AN EFFICIENT NEWTON-KRYLOV METHOD FOR THE EULER AND NAVIER-STOKES EQUATIONS

by

ALBERTO PUEYO

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

AN EFFICIENT NEWTON-KRYLOV METHOD FOR THE EULER AND NAVIER-STOKES EQUATIONS

Doctor of Philosophy, 1998

Alberto Pueyo

Graduate Department of Aerospace Science and Engineering

University of Toronto

Abstract

An efficient inexact-Newton-Krylov algorithm is presented for the computation of steady compressible aerodynamic flows on structured grids. The spatial discretization consists of a second-order centered-difference operator with the second and fourth-difference dissipation model of Jameson et al. The Baldwin-Lomax algebraic model is used for turbulent flows. The thin-layer Navier-Stokes equations are linearized using Newton's method. Preconditioned restarted GMRES in matrix-free form is used to solve the linear system arising at each Newton iteration. The preconditioner is formed using an incomplete factorization of an approximate-Jacobian matrix after applying a reordering technique.

An optimization study is presented to obtain an efficient parameter-free solver for a wide range of flows. An inexact-Newton strategy that avoids oversolving is established. Comparison between different preconditioners of the incomplete-lower-upper factorization family is presented. The best performance/memory ratio was obtained for the Block-Fill ILU(2) preconditioner. A parametric optimization of the approximate-Jacobian used to produce well-conditioned LU factors is also shown. Different reordering techniques are considered; results show that the Reverse Cuthill-McKee is the most efficient technique.

The algorithm has been successfully applied to a wide range of test cases which include inviscid, laminar, and turbulent aerodynamic flows. In all cases except one, convergence of the residual to 10^{-12} is achieved with a CPU cost equivalent to fewer than 1000 function evaluations. The sole exception is a low Mach number case where some form of local preconditioning is needed. Several other efficient implicit solvers have been applied to the same test cases, and the matrix-free inexact-Newton-GMRES algorithm is seen to be the fastest and most robust of the methods studied.

Acknowledgments

These few years that I have spent at UTIAS have been a wonderful academic and human experience to which many people have contributed in different ways. I deeply thank each and everyone for their contribution.

I am grateful in a very special way to my supervisor Professor D. W. Zingg for his gentle and demanding guidance during my doctoral studies. His constant motivation, his comments, suggestions and insights while respecting my own initiative, have been very inspiring. In spite of having a large research group, his total availability and dedication for each one of his students has been greatly appreciated. I finally thank him for sharing so many good moments of discussion, which gave me the opportunity to learn a great deal from him, both academically and humanly.

I would like to thank the members of my Doctoral Advisory Committee, Professor Gottlieb and Professor Hansen for their various suggestions and insights that have helped me throughout my research.

There is a good researcher and friend to whom I am particularly thankful too, Dr. Laura Dutto. I appreciate her patience when I have bombarded her with questions. The many interesting discussions that we have had have been very enlightening for me. I'm also thankful to her for providing me the necessary subroutines to test certain reordering algorithms.

I would like to thank Dr. Maryse Page, a friend and colleague, for having pointed me to some good literature at the beginning of my research.

Stan DeRango deserves a special place in my gratitude. His true friendship, his constant good humour and the fact that he has always been ready to give a hand has greatly facilitated my work and made it very enjoyable.

There are three other guys in the CFD group whose help has been very precious. I am referring to the three Linux *gurus*, Jason, Todd and Mike, who have helped me constantly since I switched to this operating platform.

A big thanks also to everyone at UTIAS, particularly to the directors and staff, the people in the CFD group, the soccer team and those working in the ASA, for making my staying at UTIAS so enjoyable.

My parents and family, as well as many friends and colleagues, deserve my gratitude in a special way for their constant support and encouragement.

Finally, I am very grateful to those who supported me financially during these years: the Government of Québec, the Government of Ontario, the University of Toronto, my supervisor and my parents.

Contents

List of Figures

List of Tables

Nomenclature

Alphanumeric

$\overline{AB}$	upper wake
$\overline{CD}$	lower wake
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_v	viscous flux in x-direction
F	inviscid flux in y-direction
F_v	viscous flux in y-direction
H	enthalpy
J	metric Jacobian
K_m	Krylov subspace of dimension m
M	Mach number
M	transformation matrix given by Eq. (3.47)
N	number of equations, equal to four times the number of nodes
P	matrix defined by Eq. (3.46)
P	parameter in the ILUT factorization
Q	vector of conservative variables
R	set of independent variables used at the boundaries
$R^{+,-}$	Riemann invariants
S	entropy
S	viscous flux in the thin-layer approximation
U	contravariant velocity
V	contravariant velocity
V_m	$N \times m$ matrix containing the basis of the Krylov subspace
V_n	normal component of the velocity
V_t	tangential component of the velocity

a	speed of sound
b	left-hand-side of the linear system solved by GMRES
c	chord of the airfoil
c_p	specific heat at constant pressure
e	total energy
e_i	internal energy
$h_{i,j}$	elements of the Hessenberg matrix
j	node coordinate in computational domain
j_{t1}	j-coordinate of upper trailing edge
j_{t2}	j-coordinate of lower trailing edge
k	node coordinate in computational domain
m	size of the Krylov subspace

$m_{i,j}$ elements of the matrix $\mathcal{M}$
n normal distance from the wall
p pressure
p level of fill-in in ILU
r_m residual of the linear system after m GMRES-iterations
u x-component of the velocity
u_τ friction velocity
v y-component of the velocity
v_j jth vector of the basis of the Krylov subspace
x coordinate in the physical domain
x vector of unknowns of the linear system
x_m mth iterate of the linear system
y coordinate in the physical domain
z_j preconditioned directions in preconditioned GMRES

Caligraphic

$\mathcal{A}$ Jacobian matrix of $\mathcal{F}$
$\tilde{\mathcal{A}}$ an approximation of $\mathcal{A}$
$\mathcal{A}_p$ preconditioned Jacobian
$\mathcal{A}_1$ first-order Jacobian
$\mathcal{A}_2$ second-order Jacobian
$\mathcal{B}$ discretized boundary conditions equations
$\mathcal{E}$ error matrix
$\mathcal{F}$ discretized system of equations. Right-hand-side
$\mathcal{I}$ identity matrix
$\mathcal{L}$ lower factor in LU factorization
$\mathcal{M}$ preconditioning matrix
$\mathcal{P}r$ Prandtl number
$\mathcal{P}r_t$ turbulent Prandtl number
$\mathcal{R}e$ Reynolds number
$\mathcal{U}$ upper factor in LU factorization

Greek

Δt local time step; Eq. (3.42)
Δt_0 local time step constant; Eq. (3.42)
Γ circulation around the airfoil

α angle of attack
γ ratio of specific heats

ϵ	scalar used to perturb state quantities in matrix-free GMRES; Eq. (3.70)
$\varepsilon^{(2)}$	second-difference dissipation coefficient (in ξ-direction)
$\varepsilon^{(4)}$	fourth-difference dissipation coefficient (in ξ-direction)
η	coordinate in the computational domain
$\tilde{\eta}_n$	relative reduction of residual in the nth Newton step
κ_t	thermal conductivity
μ	dynamic viscosity
μ_t	eddy viscosity
ξ	coordinate in the computational domain
ρ	density
σ	artificial dissipation constant in $\mathcal{A}_1$; Eq. (3.41)
$\sigma^{(\xi)}$	spectral radius (in the ξ-direction)
τ	parameter in the ILUT factorization
τ_w	shear stress at the wall
ω	vorticity

Abbreviations

ADI	Alternating Direction Implicit
AF	Approximate Factorization
BCG	Bi-Conjugate Gradient
Bi-CGSTAB	Bi-Conjugate Gradient Stabilized
BILU	Block Incomplete Lower-Upper factorization
BFILU	Block-Fill Incomplete Lower-Upper factorization
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrich-Lewy number
CG	Conjugate Gradient
CGS	Conjugate Gradient Squared
CPU	Central Process Unit
CSR	Compressed Sparse Row storing format
DB	Double Bandwidth ordering
DD1	Domain Decomposition ordering 1
DD2	Domain Decomposition ordering 2
FOM	Full Orthogonalization Method
GMRES	Generalized Minimal Residual
GS	Gauss-Seidel
ILU	Incomplete Lower-Upper factorization
ILUT	Incomplete Lower-Upper factorization with threshold
LGS	Line Gauss-Seidel
LUSSOR	Lower-Upper Symmetric Successive Over-Relaxation
MN	Minimum Neighbouring reordering
MSR	Modified Sparse Row storing format
NAT	Natural ordering

QMR	Quasi-Minimal Residual
RCM	Reverse Cuthill-McKee reordering
SIP	Strongly Implicit Procedure
SSOR	Symmetric Successive Over-Relaxation
TFQMR	Transposed Free Quasi-Minimal Residual

f.e.	function evaluations
i-it	inner-iterations or GMRES-iterations
mf	matrix free
nnz	number of nonzeros
o-it	outer-iterations or Newton-iterations
tm	time marching

Chapter 1

Introduction

1.1 Background

Prior to the mid 60's, different mathematical formulations had been developed to guide the design process in aerodynamics. Well known examples are the airfoil theory of Kutta and Joukowski, the wing and boundary layer theories of Prandtl, and Hayes' theory of linearized supersonic flow. These methods, which require significant simplifying assumptions, could not be used for quantitative studies of complex flows and configurations. Therefore, the development of aerodynamics had to rely heavily on experimental work. The primary tool in aerodynamic design was the wind tunnel, where shapes were tested and modified before building flying prototypes. However, experimental design is quite expensive. For example, 20,000 hours of wind tunnel testing were expended in the development of the General Dynamics F111 and the Boeing 747 [1].

The computer opened a new avenue for the development of more sophisticated mathematical models that could simulate flows of interest to a much higher level of accuracy. This development has made of computational fluid dynamics (CFD) a key tool in aerodynamic design.

In the last 30 years, a great effort has been made to come up with computational methods that can predict a wide range of complex flows. Unfortunately, in many cases they require a great amount of computer time, which limits their use in practical applications. The purpose of our study is to find an efficient algorithm to compute inviscid and viscous solutions for steady flows around airfoils.

Section 1.2 consists of an overview of the most efficient numerical methods that have been developed over the years. The following section is a review of Newton-Krylov methods, the specific type of methods that we are interested in. A section discussing the objectives of the thesis closes the present chapter. The main portion of the thesis follows, divided into five chapters. The governing equations, including the turbulence model and boundary conditions, are presented in chapter 2. Chapter 3 contains a detailed description of the numerical algorithm. Optimization of the algorithm for airfoil calculations is presented in chapter 4. In chapter 5, algorithm efficiency and performance comparison with other solvers are discussed. The last chapter includes conclusions, main contributions and recommendations of our study.

1.2 Review of some classic iterative methods for steady flows

Many algorithms have been proposed for efficient computation of steady aerodynamic flows. Their development has followed two distinct paths: explicit methods and implicit methods. Modern implicit and explicit codes show similar performance with regard to total computational time. In fact, it is hard to draw a clear line between the two approaches: most explicit methods have some sort of implicit scheme built-in to accelerate convergence such as implicit residual smoothing, and many implicit methods have some sort of approximation added to make iterations cheaper or to ensure convergence.

Explicit methods are easier to code and less computationally intensive, but they present stability limits. They can be traced back to the early work of MacCormack [2] who, in 1969, introduced a predictor-corrector algorithm that for a number of years remained as one of the most efficient algorithms. For large problems and for stiff turbulent flow problems, the convergence rates of these methods degrade rapidly.

The development of multigrid techniques dramatically accelerated the convergence of explicit algorithms. This technique was introduced for transonic potential flows in the late 70's [3, 4]. Ni [5] and Jameson [6] extended the application of multigrid to the Euler equations. Application to the Navier-Stokes equations was done a few years later by Martinelli et al. [7]. Mavriplis [8] and other researchers have developed similar schemes. Jameson's approach generally includes an explicit multi-stage iterative method, local time-stepping, and implicit residual smoothing. This approach has received considerable use for aerodynamic flows [1].

Implicit methods permit larger time steps, computing the solution in far less iterations. On the other hand, the cost per iteration can be significantly higher, since a large linear system of equations has to be solved at each time step. The earliest implicit methods were based on the Alternating Direction Implicit (ADI) scheme pioneered by Douglas and Gunn [9] and Peaceman and Rachford [10]. Stone [11] introduced the Strongly Implicit Procedure (SIP) in 1968. The Approximate Factorization methods (AF), which are among the most popular and efficient implicit solvers, were introduced for the Euler and Navier-Stokes equations by Beam and Warming [12] and Briley and McDonald [13] in the mid 70's. Steger [14] used this algorithm, which reduces the work of a two-dimensional implicit operator to that of two one-dimensional implicit operators, in the well-known flow solver ARC2D. The computational work of this algorithm was further decreased by introducing a diagonalization of the blocks in the implicit operators as developed by Pulliam and Chaussee [15]. ARC2D was further developed by Pulliam [16] with the addition of local time-stepping, and grid sequencing. Recently, multigrid acceleration has been added [17, 18], increasing the convergence rate by factors of three to six, making this approach a very efficient one.

Another class of implicit solvers, named Upwind Relaxation solvers, was introduced in the mid-80's. They were studied by several authors: Chakravarthy [19], Van Leer and Mulder [20], Thomas and Walters [21], and Walters and Dwoyer [22] to name some. Jameson and Yoon [23] developed a Lower-Upper Implicit scheme which they proved to be related to the family of Upwind Relaxation solvers. A multigrid method was combined with this scheme to speed up convergence. The scheme was eventually replaced by an explicit scheme, the Lower-Upper Symmetric-Gauss-Seidel method [24], increasing convergence by 30%.

Many of the fastest available iterative methods rely on the multigrid acceleration technique to achieve good convergence performance. Unfortunately, multigrid convergence can slow down if high aspect ratio cells are present. In the late 80's, several authors have considered using Newton's method as a possible alternative for steady flows due to its property of quadratic convergence. At each Newton step, a large linear system of equations has to be solved. Some examples of Newton's method using a direct solver for the linear systems of equations can be found in Refs. [25] to [30]. This approach was found to be robust, but memory and the CPU time required to reach steady state are not competitive with the methods mentioned earlier. On the other hand, quasi-Newton methods have shown promise. Quasi-Newton methods can be classified as inexact-Newton methods or approximate-Newton methods. In an inexact-Newton method, the large linear system arising at each Newton step is solved approximately, using an iterative solver. Dembo et al. [31] presented theoretical results regarding the precision to which the linear system must be solved to preserve superlinear or quadratic convergence of the Newton process. In an approximate-Newton method, the functional Jacobian is simplified, thus producing an approximate linearization. The linear system is again solved iteratively. Newton-like schemes have great potential for becoming very efficient solvers but the challenge is to significantly reduce the cost per iteration. A new family of iterative methods called Krylov methods, opened the door to advances in solvers based on Newton's method. An overview of Newton-Krylov schemes is presented in the following section.

1.3 Newton-Krylov methods

Krylov subspace methods are iterative methods to solve linear and non linear systems of equations, searching for the solution within a Krylov subspace. The Conjugate Gradient method of Hestenes and Stiefel [32] is the oldest and best known method of this class. It is applicable only to Hermitian positive definite matrices, which greatly limits its use in CFD applications. Fortunately, many Krylov methods have been developed for non-Hermitian matrices. Some examples are the Full Orthogonalization Method (FOM) [33], ORTHORES [34], ORTHOMIN [35], Generalized Minimal Residual (GMRES) [36], Bi-Conjugate Gradient (BCG) [37, 38], Conjugate Gradient Squared (CGS) [39], Bi-Conjugate Gradient Stabilized (Bi-CGSTAB) [40] and Quasi-Minimal Residual (QMR) [41]. A summary of these methods can be found in Refs. [42] and [43]. These methods are typically used to solve the linear system of equations at each Newton step. The use of a preconditioner, which transforms the linear system into one that is better conditioned and thus easier to solve by the iterative solver, is necessary in many practical applications.

The use of Krylov iterative methods in CFD started in the early 80's with the work of Wong and Hafez [44], Wong [45] and Prince [46], with applications to the potential flow equations. Since then, a number of authors have applied these methods to different flow problems. We are particularly interested in the research done on Krylov methods combined with Newton's method to solve the Euler and Navier-Stokes equations.

Wigton et al. [47] were the first to solve the Navier-Stokes equation using a Krylov subspace method. They made use of nonlinear GMRES, which can be viewed as a Newton linearization in which GMRES is used to solve the linear system of equations at each Newton step, and where the matrix-vector multiplications are replaced by a Frechet derivative. Instead of using a preconditioner to improve GMRES convergence, as is usually done, they solved a set of already preconditioned equations. The Euler equations were preconditioned using existing solvers, such as ARC2D [16] and FLO53P [48]. In our view, it is a very relevant work, not only because they were the first to apply a Krylov method to the Euler equations, but also because they were the first to use a matrix-free implementation of the Krylov solver and to the preconditioner.

Venkatakrishnan [49] developed an approximate-Newton-Krylov method for structured grids, with emphasis on vector performance issues. He concluded that his method was competitive with other existing methods. Together with Mavriplis [50], he extended the work to unstructured grids. They tested the solver with inviscid as well as with laminar and turbulent viscous flows. The approximations that they introduced in Newton's linearization are at the level of the artificial dissipation and the viscous fluxes. They only used first-order artificial dissipation on the left-hand side due to storage considerations. Regarding the viscous fluxes, the laminar viscosities, computed with Sutherland's law, and the turbulence model, which was nondifferentiable, were not linearized. Therefore, quadratic convergence was not attained. In order to reduce the stiffness of the Jacobian, they also added a time step term to the diagonal. The time-step was taken to be inversely proportional to the L_2 norm of the residual; it also had an upper limit. The approximations introduced in the linearization make the linear systems easier to solve for GMRES, which requires fewer iterations to converge, but the number of Newton-iterations increases substantially. The linear systems were solved to a moderate degree of precision. They tested three preconditioners: block-diagonal, Incomplete Lower-Upper factorization with no-fill (ILU(0)) and Symmetric Successive Over-Relaxation (SSOR). They concluded that GMRES/ILU(0) was the best approach. This conclusion was particularly evident as the size and the stiffness of the problem increased. They found their strategy competitive with explicit multigrid solvers. Further development of the method as well as considerations on parallelization are discussed in Ref. [51].

At about the same time, Dutto [52] used nonlinear GMRES to solve the system of equations that results from applying implicit time-marching methods to the Navier-Stokes equations, to solve inviscid and laminar viscous flows. The advantage of this approach is that it is applicable to unsteady calculations. For steady-state calculations, the time step provides a way of controlling the stiffness of the problem. She used a restarted version of GMRES in order to control memory usage. At each restart, she not only updated the residual vector to build a new Krylov subspace, but she also updated the solution used as the reference state in the Frechet derivatives, which is similar to modifying the Jacobian in the equivalent linear system that is solved. Therefore, a complete new linear system was solved at each restart of GMRES. This can be seen as doing several Newton iterations at each time step as

GMRES restarts. Therefore, there are three levels of iterations: the implicit time-marching method, several Newton steps to converge each time step and GMRES to solve the Newton-linearized system. Other related works of the same author include a study of the impact that the ordering of the unknowns has on the performance of GMRES [53] and a study of parallelizable block diagonal preconditioners [54]. Lately, Dutto et al. [55] have developed an efficient two-level parallelizable preconditioner which consists of two independent approximations of the system matrix: a block-diagonal preconditioner combined with a coarser matrix built using algebraic multigrid methods.

Johan et al. [56] developed a solution algorithm for implicit time-marching schemes. The basic algorithm is similar to that of Dutto. They use matrix-free GMRES(20) with block diagonal preconditioning. In order to increase robustness, they use linesearch backtracking and an automatic time-increment algorithm. The CFL number is always kept relatively low to improve the condition number of the linear systems. The solver was applied to inviscid and laminar viscous flows.

Ajmani et al. [57, 58] used an approximate-Newton linearization, solving the linear system with preconditioned GMRES. They compared Block ILU(0) (BILU(0)) and Lower-Upper SSOR (LUSSOR) as preconditioners, concluding than the latter was more efficient. The new solver was compared to the conventional implicit line Gauss-Seidel solver and an Approximate Factorization solver in the context of laminar flows around a hypersonic cylinder and a transonic turbine cascade. They found that the Newton-Krylov approach was much faster than the two classical solvers. Even if we take into account that the Approximate Factorization solver did not include the diagonal form, which considerably speeds up the algorithm, their results proved that Newton-Krylov methods are competitive with standard algorithms. In more recent years, Ajmani and Liou [59] compared GMRES, Bi-CGSTAB and QMR in parallel architectures. Results indicated that GMRES seems to be the solver of choice.

Habashi et al. [60] used an approximate-Newton strategy combined with CGS to solve laminar viscous flows. They used ILU(0) as a preconditioner. A finite time step was used in the Jacobian matrix to improve its condition number. They showed that the performance of the algorithm scales well with the number of unknowns.

Hixon and Sankar [61] applied Wigton's approach to unsteady calculations. At each time step, the system of nonlinear equations is preconditioned by an ADI algorithm. The preconditioned system of equations is solved by nonlinear GMRES. When multigrid was also added, the code speeded up for steady calculations but no appreciable gain was noticed for unsteady calculations.

Orkwis [62] did a very interesting evaluation of the performance of an exact Newton method and several quasi-Newton methods: an inexact-Newton method, an approximate-Newton method solving the linear systems exactly, and an approximate-Newton method solving the linear systems inexactly. CGS was used for the inexact matrix inversions. The quasi-Newton methods were preconditioned with ILU(0) applied to the corresponding Jacobian and allowing fill-in within the 4×4 blocks . He used a supersonic turbulent viscous flow over a flat plate as test case. He concluded that inexact matrix inversions with large subiterate convergence tolerances were faster in terms of CPU time. For the same level of tolerance, the approximate-Newton method and the inexact-Newton method were equally fast. He reported that CGS failed to converge for a problem with a strong shock.

Lin et al. [63] used an approximate-Newton algorithm to test three Krylov solvers, CGS, Bi-CGSTAB and TFQMR, in the context of turbulent axisymmetric flows. They incorporated the $k - \epsilon$ two-equation turbulence model. They concluded that Bi-CGSTAB and TFQMR were slightly faster than CGS. The Newton-Krylov method was more efficient than an Approximate Factorization method.

Knoll and McHugh [64, 65] compared the performance of standard and matrix-free implementations of an inexact-Newton-Krylov method. CGS, TFQMR, Bi-CG and GMRES were included in their study. They used a incompressible steady flow in a cavity as test case. They concluded that the matrix-free implementation was strongly dependent upon grid size and the choice of Krylov method. GMRES appeared to be superior to the other three solvers in the matrix-free implementation.

Degrez and Issman [66] developped an inexact matrix-free Newton-Krylov solver for the Navier-Stokes equations. The preconditioner for the Krylov solver was based on an Approximate Directional Factorization (ADF) or an Approximate LU factorization of a first-order approximation of the inviscid flux balance. Comparisons with the ADF algorithm were made for a flow in a chanel, a flow over a flat plate and a flow over a hypersonic ramp. The Newton-GMRES algorithm was considerably faster. Recently, they have used a multigrid algorithm as preconditioner for the matrix-free GMRES [67].

Following Wigton's approach, Hager and Lee [68, 69] tested ADI, ILU(0) and a four-stage Runge-Kutta solver, with and without multigrid, as preconditioners for the Euler equations for nonlinear GMRES. They used a supersonic flow over a ramp as test case. They concluded that GMRES does not consistently improve the convergence of the three schemes when they were used with multigrid.

Luo et al. [69] used a classical approximate-Newton approach with BILU(0) as preconditioner to solve 2D inviscid and laminar viscous flows, as well as some 3D inviscid flows. Jorgenson and Pletcher [70] tested three different Krylov solvers preconditioned with ILU as an alternative to the implicit Gauss-Seidel scheme used in their laminar viscous flow solver. Local preconditioning was added to handle low Mach number laminar viscous flows. GMRES was significantly faster than the other solvers.

Rogers [71] wrote an approximate-Newton-GMRES solver for incompressible flows. He compared this approach with Point-Jacobi Relaxation, Gauss-Seidel Relaxation and BILU(0). He used these solvers as preconditioners for GMRES. He concluded that GMRES preconditioned with BILU(0) outperformed all other methods by at least a factor of 2.

Barth and Linton [72] developed a matrix-free Newton-GMRES method preconditioned with ILU(0) for compressible 2D and 3D turbulent viscous flows. They presented a new technique for constructing matrix-vector products which is an exact calculation of the directional derivatives. For 3D calculations, their code was implemented on a parallel architecture using a message protocol with favorable scalability characteristics.

Forsyth and Jiang [73] compared different standard quasi-Newton methods in the context of inviscid two-dimensional flows. The linear systems were solved using CGSTAB preconditioned with ILU. Fill-in within the 4×4 blocks was allowed in the factorization. Several levels of fill-in were tested. The preconditioner was built from the same matrix used in the linear systems. They concluded that the approximate-Newton method failed to converge for supersonic flows with strong shocks, and that it was slower than the inexact-Newton method for the other cases. However, the inexact-Newton method required a fill-in level of 2 or more in the ILU factorization to converge. Considering that they used the high-order Jacobian for the factorization, the storage required by the preconditioner was quite high. They reached similar conclusions when they extended their work to laminar viscous flows [74].

Cai et al. [75] developed a Newton-Krylov method, preconditioned with an overlapping Schwarz domain decomposition which relies primarily on local information for data parallel concurrency. They claimed that this strategy was well suited for solving nonlinear elliptic systems in high-latency, distributed-memory environments. They applied their solver to incompressible inviscid flows. McHugh et al. [76] also developed a Schwarz-preconditioned matrix-free Newton-Krylov algorithm for low speed combustion flows.

Nielsen et al. [77] applied a Newton-Krylov scheme to an unstructured Euler code for two and three dimensions. The implicit-Euler time marching method was used, gradually increasing the the time step until Newton convergence was obtained. They evaluated three different methods to define the increasing time step. They also presented an effective choice for the perturbation constant used in the finite difference used in matrix-free GMRES. Comparisons with Barth and Linton's [72] matrix-free method showed that both methods have similar convergence in terms of CPU time. They compared the Newton-Krylov method with a Gauss-Seidel 3-level W-multigrid method. The Newton-Krylov method required more computer time, but if mesh sequencing was used for the first two orders of magnitude, their performance was comparable.

Anderson et al. [78] presented a comparison of different quasi-Newton-Krylov methods with a multigrid Gauss-Seidel scheme for incompressible inviscid flows. The quasi-Newton methods were used to solve the nonlinear equations resulting from employing the implicit Euler time marching method. The approximate-Newton method and the matrix-free Newton-GMRES method preconditioned with a BILU(0) factorization of the approximate-Jacobian, had a similar performance in terms of CPU time. Both methods converged faster when mesh sequencing or multigrid was added. Nevertheless, the Gauss-Seidel scheme with multigrid was faster and required less memory than the quasi-Newton Krylov methods.

Choquet et al. [79] solved the Navier-Stokes equations for laminar flows over airfoils and for a hypersonic reactive two-dimensional viscous flow. They applied matrix-free GMRES to an implicit-time marching method. Diagonal preconditioning was used for GMRES. 25 to 50 search directions were required. Search backtracking combined with a moderate CFL number ensured robustness. Their solver showed comparable performance and CPU time with a point Jacobi solver. Choquet [80] also developed a matrix-free preconditioner for the matrix-free Newton-GMRES method. The main idea was to reuse the Krylov subspace information to build a preconditioner that can be used across the Newton iterations and the time steps. Experimental results showed that the preconditioner was effective across the time steps and only slightly effective across the Newton iterations. Tests were done on inviscid and laminar viscous unsteady and steady flows using a low CFL number.

Delanaye et al. [81, 82] used a matrix-free Newton-GMRES method in the context of a new quadratic reconstruction finite-volume scheme. For steady flows, they employed the implicit Euler time marching method in order to control the stiffness of the matrix with the time step parameter. For unsteady calculations, they used the trapezoidal implicit method or the three-point backward implicit method. BILU(1) and BILU(2) applied to the approximate Jacobian were as efficient preconditioners for GMRES as BILU(0) applied to the exactly-linearized Jacobian.

Ollivier-Gooch [83] used Wigton's approach to solve the Euler equations preconditioned locally via block Jacobi. The totally matrix-free Newton-GMRES solver was applied to the change in solution over a multigrid cycle driven by a three-stage Runge-Kutta scheme. Matrix-free GMRES was applied after the maximum-residual had dropped four or five orders of magnitude using the multigrid scheme.

