto the blunt faces, which an H-grid cannot.

References

[1] P. Adami, V. Michelassi, and F. Martelli, *Performances of a Newton-Krylov scheme against implicit and multigrid solvers for inviscid flows*, AIAA Paper 98-2429, Albuquerque, NM, June 1998.

[2] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, PA, 1994.

[3] R. Beam and R. Warming, *An implicit finite-difference algorithm for hyperbolic systems in conservation law form*, J. Comp. Phys., 22 (1976), pp. 87–110.

[4] S. Benzoni-Gavage and J. Coulombel, *Boundary conditions for Euler equations*, AIAA Journal, 41 (2003), pp. 56–63.

[5] M. Blanco and D. W. Zingg, *Fast Newton–Krylov method for unstructured grids*, AIAA Journal, 36 (1998), pp. 607–612.

[6] T. Chisholm and D. W. Zingg, *A Newton-Krylov algorithm for turbulent aerodynamic flows*, AIAA Paper 2003–0071, Reno, NV.

[7] ——, *A fully coupled Newton-Krylov solver for turbulent aerodynamic flows*, Paper 333, ICAS 2002, Toronto, ON, Sept. 2002.

[8] ——, *Start-up issues in a Newton-Krylov algorithm for turbulent aerodynamic flows*, AIAA Paper 2003–3708, Orlando, FL, June 2003.

[9] E. Chow and Y. Saad, *Experimental study of ILU preconditioners for indefinite matrices*, Journal of Computational and Applied Mathematics, 86 (1997), pp. 387–414.

[10] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in 24th National Conference of the Association for Computing Machinery, no. ACM P-69, New York, 1969, Brandon Press, pp. 157–172.

[11] S. De Rango, *Higher-Order Spatial Discretization for Turbulent Aerodynamic Flows*, PhD thesis, University of Toronto, 2001.

[12] S. De Rango and D. W. Zingg, *Higher-order spatial discretization for turbulent aerodynamic computations*, AIAA Journal, 39 (2001), pp. 1296–1304.

[13] L. C. Dutto, *The effect of ordering on preconditioned GMRES algorithm, for solving the compressible Navier–Stokes equations*, International Journal for Numerical Methods in Engineering, 36 (1993), pp. 457–497.

[14] B. Epstein, T. Rubin, and S. Seror, *Accurate multiblock Navier-Stokes solver for complex aerodynamic configurations*, AIAA Journal, 41 (2003), pp. 582–594.

[15] D. Feng and T. Pulliam, *Tensor-GMRES method for large systems of nonlinear equations*, SIAM Journal on Optimization, 7 (1997), pp. 757–779.

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

[17] A. Gogoi and K. Rao, *Validation of a multi-block solver on aerospace test cases*, Technical Report 560-017, Aeronarutical Development Agency, 2003.

[18] D. Hall, *Three-dimensional elliptic grid generation*, Master's thesis, University of Toronto, 1992.

[19] ——, *User's Manual for the GEM3D Elliptic Grid Generator*, University of Toronto, April 1992.

[20] C. Hirsch, *Numerical Computation of Internal and External Flows*, vol. 2, John Wiley & Sons, England, 1994.

[21] A. Jameson, *Solution of the Euler equations for two dimensional transonic flow by a multigrid method*, Applied Mathematics and Computation, 13 (1983), pp. 327–356.

[22] A. Jameson, *CFD for aerodynamic design and optimization: Its evolution over the last three decades*, AIAA Paper 2003-3438, Stanford University, June 2003.

[23] A. Jameson, W. Schmidt, and E. Turkel, *Numerical solutions of the Euler equations by finite volume methods using Runge-Kutta time stepping*, AIAA Paper 81–1259, June 1981.

[24] J. Kim and D. Lee, *Characteristic interface conditions for multiblock high-order computation on singular structured grid*, AIAA Journal, 41 (2003), pp. 2341–2348.

[25] D. Knoll and D. Keyes, *Jacobian-free Newton-Krylov methods: a survey of approaches and applications*, Journal of Computational Physics, 193 (2004), pp. 357–397.

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

[27] T. Leung, *Parallel implementation of a Newton-Krylov flow solver on unstructured grids*, Master's thesis, University of Toronto, 2004.

[28] H. Lomax, T. H. Pulliam, and D. W. Zingg, *Fundamentals of Computational Fluid Dynamics*, Springer–Verlag, Berlin, Heidelberg, 2001.