In the context of inviscid calculations on unstructured grids, Blanco and Zingg [84] made some comparisons between an approximate-Newton method and two inexact-Newton methods, one building the high-order Jacobian and the other with a matrix-free implementation of the Krylov solver. In all cases, an ILU factorization of the lower-order Jacobian was used to build the preconditioner. Results showed the superiority of the inexact-Newton method over the approximate-Newton method. The matrix-free implementation was also faster than the standard implementation. It was suggested that for transonic flows, the approximate-Newton method should be used to reduce the initial residual by three orders of magnitude, before switching to the inexact-Newton method. A level of fill-in equal to 4 was found to be optimal for this method.

Mavriplis [85] applied Wigton's approach to his low-Mach number preconditioned directional-coarsening line-implicit smoother multigrid scheme. He employed 20 or 30 search directions for GMRES. The addition of GMRES to his solver nearly doubled the convergence rate in some of the cases tested.

A summary of the above research efforts is shown in Table 1.1. We indicate the earliest reference to the work of that particular author and, occasionally, another relevant reference. What appears as a matrix-free implementation of GMRES, i.e. *mf*, is often called *nonlinear* GMRES by other authors; *mf-tm* means a matrix-free implementation applied to an implicit time-marching method; $\mathcal{A}_2$ means that the exact Jacobian is used for standard matrix-vector products, while an approximate Jacobian is used when indicated by $\mathcal{A}_1$; *mf*-$\mathcal{A}_1$ indicates that a matrix-free approach is used but with a some modification to the function evaluation that makes the algorithm an approximate-Newton method. The applications consist of compressible turbulent flows unless otherwise specified.

1.4 Objectives

The previous section discussed a wide variety of Newton-Krylov schemes. Some of them appear to be very promising. For example, Venkatakrishnan and Mavriplis [50] found their approximate-Newton strategy to be competitive with their state of the art multigrid solver. However, we believe that the full potential of quasi-Newton methods has not been realized. Thus our objective is to develop and optimize a highly efficient Newton-Krylov solver for aerodynamic calculations, and to compare it with well-established solvers, such as the approximate factorization solver ARC2D used at NASA.

An important aspect of Newton-Krylov solvers is that, since there is a wide range of options available in simplifying the system Jacobian matrix, preconditioning the system, and iteratively solving the system, there are several parameters that need to be chosen. It is not possible to predict an optimal set for a particular problem. Therefore, another main objective in our research is to find an optimized set of parameters and strategies which will make the resulting quasi-Newton method able to efficiently handle a great variety of flows, without having to readjust those choices.

These issues are addressed here in the context of inviscid, laminar, and turbulent flows over airfoils using a centered finite-difference operator with non-linear artificial dissipation.

Year	Author(s)	Jac.	Solver -s- & Preconditioner -p-	Application	Performance comparison
85	Wigton, L.B. et al. [47]	mf	GMRES(20) -s- Precond. Eqs.	airfoils inviscid	without GMRES
90	Venkatakrishnan, V. Mavriplis, D. [49, 50]	$\mathcal{A}_1$	GMRES -s- ILU(0) of $\mathcal{A}_1$ -p-	airfoils; inviscid and viscous	adaptive Chebychev SSOR, ILU(0), MG
90	Dutto, L.C. [52, 53]	mf-tm	GMRES -s- BFILU of $\mathcal{A}_1$ -p-	airfoils, inlet; unsteady laminar	several orderings CG-S, Bi-CGSTAB
91	Johan, Z. et al. [56]	mf-tm	GMRES(20) -s- Block diagonal -p-	2D bodies inv. and laminar	
91	Ajmani, K. et al. [57, 58]	$\mathcal{A}_1$	GMRES -s- BILU(0) of $\mathcal{A}_1$ -p- LUSSOR -p-	hypersonic cylinder turbine cascade laminar	Line Gauss Seidel Approx. Factoriz.
91	Habashi, W.G et al. [60]	$\mathcal{A}_1$	CGS -s- ILU(0) of $\mathcal{A}_1$ -p-	flow on diffuser $\mathcal{R}e = 1000$	
92	Hixon, R. Sankar, L.N. [61]	mf-tm MG	GMRES -s- Precond. Eqs.	airfoils, unsteady viscous	
93	Orkwis, P.D. [62]	$\mathcal{A}_1$,$\mathcal{A}_2$	CGS -s- BFILU(0) of $\mathcal{A}$ -p-	flat plate supersonic, viscous	Newton & quasi-Newton
93	Lin, H. et al. [63]	$\mathcal{A}_1$	several Krylov solvers BILU of $\mathcal{A}_1$ -p-	axisymmetric, viscous	Approx. Factoriz.
93	Knoll, D.A. McHugh, P.R. [64]	$\mathcal{A}_2$,mf	several Krylov solvers ILU(0) of $\mathcal{A}_2$-p-	flow in cavity incompr.	Standard vs. mf
94	Degrez, G. Issman, E. [66]	mf	GMRES -s- ADF & ALU of $\mathcal{A}_1$ -p-	channel, flat plate, hypers. ramp, viscous	Approx. Directional Factorization
94	Hager, J.O. Lee, K.D. [68]	mf	GMRES(k) -s- Precond. Eqs.	wedge; inviscid supersonic flow	
94	Luo, H. et al. [69]	$\mathcal{A}_1$	GMRES -s- BILU(0) of $\mathcal{A}_1$ -p-	airfoils and aircraft laminar 2D inviscid 3D	
94	Jorgenson, P.C.E. Pletcher, R.H. [70]	$\mathcal{A}_2$	several Krylov solvers ILU -p-	internal, laminar compressible and incompr.	point and block Gauss-Seidel
95	Rogers, S.E. [71]	$\mathcal{A}_1$	GMRES -s- several precond.	airfoils and channel incompr. viscous	Jacobi, Line-GS, BILU(0)
95	Barth, T.J. Linton, S.W. [72]	mf, $\mathcal{A}_2$	GMRES -s- ILU(0) of $\mathcal{A}_1$ -p-	airfoils and wings viscous	
95	Forsyth, P.A. Jiang, H. [73]	$\mathcal{A}_1$,$\mathcal{A}_2$	CGSTAB -s- BFILU of $\mathcal{A}_2$ -p-	airfoils inviscid	approx. Newton levels of fill-in
95	Nielsen, E.J. et al. [77]	mf	GMRES(20) -s- ILU(0) of -$\mathcal{A}_1$ -p-	airfoils, aircrafts inviscid	point GS with multigrid
95	Cai, X. et al. [75]	mf	GMRES(25) -s- Schwarz + BILU(0) -p-	airfoils incompr., inviscid	approx. Newton
95	Anderson, W.K. et al. [78]	mf-$\mathcal{A}_1$ MG	GMRES(25) -s- BILU(0) -p-	airfoils, inviscid & viscous, incompr. 2D & 3D	point GS with multigrid
95	Choquet, R. [80]	mf	GMRES -s- matrix-free -p-	airfoils laminar	
95	Delanaye, M. et al. [81]	mf-tm	GMRES -s- BILU(1) -p-	airfoils laminar	
95	Ollivier-Gooch, C.F. [83]	mf	GMRES -s- Precond. Eqs.	airfoils, inviscid and laminar	without GMRES
97	Blanco, M. Zingg, D.W. [84]	mf, $\mathcal{A}_2, \mathcal{A}_1$	GMRES(25) -s- ILU(4) of $\mathcal{A}_1$	airfoils inviscid	standard vs. mf levels of fill-in
97	Dutto, L.C. et al. [55]	$\mathcal{A}_2$	GMRES -s- Two-level precond.	airfoils, 3D cavity laminar	
97	Mavriplis, D.J. [85]	mf	GMRES -s- Precond. Eqs.	airfoils viscous	

Table 1.1: Summary of published Newton-Krylov methods for the Euler and Navier-Stokes equations.

Chapter 2

Governing equations

In this chapter we present the governing equations of air flows around airfoils, in non-dimensional form. A brief description of the transformation into generalized curvilinear coordinates follows in Section 2.2. The thin-layer approximation is described in Section 2.3, and the Baldwin-Lomax turbulence model in Section 2.4. A description of the boundary conditions closes this chapter.

2.1 The Navier-Stokes equations

The governing equations for aerodynamic flows are the Navier-Stokes equations. We write them as a function of the non-dimensional Cartesian conservative variables given by

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ e \end{bmatrix} \tag{2.1}$$

where we scale the dimensional variables, density ($\tilde{\rho}$), velocity ($\tilde{u}, \tilde{v}$) and total energy ($\tilde{e}$), as

$$\rho = \frac{\tilde{\rho}}{\tilde{\rho}_\infty}, \quad u = \frac{\tilde{u}}{\tilde{a}_\infty}, \quad v = \frac{\tilde{v}}{\tilde{a}_\infty}, \quad e = \frac{\tilde{e}}{\tilde{\rho}_\infty \tilde{a}_\infty^2} \tag{2.2}$$

where ∞ refers to free-stream quantities and a is the speed of sound, which for ideal fluids is $a^2 = \gamma p/\rho$. The ratio of specific heats, γ, is taken as 1.4 for air. The total energy per unit volume is given by the internal energy and the kinetic energy

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

Using the equation of state for a perfect gas, pressure is related to the conservative flow variables as follows:

$$p = (\gamma - 1)\left(e - \frac{1}{2}\rho(u^2 + v^2)\right) \tag{2.4}$$

With this set of variables, the conservative form of the Navier-Stokes equations for a steady two-dimensional flow is:

$$\partial_x E + \partial_y F = M_\infty \mathcal{R}e^{-1}(\partial_x E_v + \partial_y F_v) \tag{2.5}$$

The inviscid and the viscous flux terms are

$$E = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(e+p) \end{bmatrix}, \quad F = \begin{bmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ v(e+p) \end{bmatrix}, \quad E_v = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ E_{v,4} \end{bmatrix}, \quad F_v = \begin{bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ F_{v,4} \end{bmatrix} \tag{2.6}$$

with

$$\begin{aligned} \tau_{xx} &= (\mu + \mu_t)(4u_x - 2v_y)/3 \\ \tau_{xy} &= (\mu + \mu_t)(u_y - v_x) \\ \tau_{yy} &= (\mu + \mu_t)(-2u_x + 4v_y)/3 \\ E_{v,4} &= u\tau_{xx} + v\tau_{xy} + (\mu \mathcal{P}r^{-1} + \mu_t {\mathcal{P}r_t}^{-1})(\gamma - 1)^{-1}\partial_x a^2 \\ F_{v,4} &= u\tau_{xy} + v\tau_{yy} + (\mu \mathcal{P}r^{-1} + \mu_t {\mathcal{P}r_t}^{-1})(\gamma - 1)^{-1}\partial_y a^2 \end{aligned} \tag{2.7}$$

where $\mu = \tilde{\mu}/\tilde{\mu}_\infty$ is the non-dimensional dynamic viscosity, μ_t is the non-dimensional turbulent eddy viscosity, $\mathcal{R}e$ is the Reynolds number, $\mathcal{P}r$ is the Prandtl number and $\mathcal{P}r_t$ is the turbulent Prandtl number. The Prandtl number is defined by

$$\mathcal{P}r = \frac{c_p \mu}{\kappa_t} \tag{2.8}$$

where κ_t is the thermal conductivity and c_p the specific heat at constant pressure. The Prandtl number is taken constant with values set to $\mathcal{P}r = 0.72$ and $\mathcal{P}r_t = 0.90$. Using the chord of the airfoil c as the reference length, we define the Reynolds number as

$$\mathcal{R}e = \frac{\rho_\infty \; c \; u_\infty}{\mu_\infty} \tag{2.9}$$

The Euler equations are obtained by setting the right hand side of Eq. (2.5) equal to zero.

2.2 Generalized curvilinear coordinate transformation

We solve the Navier-Stokes equations numerically using a structured C-grid such as the one shown in Figure 2.1. The equations are first transformed from Cartesian coordinates to generalized curvilinear coordinates. As shown in Figure 2.2, the resulting computational space is a rectangular domain. The transformation, given by

$$\xi = \xi(x, y), \qquad \eta = \eta(x, y) \tag{2.10}$$

is chosen so that the grid spacing in the computational space is uniform and equal to one. It should be noted that there is a one to one correspondence between grid points in the original physical space and the ones in computational space, except for the nodes at the wakecut and the trailing edge, which map into two nodes in the computational space.

The details of the transformation can be found in [16]. Eq. (2.5) becomes

$$\partial_\xi \hat{E} + \partial_\eta \hat{F} = M_\infty \mathcal{R}e^{-1}(\partial_\xi \hat{E}_v + \partial_\eta \hat{F}_v) \tag{2.11}$$

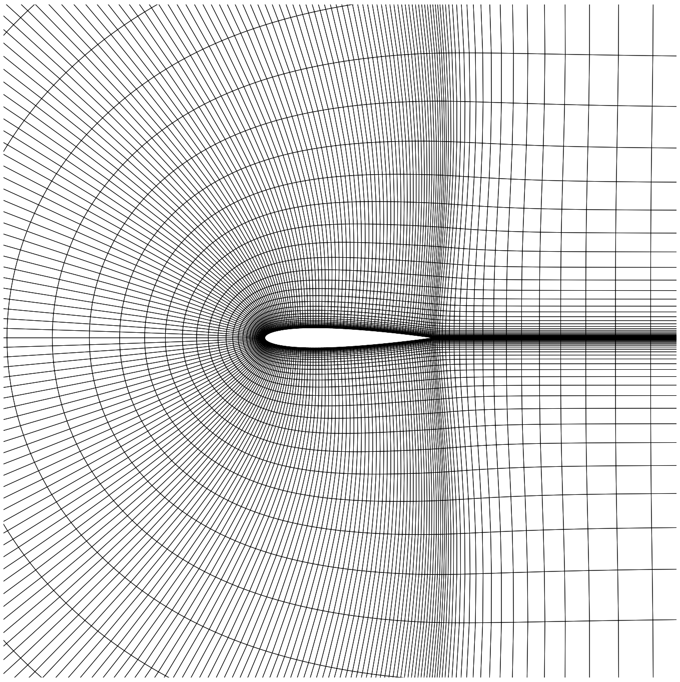


Figure 2.1: "C" grid for a NACA 0012 airfoil.

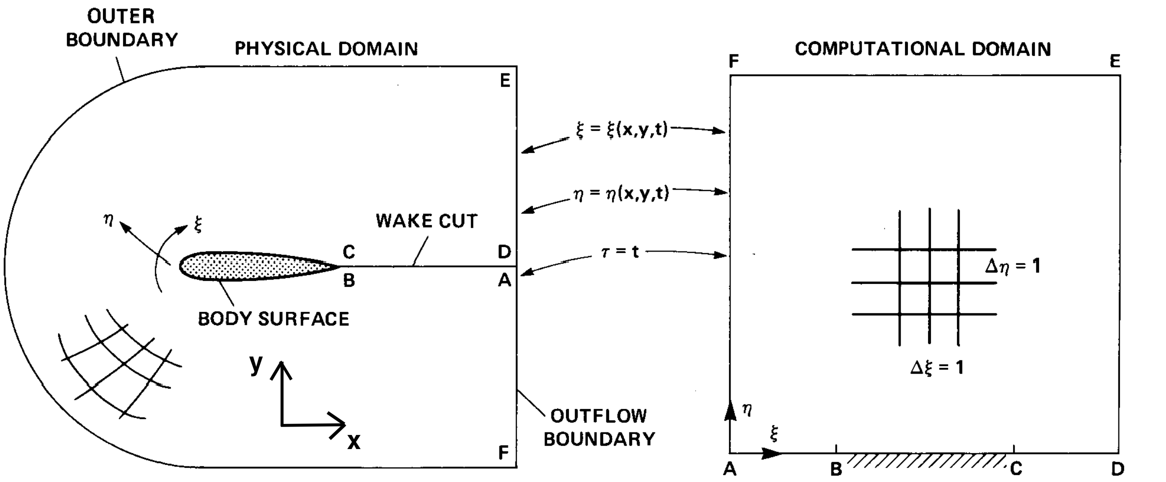


Figure 2.2: Generalized curvilinear coordinate transformation. (Supplied by Tom Pulliam, NASA Ames.)

where the vector of unknowns is

$$\hat{Q} = J^{-1} \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ e \end{bmatrix} \tag{2.12}$$

and

$$\hat{E} = J^{-1} \begin{bmatrix} \rho U \\ \rho U u + \xi_x p \\ \rho U v + \xi_y p \\ (e+p)U \end{bmatrix}, \qquad \hat{F} = J^{-1} \begin{bmatrix} \rho V \\ \rho V u + \eta_x p \\ \rho V v + \eta_y p \\ (e+p)V \end{bmatrix} \tag{2.13}$$

with the contravariant velocities

$$U = \xi_x u + \xi_y v, \quad V = \eta_x u + \eta_y v \tag{2.14}$$

The variable J represents the metric Jacobian of the transformation:

$$J^{-1} = (x_\xi y_\eta - x_\eta y_\xi) \tag{2.15}$$

The viscous flux terms are $\hat{E_v} = J^{-1}(\xi_x E_v + \xi_y F_v)$ and $\hat{F_v} = J^{-1}(\eta_x E_v + \eta_y F_v)$. The stress terms are

$$\begin{aligned}
\tau_{xx} &= (\mu + \mu_t)(4(\xi_x u_\xi + \eta_x u_\eta) - 2(\xi_y v_\xi + \eta_y v_\eta))/3 \\
\tau_{xy} &= (\mu + \mu_t)(\xi_y u_\xi + \eta_y u_\eta + \xi_x v_\xi + \eta_x v_\eta) \\
\tau_{yy} &= (\mu + \mu_t)(-2(\xi_x u_\xi + \eta_x u_\eta) + 4(\xi_y v_\xi + \eta_y v_\eta))/3 \\
E_{v,4} &= u\tau_{xx} + v\tau_{xy} + (\mu \mathcal{P}r^{-1} + \mu_t \mathcal{P}{r_t}^{-1})(\gamma - 1)^{-1}(\xi_x \partial_\xi a^2 + \eta_x \partial_\eta a^2) \\
F_{v,4} &= u\tau_{xy} + v\tau_{yy} + (\mu \mathcal{P}r^{-1} + \mu_t \mathcal{P}{r_t}^{-1})(\gamma - 1)^{-1}(\xi_y \partial_\xi a^2 + \eta_y \partial_\eta a^2)
\end{aligned} \tag{2.16}$$

2.3 Thin-layer approximation

In flows with high Reynolds numbers where the flow is attached or just mildly separated, the viscous terms associated with derivatives along the body are negligible.

For this reason, and in order to save storage and CPU time, highly stretched grids are used to resolve the normal gradients of the flow near the rigid surfaces, without resolving the diffusion terms involving derivatives parallel to those surfaces. If we drop all the viscous derivatives in the ξ direction in Eq. (2.11), we obtain the thin-layer Navier-Stokes equations. Unlike in the boundary layer equations, the normal momentum equation is solved and no assumptions are made regarding the pressure. The thin-layer equations are

$$\partial_\xi \hat{E} + \partial_\eta \hat{F} = M_\infty \mathcal{R}e^{-1} \partial_\eta \hat{S} \tag{2.17}$$

where

$$\hat{S} = J^{-1} \begin{bmatrix} 0 \\ \eta_x m_1 + \eta_y m_2 \\ \eta_x m_2 + \eta_y m_3 \\ \eta_x (u m_1 + v m_3 + m_4) + \eta_y (u m_2 + v m_3 + m_5) \end{bmatrix} \tag{2.18}$$

with

$$\begin{aligned} m_1 &= (\mu + \mu_t)(4\eta_x u_\eta - 2\eta_y v_\eta)/3 \\ m_2 &= (\mu + \mu_t)(\eta_y u_\eta + \eta_x v_\eta) \\ m_3 &= (\mu + \mu_t)(-2\eta_x u_\eta + 4\eta_y v_\eta)/3 \\ m_4 &= (\mu \mathcal{P}r^{-1} + \mu_t \mathcal{P}r_t^{-1})(\gamma - 1)^{-1} \eta_x \partial_\eta (a^2) \\ m_5 &= (\mu \mathcal{P}r^{-1} + \mu_t \mathcal{P}r_t^{-1})(\gamma - 1)^{-1} \eta_y \partial_\eta (a^2) \end{aligned} \tag{2.19}$$

2.4 Turbulence model

The effects of turbulence can be approximated by adding an eddy viscosity term μ_t to the dynamic viscosity μ in the fashion shown in Eqs. (2.7), (2.16) and (2.19). Turbulence models differ in the way that μ_t is calculated. In our study, we use the Baldwin-Lomax [86] two-layer algebraic eddy viscosity model which is patterned after that of Cebeci and Smith [87]. The modifications introduced avoid the need for finding the edge of the boundary layer. The eddy viscosity is given by

$$\mu_t = \begin{cases} (\mu_t)_{inner} & n \leq n_{cr} \\ (\mu_t)_{outer} & n_{cr} < n \end{cases} \tag{2.20}$$

where n is the normal distance from the wall and n_{cr} is the smallest value of n at which values from the inner and outer formulas are equal.

In the inner region, the Prandtl-Van Driest formulation is used

$$(\mu_t)_{inner} = \rho\, l^2\, |\omega| \tag{2.21}$$

where

$$l = k\, n\, [1 - e^{-n^+/A^+}] \tag{2.22}$$

The magnitude of the vorticity is given by

$$|\omega| = |u_y - v_x| \tag{2.23}$$

and the law-of-the-wall coordinate n^+ is

$$n^+ = \frac{\rho_w\, u_\tau\, n}{\mu_w} = \frac{\sqrt{\rho_w\, \tau_w}\;\, n}{\mu_w} \tag{2.24}$$

The subscript w denotes values at the wall, u_τ is the friction velocity, $\sqrt{\tau_w/\rho_w}$, and τ_w is the shear stress at the wall.

For the outer region,

$$(\mu_t)_{outer} = K\, C_{cp}\, \rho\, F_{wake}\, F_{kleb}(n) \tag{2.25}$$

where K is the Clauser constant, C_{cp} is an additional constant, and

$$F_{wake} = \min \begin{cases} n_{max}\, F_{max} \\ C_{wk}\, y_{max}\, u_{dif}^2 / F_{max} \end{cases} \tag{2.26}$$

The values of F_{max} and y_{max} are determined from the function

$$F(n) = n\,|\omega|\,[1 - e^{-n^+/A^+}] \tag{2.27}$$

In wakes, the exponential term of Eq. (2.27) is set equal to zero. The value n_{max} is the value of n at which $F(n)$ reaches its maximum F_{max} in a profile. The function $F_{kleb}(n)$ is the Klebanoff intermittency factor

$$F_{kleb}(n) = \left[1 + 5.5\left(\frac{C_{kleb}\,n}{n_{max}}\right)^6\right]^{-1} \tag{2.28}$$

The value of u_{dif}, the difference between maximum and minimum total velocity in the profile, is given by

$$u_{dif} = \begin{cases} \left(\sqrt{u^2+v^2}\right)_{max} & \text{in boundary layers} \\ \left(\sqrt{u^2+v^2}\right)_{max} - \left(\sqrt{u^2+v^2}\right)_{min} & \text{in wakes} \end{cases} \tag{2.29}$$

The constants that appear in the above equations were determined by Baldwin and Lomax by requiring agreement with the Cebeci formulation for constant pressure boundary layers at transonic speeds. They are

$$\begin{aligned} A^+ &= 26 \\ C_{cp} &= 1.6 \\ C_{kleb} &= 0.3 \\ C_{wk} &= 0.25 \\ k &= 0.4 \\ K &= 0.0168 \end{aligned}$$

2.5 Boundary conditions

The computational domain for an external flow around an airfoil described in Figure 2.2 presents three types of boundaries: body surface boundaries, far-field boundaries and the wakecut. Properly speaking, the wakecuts are not boundaries. They are just interior nodes that need a different consideration. This point will be discussed in Section 2.5.4. The interior differencing scheme requires the solution at the boundaries. Where this solution is not provided by boundary conditions, it is determined by extrapolation from the interior of the domain. These additional equations are often called "numerical boundary conditions." They cannot be imposed arbitrarily; they have to be based on stability and accuracy considerations.

Before describing the boundary conditions, we need to define the tangent and normal directions at each boundary. We define the tangent $\mathbf{t}$ in the positive sense of ξ at the surfaces $\overline{ABCD}$ and $\overline{FE}$, and in the positive sense of η at the surfaces $\overline{DE}$ and $\overline{AF}$. Since the grid is not orthogonal, the normal does not have, in general, the same direction as the corresponding η or ξ direction; the normal $\mathbf{n}$ is perpendicular to the tangent and positive in the same sense of the numbering of the nodes. This is illustrated in Figure 2.3. The resulting normal and tangential components of the velocity are

$$
\left.\begin{aligned}
V_n &= \frac{\eta_x\, u + \eta_y\, v}{\sqrt{\eta_x^2 + \eta_y^2}} = \overline{\eta}_x u + \overline{\eta}_y v \\
V_t &= \frac{\eta_y\, u - \eta_x\, v}{\sqrt{\eta_x^2 + \eta_y^2}} = \overline{\eta}_y u - \overline{\eta}_x v
\end{aligned}\right\} \text{ at } k = 1 \text{ and } k = k_{max}
$$

$$
\left.\begin{aligned}
V_n &= \frac{\xi_x\, u + \xi_y\, v}{\sqrt{\xi_x^2 + \xi_y^2}} = \overline{\xi}_x u + \overline{\xi}_y v \\
V_t &= \frac{-\xi_y\, u + \xi_x\, v}{\sqrt{\xi_x^2 + \xi_y^2}} = -\overline{\xi}_y u + \overline{\xi}_x v
\end{aligned}\right\} \text{ at } j = 1 \text{ and } j = j_{max}
\tag{2.30}
$$

2.5.1 Body surface

At the body surface (line $k = 1$ between points B and C in Figure 2.2), tangency must be satisfied for inviscid flows and the no-slip condition must be satisfied for viscous flows. Since the interior scheme requires four boundary conditions, we need

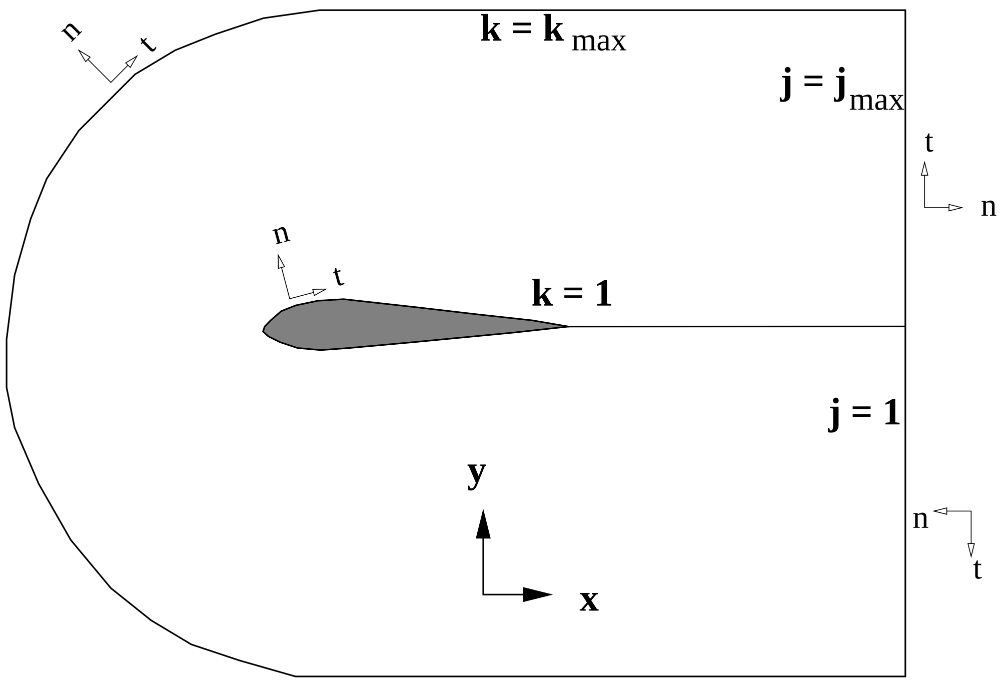


Figure 2.3: Normal and tangential directions at the boundaries.

to impose three more conditions for inviscid flows and two more for viscous flows. For inviscid flows, the four equations that we impose are

- velocity tangent to the body surface, $V_n = 0$.
- extrapolation of the tangential component of the velocity V_t from the interior.
- extrapolation of the pressure from the interior.
- stagnation enthalpy, $(e + p)/\rho$, set to free-stream value, H_∞ (steady flow).

For viscous flows, the four equations are

- two equations from imposing the no-slip condition, $u = 0$ and $v = 0$.
- gradient of p normal to the wall set to zero.
- either adiabatic or isothermal condition of the surface. We use adiabatic conditions in all calculations.

The extrapolation scheme used in each case will be discussed in Section 3.2.2.

boundary	inflow					outflow				
	V_n	R^+	R^-	S	V_t	V_n	R^+	R^-	S	V_t
$j = 1$	> 0	∞	$int.$	∞	∞	< 0	∞	$int.$	$int.$	$int.$
$j = j_{max}$	< 0	$int.$	∞	∞	∞	> 0	$int.$	∞	$int.$	$int.$
$k = k_{max}$	< 0	$int.$	∞	∞	∞	> 0	$int.$	∞	$int.$	$int.$

Table 2.1: Subsonic inflow and outflow boundary conditions: values of the Riemann invariants, entropy and tangential velocity are set to free-stream values (∞) or are extrapolated from the interior (*int.*) depending on the sign of V_n.

2.5.2 Far-field boundaries

Inviscid flows

For inviscid flows, locally one-dimensional Riemann invariants as well as V_t and entropy $S = \ln(p/\rho^\gamma)$ are used at the far-field boundaries. These four values are set to free-stream values or they are extrapolated from the interior flow variables depending on the slope of the corresponding characteristic. For the Riemann invariants

$$R^- = V_n - \frac{2a}{\gamma - 1} \qquad \text{and} \qquad R^+ = V_n + \frac{2a}{\gamma - 1} \tag{2.31}$$

the slopes of the corresponding characteristics are $V_n - a$ and $V_n + a$, and for the other two variables, V_n. For *supersonic* conditions, the four characteristics travel in the same direction. Therefore, the four variables are set to free-stream conditions for a supersonic inflow and they are extrapolated from the interior at a supersonic outflow. The same pattern holds for V_t and S in the *subsonic* regime because the slope of the corresponding characteristic has the same sign as for supersonic flows. The Riemann invariants require more careful attention in the subsonic regime. The way we have defined the normal and the tangent at each boundary, the logic for the Riemann invariants does not change whether it is an inflow or an outflow condition. The logic for applying the subsonic boundary conditions is shown in Table 2.1.

Viscous flows

At the $k = k_{max}$ boundary, the conditions for viscous inflow and outflow are determined in the same fashion as for inviscid flows. At the two downstream boundaries, $j = 1$ and $j = j_{max}$, the entropy gradients associated with convection of the wake make the characteristic analysis used for inviscid flows inappropriate. Experience indicates that simple zeroth-order extrapolation of ρ, ρu, ρv and p can be used, provided that non-reflective conditions are applied at other far-field boundaries in the domain.

2.5.3 Circulation correction

For lifting airfoils, the far-field boundary may affect the solution, unless it is placed very far away, which would require more nodes in the grid. In order to minimize the effect of the far-field boundary, Pulliam [16], following Salas et al. [88], added a compressible potential vortex solution as a perturbation to the free-stream velocity giving

$$\begin{aligned} u_f &= u_\infty + \frac{\beta \Gamma sin(\theta)}{2\pi r[1 - M_\infty^2 sin^2(\theta - \alpha)]} \\ v_f &= v_\infty - \frac{\beta \Gamma cos(\theta)}{2\pi r[1 - M_\infty^2 sin^2(\theta - \alpha)]} \end{aligned} \tag{2.32}$$

where $\Gamma = \frac{1}{2} M_\infty c C_l$, c is the chord of the airfoil, C_l the coefficient of lift, M_∞ the free-stream Mach number, α the angle of attack, $\beta = \sqrt{1 - M_\infty^2}$ and r and θ are the polar coordinates to the point of application on the far-field boundary relative to the quarter-chord point on the airfoil chord line. The speed of sound is also corrected to enforce constant free-stream enthalpy at the boundary:

$$a_f^2 = (\gamma - 1)\left(H_\infty - \frac{u_f^2 + v_f^2}{2}\right) \tag{2.33}$$

Pulliam [16] shows that with the far-field vortex correction, the lift has virtually no variation with the distance to the outer boundary for subcritical flows, and very small variation for transonic flows. Zingg's [89] grid studies confirm that an outer boundary position of 12 chords introduces virtually no error in lift and and small errors in drag relative to a far-field boundary set at 96 chords.

2.5.4 Wakecuts

Points B and C in Figure 2.3 represent the trailing edge of the airfoil. They correspond to nodes $(j_{t1}, 1)$ and $(j_{t2}, 1)$ in the grid. The wakecut that appears in the physical domain in Figure 2.2 corresponds to two wakecuts, $\overline{AB}$ and $\overline{CD}$, in the computational domain. Therefore there are two sets of grid-nodes in the computational domain which correspond to only one set of nodes in the physical domain. The wakecuts are defined as

$$\text{nodes } (j,k) \text{ such that } \begin{cases} 1 < j < j_{t1} & k = 1 & \text{wakecut } \overline{AB} \\ j_{t2} < j < j_{max} & k = 1 & \text{wakecut } \overline{CD} \end{cases} \tag{2.34}$$

Nodes at wakecuts are not boundary nodes. They are interior nodes whose neighbours are not nearby in the database. For example, when nodes from wakecut $\overline{CD}$ need information from $k-1$ we get it from across the wakecut:

$$Q_{j,k-1} \equiv Q_{j_{max}-j+1,k+1} \tag{2.35}$$

At wakecut $\overline{AB}$, we just impose that values of Q are equal to the ones computed at wakecut $\overline{CD}$. In other words, at wakecut $\overline{AB}$ the Navier-Stokes equations are replaced by

$$Q_{j,1} = Q_{j_{max}-j+1,1} \tag{2.36}$$

Chapter 3

Algorithm description

We begin this chapter by describing the spatial discretization used in the interior of the domain. This is followed by a description of the linearization of the resulting equations using Newton and approximate-Newton methods. The linear system of equations that arises at each Newton step is solved using GMRES, a Krylov iterative solver that is introduced in Section 3.4. The matrices that result from Newton linearization are very ill-conditioned and are not diagonally dominant. These characteristics make the linear systems hard to solve without preconditioning. Preconditioning techniques are discussed in Section 3.5, and reordering techniques that affect the preconditioner are introduced in the last section of this chapter.

3.1 Spatial discretization

The aerodynamic problems that we intend to solve have been modeled by the set of equations (2.17) to (2.19), together with the turbulence model and the boundary conditions. As a first step in solving this set of non-linear partial differential equations, we have to transform them into a system of algebraic equations. The second-order centered-difference operator used to approximate the differential operators ∂_ξ and ∂_η is described in Section 3.1.1. The second and fourth-difference dissipation model of Jameson et al. [90] is added to maintain stability and to prevents oscillations at shocks; it is described in Section 3.1.2. The resulting scheme is second-order accurate in space, except in the vicinity of shocks, where it is first-order. The spatial discretization is thus identical to that in ARC2D, the implicit finite difference Euler and Navier-Stokes solver for structured grids developed by Steger [14] and Pulliam [16].

3.1.1 Finite differencing

Let us recall that the computational domain described in Section 2.2 has a uniform grid spacing equal to unity. The variable $\hat{Q}$ at a grid point j, k is represented by

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

The finite difference operators that we use in this Section are defined as follows:

$$\left\{\begin{array}{lcll} \delta_\xi\, q_{j,k} & = & \left(q_{j+1,k} - q_{j-1,k}\right)/2 & \text{second-order central difference} \\ \nabla_\xi\, q_{j,k} & = & q_{j+1,k} - q_{j,k} & \text{first-order forward difference} \\ \Delta_\xi\, q_{j,k} & = & q_{j,k} - q_{j-1,k} & \text{first-order backward difference} \end{array}\right\} \tag{3.2}$$

The partial derivatives of the inviscid fluxes in Eq. (2.17), $\partial_\xi \hat{E}$ and $\partial_\eta \hat{F}$, are approximated by the second-order difference operator

$$\delta_\xi \hat{E}_{j,k} = \frac{\hat{E}_{j+1,k} - \hat{E}_{j-1,k}}{2} \qquad \text{and} \qquad \delta_\eta \hat{F}_{j,k} = \frac{\hat{F}_{j,k+1} - \hat{F}_{j,k-1}}{2} \tag{3.3}$$

The viscous terms in the equation take the form

$$\partial_\eta \left(\alpha_{j,k}\, \partial_\eta \beta_{j,k}\right) \tag{3.4}$$

The derivative $\partial_\eta \beta_{j,k}$ is approximated by a central difference at half nodes; the second derivative is approximated by a central difference at the grid nodes using the values computed at half nodes. Therefore, we can approximate Eq. (3.4) by

$$\Delta_\eta \left(\alpha_{j,k+\frac{1}{2}}\, \nabla_\eta \beta_{j,k}\right) = \alpha_{j,k+\frac{1}{2}}(\beta_{j,k+1} - \beta_{j,k}) - \alpha_{j,k-\frac{1}{2}}(\beta_{j,k} - \beta_{j,k-1}) \tag{3.5}$$

Values of α at half nodes are computed by averaging the values at the closest grid nodes,

$$\alpha_{j,k+\frac{1}{2}} = \frac{\alpha_{j,k+1} + \alpha_{j,k}}{2} \tag{3.6}$$

3.1.2 Artificial dissipation

It is well-known that central difference schemes experience odd and even point decoupling. Therefore, the high frequency modes have to be damped using artificial dissipation in order to achieve convergence. The elimination of oscillations in the neighborhood of shock waves also requires the use of artificial dissipation. There are several suitable schemes, such as the scalar second and fourth-difference dissipation model of Jameson et al. [90], the scalar model with scaling based on Mach number and vorticity of Hall [91], the scalar model with scaling based on the cell Reynolds number of Frew and Zingg [92] and the matrix dissipation model of Swanson and Turkel [93]. We use the first one in order to converge to the same solutions as ARC2D.

The description of the scheme is done for the terms in the ξ direction. Similar formulas can be written for the η direction. Written as difference operators, the second and fourth-difference dissipation terms are

$$D_{j,k}^{(2\xi)} = \Delta_\xi \left[\varepsilon_{j+\frac{1}{2},k}^{(2)} \nabla_\xi \left(J_{j,k} \hat{Q}_{j,k} \right) \right] \tag{3.7}$$

$$D_{j,k}^{(4\xi)} = \Delta_\xi \left[\varepsilon_{j+\frac{1}{2},k}^{(4)} \nabla_\xi \Delta_\xi \nabla_\xi \left(J_{j,k} \hat{Q}_{j,k} \right) \right] \tag{3.8}$$

The second-difference dissipation coefficients are

$$\varepsilon_{j+\frac{1}{2},k}^{(2)} = 2 \left(\sigma^{(\xi)} J^{-1} \epsilon^{(2)} \right)_{j+\frac{1}{2},k} \tag{3.9}$$

where $\sigma^{(\xi)}$ is the spectral radius scaling of the flux Jacobian matrix $\partial \hat{E} / \partial \hat{Q}$ (matrix $\partial \hat{F} / \partial \hat{Q}$ in the η direction)

$$\sigma_{j,k}^{(\xi)} = |U| + a\sqrt{\xi_x^2 + \xi_y^2} \tag{3.10}$$

and

$$\epsilon^{(2)}_{j,k} = \kappa_2 \Delta t \ \mathrm{Max}(\Upsilon_{j+1,k}, \Upsilon_{j,k}, \Upsilon_{j-1,k}) \tag{3.11}$$

The pressure gradient coefficient Υ is used to scale the second-difference dissipation so that its value is increased near shocks in order to avoid overshoots. The pressure gradient coefficient is defined as

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

The fourth-difference dissipation coefficients are

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

To avoid oscillations near shocks, this logic switches $\varepsilon^{(4)}$ off when the second-difference dissipation coefficient is larger than a certain value. In the present work, the values of κ_2 in Eq. (3.11) and κ_4 in Eq. (3.13) are fixed to 0.5 and 0.01 respectively.

Using Eq. (3.7), and noting that $J\hat{Q} = Q$, we obtain the following expression,

$$D^{(2\xi)}_{j,k} = \varepsilon^{(2)}_{j+\frac{1}{2},k} Q_{j+1,k} - \left(\varepsilon^{(2)}_{j+\frac{1}{2},k} + \varepsilon^{(2)}_{j-\frac{1}{2},k}\right) Q_{j,k} + \varepsilon^{(2)}_{j-\frac{1}{2},k} Q_{j-1,k} \tag{3.14}$$

Similarly, Eq. (3.8) becomes,

$$\begin{aligned} D^{(4\xi)}_{j,k} \ = \ & \varepsilon^{(4)}_{j+\frac{1}{2},k} Q_{j+2,k} - \left(3\varepsilon^{(4)}_{j+\frac{1}{2},k} + \varepsilon^{(4)}_{j-\frac{1}{2},k}\right) Q_{j+1,k} + \\ & \left(3\varepsilon^{(4)}_{j+\frac{1}{2},k} + 3\varepsilon^{(4)}_{j-\frac{1}{2},k}\right) Q_{j,k} - \left(\varepsilon^{(4)}_{j+\frac{1}{2},k} + 3\varepsilon^{(4)}_{j-\frac{1}{2},k}\right) Q_{j-1,k} + \varepsilon^{(4)}_{j-\frac{1}{2},k} Q_{j-2,k} \end{aligned} \tag{3.15}$$

The stencil of Eq. (3.15) cannot be used at the first interior node (i.e., $j = 2$ and $j = j_{max} - 1$); it has to be modified to a one-sided second-order stencil. For example, at $j = 2$,

$$\begin{aligned} D^{(4\xi)}_{j,k} \ = \ & \varepsilon^{(4)}_{j+\frac{1}{2},k} Q_{j+2,k} - \left(3\varepsilon^{(4)}_{j+\frac{1}{2},k} + \varepsilon^{(4)}_{j-\frac{1}{2},k}\right) Q_{j+1,k} + \\ & \left(3\varepsilon^{(4)}_{j+\frac{1}{2},k} + 2\varepsilon^{(4)}_{j-\frac{1}{2},k}\right) Q_{j,k} - \left(\varepsilon^{(4)}_{j+\frac{1}{2},k} + \varepsilon^{(4)}_{j-\frac{1}{2},k}\right) Q_{j-1,k} \end{aligned} \tag{3.16}$$

Similar stencils are used at the other boundaries. In the η direction, at wakecut $\overline{\mathbf{CD}}$ and at nodes located one row above both wakecuts, $Q_{j,k-2}$ and $Q_{j,k-1}$ have to be replaced by the corresponding values across the wakecut, as shown in Section 2.5.4.

3.1.3 Boundary conditions

Body surface

The boundary conditions at the body surface are described in Section 2.5.1. For an *inviscid* flow, we use a first-order extrapolation for V_t and for the pressure. Note that, in order to reduce errors in extrapolation, the velocities used in Eqs. (2.30) do not contain the term J^{-1}: they are the velocities in the physical domain. In order to eliminate e from the stagnation enthalpy, we make use of Eq. (2.4), the equation of state. Thus, the boundary conditions are given by

$$(\overline{\eta}_x u + \overline{\eta}_y v)_{j,1} = 0 \tag{3.17}$$

$$(\overline{\eta}_y u - \overline{\eta}_x v)_{j,1} - 2(\overline{\eta}_y u - \overline{\eta}_x v)_{j,2} + (\overline{\eta}_y u - \overline{\eta}_x v)_{j,3} = 0 \tag{3.18}$$

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

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

For *viscous* flows, the boundary conditions are

$$\rho_{j,1} - \rho_{j,2} = 0 \tag{3.21}$$

$$(\rho u)_{j,1} = 0 \tag{3.22}$$

$$(\rho v)_{j,1} = 0 \tag{3.23}$$

$$p_{j,1} - p_{j,2} = 0 \tag{3.24}$$

where Eq. (3.21) represents the adiabatic condition at the wall.

Far-field boundary

At $k = k_{max}$ the boundary conditions are common for viscous and inviscid flows. Following the logic shown in Table 2.1, and using a zeroth-order extrapolation in space, the first two equations are

$$\left(\overline{\eta}_x u + \overline{\eta}_y v - \frac{2}{\gamma - 1}\sqrt{\frac{\gamma p}{\rho}}\right)_{j,k_{max}} - \left(V_n - \frac{2a}{\gamma - 1}\right)_{\infty} = 0 \tag{3.25}$$

$$\left(\overline{\eta}_x u + \overline{\eta}_y v + \frac{2}{\gamma - 1}\sqrt{\frac{\gamma p}{\rho}}\right)_{j,k_{max}} - \left(\overline{\eta}_x u + \overline{\eta}_y v + \frac{2}{\gamma - 1}\sqrt{\frac{\gamma p}{\rho}}\right)_{j,k_{max}-1} = 0 \tag{3.26}$$

For an inflow condition, the other two equations are

$$\left(\frac{\rho^{\gamma}}{p}\right)_{j,k_{max}} - S_{\infty} = 0 \tag{3.27}$$

$$\left(\overline{\eta}_y u - \overline{\eta}_x v\right)_{j,k_{max}} - (V_t)_{\infty} = 0 \tag{3.28}$$

For an outflow condition, Eqs. (3.27) and (3.28) are replaced by

$$\left(\frac{\rho^{\gamma}}{p}\right)_{j,k_{max}} - \left(\frac{\rho^{\gamma}}{p}\right)_{j,k_{max-1}} = 0 \tag{3.29}$$

$$\left(\overline{\eta}_y u - \overline{\eta}_x v\right)_{j,k_{max}} - \left(\overline{\eta}_y u - \overline{\eta}_x v\right)_{j,k_{max-1}} = 0 \tag{3.30}$$

For an *inviscid* flow, boundary conditions at $j = 1$ and at $j = j_{max}$ are very similar to the one at $k = k_{max}$; we just need to keep in mind the logic shown in Table 2.1.

The boundary conditions at $j = 1$ and $j = j_{max}$ for a *viscous* flow consist of a zeroth-order extrapolation of $\rho, \rho u, \rho v$ and p. At $j = 1$, they are given by

$$\rho_{1,k} - \rho_{2,k} = 0 \tag{3.31}$$

$$(\rho u)_{1,k} - (\rho u)_{2,k} = 0 \tag{3.32}$$

$$(\rho v)_{1,k} - (\rho v)_{2,k} = 0 \tag{3.33}$$

$$p_{1,k} - p_{2,k} = 0 \tag{3.34}$$

To take into account the circulation correction, the values of u, v and a at ∞ as well as the corresponding V_t and V_n should be replaced by the values given by Eqs. (2.32) and (2.33).

3.2 Linearization: Newton's method

The spatial discretization of the nonlinear partial differential equations and the boundary conditions leads to a nonlinear system of algebraic equations of the form

$$\mathcal{F}(\hat{Q}) = 0 \tag{3.35}$$

In the introduction, we discussed a number of schemes that could be used to solve this set of equations. We also justified the use of quasi-Newton methods based on their great potential to become efficient solvers. These methods are based on Newton's linearization in which

$$\mathcal{F}^{(n+1)} \approx \mathcal{F}^{(n)} + \mathcal{A}^{(n)} \Delta\hat{Q}^{(n)} = 0 \tag{3.36}$$

where

$$\mathcal{F}^{(n)} \equiv \mathcal{F}(\hat{Q}^{(n)}) \qquad \text{and} \qquad \Delta\hat{Q}^{(n)} \equiv \hat{Q}^{(n+1)} - \hat{Q}^{(n)}$$

$\mathcal{A}^{(n)}$ is the Jacobian matrix of $\mathcal{F}$, which is given by

$$\mathcal{A} = \frac{\partial \mathcal{F}(\hat{Q})}{\partial \hat{Q}} \tag{3.37}$$

evaluated at $\hat{Q}^n$. The nonlinear system of equations has been replaced by a series of systems of linear equations of the form

$$\mathcal{A}^{(n)}\, \Delta\hat{Q}^{(n)} = -\mathcal{F}^{(n)} \tag{3.38}$$

If $\mathcal{A}$ corresponds to an exact linearization of $\mathcal{F}$, Eq. (3.36) represents a true Newton linearization. We will refer to this Jacobian as $\mathcal{A}_2$. If the functional Jacobian is simplified, thus producing an approximate-linearization, we get an approximate-Newton method. The motivation for using an approximate-Jacobian is the possibility of using far less storage and/or building a Jacobian that is better conditioned and that is more diagonally dominant, which will benefit the iterative solver, as we will see.

It should be noted that, for the present algorithm, an exact analytical linearization of the equations cannot be obtained, due to the impossibility of linearizing terms such as the spectral-radius in Eq. (3.9), the switch between the second-difference and the fourth-difference artificial dissipation that appears in Eq. (3.13) and the turbulence model. As we will see, the linearization of the far-field circulation correction poses some problems as well. It is possible to freeze all these terms, but, in that case, second-order convergence can no longer be reached. Nevertheless we will still refer to this approximation as the second-order Jacobian $\mathcal{A}_2$. Alternatively, the second-order Jacobian can be computed numerically, overcoming the difficulties mentioned earlier.

3.2.1 Linearization of the interior scheme

The functional $\mathcal{F}$ at the interior nodes consists of

$$\begin{aligned}\mathcal{F}_{j,k} \;=\; & \frac{1}{2}\left[\hat{E}_{j+1,k} - \hat{E}_{j-1,k} + \hat{F}_{j,k+1} - \hat{F}_{j,k-1}\right] - \\ & \mathcal{R}e^{-1}\left[\hat{S}_{j,k+\frac{1}{2}} - \hat{S}_{j,k-\frac{1}{2}}\right] + D^{(2\xi)}_{j,k} + D^{(4\xi)}_{j,k} + D^{(2\eta)}_{j,k} + D^{(4\eta)}_{j,k}\end{aligned} \tag{3.39}$$

Second-order Jacobian

The linearization of Eq. (3.39) leads to nine 4×4 blocks in the rows corresponding to the interior nodes. The location of the blocks within the matrix depends on the chosen ordering for the nodes. Ordering schemes will be discussed in Section 3.6. The nine blocks are given by

$$
\begin{aligned}
B^{-2,\xi}_{j,k} &= \frac{\partial D^{(4\xi)}_{j,k}}{\partial \hat{Q}_{j-2,k}} \\
B^{-1,\xi}_{j,k} &= \frac{\partial D^{(4\xi)}_{j,k}}{\partial \hat{Q}_{j-1,k}} + \frac{\partial D^{(2\xi)}_{j,k}}{\partial \hat{Q}_{j-1,k}} - \frac{1}{2}\frac{\partial \hat{E}_{j-1,k}}{\partial \hat{Q}_{j-1,k}} \\
B^{-2,\eta}_{j,k} &= \frac{\partial D^{(4\eta)}_{j,k}}{\partial \hat{Q}_{j,k-2}} \\
B^{-1,\eta}_{j,k} &= \frac{\partial D^{(4\eta)}_{j,k}}{\partial \hat{Q}_{j,k-1}} + \frac{\partial D^{(2\eta)}_{j,k}}{\partial \hat{Q}_{j,k-1}} - \frac{1}{2}\frac{\partial \hat{F}_{j,k-1}}{\partial \hat{Q}_{j,k-1}} - M_\infty \mathcal{R}e^{-1}\frac{\partial \hat{S}_{j,k-\frac{1}{2}}}{\partial \hat{Q}_{j,k-1}} \\
B^{0}_{j,k} &= \frac{\partial D^{(4\xi)}_{j,k}}{\partial \hat{Q}_{j,k}} + \frac{\partial D^{(2\xi)}_{j,k}}{\partial \hat{Q}_{j,k}} + \frac{\partial D^{(4\eta)}_{j,k}}{\partial \hat{Q}_{j,k}} + \frac{\partial D^{(2\eta)}_{j,k}}{\partial \hat{Q}_{j,k}} + \\
&\qquad M_\infty \mathcal{R}e^{-1}\left[\frac{\partial \hat{S}_{j,k-\frac{1}{2}}}{\partial \hat{Q}_{j,k}} + \frac{\partial \hat{S}_{j,k+\frac{1}{2}}}{\partial \hat{Q}_{j,k}}\right] \\
B^{+1,\eta}_{j,k} &= \frac{\partial D^{(4\eta)}_{j,k}}{\partial \hat{Q}_{j,k+1}} + \frac{\partial D^{(2\eta)}_{j,k}}{\partial \hat{Q}_{j,k+1}} - \frac{1}{2}\frac{\partial \hat{F}_{j,k+1}}{\partial \hat{Q}_{j,k+1}} - M_\infty \mathcal{R}e^{-1}\frac{\partial \hat{S}_{j,k+\frac{1}{2}}}{\partial \hat{Q}_{j,k+1}} \\
B^{+2,\eta}_{j,k} &= \frac{\partial D^{(4\eta)}_{j,k}}{\partial \hat{Q}_{j,k+2}} \\
B^{+1,\xi}_{j,k} &= \frac{\partial D^{(4\xi)}_{j,k}}{\partial \hat{Q}_{j+1,k}} + \frac{\partial D^{(2\xi)}_{j,k}}{\partial \hat{Q}_{j+1,k}} - \frac{1}{2}\frac{\partial \hat{E}_{j+1,k}}{\partial \hat{Q}_{j+1,k}} \\
B^{+2,\xi}_{j,k} &= \frac{\partial D^{(4\xi)}_{j,k}}{\partial \hat{Q}_{j+2,k}}
\end{aligned}
\tag{3.40}
$$

The coefficients of the artificial dissipation, as well as μ and μ_t are treated as constants in the linearization. The Jacobians of the artificial dissipation are 4×4 diagonal matrices, easy to compute from Eqs. (3.14) to (3.17). The flux Jacobians corresponding to $\hat{E}, \hat{F}$ and $\hat{S}$ are shown in Appendix A.

First-order Jacobian

A simple approximation to $\mathcal{A}_2$ with far fewer nonzero entries, which we designate $\mathcal{A}_1$, is obtained by using only second-difference dissipation in forming the matrix. The new matrix is more diagonally dominant because we are adding a large amount of first-order dissipation with stencil (1,-2,1), whereas A_2 has second-order dissipation with stencil (-1,4,-6,4,-1). The coefficient of the second-difference dissipation is given by

$$\varepsilon_2^l = \varepsilon_2^r + \sigma \varepsilon_4^r \tag{3.41}$$

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. Since blocks $B_{j,k}^{\pm 2}$ do not exist anymore, the resulting matrix has five blocks per node instead of nine.

A second approximation is introduced to make the matrix more diagonally dominant by adding a term to the diagonal. If we had used the implicit Euler time-marching method with a suitable linearization applied to the unsteady Navier-Stokes equations instead of Newton's method applied to the steady equations, we would have obtained the same functional Jacobian except for an extra term $1/\Delta t$ in the diagonal. In other words, the implicit Euler time-marching method reduces to Newton's method when using an infinite time step [94]. The smaller the time step, the more diagonally dominant the Jacobian, which improves the convergence of the inner iterations. Therefore, we add the equivalent of a local time step term to the diagonal of $\mathcal{A}_1$. Saleem et al. [95] have shown that, in the context of ARC2D, the time step based on the metric Jacobian is the optimal strategy and it is the one that we adopt, with Δt given by,

$$\Delta t_{j,k} = \frac{\Delta t_0}{1 + \sqrt{J_{j,k}}} \tag{3.42}$$

where Δt_0 is a constant that will be chosen experimentally to maximize convergence.

3.2.2 Linearization of the boundary conditions

To achieve Newton convergence, the boundary conditions have to be treated implicitly. Implementation of the far field circulation correction in an implicit manner is difficult since the vortex strength is proportional to the lift coefficient. This leads to coupling between every point in the far field and the ones on the airfoil surface, which adds more non zeros outside of the banded structure. Therefore, we introduce an approximation similar to the one introduced for the turbulence model and the artificial dissipation: when computing the Jacobian analytically, we treat the vortex strength as a constant. This difficulty can also be overcome by numerically computing the Jacobian.