[29] H. Luo, J. Baum, and R. Löhner, *A fast, matrix-free implicit method for compressible flows on unstructured grids*, Journal of Computational Physics, 146 (1998), pp. 664–690.

[30] L. Manzano, *Implementation of multigrid for aerodynamic computations on multi-block grids*, Master's thesis, University of Toronto, 1999.

[31] L. Manzano, J. Lassaline, P. Wong, and D. W. Zingg, *A Newton-Krylov algorithm for the Euler equations using unstructured grids*, Paper 2003-0274, University of Toronto, 2003.

[32] D. J. Mavriplis, *Three dimensional unstructured multigrid for the Euler equations*, AIAA Paper 91-1549, 1991.

[33] G. May, E. van der Weide, A. Jameson, and L. Martinelli, *Drag prediction of the DLR-F6 configuration*, AIAA Paper 2004–0396, Reno, NV, Jan. 2004.

[34] P. McHugh and D. Knoll, *Comparison of standard and matrix-free implementations of several Newton-Krylov solvers*, AIAA Journal, 32 (1994), pp. 2394–2400.

[35] B. Morini, *Convergence behaviour of inexact Newton methods*, American Math Society, 68 (1999), pp. 1605–1613.

[36] W. Mulder and B. van Leer, *Experiments with an implicit upwind method for the Euler equations*, Journal of Computational Physics, 59 (1985), pp. 232–246.

[37] T. Nelson, *Numerical Solutions of the Navier-Stokes Equations for High-Lift Airfoil Configurations*, PhD thesis, University of Toronto, 1994.

[38] T. Nelson and D. W. Zingg, *Fifty years of aerodynamics: Successes, challenges, and opportunities*, CAS J., 50 (2004), pp. 61–84.

[39] M. NEMEC, *Optimal Shape Design of Aerodynamic Configurations: A Newton-Krylov Approach*, PhD thesis, University of Toronto, 2003.

[40] M. NEMEC AND D. W. ZINGG, *Newton-Krylov algorithm for aerodynamic design using the Navier-Stokes equations*, AIAA Journal, 36 (2002), pp. 1146–1154.

[41] M. NEMEC AND D. W. ZINGG, *TORNADO user's guide*, 2002.

[42] E. NIELSEN, K. ANDERSON, R. WALTERS, AND D. KEYES, *Application of Newton-Krylov methodology to a three-dimensional unstructured Euler code*, AIAA Journal, (1995).

[43] E. ORAN, *Matchsticks, scramjets, and black holes: Numerical simulation faces reality*, AIAA Journal, 40 (2002), pp. 1481–1494.

[44] P. ORKWIS, *Comparison of Newton's and quasi-Newton's method solvers for the Navier-Stokes equations*, AIAA Journal, 31 (1993), pp. 832–836.

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

[46] A. PUEYO AND D. W. ZINGG, *Efficient Newton-Krylov solver for aerodynamic computations*, AIAA Journal, 36 (1998), pp. 1991–1997.

[47] A. PUEYO AND D. W. ZINGG, *Improvements to a Newton-krylov solver for aerodynamic flows*, aiaa paper, University of Toronto, 1998.

[48] T. PULLIAM, *Efficient solution methods for the Navier-Stokes equations*, Lecture Notes For The Von Karman Institute For Fluid Dynamics Lecture Series: *numerical techniques for viscous flow computation in turbomachinery bladings*, NASA Ames Research Center, January 1986.

[49] T. PULLIAM, S. ROGERS, AND T. BARTH, *Practical aspects of Krylov-subspace iterative methods in CFD*, AGARD CFD Symposium, NASA Ames Research Center, October 1995.

[50] T. PULLIAM AND J. STEGER, *Implicit finite-difference simulations of three-dimensional compressible flow*, AIAA Journal, 18 (1980), pp. 159–167.

[51] Y. SAAD, *Sparskit: a basic tool kit for sparse matrix computations.* http://www-.users.cs.umn.edu/∼saad/software.html, 1994.

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

[53] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear problems*, SIAM J. Sci. Stat. Comp., 7 (1986), pp. 856–869.

[54] V. Schmitt and F. Charpin, *Pressure distributions on the ONERA-M6 wing at transonic Mach numbers*, paper, Office National D'Etudes et de Recherches Aerospatiales, 1979.