The boundary conditions were introduced in Section 2.5 and the discretized form has been presented in Section 3.1.3. The discretized equations can be written as

$$\mathcal{B}(R) = 0 \tag{3.43}$$

where R is the set of variables chosen to write the equations. We found it convenient to use

$$R = \begin{bmatrix} \rho \\ u \\ v \\ p \end{bmatrix} \tag{3.44}$$

Applying Newton's linearization to Eq. (3.43), we obtain

$$P\Delta R = -\mathcal{B}^{(n)} \tag{3.45}$$

where

$$P = \left(\frac{\partial \mathcal{B}}{\partial R}\right)^{(n)} \tag{3.46}$$

Since the unknowns in the global system are $\Delta\hat{Q}$, we have to change variables in Eq. (3.45). The Jacobian matrix $M = \partial Q/\partial R$ of the transformation between the variables ΔQ and ΔR is defined by

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ u & \rho & 0 & 0 \\ v & 0 & \rho & 0 \\ \dfrac{u^2+v^2}{2} & \rho u & \rho v & \dfrac{1}{\gamma-1} \end{bmatrix} \tag{3.47}$$

Therefore, Eq. (3.45) becomes

$$PM^{-1}J\Delta\hat{Q} = -\mathcal{B}^{(n)} \tag{3.48}$$

where

$$M^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\dfrac{u}{\rho} & \dfrac{1}{\rho} & 0 & 0 \\ -\dfrac{v}{\rho} & 0 & \dfrac{1}{\rho} & 0 \\ (\gamma-1)\dfrac{u^2+v^2}{2} & -(\gamma-1)u & -(\gamma-1)v & \gamma-1 \end{bmatrix} \tag{3.49}$$

Body surface

For an *inviscid* flow, boundary conditions at the body surface are given by Eqs. (3.17) to (3.20). Since we are using a first-order extrapolation at the body surface, Eq. (3.48) takes the form

$$P_{j,1}M_{j,1}^{-1}J_{j,1}\Delta\hat{Q}_{j,1} - 2P_{j,2}M_{j,2}^{-1}J_{j,2}\Delta\hat{Q}_{j,2} + P_{j,3}M_{j,3}^{-1}J_{j,3}\Delta\hat{Q}_{j,3} = -\mathcal{B}^{(n)} \tag{3.50}$$

where

$$P_{j,1} = \begin{bmatrix} 0 & \overline{\eta}_x & \overline{\eta}_y & 0 \\ 0 & \overline{\eta}_y & -\overline{\eta}_x & 0 \\ 0 & 0 & 0 & 1 \\ \frac{1}{2}(u^2+v^2) - H_\infty & \rho u & \rho v & \gamma(\gamma-1)^{-1} \end{bmatrix}_{j,1} \tag{3.51}$$

$$P_{j,2} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \overline{\eta}_y & -\overline{\eta}_x & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{j,2} \tag{3.52}$$

$$P_{j,3} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \overline{\eta}_y & -\overline{\eta}_x & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{j,3} \tag{3.53}$$

For a *viscous* flow, Eq. (3.48) is now given by

$$P_{j,1}M_{j,1}^{-1}J_{j,1}\Delta\hat{Q}_{j,1} - P_{j,2}M_{j,2}^{-1}J_{j,2}\Delta\hat{Q}_{j,2} = -\mathcal{B}^{(n)} \tag{3.54}$$

Since Eqs. (3.21) to (3.23) are expressed in terms of the conservative variables, the product $P_{j,1}M_{j,1}^{-1}$ is the identity matrix for the first three rows. Thus,

$$P_{j,1}M_{j,1}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ m_{4,1} & m_{4,2} & m_{4,3} & m_{4,4} \end{bmatrix}_{j,1} \tag{3.55}$$

$$P_{j,2}M_{j,2}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ m_{4,1} & m_{4,2} & m_{4,3} & m_{4,4} \end{bmatrix}_{j,2} \tag{3.56}$$

where $[m_{4,1}, m_{4,2}, m_{4,3},\ m_{4,4}]$ is the last row of M^{-1} given in Eq. (3.49).

Far-field boundary

We show the linearized equations at k_{max}, which are the same for viscous and inviscid flows, but they apply also at $j = 1$ and $j = j_{max}$ for inviscid flows. After the linearization, we obtain an equation similar to (3.54) where $P_{j,k_{max}}$ is the same for inflow and outflow conditions; it is given by:

$$P_{j,k_{max}} = \begin{bmatrix} \dfrac{a}{(\gamma-1)\rho} & \overline{\eta}_x & \overline{\eta}_y & \dfrac{-\gamma}{(\gamma-1)a\rho} \\ \dfrac{-a}{(\gamma-1)\rho} & \overline{\eta}_x & \overline{\eta}_y & \dfrac{\gamma}{(\gamma-1)a\rho} \\ \dfrac{\gamma\rho^{\gamma-1}}{p} & 0 & 0 & \dfrac{-\rho^{\gamma}}{p^2} \\ 0 & \overline{\eta}_y & -\overline{\eta}_x & 0 \end{bmatrix}_{j,k_{max}} \tag{3.57}$$

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

$$P_{j,k_{max}-1} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \dfrac{-a}{(\gamma-1)\rho} & \overline{\eta}_x & \overline{\eta}_y & \dfrac{\gamma}{(\gamma-1)a\rho} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{j,k_{max}-1} \tag{3.58}$$

and for an outflow condition, the last two rows are given by the last two rows of (3.57), but evaluated at $(j, k_{max} - 1)$.

The linearization of the boundary condition equations at $j = 1$ and $j = j_{max}$ for a *viscous* flow gives an equation similar to (3.54) where the products $P_{1,k}M_{1,k}^{-1}$ and $P_{2,k}M_{2,k}^{-1}$ are equal to the matrix shown in (3.55).

Scaling the equations

The coefficients of $\Delta\hat{Q}$ in Eq. 3.48 are not of the same order of magnitude as the coefficients at the interior nodes. This causes some of the eigenvalues of the Jacobian to become extremely large and slows down the convergence of GMRES, which may even stagnate. An appropriate scaling of the equations at the boundaries overcomes the problem. After pivoting within the diagonal block to place the biggest element of each column in the diagonal, we normalize each equation by the diagonal element.

3.3 Start up

For some flow cases, especially those with shocks, the early Newton iterations can diverge. Different relaxation techniques have been suggested to overcome this difficulty. One way to relax the solution is to damp the Newton updates to prevent the calculation of non-physical variable values [65]. For example,

$$\hat{Q}^{(n+1)} = \hat{Q}^{(n)} + (1-\theta)\Delta\hat{Q}^{(n)} \ , \qquad \theta \in (0,1] \tag{3.59}$$

where θ takes initially small values and increases gradually towards unity.

An alternative to this technique is to use the unsteady form of the Navier-Stokes equations which can be written as

$$\frac{d\hat{Q}}{dt} = \mathcal{F}(\hat{Q}) \tag{3.60}$$

and to apply implicit Euler time-differencing. As mentioned in Section 3.2.1, this is equivalent to adding a term to the diagonal of the Jacobian $\mathcal{A}$. A finite time step can be used initially and, as $\Delta t \to \infty$, the Newton method is obtained. This strategy is used by many authors, e.g., Mulder and Van Leer [96], Orkwis [62] and Barth [76] to name a few. In our experience [97], it is more efficient to use an approximate Jacobian for the first two orders of magnitude reduction in outer residual.

However, when a finite time step must be used, a cheaper relaxation algorithm can be employed [77, 98], significantly reducing computing time. This is particularly true for transonic flows, where many outer iterations at low Δt can be needed before fast convergence can be achieved. This can be computationally expensive even when using an approximate-Newton method. In the present study, we replace the approximate-Newton method used in Ref. [97] by an approximately-factored algorithm of ARC2D in diagonal form with two levels of grid sequencing. It is also used for the first two orders of magnitude residual reduction, but limiting the number of iterations for cases where the approximately-factored algorithm shows slow convergence. Limiting the number of iterations to 150 on the coarse grid, and five on the fine grid seems to give good performance. This strategy reduces the CPU time of the start up by a factor of two to three compared to the approximate-Newton strategy.

3.4 Solvers for the linear problem

Direct solvers are more robust than iterative solvers and require fewer parameters. The drawback is the higher computational complexity and the need of significantly larger storage capacity. For these reasons, we do not consider them in the present work. For further discussion of direct solvers in CFD applications, we refer the reader to [28].

The alternative is to use iterative solvers. There are several effective iterative solvers for non-symmetric linear systems available, as reviewed by Dutto [42], Barrett et al. [43] and Page [99], among others. It is very difficult to establish general rules about which one is *the best method*. This depends on the particular problem one is attempting to solve. Nevertheless, for the type of systems arising in CFD applications, preconditioned Krylov methods have shown better convergence properties than classical stationary methods such as Jacobi, Gauss-Seidel or SOR. Among Krylov solvers, GMRES, developed by Saad and Schultz [36], is the most popular one, being, on average, faster than other Krylov solvers. We have not done a systematic study of different Krylov solvers, but in a few tests comparing GMRES with bi-CGSTAB and CGS, we found GMRES faster for our applications.

3.4.1 GMRES

For any linear system of equations of the form,

$$\mathcal{A}x = b \tag{3.61}$$

GMRES has the property of finding, at every step, 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

$$K_m = \text{span}\{v_1, \mathcal{A}v_1, \mathcal{A}^2 v_1, ..., \mathcal{A}^{m-1}v_1\} \tag{3.62}$$

The vector v_1 is defined as

$$v_1 = \frac{r_0}{||r_0||_2} = \frac{b - \mathcal{A}x_0}{||b - \mathcal{A}x_0||_2} \tag{3.63}$$

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:

$$\left.\begin{array}{ll} For & j = 1, 2, ..., m \quad do: \\ & h_{i,j} = (\mathcal{A}v_j, v_i), i = 1, 2, ..., j \\ & \hat{v}_{j+1} = \mathcal{A}v_j - \sum_{i=1}^{j} h_{i,j} v_j \\ & h_{j+1,j} = ||\hat{v}_{j+1}|| \\ & v_{j+1} = \hat{v}_{j+1}/h_{j+1,j} \end{array}\right\} \text{Arnoldi's method} \tag{3.64}$$

From the above process, it is easy to show that

$$\mathcal{A}V_m = V_{m+1}\bar{H}_m \tag{3.65}$$

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

$$x = x_0 + V_m y \tag{3.66}$$

where y is a vector of dimension m. Making use of Eqs. (3.65) and (3.66), the L_2 norm of the residual can be written as a function of y:

$$\begin{aligned} \|r(y)\|_2 &= \|b - \mathcal{A}x\|_2 \\ &= \|b - \mathcal{A}(x_0 + V_m y)\|_2 \\ &= \|r_0 - \mathcal{A}V_m y\|_2 \\ &= \|\beta v_1 - V_{m+1}\bar{H}_m y\|_2 \\ &= \|V_{m+1}(\beta e_1 - \bar{H}_m y)\|_2 \end{aligned} \tag{3.67}$$

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,

$$\|r(y)\|_2 = \|\beta e_1 - \bar{H}_m y\|_2 \tag{3.68}$$

The third step, consisting of finding the x that minimizes the residual, is reduced to finding the y_m such that the function $\|r(y)\|_2$ is minimized. This is quite inexpensive, since it is a $(m+1) \times m$ least-squares problem with m very small compared to N. It should be noted that, given the structure of $\bar{H}_{m+1}$, the least-squares problem can be solved very inexpensively by applying simple rotations to the Hessenberg matrix to transform it into an upper triangular matrix. Once we have y_m, we form x_m using Eq. (3.66).

Another important property of GMRES is that the norm of the residual is directly available, if we apply the rotations every time that we compute a new column of the Heissenberg matrix, which means, after adding a new vector v_j to the base. Therefore, there is no need to form x_j, the corresponding residual, $r_j = b - \mathcal{A}x_j$, and evaluate its L_2 norm in order to check convergence. If the rotations transform the vector βe_1 of Eq. (3.68) into $(\gamma_1, \ldots, \gamma_{j+1})^T$, it is easy to prove that

$$\|r_j\|_2 = \quad | \gamma_{j+1} | \tag{3.69}$$

For the proof, we refer the reader to Ref. [100], pages 29 and 30.

GMRES is guaranteed to converge in at most $k = N$ steps. This is impractical because N is large and it is not possible to find a short vector recursion when building the orthonormal basis of K_m, which means that work and storage requirements increase at every new search direction v_j. Storage increases linearly with the number of search directions and CPU time increases quadratically. To overcome this problem, we can use the algorithm iteratively: we can restart it after $m \ll N$, and use x_m as the initial guess when we restart. This is the restarted version of GMRES denoted by GMRES(m). If $\mathcal{A}$ is nearly positive definite, m does not have to be too large for convergence of GMRES(m). But for indefinite problems, such as the ones solved here, GMRES(m) may stagnate (i.e., not converge). This can be overcome by using a preconditioner, which will be discussed in the next section. The pseudocode for the restarted GMRES(m) is described in Figure 3.1.

In our applications, $\mathcal{A}$ is the $(N \times N)$ matrix in Eq. (3.38), with $N = 4 \times j_{max} \times k_{max}$, $b = -\mathcal{F}$ and $x = \Delta\hat{Q}$. We choose $x_0 = \Delta\hat{Q}_0 = 0$ in all cases. After testing different values of m, we found that, for our applications, limiting its value to 20 does not significantly degrade the convergence rate.

3.4.2 Matrix-free GMRES

Since GMRES requires only matrix-vector products, the algorithm can be implemented without forming the Jacobian matrix explicitly: second-order centred-difference as well as first-order forward-difference of the fluxes can be employed to approximate the matrix-vector products. It was already shown in [97] that such a matrix-free approach can be advantageous, from the point of view of both performance and storage. Since we do not need to linearize the Jacobian analytically and we use only

1. *Start*: Choose x_0 and compute $r_0 = b - \mathcal{A}x_0$ and $v_1 = r_0/\|r_0\|_2$

2. *Iterate*: For $j = 1, \ldots, m$ do

   $w_j = \mathcal{A}v_j$
   $h_{i,j} = (w_j, v_i), i = 1, 2, \ldots, j$
   $\hat{v}_{j+1} = w_j - \sum_{i=1}^{j} h_{i,j}v_j$
   $h_{j+1,j} = \|\hat{v}_{j+1}\|$
   $v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$
   Perform rotation to $\bar{H}_{j+1,j}$ and to rhs.
   if $\|r_j\|_2$ small enough, then stop

3. *Form the approximate solution*:

   Solve for y_m
   Form $x_m = x_0 + V_m y_m$

4. *Restart*:

   Compute $r_m = b - \mathcal{A}x_m$; if convergence is satisfied then stop
   else compute $x_0 \leftarrow x_m, v_1 \leftarrow r_m/\|r_m\|_2$ and go to 2

Figure 3.1: Algorithm for the restarted GMRES(m) iterative method.

evaluations of the fluxes, the switch between second and fourth-difference artificial dissipation, the turbulence model and the use of a far-field circulation correction can easily be included in the Jacobian.

We use the forward-difference, which requires only one right-hand side evaluation per iteration instead of two when using centred-difference. Therefore,

$$\mathcal{A}v \simeq \frac{\mathcal{F}(\hat{Q}+\varepsilon v)-\mathcal{F}(\hat{Q})}{\varepsilon} \tag{3.70}$$

where ε is a small scalar used to perturb the state quantities $\hat{Q}$ in the direction of v. The performance of this technique is very sensitive to ε, especially when using forward differencing [95]. A large value of ε can result in an unstable process due to an inaccurate approximation, while a very small value can lead to difficulties with round-off error. An effective strategy proposed by Nielsen et al. [77] involves choosing ε such that

$$\varepsilon\,\bar{v} \;\simeq\; \sqrt{\varepsilon_m} \tag{3.71}$$

where $\bar{v}$ is the root mean square of v and ε_m is the value of "machine zero" for the hardware being used. We use the L_2 norm of v, which gives identical results.

In order to make a distinction between the Newton iterations on the non-linear problem and the GMRES iterations on the linear one, the terms "outer iterations" and "inner iterations", respectively, will be used.

3.5 Preconditioners

A weakness of iterative solvers, relative to direct solvers, is their lack of robustness. Preconditioning is an effective technique to improve both efficiency and robustness. It consists of transforming the linear system into one that is easier to solve by an iterative solver. The convergence rate of a solver like GMRES is determined by the spectrum of the matrix that we are inverting. The ideal spectrum is to have all the eigenvalues equal to one, which is the identity matrix spectrum. This suggests that a good preconditioner should transform the original matrix into another one that is as close as possible to the identity matrix. To illustrate this point, we show in Figure 3.2 how the original eigenvalues of a typical matrix $\mathcal{A}$ that arises in our applications are clustered around 1 when a preconditioner is applied to it.

In the most general case, known as *right and left preconditioning* or *preconditioning by both sides*, the original system $\mathcal{A}x = b$ is transformed into

$$\mathcal{M}_l^{-1}\mathcal{A}\mathcal{M}_r^{-1}\mathcal{M}_r x = \mathcal{M}_l^{-1}b \tag{3.72}$$

where $\mathcal{A}_p = \mathcal{M}_l^{-1}\mathcal{A}\mathcal{M}_r^{-1}$ should approximate $\mathcal{I}$. The condition number of $\mathcal{A}_p$ is smaller than that of $\mathcal{A}$ and the iterative solver will produce a better convergence rate.

Many authors insist on the fact that one of the conditions that a *good* preconditioner given by

$$\mathcal{M} = \mathcal{A} - \mathcal{E} \tag{3.73}$$

with $\mathcal{E}$ being an error matrix, should be that $\mathcal{M}$ is as close to $\mathcal{A}$ as possible. Some studies have shown that the number of iterations is related to the norm of the error matrix $\mathcal{E}$ [101, 53]. But, as we will see, this is not always the case. If we have a look at Eq. (3.72), it is easy to see that, if we use a one side (right or left) preconditioning, a good preconditioner is such that the matrix $\mathcal{M}_l^{-1}$ or $\mathcal{M}_r^{-1}$ approximates $\mathcal{A}^{-1}$. The fact that $\mathcal{M}$ is a good approximation of $\mathcal{A}$ *does not* guarantee that $\mathcal{M}^{-1}$ is a good approximation of $\mathcal{A}^{-1}$. This is particularly true for incomplete factorizations of non-M matrices [102].

One of the advantages of using right preconditioning is that the residual of the preconditioned system is the same as the residual of the unpreconditioned system. This is important considering that stopping criteria should be based on the residual of the unpreconditioned system. Thus we use right preconditioning in our application. When dropping left preconditioning from Eq. (3.72), it becomes

$$\mathcal{A}\mathcal{M}_r^{-1}\mathcal{M}_r x = b \tag{3.74}$$

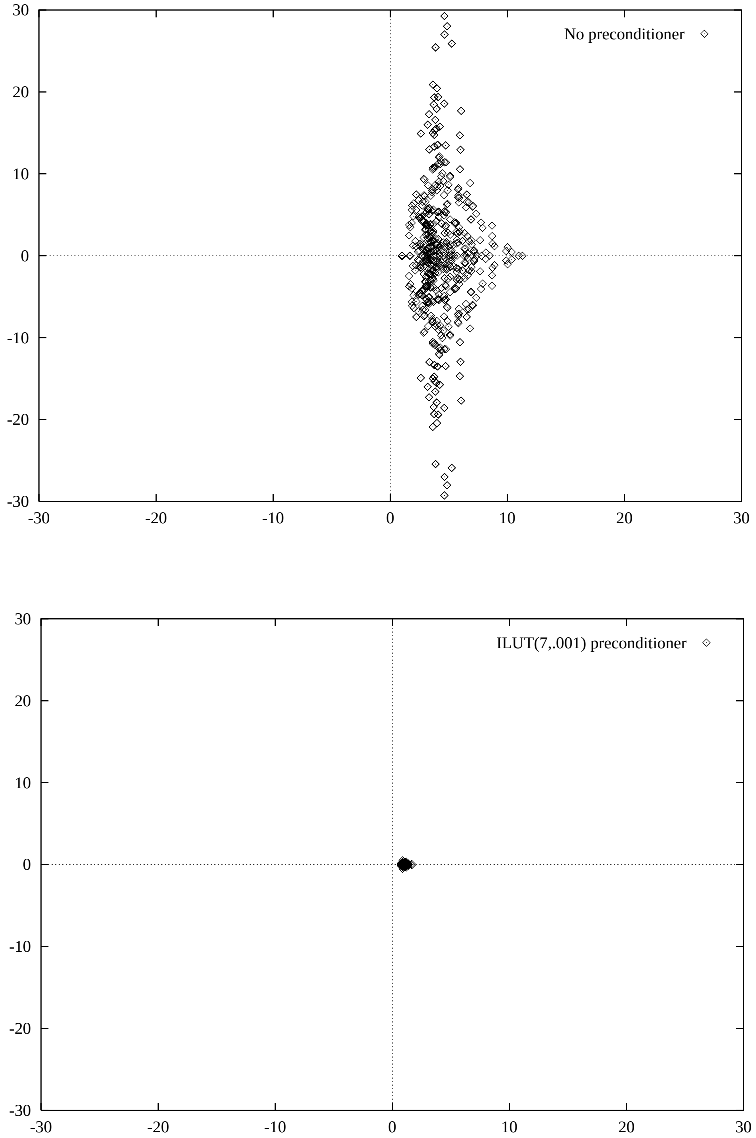


Figure 3.2: Eigenvalues of a non-preconditioned matrix $\mathcal{A}$ and of the matrix preconditioned from the left with an ILU-type preconditioner $\mathcal{M}_l^{-1}\mathcal{A}$.

1. *Start*: Choose x_0 and compute $r_0 = b - \mathcal{A}x_0$ and $v_1 = r_0/\|r_0\|_2$

2. *Iterate*: For $j = 1, \ldots, m$ do

   Preconditioning: $z_j = \mathcal{M}^{-1}v_j$
   $w_j = \mathcal{A}z_j$
   $h_{i,j} = (w_j, v_i), i = 1, 2, \ldots, j$
   $\hat{v}_{j+1} = w_j - \sum_{i=1}^{j} h_{i,j}v_j$
   $h_{j+1,j} = \|\hat{v}_{j+1}\|$
   $v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$
   Perform rotation to $\bar{H}_{j+1,j}$ and to rhs.
   if $\|r_j\|_2$ small enough, then stop

3. *Form the approximate solution*:

   Solve for y_m
   Solve $u_m = \mathcal{M}^{-1}(V_m y_m)$
   Form $x_m = x_0 + u_m$

4. *Restart*:

   Compute $r_m = b - \mathcal{A}x_m$; if satisfied then stop
   else compute $x_0 \leftarrow x_m, v_1 \leftarrow r_m/\|r_m\|_2$ and go to 2

Figure 3.3: The preconditioned GMRES(m) algorithm.

The preconditioned GMRES(m) algorithm is shown in Figure 3.3; it can be observed that we do not need to form the matrix $\mathcal{A}\mathcal{M}^{-1}$, which would be very costly: we multiply the vector v_j by $\mathcal{M}^{-1}$ and then by $\mathcal{A}$.
We should note that applying the preconditioner

$$z_j = \mathcal{M}^{-1} v_j \tag{3.75}$$

is equivalent to solving

$$\mathcal{M} z_j = v_j \tag{3.76}$$

Therefore, we do not need to compute and store $\mathcal{M}^{-1}$, which may be a dense matrix even though $\mathcal{M}$ may be sparse; we just need to solve Eq. (3.76) by either

i) finding a matrix $\mathcal{M}$ such that it is easier to invert than $\mathcal{A}$ while $\mathcal{M}^{-1}$ remains a *good* approximation of $\mathcal{A}^{-1}$, or

ii) using an iterative method, with $\mathcal{M} = \mathcal{A}$ or some *good* approximation of it for which the iterative solver converges.

In the first case, since we are exactly inverting $\mathcal{M}$, the preconditioner is the same at each step and in computing the solution, we only need to apply $\mathcal{M}^{-1}$ to the linear combination $V_m y_m$. Approximate factorization (AF) and the incomplete upper-lower factorization (ILU) family are among the most efficient preconditioners of this type.

On the second type of preconditioners, using an iterative solver to inexactly invert $\mathcal{M}$ means that we have a different $\mathcal{M}_j^{-1}$ at each step. Therefore, we cannot use the expression

$$x_m = x_0 + \mathcal{M}^{-1} V_m y_m \tag{3.77}$$

It has to be replaced by

$$x_m = x_0 + Z_m y_m \tag{3.78}$$

where Z_m is the matrix containing the m vectors $z_j = M_j^{-1} v_j$. Therefore, not only the vectors v_j need to be stored, as in the standard GMRES implementation, but the preconditioned vectors z_j need to be stored as well; they are used to update the solution. This variant of GMRES, introduced by Saad [103], where the preconditioner may be different at each iteration is known as flexible GMRES (FGMRES).

Some of the most popular iterative solvers used as preconditioners are the point, line and block versions of Jacobi, Gauss-Seidel and SSOR (successive over-relaxation) iterative methods, but many other solvers can be used as preconditioners. Even GMRES could be used as preconditioner as in the nested GMRES method proposed by Van der Vorst and Vuik [104]. For further discussion about these techniques and preconditioning in general, we refer the reader to the papers of Axelsson [105] and Saad [106].

The costs associated with a preconditioner are

i) Forming the preconditioning matrix $\mathcal{M}$

ii) Solving the system given by Eq. (3.76)

iii) Additional storage to store $\mathcal{M}$

Therefore, in choosing a preconditioner, we should try to minimize these costs, while significantly reducing the number of iterations required by GMRES compared with the unpreconditioned system.

In this work, we use preconditioners of the ILU-type which have proven to be reliable in CFD applications. Two of them are described in the following sections.

3.5.1 Incomplete LU factorization preconditioners

Incomplete LU factorizations are often regarded as efficient preconditioners for Krylov solvers. In an incomplete factorization, we approximate the matrix $\mathcal{A}$ by a matrix $\mathcal{M}$ such that

$$\mathcal{A} = \mathcal{M} - \mathcal{E} = \mathcal{L}\ \mathcal{U} - \mathcal{E} \tag{3.79}$$

where $\mathcal{L}$ is a lower-triangular matrix and $\mathcal{U}$ is an upper-triangular matrix. The factors are computed using a Gaussian elimination process or any other alternative process applied to the matrix $\mathcal{A}$ or to some *reasonable* approximation of it. The factorization may be more or less accurate, depending on how many new non-zero entries we retain in the factorization compared to the original matrix. The cost of forming the preconditioner and storage goes up when we allow more fill-in in $\mathcal{L}$ and $\mathcal{U}$; on the other hand, increases in robustness and efficiency often justify more accurate factorizations, particularly when memory is not an issue and when the preconditioner is going to be used in solving several systems, since the cost of forming the preconditioner is going to be amortized.

ILU factorization preconditioners were originally developed for M-matrices. Even if they have been successfully used in much more general cases, we should be aware of the possible problems that we may face. For example, in the case of nonsymmetric matrices, such as the ones arising in our applications, the incomplete factors $\mathcal{L}$ and $\mathcal{U}$ may be more ill-conditioned than the original matrix and the long recurrences associated with backward and forward solves may be unstable [102, 107, 108]. Since diagonally dominant matrices tend to produce well conditioned factors [100], one possible way to improve the preconditioner $\mathcal{M} = \mathcal{L}\ \mathcal{U}$ is to compute it from a matrix $\tilde{\mathcal{A}}$ which is more diagonally dominant than $\mathcal{A}$ while remaining a reasonable approximation of it. Numerical experiments shown in Section 4.4.1 confirm this thesis.

ILU preconditioners have also been applied to indefinite matrices. However, they may present even more severe problems than the ones mentioned above:

1. Inaccuracy due to very small pivots

2. Unstable triangular solves, which may or may not be related to small pivots

For further discussion of these problems, we refer the reader to reference [102].

The difficulties that we face in devising an efficient preconditioner using an ILU factorization, make particularly important the choice of the matrix $\tilde{\mathcal{A}}$, as well as the strategy used during the factorization. There have been two distinct strategies to forming such incomplete factorizations: level of fill-in and threshold strategies.

3.5.2 Level of fill-in: ILU(p)

The first approach, named ILU(p), uses only the graph of the matrix to determine which entries to keep in the factorization. A level of fill-in is attributed to each element that appears in the factorization. During the Gaussian-elimination process, the element is dropped if its level exceeds a certain threshold p. The way this is done in practice consists of assigning a level of fill-in equal to 0 to the nonzero elements of the original matrix $\tilde{\mathcal{A}}$ used to build the preconditioner. When a new nonzero element

$$m_{i,j} = -m_{i,k} \times m_{k,j}$$

is created in the factorization, the level of fill-in assigned to it is defined as

$$\text{level}\left(m_{i,j}\right) = \text{level}\left(m_{i,k}\right) + \text{level}\left(m_{k,j}\right) + 1 \tag{3.80}$$

When $p = 0$, the nonzero pattern of the preconditioner $\mathcal{M}$ corresponds to that of the original matrix $\tilde{\mathcal{A}}$. For other values of p, it is difficult to predict the amount of fill-in that will be generated. For a diagonally dominant matrix, the higher the level of fill-in of an element, the smaller its magnitude [100], which suggests that this is an appropriate strategy for this kind of matrix. Unfortunately, this may not be the case for more general matrices. The algorithm corresponding to ILU(p) is shown in Figure 3.4.

3.5.3 Threshold strategies. ILUT(P,τ)

In other incomplete factorization techniques, the drop-off rule is based on the numerical value of the elements introduced in the factorization rather than on their fill levels. These are known as threshold strategies. Unfortunately, the amount of fill-in is also hard to predict for this approach, and this preconditioner is far more expensive to form than ILU(p).

ILUT(P,τ), developed by Saad [109], is a class of LU factorization that lies between the level of fill-in strategy and the threshold strategy. Two rules are used to

For all nonzero elements $\tilde{a}_{i,j}$ do

$u_{i,j} = \tilde{a}_{i,j}$, and $\text{lev}(u_{i,j}) = 0$

For $i = 2, \ldots, N$ do

For $k = 1, \ldots, i-1$ and if $u_{i,k} \neq 0$ do

Compute $l_{i,k} = u_{i,k}/u_{k,k}$ and $\text{lev}(l_{i,k}) = \text{lev}(u_{i,k})$

For $j = 1, \ldots, N$ do

$u_{i,j} = u_{i,j} - l_{i,k}u_{k,j}$

$\text{lev}(u_{i,j}) = \min\ \{\text{lev}(u_{i,j}),\ \text{lev}(l_{i,k}) + lev(u_{k,j}) + 1\ \}$

Replace any element in row i with $\text{lev}(u_{i,j}) > p$ by a zero

Figure 3.4: Algorithm of the incomplete factorization ILU(p)

determine which elements should be dropped in a given row. The first rule consists of dropping any element smaller than a relative tolerance determined by τ and a norm of the original matrix. The second rule is controlled by the parameter P: if P_l and P_u are the numbers of non-zeros on the lower and upper part of a given row of the original matrix $\tilde{\mathcal{A}}$, at most the largest $P+P_l$ and $P+P_u$ are kept in the lower and upper part of the preconditioning matrix. This rule allows us to control the maximum number of elements per row and thus the memory usage. Figure 3.5 shows the pseudocode corresponding to ILU(P,τ).

3.5.4 BFILU(p)

The matrices arising from the linearization of the Navier-Stokes equations present a block structure, with blocks of size 4×4. That is why block versions of ILU(p) are quite popular in this type of application. As a matter of fact, in cases where a low level of fill-in is used, i.e., $p = 0$ or $p = 1$, the iterative solver may fail with the scalar version and converge with the block version.

For $i = 1, \ldots, N$ do

Compute average norm of elements in row i: $\alpha_i = \dfrac{\|\tilde{a}_{i,*}\|_2}{nnz}$

For $k = 1, N$ do

If $k = i$ then $u_{i,i} = \tilde{a}_{i,i}$

else if $\dfrac{\mid \tilde{a}_{i,k} \mid}{\alpha_i} > \tau$ then

If $k < i$ then $l_{i,k} = \tilde{a}_{i,k}$

If $k > i$ then $u_{i,k} = \tilde{a}_{i,k}$

For $k = 1, \ldots, i-1$ and if $l_{i,k} \neq 0$ do

$l_{i,k} = l_{i,k}/u_{k,k}$

If $\dfrac{\mid l_{i,k} \mid \alpha_k}{\alpha_i} > \tau$ then

For $j = k+1, \ldots, N$ and if $u_{k,j} \neq 0$ do

If $u_{i,j} \neq 0$ then $u_{i,j} = u_{i,j} - l_{i,k}u_{k,j}$

else if $\frac{l_{i,k}u_{k,j}}{\alpha_i} > \tau$ then $u_{i,j} = -l_{i,k}u_{k,j}$

Keep the $P + P_l$ biggest elements in L

Keep the $P + P_u$ biggest elements in U

Compute new α_i using all nnz except for the diagonal

Figure 3.5: Algorithm of the incomplete factorization ILU(P, τ).

An alternative approach is to use the scalar version, but treat the zeros within the 4×4 blocks as if they were nonzeros, in other words, allowing fill-in in those positions. We call this strategy Block-Fill ILU(p) (BFILU(p)). Orkwis [62] reports than in his application, CGS (Conjugate Gradient Squared) did not converge with ILU(0) but it did converge with BFILU(0). Because of the storing format that we use namely Compressed Sparse Row, CSR [110], we find this approach more convenient than block ILU(p).

3.6 Ordering of unknowns

The ordering of the unknowns plays an important role in the convergence of the preconditioned iterative solver [50, 53]. It can greatly affect the quality of the incomplete factorization. We have considered a number of ordering algorithms. The most significant ones are described in the following paragraphs.

3.6.1 Natural ordering

The first one (referred to as NAT) is the classical jk ordering described in [16]: the loop in k is inside the loop in j. This ordering produces the smallest bandwidth ($4 \times k_{max}$ for a second order matrix) since for our applications $k_{max} \ll j_{max}$. Looping the indices in the inverse order (j-loop within the k-loop) leads to very slow convergence and higher CPU in the factorization due to a much bigger bandwidth ($4 \times j_{max}$). Therefore, the kj-loop is not considered here.

In applications like ours where C-grids are used, like the one shown in Figure 2.1, the nodes at the wakecut need some special attention. In the case of approximate-factored algorithms such as ARC2D, they require some shuffling in the data before solving in the normal direction to keep the banded structure of the matrix. A less computationally intense alternative is not to solve at those nodes, and to compute the average values of the primitive variables at both sides of the wakecut. When Newton convergence is sought, either approach has to be exactly linearized, which will add some nonzeros to the matrix outside of the diagonals. These nonzeros create a fair amount of nonzeros during the factorization which may not be retained in an incomplete factorization, reducing the quality of the preconditioner.

3.6.2 Orderings based on domain decomposition

To overcome the difficulty of the nonzeros that appear outside of the main diagonals, a second ordering, labeled DD1 (domain decomposition 1), was devised. It is shown schematically in Figure 3.6 that the computational domain is divided in two zones, one that contains the wakecut nodes and another one that contains the rest of the

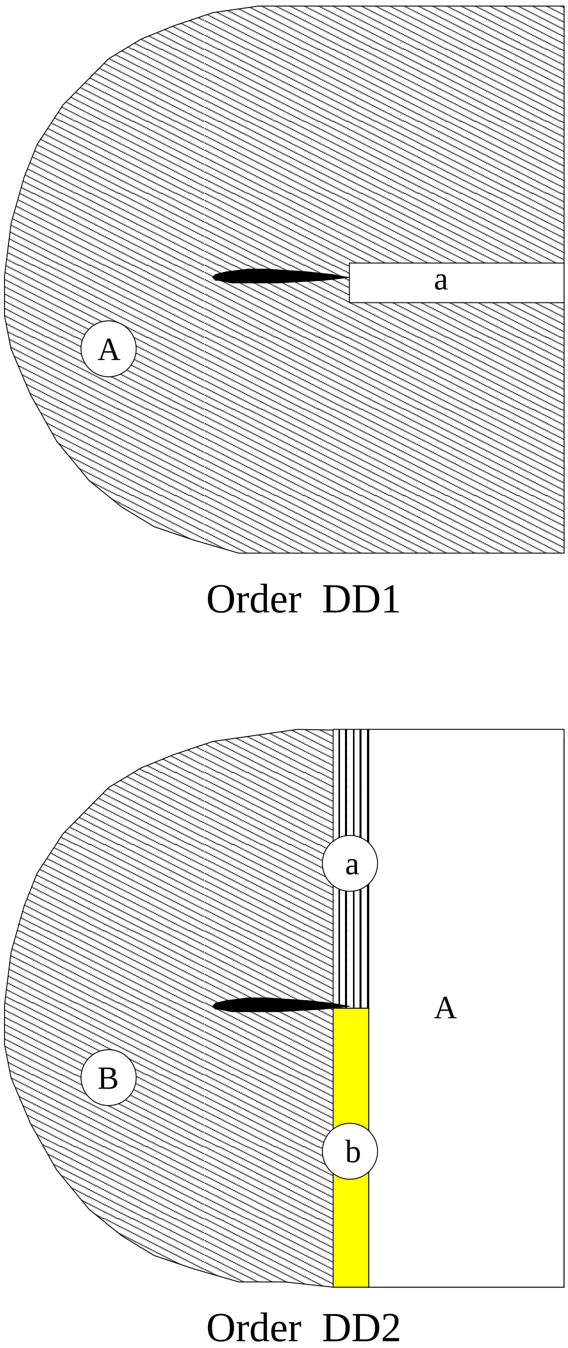


Figure 3.6: Schematical distribution of zones in orderings DD1 and DD2.

nodes. The numbering pattern is similar to the one used in NAT: we number the nodes of zone A following the jk-loop, followed by the nodes of zone a. This ordering leads to a decoupling of both zones across the wakecut. The matrix presents, for the most part, a similar block banded structure as before, with a bandwidth of $4 \times k_{max}$. Zone a covers $[j = 1 : j_{t1} - 1, k = 1 : 3] \cup [j = j_{t2+1} : j_{max}, k = 1 : 2]$.

Another ordering based on domain decomposition that avoids the problems introduced in the band-structure by the wakecut is also shown in Figure 3.6. This ordering, that we name DD2, is based in two zones A and B divided by the separators a and b. The separators are two cells wide. This ordering leads to a two totally independent computational zones A (of bandwidth $2 \times 4 \times k_{max}$) and B (of bandwidth $4 \times k_{max}$). The off-diagonal blocks, which are few in number, are produced by the separator zones; they may affect the factorization but to a lesser degree than in the case of the natural ordering.

3.6.3 Double bandwidth

Another alternative consists in numbering the nodes across the wakecut. We designate it DB (double bandwidth). The downside is that the bandwidth is twice as large, which can affect the quality of the factorization.

3.6.4 Reverse Cuthill-McKee

In order to obtain more consistent performance, we test two of the reordering algorithms typically used for unstructured grids. Note that the difference between the reordering algorithms and the previous ordering algorithms is that the reordering algorithms need to assume an initial ordering. The final ordering depends on the initial ordering that is provided to the algorithm. The first reordering algorithm that we will test is the Reverse Cuthill-McKee (RCM) strategy [111], a well-known bandwidth reduction algorithm.

3.6.5 Minimum neighbouring

The minimum neighbouring algorithm [112] (MN) is a modification of the minimum degree reordering of George and Liu [113]. The minimum degree algorithm was designed to minimize the fill-in produced in a factorization. However, this does not guarantee that, in an incomplete factorization, we are not throwing away important terms. The modifications introduced with the minimum neighbouring algorithm aim at keeping in the factors $\mathcal{L}$ and $\mathcal{U}$ those nonzeros that account for more operations during the factorization. In other words, the algorithm tries to minimize the amount of information that is thrown away during the incomplete factorization. This is equivalent to saying that it is trying to decrease the norm of the matrix $\mathcal{E}$ that appears in Eq. (3.73). But, as was pointed out earlier, for matrices that are not diagonal dominant, this does not ensure that the norm of $\mathcal{E}\mathcal{M}^{-1}$ is going to be reduced too.

In Figures 3.7 to 3.9, we show the structure of the matrices that arise from using the natural ordering as well as the domain decomposition orderings DD1 and DD2 for a 55×7 mesh with implicit boundary conditions and implicit wake cut. Figure 3.10 shows the matrix resulting after using the double bandwidth ordering. And Figures 3.11 and 3.12 show the structure of the matrix after applying RCM and minimum neighbouring methods respectively to the matrix shown in Figure 3.10.

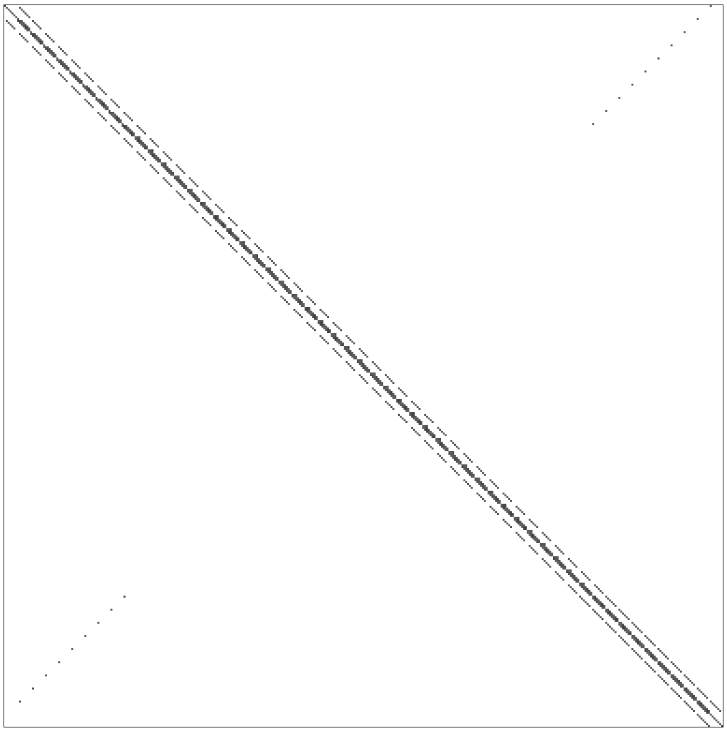


Figure 3.7: Matrix that arises from using the natural ordering, NAT.

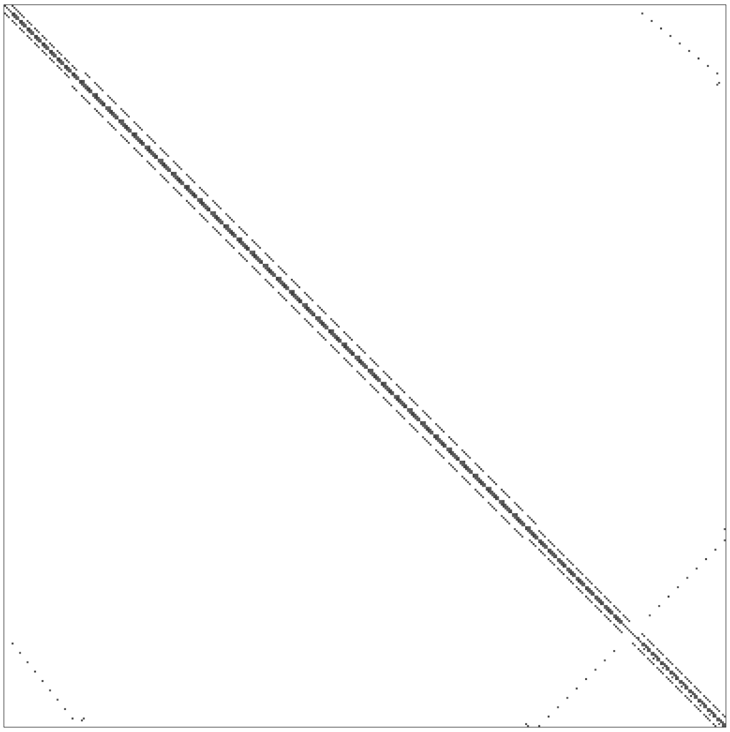


Figure 3.8: Matrix that arises from using the domain decomposition ordering DD1.

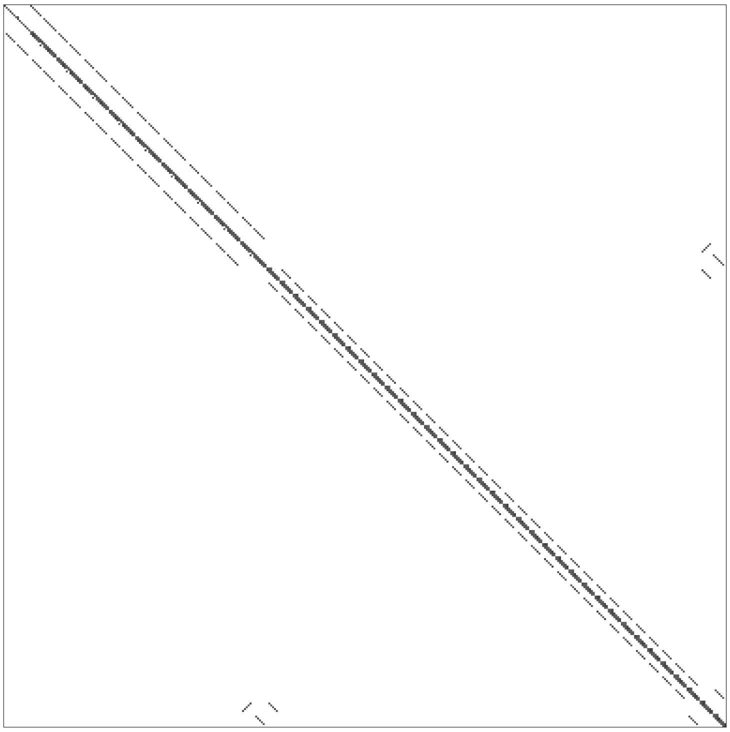


Figure 3.9: Matrix that arises from using the domain decomposition ordering DD2.

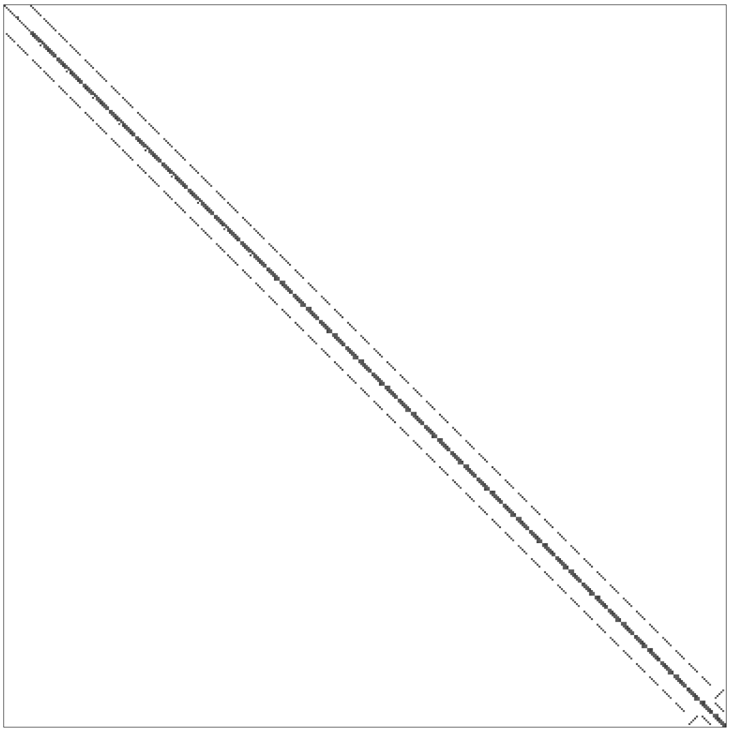


Figure 3.10: Matrix that arises from using the double bandwidth ordering, DB.

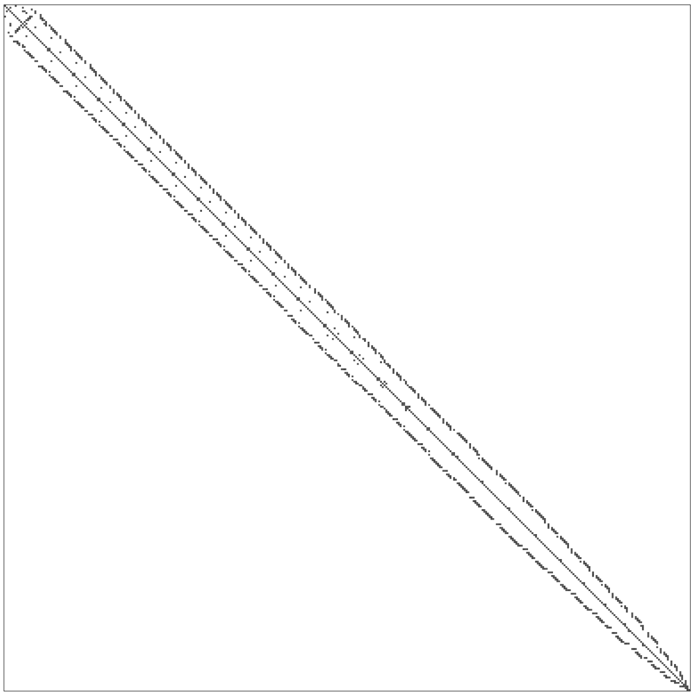


Figure 3.11: Matrix that arises from applying the RCM reorderings to an initial double bandwidth ordering.

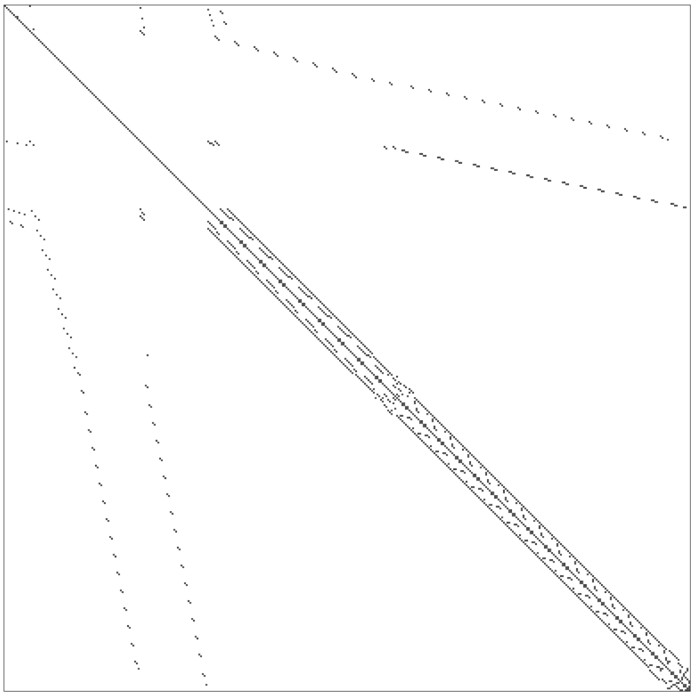


Figure 3.12: Matrix that arises from applying the minimum neighbouring reordering to an initial double bandwidth ordering.

Chapter 4

Algorithm optimization

We have seen that there are several parameters and choices that have to be optimized regarding the reordering algorithm, the preconditioner, the iterative solver and the inexact-Newton approach. It is a very tedious task, not to say an impossible one, to try to find the *ideal* set of parameters, since the choice of one influences the others. Also they may change from case to case and from grid to grid. Nevertheless, attempts can be made to choose a set that makes the code efficient and robust. After introducing the cases that we use in our study and a commentary on the units used for comparison, a detailed study of the optimization of the code is presented.

4.1 Test cases

A total of eight cases are considered in order to optimize the algorithm described in the previous chapter, as well as to study its performance and compare it to other well established solvers. The chosen cases include a wide variety of flows around airfoils: two inviscid flows, one laminar, and five turbulent. The turbulent cases include one in the incompressible regime, two other subsonic cases at a higher Mach number and two transonic flows. In all cases except for one transonic turbulent case, we use the NACA 0012 airfoil. For the last case, we use the RAE 2822 airfoil. The parameters defining the test cases are given in Table 4.1. The transonic turbulent flow conditions listed as case 6 are used for the comparisons unless stated otherwise. The initial condition is freestream flow.

case	airfoil	M	α	Re	tr.up	tr.low	grid nodes
1	NACA 0012	0.63	2.00	invisc	—	—	9711
2	NACA 0012	0.80	1.25	invisc	—	—	9711
3	NACA 0012	0.80	5.00	5.00e2	—	—	12201
4	NACA 0012	0.30	0.00	2.88e6	0.43c	0.43c	16881
5	NACA 0012	0.30	6.00	2.88e6	0.05c	0.80c	16881
6	NACA 0012	0.70	1.49	9.00e6	0.05c	0.05c	16881
7	NACA 0012	0.16	12.00	2.88e6	0.01c	0.95c	16881
8	RAE 2822	0.729	2.31	6.50e6	0.03c	0.03c	15729

Table 4.1: Parameters for the eight flows studied. The column *tr.up* is the transition point at the upper surface of the airfoil and *tr.low* is the transition point at the lower surface.

For the inviscid cases, the grid has 249×39 nodes with the wall spacing set to 2×10^{-3} chords. For the laminar case, the grid has 249×49 nodes and a wall spacing of 5×10^{-4} chords. A 331×51 grid with the wall spacing set to 1×10^{-5} chords is used for the NACA 0012 turbulent cases. For the RAE airfoil case, the grid has 321×49 nodes with similar wall spacing. These grids provide reasonable numerical accuracy for the flows considered. Mach contours, surface pressure coefficients, as well as lift and drag for the eight cases, are presented in Appendix B.

4.2 Units for comparing efficiency

When comparing the speed of different algorithms, CPU time is the appropriate unit. However, this is dependent on the computer, the compiler, and the coding details. Although it is by no means perfect, the number of function evaluations (or right-hand-side evaluations) required to reduce the residual by a given amount is a useful unit for assessing the speed of an iterative algorithm [114]. This unit allows the relative performance of different algorithms to be compared across various platforms, compilers, and flux evaluation methods. Shortcomings of this choice are that it tends to favour expensive flux evaluation methods (overhead appears smaller) and there is some arbitrariness as to what is included in a function evaluation. For example, local time stepping and the circulation correction are optional. It is also important to notice that the memory bandwidth of the computer, which may differ substantially from one machine to another, can have an important effect on the relative cost of an inner iteration to a function evaluation, since not all the operations are equally affected by the speed of the access to memory.

For most of our comparisons, we will use the number of function evaluations as our basic unit. In the function evaluation we include the flux evaluation, pressure field update, the computation of the artificial dissipation coefficients, the computation of the molecular and eddy viscosity and the evaluation of the right-hand-side at the boundaries. This permits comparison with other solvers. Since all of the methods compared in this work use the same right-hand-side, the number of function evaluations translates directly into CPU time on a given computer. All the solvers and cases are run on a Pentium Pro 180.

4.3 Inexact-Newton solver

Newton's method, which we have described in section 3.2, approximates the non-linear system of equations by a succession of linear systems of equations. An inexact-Newton method is an extension of Newton's method where, at each step n, we solve the linear system, given by Eq. (3.38), in an inexact fashion, using an iterative solver. A decision has to be made regarding how accurately we need to solve the linear systems of equations.

An inexact-Newton method applied to Eq. (3.38) can be written as

$$\|\mathcal{F}^{(n)} + \mathcal{A}^{(n)}\Delta\hat{Q}^{(n)}\| \;\leq\; \tilde{\eta}_n\|\mathcal{F}^{(n)}\| \tag{4.1}$$

which implies finding a $\Delta\hat{Q}^{(n)}$ such that the initial residual $\mathcal{F}^{(n)}$ is reduced by a factor of $\tilde{\eta}_n$. If $\tilde{\eta}_n = 0$, we recover Newton's method. The local convergence of the inexact-Newton method is controlled by $\tilde{\eta}_n$. Dembo et al. [31] showed that, under certain assumptions, 1) linear convergence with an asymptotic rate constant no greater than $\tilde{\eta}_{max}$ is obtained if $0 \leq \tilde{\eta}_n \leq \tilde{\eta}_{max} < 1$ for each $\tilde{\eta}_n$, 2) superlinear convergence is obtained if $lim_{n\to\infty}\tilde{\eta}_n = 0$, and 3) quadratic convergence is obtained if $\tilde{\eta}_n = 0(\| \mathcal{F}^{(n)} \|)$.

There is a second issue associated with $\tilde{\eta}_n$. To illustrate it, several levels of reduction of the residual of the linear problem arising at each quasi-Newton step have been tested. The convergence histories for six values of $\tilde{\eta}$ are shown in Figure 4.1. Strict inner tolerances reduce the number of outer iterations, but there is an increased number of inner iterations, giving an overall increase in total CPU time needed to converge. The results indicate that strict inner tolerances reduce the speed of the solver. To understand the reason let us recall that we are making the following approximation to the non-linear system

$$\mathcal{F}(\hat{Q} + \Delta\hat{Q}) \approx \mathcal{F}(\hat{Q}) + \mathcal{A}(\hat{Q})\Delta\hat{Q} \tag{4.2}$$

During the first few iterations, the solution is far from the converged solution, and thus the linear approximation of $\mathcal{F}$ can be very inaccurate. The use of a strict tolerance (i.e., a small value of $\tilde{\eta}_n$) during these early iterations is thus not beneficial to the rate of convergence and wastes CPU time. This is known as oversolving [115]. To illustrate this fact, in Figure 4.2 we show the logarithms of the norms of $\mathcal{F}$ and its linear approximation versus the number of GMRES iterations when reducing the inner residual by five of orders of magnitude. At each new inexact-Newton step, there is a point where further reduction of the residual in the linear problem does not reduce the outer residual. All the GMRES iterations done to reduce the inner residual below that point represent wasted effort, oversolving the linear system: at the next inexact-Newton step, the inner residual jumps up. The oversolving disappears as we approach the solution.