[55] M. Spel and W. Dieudonne, *Mistral code validation*, Report RTECH 030807 001-R13, R. Tech, September 2003.

[56] J. Steger, *Implicit finite difference simulation of flow about arbitrary geometries with application to airfoils.* AIAA Paper 77-665, June 1977.

[57] R. C. Swanson and E. Turkel, *On central-difference and upwind schemes*, J. Comp. Phys., 101 (1992), pp. 292–306.

[58] E. Tinoco and T. Su, *Drag prediction with the Zeus/CFL3D system*, AIAA Paper, Reno, NV, Jan. 2004.

[59] J. W. van der Burg and E. T. A. van der Weide, *Short turnaround time turbulent flow computations for complete aircraft configurations*, ICAS 2002 Congress, National Aerospace Laboratory NLR, 2002.

[60] V. Venkatakrishnan and D. Mavriplis, *Implicit solvers for unstructured meshes*, Journal of Computational Physics, (1993), pp. 83–91.

[61] M. Vinokur, *An analysis of finite-difference and finite-volume formulations of conservation laws*, Journal of Computational Physics, 81 (1989), pp. 1–52.

[62] D. W. Zingg, S. De Rango, A. Pueyo, M. Nemec, and T. Chisholm, *Advances in Algorithms for Computing Aerodynamic Flows in Frontiers of Computational Fluid Dynamics*, World Scientific Publishing Company, 2000.

Appendix A

Contravariant velocity derivatives

This section provides an explanation in the calculation of the contravariant velocity derivatives that are used in the boundary conditions described in Section 3.3.1.

After the system of equations given in Eq. 2.13 is solved for V_n, V_{t_1} and V_{t_2} the system will have the form

$$\begin{bmatrix} V_{t_1} \\ V_{t_2} \\ V_n \end{bmatrix} = [\theta] \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{A.1}$$

where $\theta = \mathcal{B}^{-1}$ from Eq. 2.13. The elements in θ are only functions of the grid metrics and are constants when derivitatives in u, v, w are applied. The system can be re-written as:

$$\begin{aligned} V_{t_1} &= \theta_{1,1}u + \theta_{1,2}v + \theta_{1,3}w \\ V_{t_2} &= \theta_{2,1}u + \theta_{2,2}v + \theta_{2,3}w \\ V_n &= \theta_{3,1}u + \theta_{3,2}v + \theta_{3,3}w \end{aligned}$$

With the θ entries treated as constant the an example of the derivatives is shown below

$$\frac{\partial V_n}{\partial u} = \theta_{3,1} \qquad \frac{\partial V_n}{\partial v} = \theta_{3,2} \qquad \frac{\partial V_n}{\partial w} = \theta_{3,3} \tag{A.2}$$

Appendix B

GMRES

The following is an outline of the operation of GMRES [53] applied to a linear system in the form $Ax = b$.

GMRES finds the iterate $x_m \in \{x_0 + K_m\}$ that minimizes the L_2 norm of the residual $r_m = b - \mathcal{A}x_m$, where x_0 is an initial guess in the iterative process, and K_m is a Krylov subspace of the form

$$K_m = \text{span}\{v_1, \mathcal{A}v_1, \mathcal{A}^2 v_1, ..., \mathcal{A}^{m-1} v_1\} \tag{B.1}$$

The vector v_1 is

$$v_1 = \frac{r_0}{\|r_0\|_2} = \frac{b - \mathcal{A}x_0}{\|b - \mathcal{A}x_0\|_2} \tag{B.2}$$

GMRES has three basic steps. First, from an initial guess x_0, it computes the vector v_1. Second, using Arnoldi's method it forms an orthogonal basis of the subspace K_m; every new direction vector $\mathcal{A}v_j$ is made orthogonal to all the previous ones and it is normalized:

$$\begin{aligned} For \quad & j = 1, 2, ..., m \quad do: \\ & h_{i,j} = (\mathcal{A}v_j, v_i), i = 1, 2, ..., j \\ & \hat{v}_{j+1} = \mathcal{A}v_j - \textstyle\sum_{i=1}^{j} h_{i,j} v_j \\ & h_{j+1,j} = \|\hat{v}_{j+1}\| \\ & v_{j+1} = \hat{v}_{j+1}/h_{j+1,j} \end{aligned} \tag{B.3}$$