Strategies for choosing a sequence of $\tilde{\eta}_n$'s leading to an efficient local rate of convergence of the inexact-Newton method, have been developed by several authors and are described in Refs. [115] to [118]. The proposed strategies show superlinear and

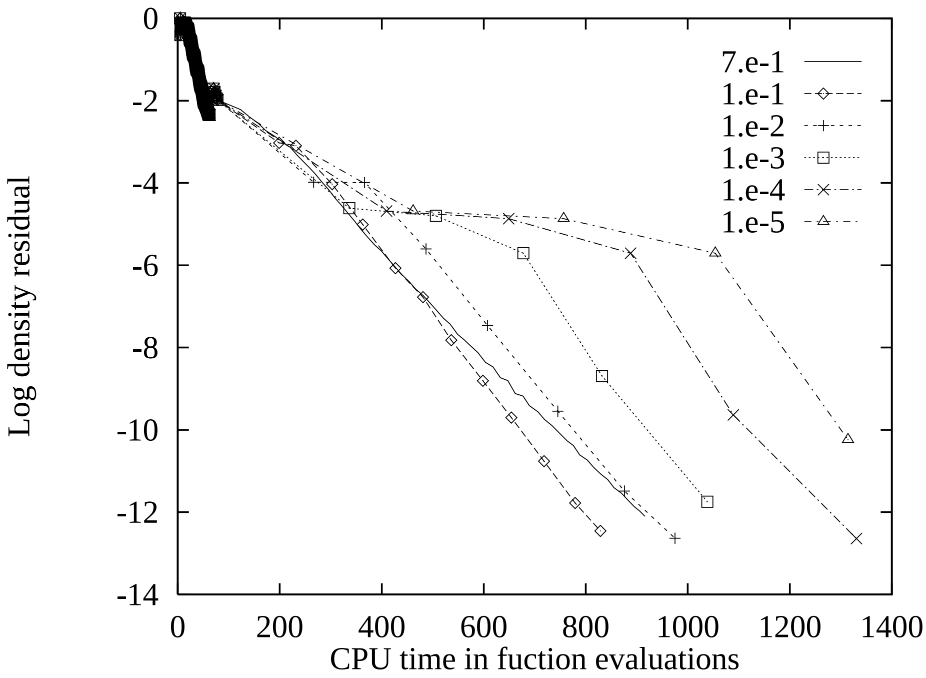


Figure 4.1: Convergence history for different levels of reduction of the inner residual using the matrix-free Newton-GMRES.

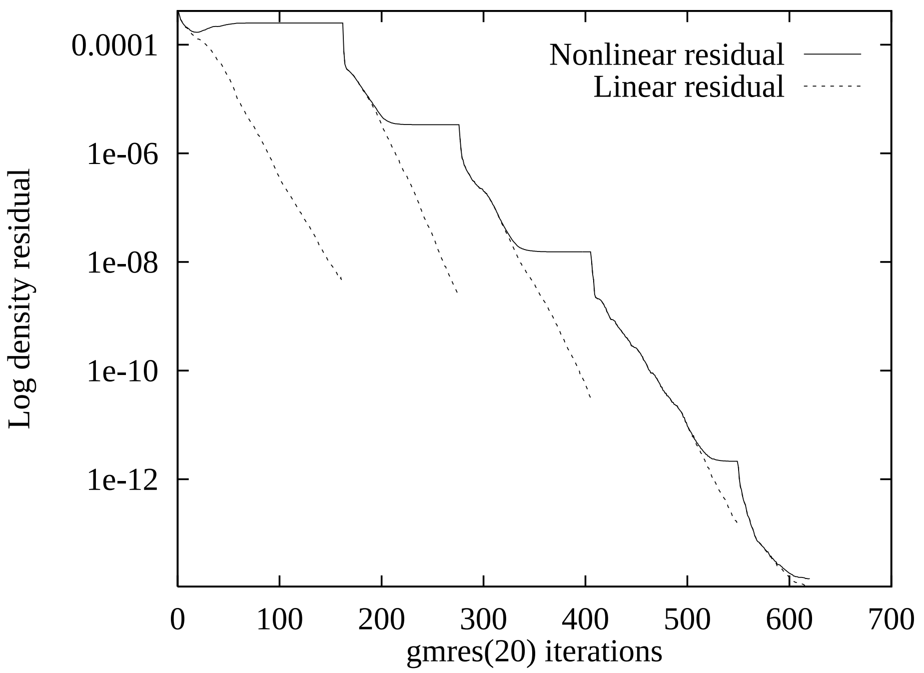


Figure 4.2: Illustration of oversolving with a reduction of the *linear residual* $\mathcal{F}(\hat{Q}) + \mathcal{A}(\hat{Q})\Delta\hat{Q}$ of five orders of magnitude: $\tilde{\eta} = 10^{-5}$. The *nonlinear residual* $\mathcal{F}(\hat{Q} + \Delta\hat{Q})$ shows no reduction for an important number of GMRES iterations.

quadratic convergence but they may not be able to avoid oversolving in some practical applications. For that reason, Eisenstat and Walker [115] introduced safegards to prevent $\tilde{\eta}_n$ from becoming too small too quickly. We find that choosing

$$\tilde{\eta}_n = \tilde{\gamma} \left(\frac{\|\mathcal{F}(\hat{Q}_n)\|}{\|\mathcal{F}(\hat{Q}_{n-1})\|} \right)^{\tilde{\alpha}} \tag{4.3}$$

with $\tilde{\gamma} \in [0,1], \tilde{\alpha} \in (1,2]$, and $\tilde{\eta}_0 \in [0,1)$ is an effective strategy when used with a safeguard. However, for our applications, we have found the following approach to give slightly better efficiency. We use $\tilde{\eta}_k = 0.5$ for the first 10 outer iterations, which guarantees no oversolving for most problems, and we then use $\tilde{\eta}_k = 0.1$ for the remaining outer iterations. This approach results in linear convergence, but savings in not having to compute $\tilde{\eta}_k$ and better behaviour regarding oversolving make it more efficient in terms of CPU time than the other approaches. It should be noted that "undersolving" at each Newton step is not particularly wasteful, while oversolving should be avoided.

4.4 Preconditioning strategies

The purpose of this section is to compare the performance of the solver with the two main preconditioners that we have described in Section 3.5, BFILU(p) and ILUT(P,τ), in order to determine the most efficient one for our applications. Different values of fill-in are considered to establish optimum performance at a limited cost of memory and CPU.

4.4.1 Comparing preconditioners

In order to compare BFILU(p) and ILUT(P,τ) with different levels of fill-in, we solve the linear system of the first Newton iteration. The preconditioners are computed from the first-order matrix. Results for case 1 are presented in Table 4.2. Similar results are obtained for other cases. Memory requirements for each preconditioner are

Preconditioner	nnz/N	cpu-form	i-it	cpu-total
BFILU(0)	19.41	0.6	360	204.4
BFILU(1)	26.45	3.4	57	33.8
BFILU(2)	33.66	5.2	44	27.8
BFILU(3)	40.90	7.4	44	29.5
BFILU(4)	44.10	8.4	43	29.7
ILUT (7,0.1)	25.15	5.2	155	91.3
ILUT (7,0.01)	27.78	8.0	94	56.3
ILUT (7,0.)	32.59	15.5	92	57.4
ILUT(12,0.1)	33.63	7.2	62	33.6
ILUT(12,0.01)	37.12	10.9	63	41.4
ILUT(15,0.3)	31.09	5.9	276	172.0
ILUT(15,0.1)	37.93	8.3	51	33.5
ILUT(15,0.01)	42.67	13.6	57	38.7
ILUT(18,0.1)	41.74	9.3	46	31.0
ILUT(18,0.2)	36.53	7.3	68	36.5

Table 4.2: Memory, CPU cost and effectiveness to reduce the inner residual by two orders of magnitude for different preconditioners.

shown as nonzeros per equation of the preconditioning matrix (nnz/N). Under "cpu-form" we indicate the CPU time (in seconds) required to compute the preconditioning matrix. The last two columns indicate the number of GMRES(20) iterations and total cpu time to reduce the initial inner residual by two orders of magnitude. Since the ordering of the unknowns has a big impact on the performance of the preconditioner, for each case we have chosen the best among the ones considered in this study. For this test case, our experience is that the optimum reordering with BFILU is RCM and that ILUT performs much better with the MN reordering than with any of the other orderings.

Results show that the ILUT preconditioners are more expensive to form than the BFILU ones. This is due to the comparison of numerical size that we have to do between the elements of the rows in order to keep the largest ones.

BFILU(p) shows a clear improvement in the performance of the solver as we allow more fill-in, up to $p = 2$. Further increase in fill-in, does not reduce the number of inner iterations, while substantially increases the memory requirements.

It seems that optimum values for the fill-in parameters are around $P = 15$ and $\tau = 0.1$. But, for the same amount of nonzeros per row, BFILU factorizations perform better than the ILUT ones.

The choice of the preconditioner has to balance computational efficiency and memory requirements. For this case, we observe that, overall, BFILU(2) is the best choice.

4.4.2 Preconditioners from the first-order and the second-order Jacobians

We have seen in section 3.5.1 that we can build the preconditioner from the same matrix $\mathcal{A}$ used for the Newton linearization, or that we can use a *reasonable* approximation of it $\tilde{\mathcal{A}}$ which could potentially produce a more efficient preconditioner. For instance, we can use the first-order Jacobian $\mathcal{A}_1$ described in section 3.2.1 to form a preconditioner, $\mathcal{M}_1$, and compare it to the preconditioner $\mathcal{M}_2$, formed from A_2, which is closer to the matrix produced by the exact linearization.

Figure 4.3 shows the convergence histories obtained with three different preconditioners for case 1 on a coarse grid with 143×20 nodes: BFILU(0) formed from the first and second-order Jacobians, and BFILU(2) formed from the first-order Jacobian. It should be noted that for this inviscid subsonic flow case, the matrix $\mathcal{A}_2$ is very close to being the exact linearization of the flux Jacobian. The number of nonzeros given by BFILU(2) using $\mathcal{A}_1$ is about the same as the number of nonzeros given by BFILU(0) using $\mathcal{A}_2$. BFILU(2) applied to $\mathcal{A}_2$ requires excessive storage and is not considered here.

The results in Figure 4.3 show that the preconditioners $\mathcal{M}_1$ built from $\mathcal{A}_1$ are more efficient than $\mathcal{M}_2$ built from the second-order Jacobian $\mathcal{A}_2$. This is examined in more detail with BFILU(0). If we multiply Eq. (3.79) by the inverse of the preconditioner, we obtain the preconditioned matrices given by:

$$\mathcal{A}\ \mathcal{M}^{-1} = \mathcal{I} - \mathcal{E}\ \mathcal{M}^{-1} \tag{4.4}$$

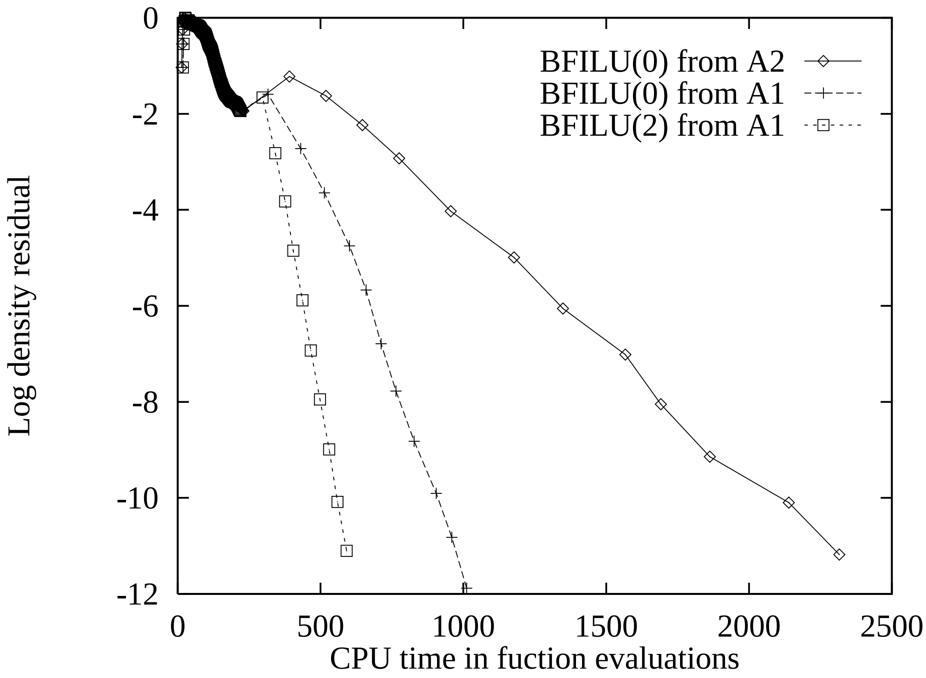


Figure 4.3: Convergence histories for case 1 in a 143 × 20 nodes grid, using three preconditioners: BFILU(0) formed from the first-order Jacobian $\mathcal{A}_1$, BFILU(0) using the second-order Jacobian $\mathcal{A}_2$ and BFILU(2) from the first-order Jacobian.

preconditioner	$\frac{\parallel \mathcal{E} \parallel_F}{\parallel \mathcal{A} \parallel_F}$	$\frac{\parallel \mathcal{E}\mathcal{M}_i^{-1} \parallel_F}{\parallel \mathcal{I} \parallel_F}$
$\mathcal{M}_1$	0.4870	4.6003
$\mathcal{M}_2$	0.2576	6676.4449

Table 4.3: Frobenius norm of the error matrix and of the preconditioned error matrix for the first- and second-order preconditioners.

Since we are solving the preconditioned system, the matrix $\mathcal{E}$ is not as important as the preconditioned error matrix $\mathcal{E}\ \mathcal{M}^{-1}$. As discussed in section 3.5.1, non-diagonally dominant matrices produce factors with inverses $\mathcal{L}^{-1}$ and $\mathcal{U}^{-1}$ which may have very large norms, causing $\mathcal{E}\ \mathcal{U}^{-1}\mathcal{L}^{-1}$ to be very large and thus adding large perturbations to the identity matrix. In that case, the eigenvalues of $\mathcal{A}\ \mathcal{M}^{-1}$ will not be nicely clustered around unity and the iterative solver will show slower convergence.

In Table 4.3, the Frobenius norms of the error matrices are presented. The results confirm that $\mathcal{M}_1$ produces an error matrix $\mathcal{E}$ that has a bigger norm than the one from $\mathcal{M}_2$, but the norm of its preconditioned error matrix is much smaller than the one from $\mathcal{M}_2$.

4.4.3 Parametric study

The results from the previous section confirm that the best factorization is not necessarily the one obtained from the matrix used at each Newton step, but from an approximation that shows better characteristics. The matrix $\mathcal{A}_1$ described in section 3.2.1 has two parameters that allow us to optimize, in some sense, the approximation.

The optimization criterion that we use is to minimize the sum of the GMRES iterations over the Newton iterations, which is equivalent to minimizing the CPU time to convergence. For this optimization process, BFILU(2) is used as preconditioner, with RCM ordering.

The parameter σ controls the amount of second-difference dissipation added to $\mathcal{A}_1$. As we increase its value, the matrix is less non-diagonally dominant and more symmetric, which will benefit the factorization. But on the other hand, the matrix will be a less accurate representation of $\mathcal{A}$. Figure 4.4 shows the total number of GMRES-iterations required to converge to machine zero for the seven compressible cases, as we vary the value of σ. As expected, there is an optimum value; for all cases, with the exception of case 4, it is equal to 5. For bigger values, the number of GMRES-iterations increases gradually and for smaller values, it increases sharply. In most cases, the code does not converge for $\sigma \leq 3$. Case 7, which is a nearly incompressible flow, behaves somewhat differently and will be discussed in section 5.1.

The other parameter, Δt_0 which appears in Eq. (3.42), is used to enhance the diagonal dominance of $\mathcal{A}_1$, which should improve the factorization properties of the matrix. Figure 4.5 shows that the influence of this parameter is, in general, much less beneficial than that of σ. Only cases 2 and 5 show some improvement in convergence when using a finite Δt_0. Values under 200 seriously affect the performance of the solver. Thus we consider 10^4 to be an optimum value for our applications.

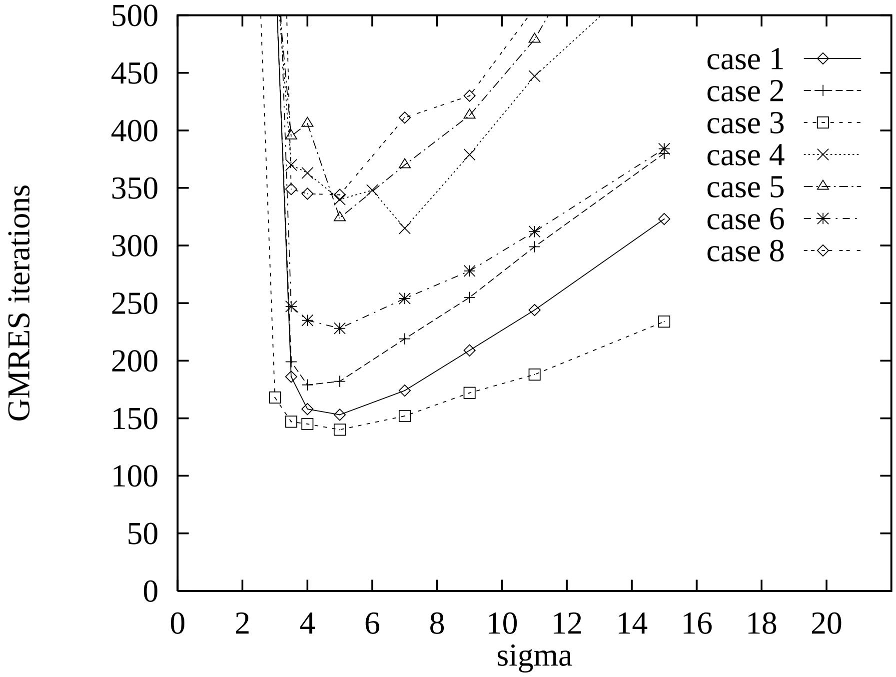


Figure 4.4: Total number of GMRES iterations required to converge to machine zero, for different values of σ for various cases.

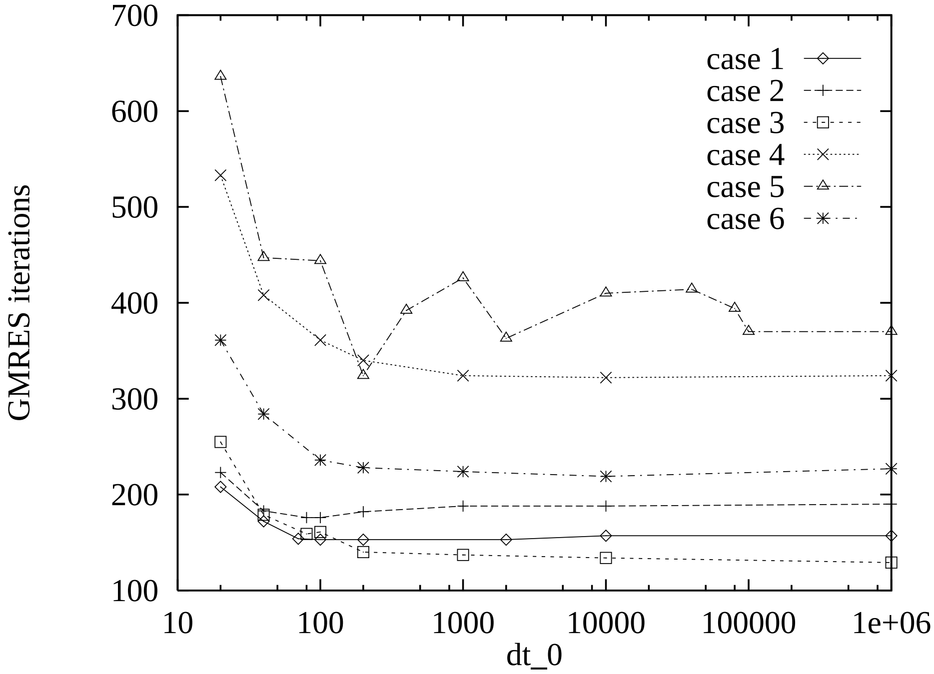


Figure 4.5: Total number of GMRES iterations required to converge to machine zero, for different values of dt_0 for cases 1 to 6.

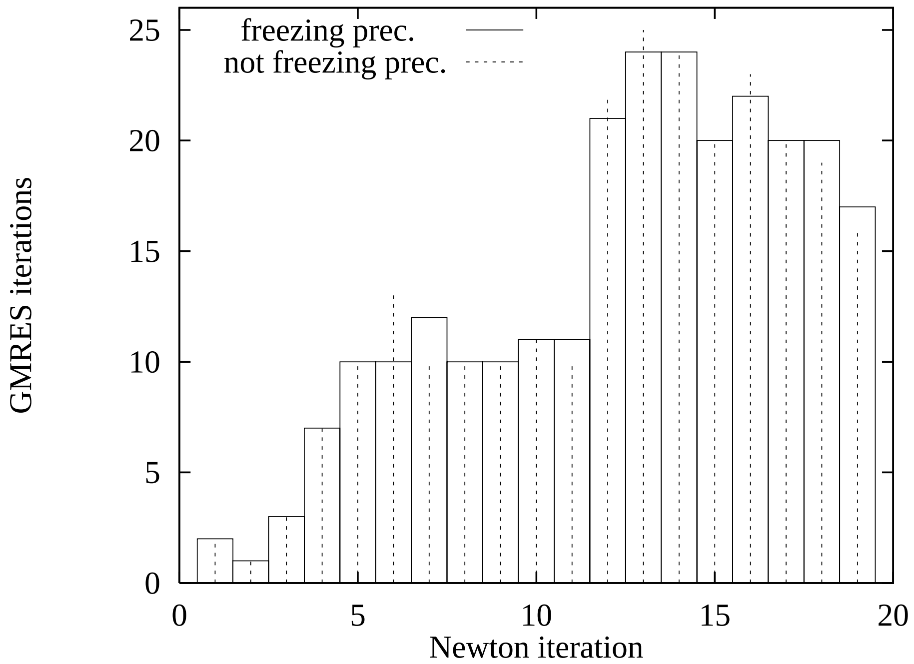


Figure 4.6: Number of GMRES iterations at each Newton iteration freezing the preconditioner (bars) and updating the preconditioner at each Newton iteration (impulses) for case 2.

4.4.4 Freezing the preconditioner

Since we are using an approximation to $\mathcal{A}$ to build the preconditioner, we may consider using the same preconditioner for all the Newton steps. In other words, we could compute the preconditioner once, at the first Newton iteration, and then freeze it. To see how this strategy affects the performance of the solver, we run one of the cases which has strong initial nonlinearities, case 2, with and without updating the preconditioner.

In Figure 4.6 we show that the number of GMRES iterations at each Newton iteration does not increase when we freeze the preconditioner. In other words, there is no gain in updating the preconditioner at each Newton iteration. At the same time, Figure 4.7 shows that there are substantial CPU savings when we do not update the preconditioner.

This is due to the fact that we have used a relaxation technique for two orders of magnitude to eliminate the most significant transients, and the flow changes do not affect the matrix in a significant way compared to the approximations that we have

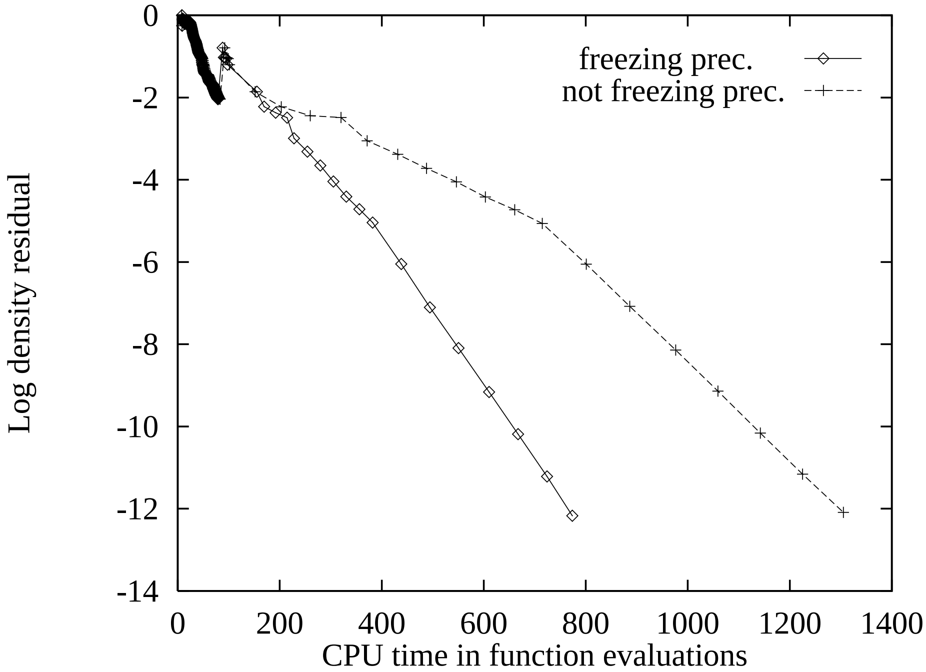


Figure 4.7: Convergence history for case 2, freezing the preconditioner and updating the preconditioner at each Newton iteration.

introduced. Therefore, computing the preconditioner only once produces important savings in CPU.

4.5 Ordering of unknowns

The six different orderings described in section 3.6, including the natural ordering (NAT), two orderings based on domain decomposition techniques (DD1 and DD2), double bandwidth (DB), Reverse Cuthill-McKee (RCM) and minimum neighbouring (MN), have been tested for cases 3 and 8. Since the optimum parameters that we have found for RCM are not necessarily the same for the other orderings, we have followed a similar process of optimization for each one of the orderings using case 3.

Convergence history plots for cases 3 and 8 are shown in Figures 4.8 and 4.9 respectively. The benefits of applying reordering techniques such as RCM and MN are clearly shown. In our experience, the performance of MN is virtually independent of the initial ordering fed to the algorithm. In contrast, the performance of RCM depends greatly on the initial ordering, with double bandwidth preferred. RCM is

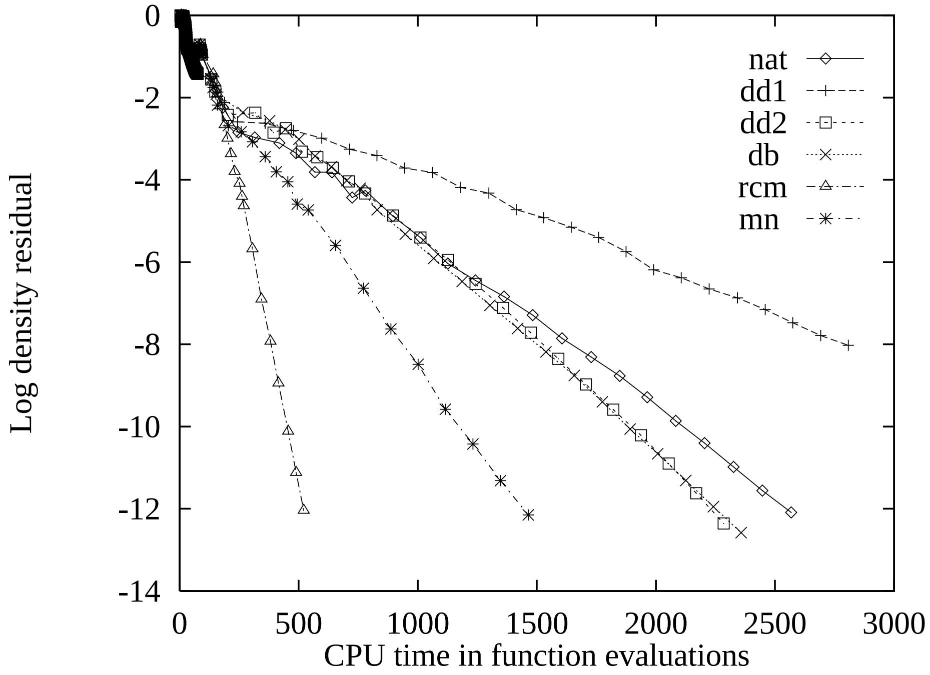


Figure 4.8: Convergence history for the six orderings for case 3, using BFILU(2) as preconditioner.

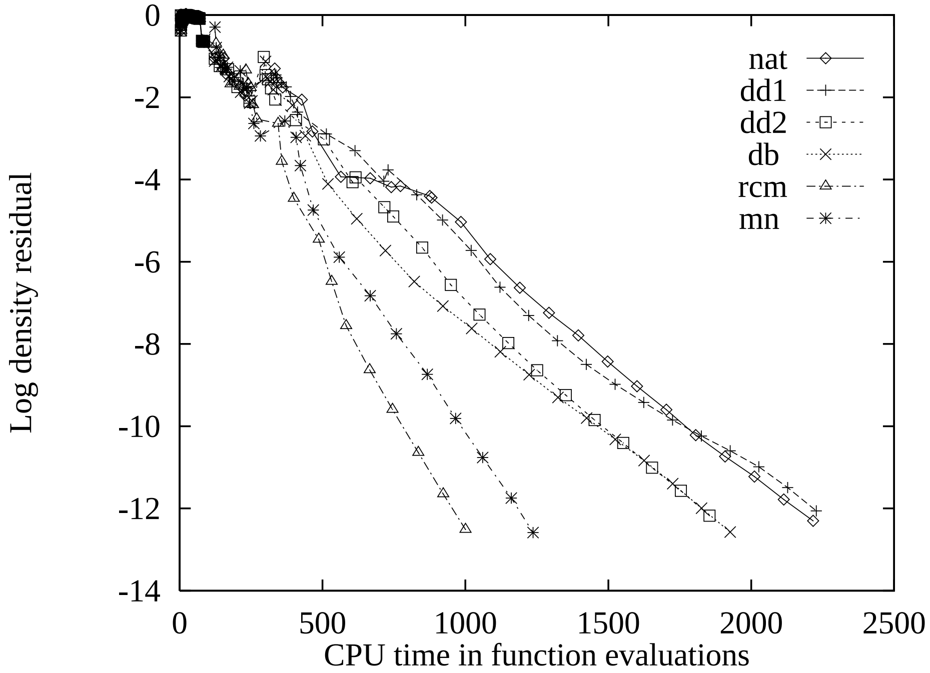


Figure 4.9: Convergence history for the six orderings for case 8, using BFILU(2) as preconditioner.

significantly faster than any other ordering for these two cases. We have compared with MN in other cases and results are generally in favour of RCM [119]. The double bandwidth algorithm and DD2 behave quite similarly to each other. The natural ordering and DD1 are the slowest ones.

4.6 Optimized algorithm

Our solver, known as PROBE, was presented in Ref. [119] and later on, in an improved version, in Ref. [120]. Here, we present an improved optimization. The following are the main strategies and parameters of this version:

- inexact-Newton strategy
- matrix-free GMRES(20)
- BFILU(2) preconditioner based on the first-order Jacobian formed using $\Delta t_0 = 10^4$ and $\sigma = 5$
- Reverse Cuthill-McKee reordering
- preconditioner computed at first iteration and not updated
- inner tolerance ($\tilde{\eta}_k$) set to 0.5 for the first ten outer iterations, 0.1 for the remainder
- approximate factorization algorithm used for 150 iterations on coarse grid or to reduce the residual two orders of magnitude initially, whichever happens first; for most cases, $\Delta t_0 = 5$ is used with the local time step definition given in Eq. 3.42.

Chapter 5

Results

In the first section of this chapter, we show the performance of PROBE for the eight cases, as well as a study of the variation of the CPU cost with the number of nodes in the grid. The second section consists of a comparison of the performance and the storage requirements of PROBE with other efficient solvers. All parameters of PROBE were fixed for all of the results presented in this chapter.

5.1 Performance of the algorithm

Ideally, a code should be able to handle different flow conditions on varying grids with relatively consistent performance. In this section, we present convergence results for all the cases. They have been run with the same parameters with the exception of case 8 because it converged only six orders of magnitude. In order to converge to machine zero, it was enough to set $\Delta t_0 = 1$ in the AF relaxation method for the coarse grid; its usual value is 5. Similar behaviour has been observed with ARC2D. This may be due to the turbulence model.

Table 5.1 gathers some statistics of the convergence histories for the eight cases that we have studied. The table includes the number of inner and outer iterations required to reduce the residual norm by twelve orders of magnitude (to machine zero), and the average number of inner iterations per outer iteration. The outer iterations include only those done using the Newton-Krylov solver and not those of the approximately-factored algorithm. CPU/f.e. time gives the total run time

	o-it	Σi-it	i-it/o-it	CPU/f.e.	CPU (min. sec.)
Case 1	18	157	8.7	615.5	1' 53.9"
Case 2	18	190	10.6	743.8	2' 17.6"
Case 3	18	129	7.2	490.7	2' 27.2"
Case 4	17	324	19.1	910.1	8' 06.0"
Case 5	19	370	19.5	990.8	8' 49.1"
Case 6	18	227	12.6	642.9	5' 43.3"
Case 7	34	1116	32.8	2824.9	25' 08.5"
Case 8	22	354	16.1	953.5	7' 47.2"

Table 5.1: Statistics for the Newton-Krylov algorithm for the cases studied; **o-it**: outer iterations, Σ**i-it**: total number of inner iterations, **i-it/o-it**: average number of inner iterations per outer iteration; **CPU/f.e.**: CPU time in equivalent function calls to reduce the residual by twelve orders of magnitude.

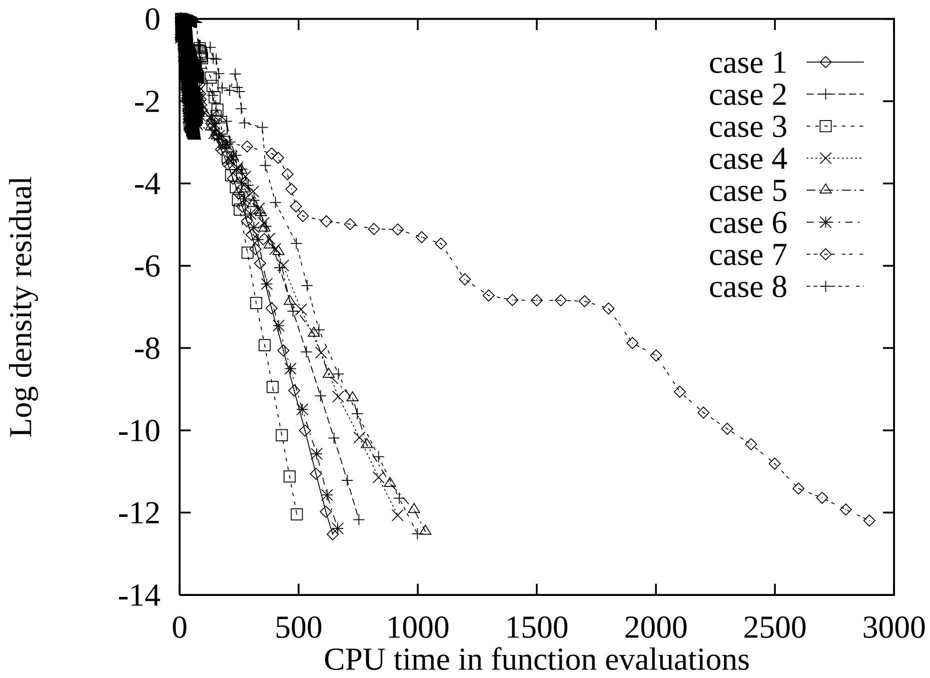


Figure 5.1: Residual history for PROBE for the 8 cases described in Table 4.1.

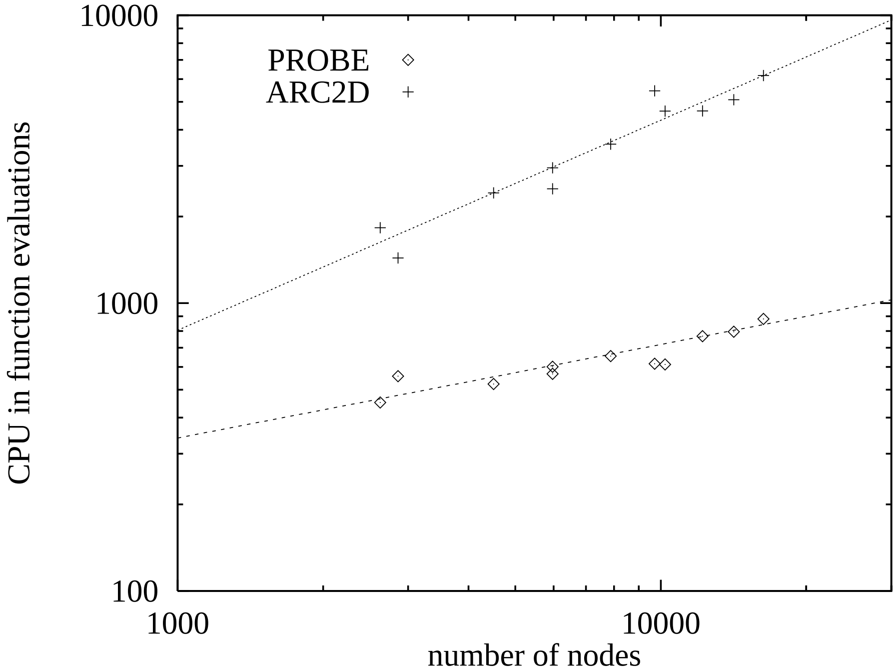


Figure 5.2: CPU time in function evaluations required to converge to machine zero as a function of the grid size, for PROBE and ARC2D using case 1.

normalized by the CPU time of a function evaluation. Figure 5.1 shows the residual histories for the cases studied.

The results show that, except for the low Mach number case, convergence is achieved in less than 1000 function evaluations. Case 7, in which the freestream Mach number is 0.16, shows a slower convergence. The number of inner iterations goes up dramatically for this case. In order to minimize oversolving in cases like this one, we always limit the maximum number of inner iterations at each quasi-Newton step to 40.

Another important consideration in the performance of a solver is how the CPU time increases with the size of the problem. To this end, we have tested case 1 using different grids. The results for PROBE and for ARC2D, which is included as a reference, are plotted in Figure 5.2. We can approximate the points in the plot by the expression

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

where ω is the CPU time in function evaluations, and κ and β are constants. Ideally β should be zero, which would mean that the CPU cost would be linear with respect to the grid size, since the cost of a function evaluation varies linearly with N. For ARC2D, $\beta = 0.73$ and for PROBE $\beta = 0.325$. Thus, the scalability of PROBE appears to be very good.

5.2 Comparison to other solvers

We compare PROBE with other efficient solvers, namely the approximate factorization algorithm of Beam and Warming as implemented in ARC2D [16], the same solver ARC2D enhanced by multigrid (ARC2D-MG), an incomplete factorization solver (BFILU(2)) and an approximate-Newton solver (approx. Newton). The spatial discretization for all the solvers is identical.

Not all the grids defined in Section 4.1 are suitable for the multigrid solver. The reason being that coarsening is done by removing every other interior node; this requires that, in order to be able to extract two coarser grids, $j_{max} - 1$ and $k_{max} - 1$ have to be divisible by 4. Therefore, for cases 1 and 2, we modified the grid to have 249×41 nodes; and for cases 4 to 7, we modified the grid to have 329×49 nodes.

5.2.1 Description of solvers

Approximate factorization (ARC2D)

The approximate factorization algorithm in diagonal form, as used in ARC2D, is explained in detail in Ref. [16]. In ARC2D, the wakecut can be treated implicitly or explicitly. In this work we consider only the explicit treatment of the wakecut, which is faster for our present test cases. ARC2D provides a useful reference because of its wide availability.

Approximate factorization with multigrid (ARC2D-MG)

Other authors [17, 18] have already shown that multigrid can substantially increase the convergence rate of the approximate factorization algorithm. For our study, we use a three level sawtooth-cycle. Four AF iterations at each level produces optimum convergence in terms of CPU time [18].

Incomplete factorization (BFILU(2))

BFILU(2) has proven to be an efficient preconditioner for our applications. It can also be used as a solver, with the same strategy as used for the preconditioner: the factorization is calculated from the first-order Jacobian at the first iteration and it is not updated. After testing different values of σ and Δt_0, we have chosen 5 and 50 respectively, which produce a good convergence rate.

Approximate-Newton (approx. Newton)

Some of the most popular approximate-Newton methods use a first-order Jacobian on the left-hand side. One of the original reasons is that this matrix requires less storage than the second-order Jacobian. Another reason for using a first-order Jacobian is that it is better conditioned than the second-order Jacobian; hence the inner iterations can converge faster. The penalty is an increased number of outer iterations. Approximate-Newton solvers which use a first-order Jacobian are typically preconditioned with a BILU(0) or a BFILU(0) factorization. With this preconditioner, the solver requires approximately the same amount of storage as our matrix-free inexact-Newton solver. The parameters σ and Δt_0 that appear in the matrix $\mathcal{A}_1$ have been optimized using case 1, following a similar process to the one used for PROBE. Values of $\sigma = 9$ and $\Delta t_0 = 20$ minimize the CPU time required to converge to machine zero. The inner residual is reduced by a factor of 5×10^{-1}. An efficient strategy which helps to reduce the overall CPU time without harming the convergence rate consists of freezing the left-hand-side and the preconditioner after the first iteration, as we do for PROBE and the BFILU(2) factorization.

5.2.2 Performance comparison

The residual convergence histories of PROBE are compared with the other four solvers in Figures 5.3 to 5.6.

For the eight cases, PROBE is substantially faster than the reference solver, ARC2D. With this set of grids, PROBE does not converge to machine zero for case 5 and ARC2D does not converge for cases 4 and 7. Therefore, we use the convergence histories obtained with the grids of Section 5.1 to calculate the relative speedup factors to converge to machine zero. Results are given in Table 5.2.

Consistent with results presented by other authors, multigrid is very effective in accelerating the approximate factorization algorithm by a factor of 3 to 6. Thus, ARC2D-MG performs quite well, with the exceptions of cases 3 and 4. Nevertheless, PROBE remains a faster solver in all of the cases studied.

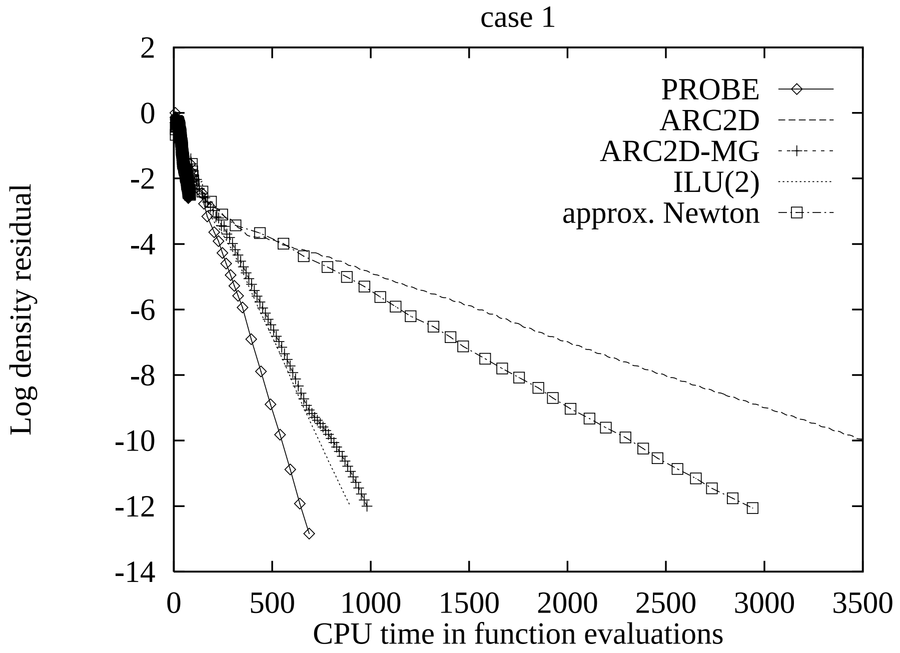
Excluding case 8 for which it diverged, BFILU(2) shows very good convergence properties. For the inviscid cases and for case 5, it performs as well as ARC2D-MG, being faster than the multigrid solver for the rest of the cases. For case 7, it is as fast as PROBE; for cases 1 to 6, PROBE is between 20% and 60% faster.

Finally, the approximate-Newton strategy preconditioned with BFILU(0) is, on average, five or six times slower than PROBE. If the preconditioner is upgraded to BFILU(2), its performance increases but the solver uses more memory than the matrix-free inexact-Newton approach and is still substantially slower.

5.2.3 Memory comparison

Two of the main needs for storage for PROBE come from the preconditioner, and the search directions to form the Krylov subspace. The preconditioner is stored in Modified Sparse Row (MSR) format [110], which requires a real and an integer array of lengths equal to the number of nonzeros and another integer array of length equal to four times the number of nodes in the grid. With BFILU(2), the number of nonzeros per node is on the order of 136. Since a real number represents one word (8 bytes) and an integer half a word, the total amount of memory required for the preconditioner is on the order of 206 words per node. The Krylov subspace with 20 search directions adds 80 words per node, which brings the total to 286 words per node. This is just a very conservative estimate of the memory needs. Work arrays and variable arrays, which depend on the particular implementation of the algorithm, have also an important impact on the amount of memory required.

In order to have an idea of the relative needs of memory of the solvers that we are considering, we use the numerical experiments of case 6, comparing the amount of resident memory that the codes occupy in the computer. This is not a perfect measure, since it depends on many factors foreign to the algorithm, but it serves as a reference. We show in Table 5.3 the words per node and the amount of memory relative to ARC2D for each solver. The high storage required by the Newton-Krylov methods and the BFILU(2) method is partially due to a not too careful allocation of memory on our part. Nevertheless, these methods require substantially more memory than the traditional solvers.



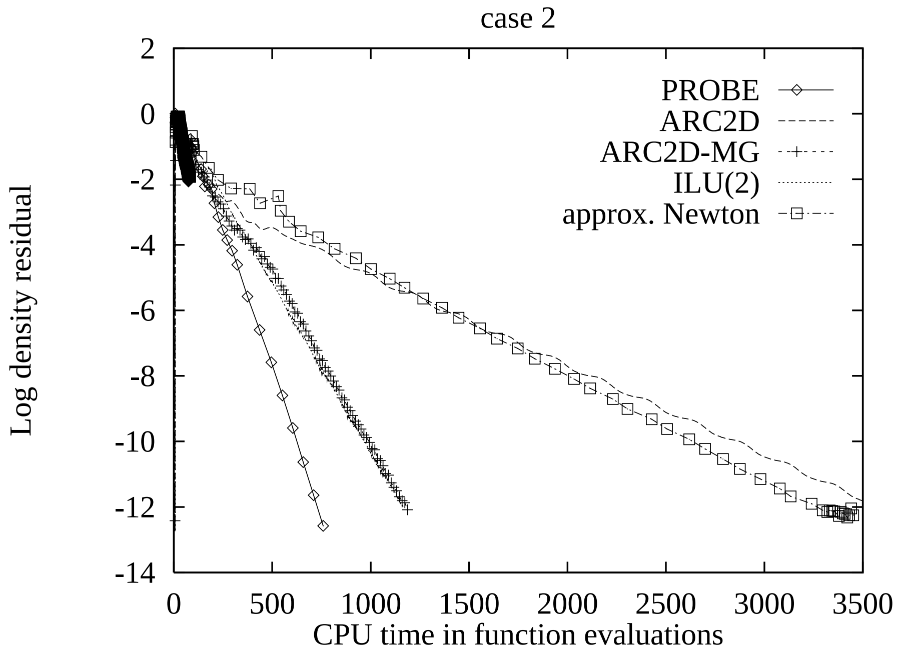


Figure 5.3: Cases 1 and 2: convergence history for the inexact-Newton-Krylov method (PROBE), the approximately-factored method (ARC2D), the approximately-factored method with three-level multigrid (ARC2D-MG), the incomplete factorization method (BFILU(2)), and the approximate-Newton method (approx. Newton).

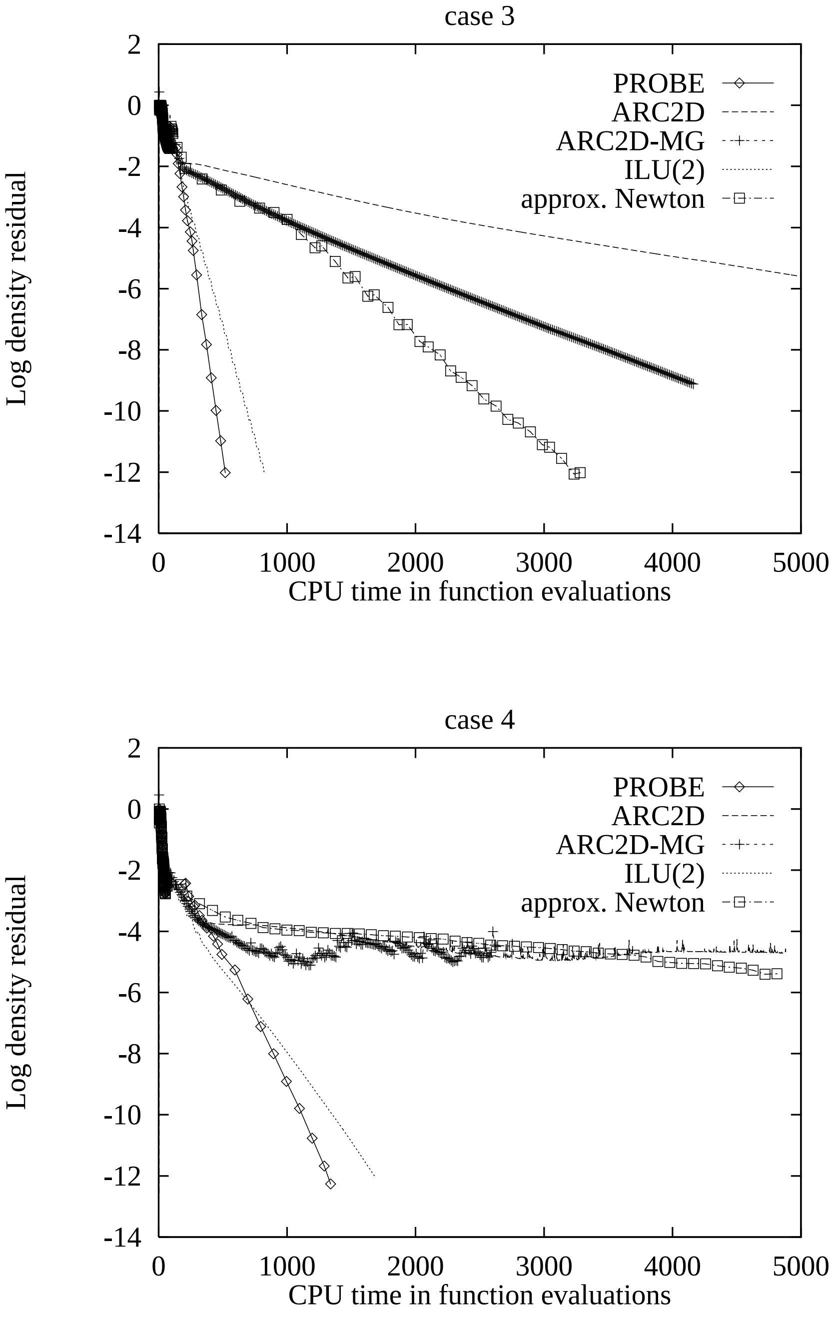
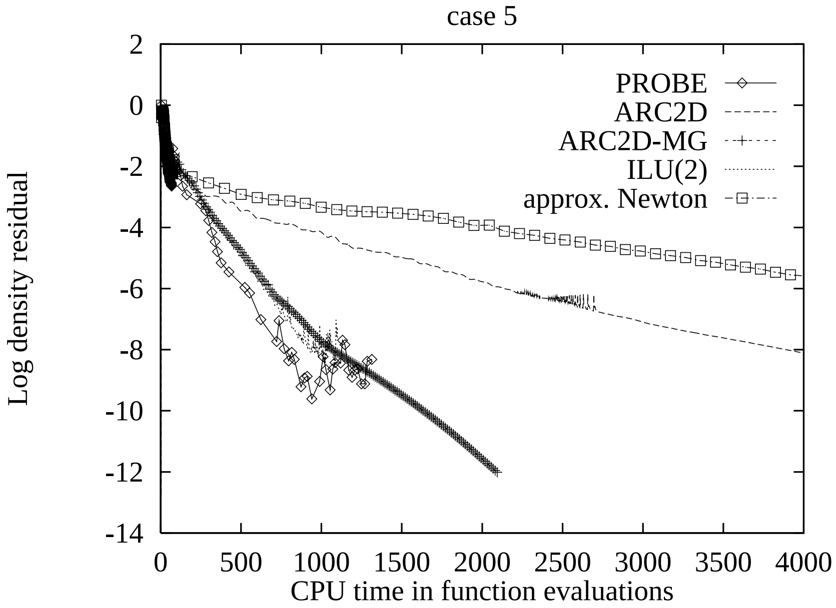


Figure 5.4: Cases 3 and 4: convergence history for the inexact-Newton-Krylov method (PROBE), the approximately-factored method (ARC2D), the approximately-factored method with three-level multigrid (ARC2D-MG), the incomplete factorization method (BFILU(2)), and the approximate-Newton method (approx. Newton).



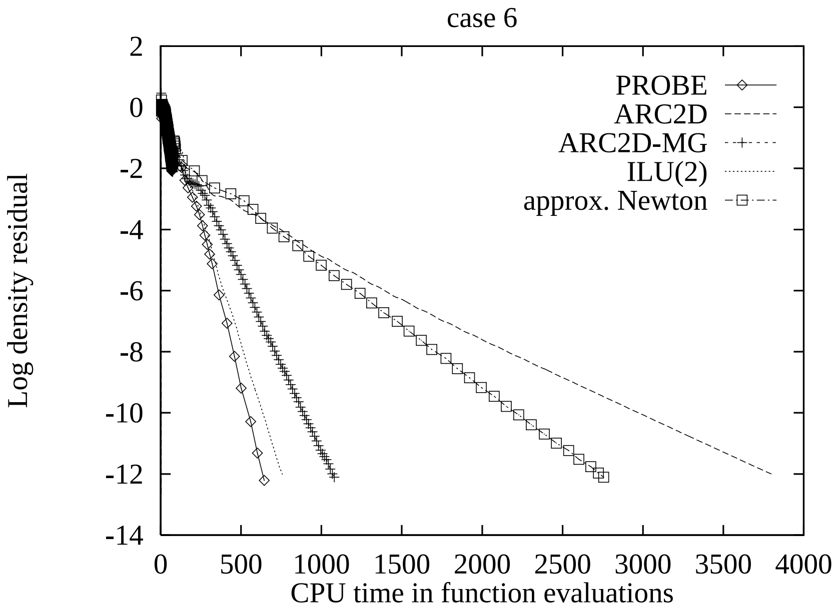


Figure 5.5: Cases 5 and 6: convergence history for the inexact-Newton-Krylov method (PROBE), the approximately-factored method (ARC2D), the approximately-factored method with three-level multigrid (ARC2D-MG), the incomplete factorization method (BFILU(2)), and the approximate-Newton method (approx. Newton).