From the above process, it is easy to show that

$$\mathcal{A}V_m = V_{m+1}\bar{H}_m \tag{B.4}$$

where V_m is a $N \times m$ matrix with column vectors $v_1, ..., v_m$ and $\bar{H}_m$ is a $(m+1) \times m$ Hessenberg matrix containing the $h_{i,j}$ coefficients computed by Arnoldi's algorithm. Any given vector

$x \in \{x_0 + K_m\}$ can be written as

$$x = x_0 + V_m y \tag{B.5}$$

where y is a vector of dimension m. Making use of Eqs. (B.4) and (B.5), the L_2 norm of the residual can be written as a function of y:

$$\begin{aligned} \|r(y)\|_2 &= \|b - \mathcal{A}x\|_2 \\ &= \|b - \mathcal{A}(x_0 + V_m y)\|_2 \\ &= \|r_0 - \mathcal{A}V_m y\|_2 \\ &= \|\beta v_1 - V_{m+1}\bar{H}_m y\|_2 \\ &= \|V_{m+1}(\beta e_1 - \bar{H}_m y)\|_2 \end{aligned} \tag{B.6}$$

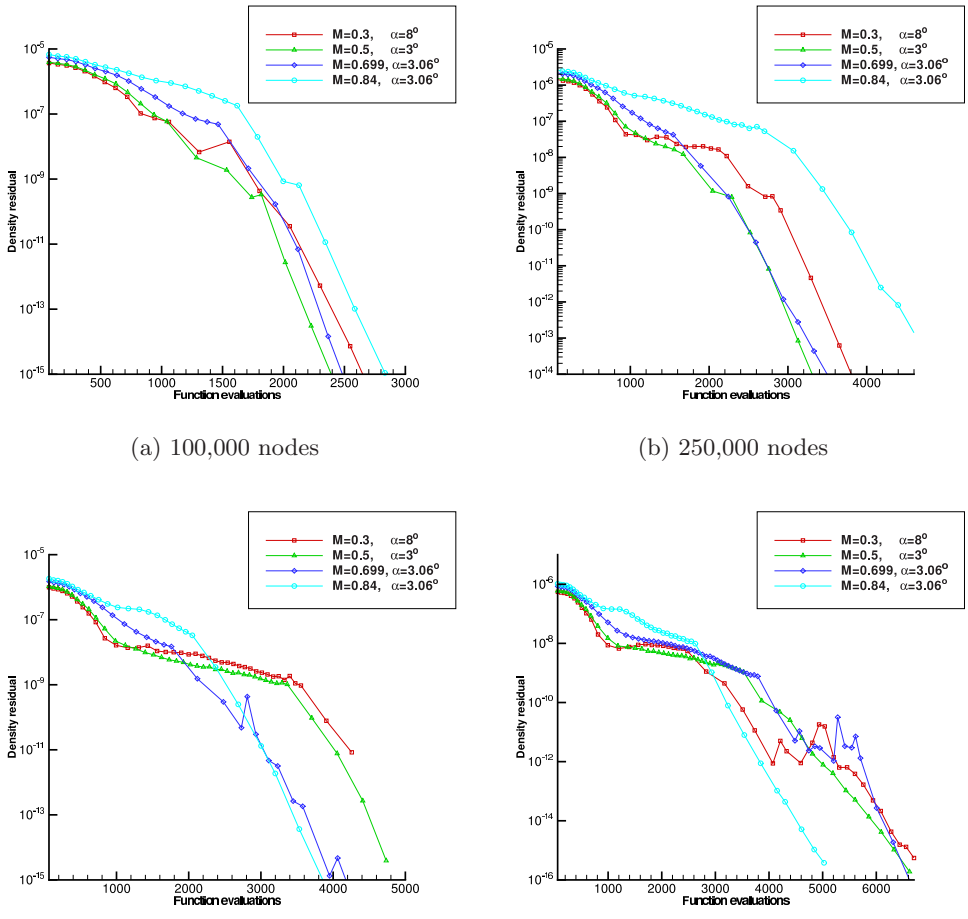
where $\beta = \|r_0\|$ and e_1 is the first column of the $m \times m$ identity matrix. Since the column-vectors of V_{m+1} are orthonormal,

$$\|r(y)\|_2 = \|\beta e_1 - \bar{H}_m y\|_2 \tag{B.7}$$

The third step is to find the x that minimizes the residual. This is reduced to finding y such that the function $\|r(y)\|_2$ is minimized. Since this is a $(m+1) \times m$ least-squares problem with m very small compared to N it is inexpensive. Once y is found, x_m is formed using Eq. B.5.

Appendix C

Additional results



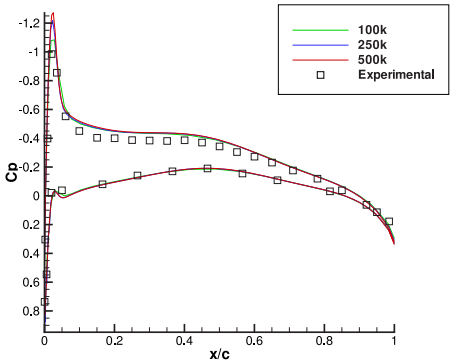
(a) 100,000 nodes

(b) 250,000 nodes

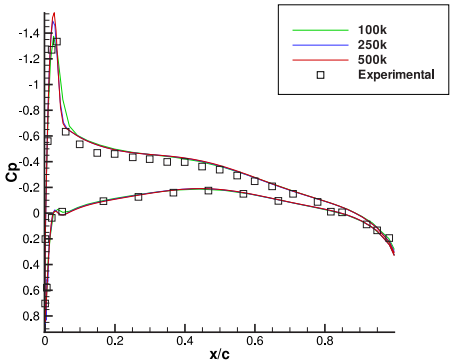
(c) 500,000 nodes

(d) 1,000,000 nodes

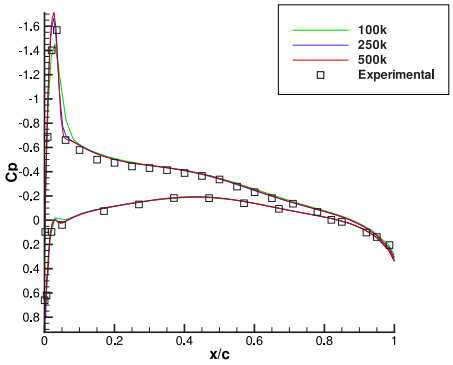
Figure C.1: Density residual vs. function evaluations



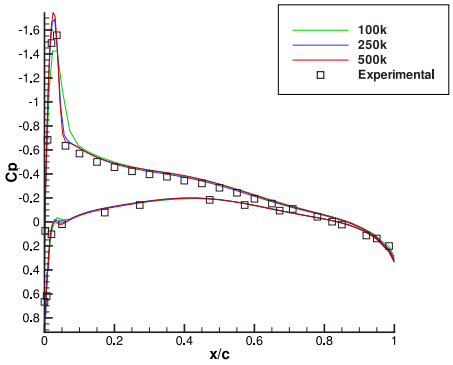
(a) 20% span



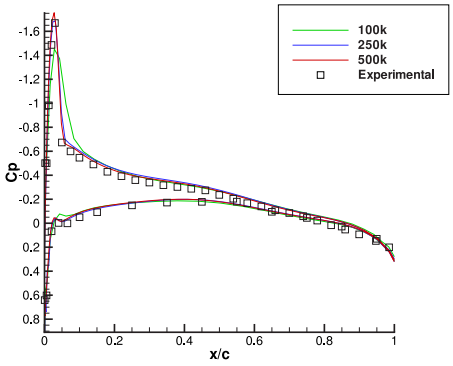
(b) 44% span



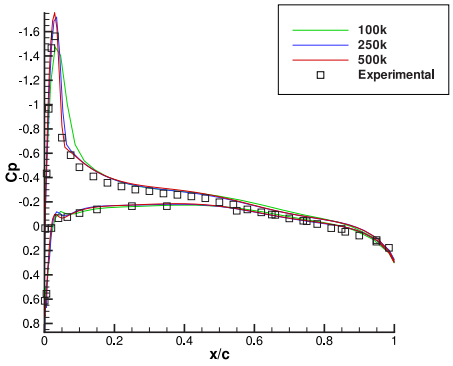
(c) 65% span



(d) 80% span



(e) 90% span



(f) 96% span

Figure C.2: ONERA M6 wing C_p distribution, $M = 0.699, \alpha = 3.06°$

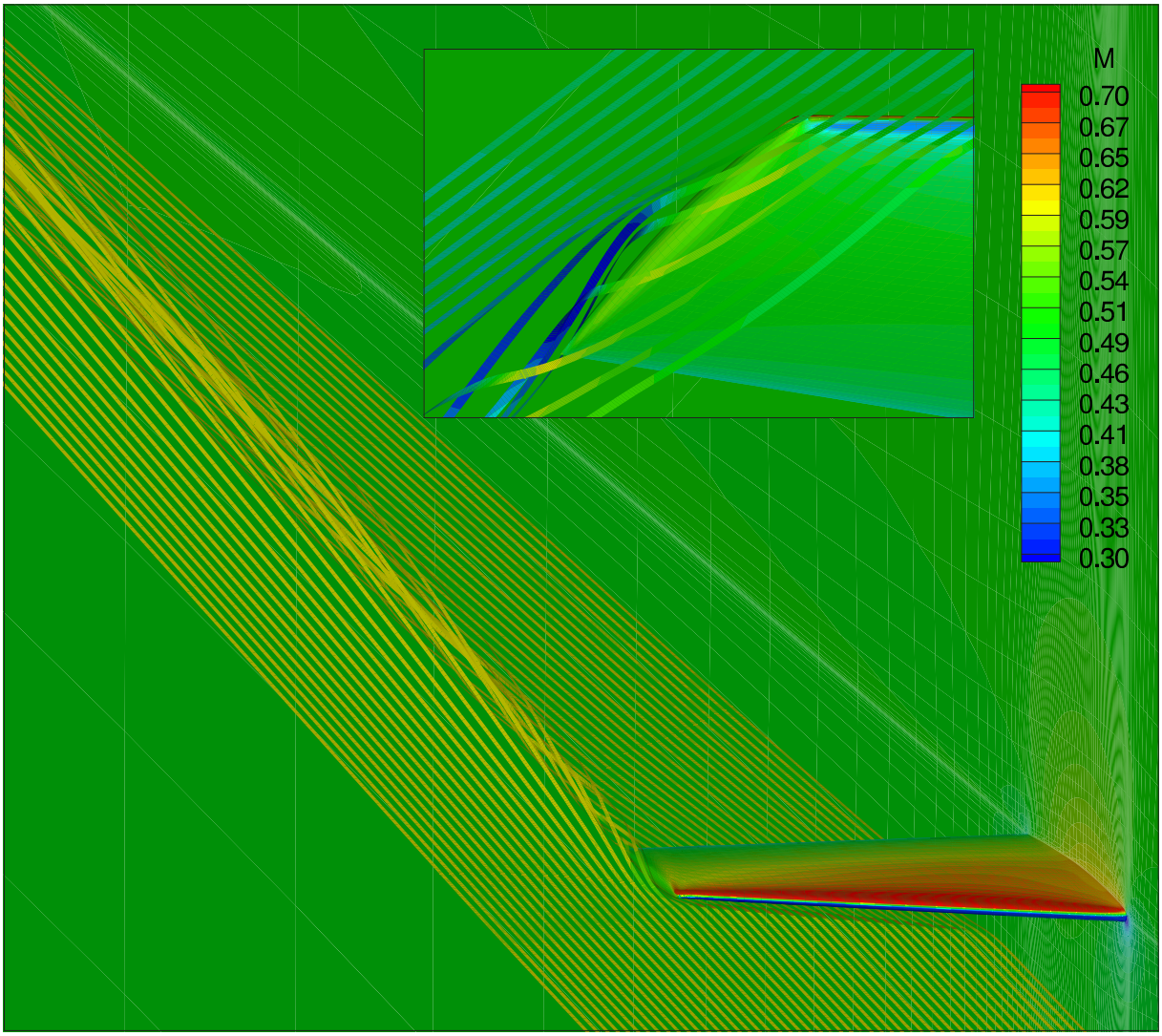


Figure C.3: ONERA M6 wing tip effects at $M = 0.5, \alpha = 5°$

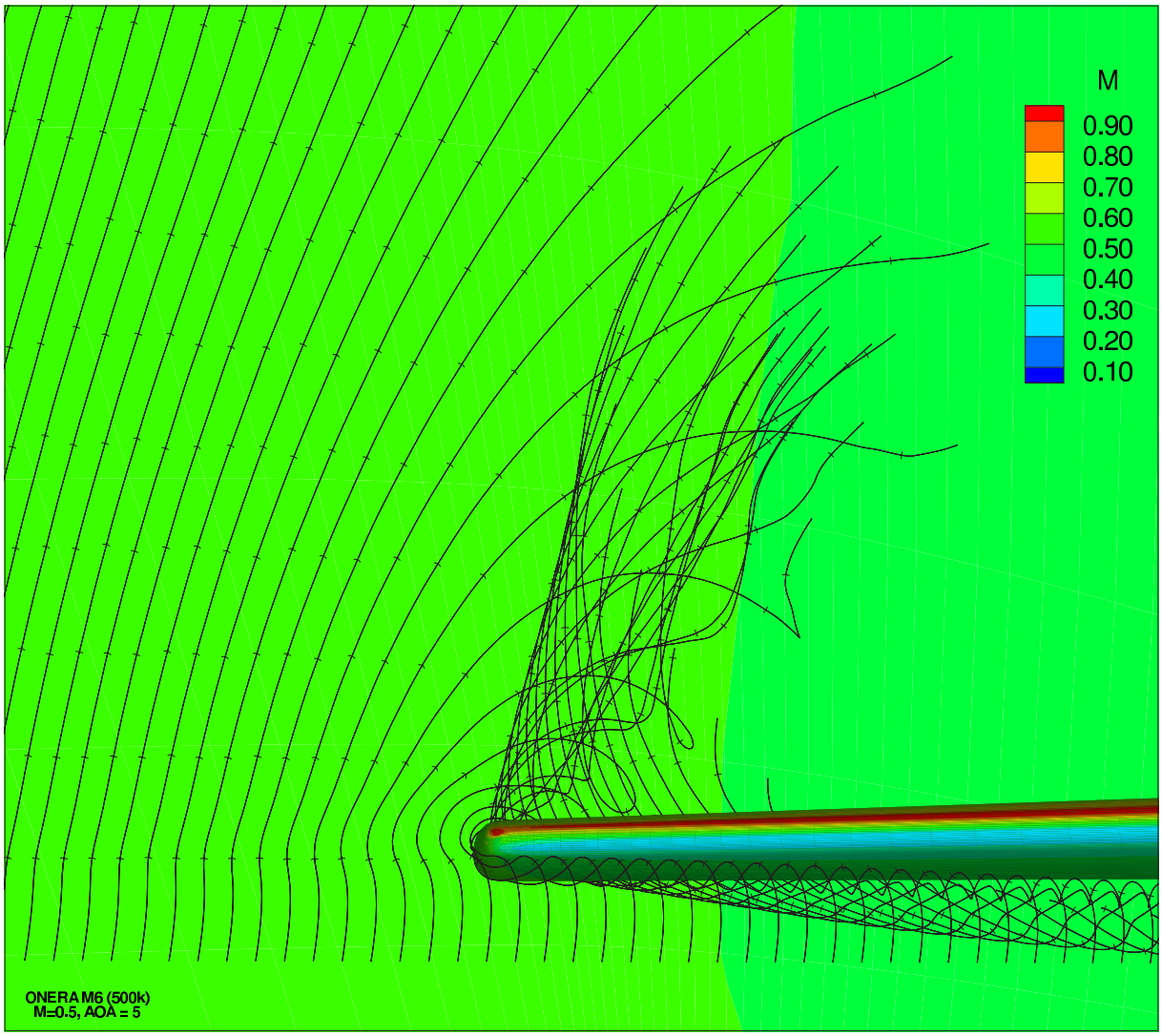


Figure C.4: ONERA M6 wing tip effects at $M = 0.5, \alpha = 5°$ (Front view)

Grid	Case	Lift Coefficient (C_L)	Drag Coefficient (C_D)
A	1	0.5911	0.031304
B	1	0.5679	0.028102
C	1	0.5670	0.027418
D	1	0.5729	0.029413
A	2	0.2274	0.002556
B	2	0.2227	0.003810
C	2	0.2210	0.003071
D	2	0.2225	0.002795
A	3	0.2552	0.005553
B	3	0.2502	0.005712
C	3	0.2491	0.004843
D	3	0.2510	0.004820
A	4	0.3048	0.011558
B	4	0.3017	0.011564
C	4	0.3014	0.010785
D	4	0.3031	0.010693

Table C.1: TYPHOON solver lift and and drag coefficients