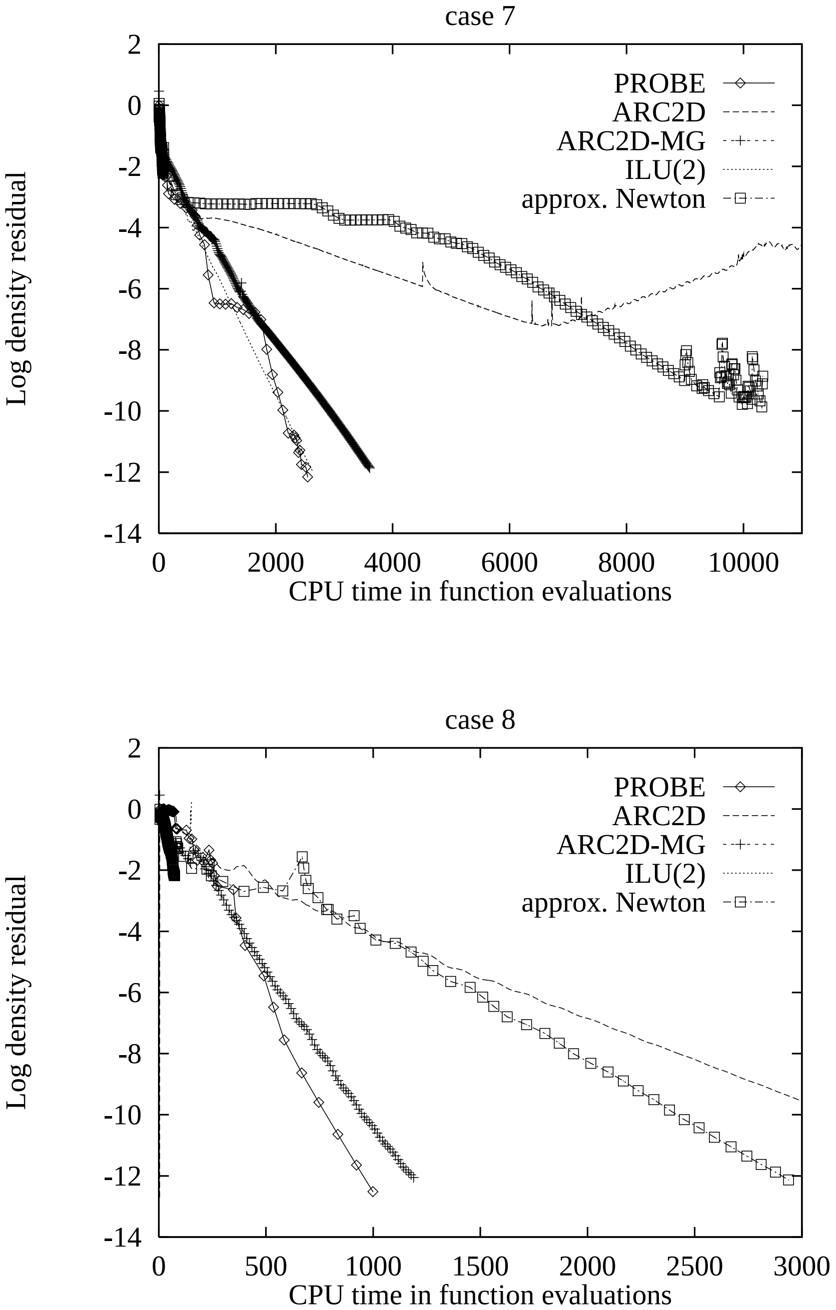


Figure 5.6: Cases 7 and 8: convergence history for the inexact-Newton-Krylov method (PROBE), the approximately-factored method (ARC2D), the approximately-factored method with three-level multigrid (ARC2D-MG), the incomplete factorization method (BFILU(2)), and the approximate-Newton method (approx. Newton).

	PROBE	ARC2D	speedup
	(*CPU in funct. eval.*)		
Case 1	615.5	5329.2	8.7
Case 2	743.8	3810.6	5.1
Case 3	490.7	13630.0	27.8
Case 4	910.1	10877.9	12.0
Case 5	990.8	7557.7	7.6
Case 6	642.9	4298.5	6.7
Case 7	2824.9	16558.2	5.9
Case 8	953.5	3892.6	4.1

Table 5.2: Comparison in performance between PROBE and ARC2D to reduce the initial residual by twelve orders of magnitude.

Solver	words/node	rel. to ARC2D
PROBE	528.04	9.69
BFILU(2)	518.76	9.51
approx. Newton	509.86	9.34
ARC2D-MG	148.00	2.71
ARC2D	54.56	1.00

Table 5.3: Comparison of the storage requirements between the five solvers, in words per node and relative to ARC2D.

Chapter 6

Conclusions, Contributions and Recomendations

6.1 Conclusions

An efficient matrix-free inexact-Newton-GMRES algorithm has been developed for steady aerodynamic flows. The thin-layer approximation is used for viscous flows. The effects of turbulence are simulated with the Baldwin-Lomax turbulence model. Spatial discretization is done using second-order centered-differences. The second and fourth-difference dissipation model of Jameson et al. [90] is added. The discretized equations are linearized using Newton's method. Preconditioned restarted GMRES in matrix-free form is used to solve the linear system arising at each Newton iteration. The preconditioner is formed using an incomplete factorization of an approximate-Jacobian matrix after applying a reordering technique. For some flow cases, especially those with shocks, the early Newton iterations can diverge and a relaxation technique has to be used to overcome this difficulty. In the present solver, an approximately factored algorithm is used to reduce the residual two orders of magnitude before switching to Newton-Krylov. The algorithm, has been successfully applied to a wide range of test cases which include inviscid, laminar, and turbulent flows.

A thorough study has been done to optimize the solver. The first part of the study consists of determining an efficient inexact-Newton strategy that avoids oversolving. This is followed by a comparison between different preconditioners of the incomplete-lower-upper factorization family. The matrix used for the factorization is carefully chosen in order to produce well-conditioned factors. Reordering techniques that improve the factorization are compared. The main conclusions of this study can be summarized as follows:

- Oversolving is avoided and optimum performance is obtained by setting the inner tolerance to 0.5 for the first ten iterations, and to 0.1 for the remainder, once the transients are less severe. As a safeguard, the total number of GMRES iterations at each Newton iteration is limited to 40.

- For the same storage requirements, Block-Fill ILU(p) factorizations (BFILU(p)) is a more efficient preconditioner than ILUT(P,τ). The best performance/memory ratio was obtained for BFILU(2).

- It is possible to produce better conditioned factorizations with a well chosen approximate-Jacobian matrix than with the exact-Jacobian matrix.

- Two parameters were used in the approximation of the Jacobian matrix. The first one, which corresponds to a local time stepping, has little beneficial influence in the quality of the preconditioner; for small values, it actually degrades the convergence of GMRES. The parameter that controls the amount of artificial dissipation in the matrix shows an optimum value that significantly improves the efficiency of the preconditioner. The behavior of these parameters and their optimum values differ from the above when the factorization is used as a solver.

- The use of a fixed preconditioner does not adversely affect convergence of the inner iterations. Therefore, the factorization is computed only once, which reduces the CPU cost substantially.

- Reverse Cuthill-McKee reordering applied to the double bandwidth ordering produces the fastest convergence of the six ordering strategies that have been studied.

The performance of the solver with optimum parameters has been tested for a wide range of flow conditions. In most cases, the solver reduces the initial residual by twelve orders of magnitude in 500 to 1000 function evaluations. The most significant exception occurs for low Mach numbers (i.e., $M_\infty = 0.16$) for which convergence is around 2800 function evaluations. The code is quite robust: with the optimized set of parameters it is capable of handling very different flow conditions and grids, some with cells which have aspect ratios on the order of 10^4. However, in certain cases, convergence may not reach machine zero, probably due to the turbulence model; a modification in the parameters of the approximate-Jacobian or on the time step of the relaxation method often overcomes the problem. Scalability has been tested for one of the cases, giving good results: the CPU cost is proportional to the size of the problem to the power of 1.325.

The solver has been compared with four other efficient algorithms including an approximate-factorization solver, a multigrid solver, a Block-Fill incomplete LU-factorization solver and an approximate-Newton solver. It has been shown that the new solver is in general more robust and significantly faster than the others. For most of the cases considered here, the speedups over the baseline approximate-factorization solver (ARC2D) are between 6 and 9. Surprising convergence results have been shown for the BFILU(2) factorization of our approximate-Jacobian matrix when used as a solver. Although less robust than the Newton-Krylov method for flows with strong shocks, it is in general very efficient, performing in some cases as well as the Newton-Krylov solver.

6.2 Contributions

Our primary contribution lies in developing a very efficient and robust Newton-Krylov method for a wide range of aerodynamic calculations. The new solver is competitive with the fastest existing solvers. This has been achieved through a careful optimization and selection of strategies at different levels of the solver.

An important contribution of our study is showing that there are approximate-Jacobian matrices that produce better conditioned factors in an ILU process, and thus better preconditioners, than the exact-Jacobian matrix. We have shown too that the approximation used in the Jacobian has a key role in the efficiency of the solver. We have presented an efficient approximation to the Jacobian, consisting of a simple modification of the artificial dissipation. The modification includes a parameter that provides some possibility of optimization. A single value of this parameter was shown to be effective for all cases studied. The quality of the preconditioner that is produced is such that it can be frozen (i.e., not updated) without deterioration in the performance of GMRES.

We have demonstrated that the reordering techniques commonly used in the context of unstructured grids, can be applied to structured grids, contributing significantly to the overall efficiency of the solver.

Finally, we have shown the potential of BFILU(2) as a solver.

6.3 Recommendations

The algorithm that we have developed constitutes a useful tool for practical aerodynamic calculations and a good platform to continue research on Newton-Krylov solvers. We suggest here some modifications that could be introduced to improve the accuracy and versatility of the solver. We propose as well some possible research avenues to make the algorithm more efficient and to reduce its storage requirements.

Accuracy could be improved in boundary layers by replacing the scalar dissipation with matrix artificial dissipation. Its impact on the condition of the LU-factors of the approximate-Jacobian matrix should be evaluated. Possible ways of optimizing the new approximate-Jacobian should be defined. If matrix dissipation reduces the quality of the preconditioner, scalar dissipation could be used for the approximate-Jacobian, while matrix dissipation would be used on the right-hand-side and in the matrix-vector products.

The turbulence model that is currently implemented in PROBE is the Baldwin-Lomax turbulence model, which is most appropriate to attached and mildly separated boundary layers. For high-lift calculations, a field-equation turbulence model should be implemented. Godin et al. [121] have shown that the one-equation turbulent model of Spalart-Allmaras [122] is quite accurate in attached flows and wakes, including merging boundary layers and wakes, while the two-equation turbulence model of Menter [123] is preferred for separated flow regions. The flow equations and the turbulence model equations have to be solved simultaneously as we are using Newton's method. Therefore, the 4×4 blocks of the Jacobian matrix will be replaced by 5×5 blocks for a one-equation model and 6×6 blocks for a two-equation model. Impact introduced by this modification on the efficiency of the preconditioner and on convergence would need to be evaluated.

In order to solve multi-element airfoil configurations, the solver should be extended to handle multiblock-grids [124] and its performance evaluated. Reordering techniques become even more relevant for these type of grids, due to the fact that the connectivity between different blocks of the grid will introduce many more non-zeros outside the main bandwidth of a natural ordering. The performance of the two main reordering techniques considered in our study, namely Reverse Cuthill-McKee and minimum neighbouring should be re-evaluated.

In terms of efficiency, it has been shown that convergence of the Newton-Krylov solver is significantly slower for lower Mach number flows than for flows in the compressible range. The use of local preconditioning could overcome the problem. Ideally, the solver should show the same level of performance for all Mach numbers.

Another way to increase the efficiency would be to give consideration to the parallelization of the solver. In this case, more research needs to be done to develop parallelizable preconditioners (see Dutto et al. [54, 55, 125] for some results in shared-memory computers). Classical preconditioners as ILU factorizations, do not parallelize well, both during the factorization and the backward-forward substitution. On the other hand, more scalable preconditioners, such as a diagonal preconditioner, are not very robust, and even if the system can be solved in parallel, the total saving in computational time compared with a better sequential preconditioner could be very small.

In addition to speed, memory use is also an important consideration in algorithm development. We have seen that PROBE requires significantly more storage than some other current solvers. This fact presently restricts the use of the solver to applications where memory use is not a dominant concern, such as two-dimensional flows. Development of an effective matrix-free preconditioner would be a significant advance.

Finally, we have shown that BFILU(2) performs very well as a solver. We think that it merits further development, perhaps in conjunction with multigrid.

References

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

[2] MacCormack, R.W., "The Effect of Viscosity on Hypervelocity Impact Cratering." AIAA Paper 69-0354, April 1969.

[3] South, J.C., and Brandt, A., "Application of a Multi-Level Grid Method to Transonic Flow Calculations." Proc. of Workshop on Transonic Flow Problems in Turbomachinery, Monterey, 1976, edited by T.C. Adamson and M.F. Platzer, Hemisphere, pp. 180–206, 1977.

[4] Jameson, A., "Acceleration of Transonic Potential Flow Calculations on Arbitrary Meshes by the Multigrid Method." Proc. AIAA 4th Computational Fluid Dynamics Conference, Williamsburg, 1979, pp. 122-146.

[5] Ni, R.H., "A Multiple Grid Scheme for Solving the Euler Equations." Proc. AIAA 5th Computational Fluid Dynamics Conference, 1981, pp. 257-264.

[6] Jameson, A., "Solution of the Euler Equations for Two Dimensional Transonic Flow by a Multigrid Method," *Applied Mathematics and Computation*, vol. 13, pp. 327–356, 1983.

[7] Martinelli, L., Jameson, A., and Grasso, F., "A Multigrid Method for the Navier-Stokes Equations." AIAA Paper 86-0208, January 1986.

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

[9] Douglas, J., and Gunn, J.E., "A General Formulation of Alternating Direction Methods," *Numerische Mathematik*, vol. 6, 1964.

[10] Peaceman, D.W., and Rachford H.H., "The Numerical Solution of Parabolic and Elliptic Differential Equations," *SIAM J.*, vol. 3, no. 1, pp. 28–41, 1955.

[11] Stone, H.L., "Iterative Solution of Implicit Approximations of Multidimensional Partial Differential Equations," *SIAM J. Num. Anal.*, vol. 5, no. 3, 1968.

[12] Beam, R., and Warming, R.F., "An Implicit Finite-Difference Algorithm for Hyperbolic Systems in Conservation Law Form," *J. Comp. Phys.*, vol. 22, pp. 87–110, 1976.

[13] Briley, W.R., and McDonald, H., "Solution of the Multi-Dimensional Compressible Navier-Stokes Equations by a Generalized Implicit Method," *J. Comp. Phys.*, vol. 74, 1977.

[14] Steger, J.L., "Implicit Finite Difference Simulation of Flow About Arbitrary Geometries with Application to Airfoils." AIAA Paper 77-665, June 1977.

[15] Pulliam, T.H., and Chaussee, D.S., "A Diagonal Form of an Implicit Approximate Factorization Algorithm," *J. Comp. Phys.*, vol. 39, p. 347, 1981.

[16] Pulliam, T.H., "Efficient Solution Methods for the Navier-Stokes Equations." Lecture Notes For The Von Karman Institute For Fluid Dynamics Lecture Series, January 1986.

[17] Jespersen D., Pulliam T., and Buning P., "Recent Enhancements to OVERFLOW." AIAA Paper 97-0644, January 1997.

[18] Chisholm, T., and Zingg, D.W., "Multigrid Acceleration of an Approximately Factored Algorithm for Aerodynamic Flows." 44th Annual Conference of the Canadian Aeronautics and Space Institute, April 1997.

[19] Chakravarthy, S.R., "Relaxation Methods for Infactored Implicit Upwind Schemes." AIAA Paper 84-0165, January 1984.

[20] Van Leer, B., and Mulder, W.A., “Relaxation Methods for Hyperbolic Equations,” Tech. Rep. 84-20, Delft University of Technology, 1984.

[21] Thomas, J.L., and Walters, R.W., “Upwind Relaxation Algorithm for the Navier-Stokes Equations.” AIAA Paper 85-1501, July 1985.

[22] Walters, R.W., and Dwoyer, “An Efficient Iteration Strategy Based on Upwind Relaxation Schemes for the Euler Equations.” AIAA Paper 85-1529, July 1985.

[23] Jameson, A., and Yoon, S., “Lower-Upper Implicit Schemes with Multiple Grids for the Euler Equations,” *AIAA J.*, vol. 25, no. 7, pp. 929–935, 1987.

[24] Yoon, S., and Jameson, A., “Lower-Upper Symmetric-Gauss-Seidel Method for the Euler and Navier-Stokes Equations,” *AIAA J.*, vol. 26, no. 9, pp. 1025–1026, 1988.

[25] Wigton, L.B.., “Application of MACSYMA and Sparse Matrix Technology to Multielement Airfoil Calculations.” AIAA Paper 87-1142, June 1987.

[26] Venkatakrishnan, V., “Newton Solution of Inviscid and Viscous Problems.” AIAA Paper 88-0413, January 1988.

[27] Hafez, M., Palaniswamy, S., and Mariani, P., “Calculations of Transonic Flows with Shocks using Newton’s Method and a Direct Solver, Part II.” AIAA Paper 88-0226, January 1988.

[28] Venkatakrishnan, V., and Barth, T.J., “Application of Direct Solvers to Unstructured Meshes for the Euler and Navier-Stokes Equations Using Upwind Schemes.” AIAA Paper 89-0364, January 1989.

[29] Orwis, P.D., “A Newton’s Method Solver for the Two-Dimensional and Axisymmetric Navier-Stokes Equations.” Ph. D. Thesis, Dept. of Mechanical and Aerospace Engineering, North Carolina State University, 1990.

[30] Bailey, H.E., and Beam, R.M., "Newton's Method Applied to Finite-Difference Approximations for the Steady-State Compressible Navier-Stokes Equations," *J. Comp. Phys.*, vol. 93, pp. 108–127, 1991.

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

[32] Hestenes, M.R., and Stiefel, E., "Methods of Conjugate Gradients for Solving Linear Systems of Equations," *J. Res. Nat. Bur. Stand.*, no. 49, pp. 409–436, 1952.

[33] Saad, Y., "Krylov Subspace Methods for Solving Large Unsymmetric Linear Systems," *Math. Comput.*, vol. 37, pp. 105–126, 1981.

[34] Young, D.M., and Jea, K.A., "Generalized Conjugate Gradient Acceleration of Nonsymmetrizable Iterative Methods," *Linear Algebra Appl.*, vol. 34, pp. 159–194, 1980.

[35] Vinsome P.K.W., "Orthomin, An Iterative Method for Solving Sparse Sets of Simultaneous Linear Equations." Proceeding of the Fourth Symposium of Reservoir Simulation, Society of Petroleum Engineers of AIME.

[36] Saad, Y., and Schultz, M.H., "GMRES: A Generalized Minimal Residual Algorithm For Solving Nonsymmetric Linear Systems," *SIAM J. Sci. Stat. Comput.*, vol. 7, no. 3, 1986.

[37] Lanczos C., "Solution of Systems of Linear Equations by Minimized Iterations," *J. Res. Nath. Bur. Stand.*, no. 49, pp. 33–53, 1952.

[38] Fletcher, R., "Conjugate Gradient Methods for Indefinite Systems." Numerical Analysis Dundee 1975, G. Watson, ed., Berlin, New York, 1976, Springer Verlag.

[39] Sonneveld, P., "CGS, a Fast Lanczos Type Solver for Nonsymmetric Linear Systems," *SIAM J. Sci. Stat. Comput.*, vol. 10, no. 1, pp. 36–52, 1989.

[40] van der Vorst, H.A., "Bi-CGSTAB: a Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems," *SIAM J. Sci. Stat. Comp.*, vol. 13, pp. 631–644, 1992.

[41] Freund, R.W., and Nachigal, N.M., "QMR: a Quasi-Minimal Residual Method for Non-Hermitian Linear Systems," *Numer. Math.*, vol. 60, pp. 315–339, 1991.

[42] Dutto, L.C., "On Iterative Methods for Solving Linear Systems of Equations," *Revue européenne des éléments finis*, vol. 2, 1993.

[43] Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and van der Vorst, H., "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods," *SIAM*, 1993.

[44] Wong, Y.S., and Hafez, M., "Application of Conjugate Gradient Methods to Transonic Finite Difference and Finite Element Calculations." AIAA Paper 81-1032, June 1981.

[45] Wong, Y.S., "Calculations of Transonic Potential Flows by a Parameter Free Procedure." AIAA Paper 83-1886, June 1983.

[46] Prince, T.C., "Conjugate Gradient Methods for Solution of Finite Element and Finite Difference Flow Problems." AIAA Paper 83-1923, June 1983.

[47] Wigton, L.B., Yu, N.J., and Young, D.P., "GMRES Acceleration of Computational Fluid Dynamics Codes." AIAA Paper 85-1494, July 1985.

[48] Jameson, A., "Transonic Flow Calculations," Tech. Rep. MAE 1651, Department of Mechanical and Aerospace Engineering, Princeton University, 1983.

[49] Venkatakrishnan, V., "Preconditioned Conjugate Gradient Methods for the Compressible Navier-Stokes Equations." AIAA Paper 90-00586, January 1990.

[50] Venkatakrishnan, V., and Mavriplis, D.J., "Implicit Solvers for Unstructured Meshes." AIAA Paper 91-1537, June 1991.

[51] Venkatakrishnan, V., "Implicit Schemes and Parallel Computing in Unstructured Grid CFD," ICASE report 95-28 CR-195071, NASA, 1995.

[52] Dutto, L.C., "Etude de préconditionnements pour la résolution, par la méthode des éléments finis, des équations de Navier-Stokes pour un fluide compressible." Thèse de Doctorat, Université Pierre et Marie Curie, Paris VI, France, 1990.

[53] Dutto, L.C., "The Effect of Ordering on Preconditioned GMRES Algorithm, for Solving the Compressible Navier-Stokes Equations," *International Journal for Numerical Methods in Engineering*, vol. 36, pp. 457–497, 1993.

[54] Dutto, L.C., Habashi, W.G., and Fortin, M., "Parallelizable Block Diagonal Preconditioners for the Compressible Navier-Stokes Equations," *Comput. Methods Appl. Mech. Engrg.*, vol. 117, pp. 15–47, 1994.

[55] Dutto, L.C., Habashi, W.G., and Fortin, M., "An Algebraic Multilevel Parallelizable Preconditioner for Large-Scale CFD Problems," *Paper to appear in Comput. Methods Appl. Mech. Engrg, Special issue in honor of Tinsley Oden, Edited by J.N. Reddy and Leszek Demkowicz*, 1997.

[56] Johan, Z., Hughes, T.J.R., and Shakib, F., "A Globally Convergent Matrix-Free Algorithm for Implicit Time-Marching Schemes Arising in Finite Element Analysis in Fuids," *Comput. Methods Appl. Mech. Engrg.*, vol. 87, pp. 281–304, 1991.

[57] Ajmani, K., "Preconditioned Conjugate Gradient Methods for the Navier-Stokes Equations." Ph.D. thesis, Virginia Polytechnic Institute and State University, 1991.

[58] Ajmani, K., Ng, W., Liou, M., "Preconditioned Conjugate Gradien Methods for the Navier-Stokes Equations," *J. Comp. Phys.*, vol. 110, pp. 68–81, 1994.

[59] Ajmani, K., and Liou, M., "Implicit Conjugate-Gradient Solvers On Distributed-Memory Architectures." AIAA Paper 95-1695, 1995.

[60] Habashi, W.G., Robichaud, M., Nguyen,V-N., Ghaly, W.S., Fortin, M., and Liu, J.W.H., "Large-Scale Computational Fluid Dynamics by the Finite Element Method." AIAA Paper 91-0120, January 1991.

[61] Hixon, R., and Sankar, L.N., "Application of a Generalized Minimal Residual Method to 2D Unsteady Flows." AIAA Paper 92-0422, January 1992.

[62] Orkwis, P.D., "Comparison of Newton's and Quasi-Newton's Method Solvers for the Navier-Stokes Equations," *AIAA J.*, vol. 31, pp. 832–836, May 1993.

[63] Lin, H., Yang, D.Y., Chieng, Ch., "Variant Bi-Conjugate Gradient Methods for the Compressible Navier-Stokes Solver with at Two-equation Model of Turbulence." AIAA Paper 93-3316, June 1993.

[64] Knoll, D.A., and McHugh, P.R., "Inexact Newton's Method Solutions to the Incompressible Navier-Stokes and Energy Equations Using Standard and Matrix-Free Implementations." AIAA Paper 93-3332, June 1993.

[65] McHugh, P.R., and Knoll, D.A., "Comparison of Standard and Matrix-Free Implementations of Several Newton-Krylov Solvers," *AIAA J.*, vol. 12, pp. 2394–2400, December 1994.

[66] Degrez, G., and Issman, E., "Acceleration of Compressible Flow Solvers by Krylov Subspace Methods." Lecture Notes For The Von Karman Institute For Fluid Dynamics Lecture Series 1994-05, 1994.

[67] Degrez, G., and Issman, E., "Multilevel Newton Iterative Solution of Euler/Navier-Stokes Equations on Unstructured Grids." AIAA Paper 97-2132, June 1997.

[68] Hager, J.O., and Lee, K.D., "The Behavior of Some Solution Acceleration Techniques in CFD." AIAA Paper 94-0175, January 1994.

[69] Luo, H., Baum, J.D., Löhner, R., and Cabello, J., "Implicit Schemes and Boundary Conditions for Compressible Flows on Unstructured Meshes." AIAA Paper 94-0816, January 1994.

[70] Jorgenson, P.C.E., and Pletcher, R.H., "An Implicit Numerical Scheme for the Simulation of Internal Viscous Flows on Unstructured Grids." AIAA Paper 94-0306, January 1994.

[71] Rogers, S.E., "A Comparison of Implicit Schemes for the Incompressible Navier-Stokes Equations with Artificial Compressibility." AIAA Paper 95-0567, June 1995.

[72] Barth, T.J., and Linton, S.W., "An unstructured Mesh Newton Solver for Compressible Fluid Flow and Its Parallel Implementation." AIAA Paper 95-0221, January 1995.

[73] Forsyth, P.A., and Jiang, H., "Iterative Methods for Full Newton Solution of the Euler Equations." Sixth International Symposium on Computational Fluid Dynamics, pp. 318–323, September 1995.

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

[75] Cai, X., Keyes, D.E., and Venkatakrishnan, V., "Newton-Krylov-Schwarz: an Implicit Solver for CFD," Tech. Rep. No. 95-87, ICASE, December 1995.

[76] McHugh, P.R., Knoll, D.A., Keyes, D.E., "Schwarz-Preconditioned Newton-Krylov Algorithm for Low Speed Combustion Problems." AIAA Paper 96-0911, January 1996.

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

[78] Anderson, W.K., Rausch, R.D., and Bonhaus, D.L., "Implicit/Multigrid Algorithms for Incompressible Turbulent Flows on Unstructured Grids." AIAA Paper 95-1740, June 1995.

[79] Choquet, R., Leyland, P., and Tefy, T., "GMRES Acceleration of Iterative Implicit Finite Element Solvers for Compressible Euler and Navier-Stokes Equations," *International Journal for Numerical Methods in Fluids*, vol. 20, pp. 957–967, 1995.

[80] Choquet, R., "A Matrix-free Preconditioner Applied to CFD," Tech. Rep. No. 2605, INRIA, Institut National de Recherche en Informatique et en Automatique, June 1995.

[81] Delanaye, M., Geuzaine, Ph., Essers, J.A., and Rogiest, P., "A Second-Order Finite-Volume Scheme Solving Euler and Navier-Stokes Equations on Unstructured Grids." AGARD 77th Fluid Dynamics Panel Symposium on Progress and Challenges in CFD Methods and Algorithms, Seville, Spain, 2-5 October, 1995.

[82] Delanaye, M., Rogiest, P., and Essers, J., "Implicit Quadratic Reconstruction Finite Volume Scheme for Compressible Flows." Third ECCOMAS CFD Conference, Paris.

[83] Ollivier-Gooch, C.F., "Towards Problem Independent Multigrid Convergence Rates For Unstructured Mesh Methods I: Inviscid and Laminar Viscous Flows." In Proceedings of the Sixth International Symposium on Computational Fluid Dynamics, September 1995.

[84] Blanco, M., and Zingg, D.W., "A Fast Solver for the Euler Equations on Unstructured Grids Using a Newton-GMRES Method." AIAA Paper 97-0331, January 1997.

[85] Mavriplis, D.J., "Multigrid Strategies for Viscous Flow Solvers on Anisotropic Unstructured Meshes." AIAA Paper 97-1952, June 1997.

[86] Baldwin, B.S., and Lomax, H., "Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows." AIAA Paper 78-257, June 1978.

[87] Cebeci, T., "Calculation of Compressible Turbulence Boundary Layers with Hear and Mass Transfer." AIAA Paper 70-741, June 1970.

[88] Salas, M., Jameson, A., and Melnik, R., "A comparative Study of the Nonuniqueness Problem of the Potential Equation." AIAA Paper 83-1888, June 1983.

[89] Zingg, D.W., "Grid Studies for Thin-Layer Navier-Stokes Computations of Airfoil Flowfields," *AIAA J.*, vol. 30, pp. 2561–2564, October 1992.

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

[91] Hall, M.G., "On the Reduction of Artificial Viscosity in Viscous flow Solutions." Frontiers of Computational Fluid Dynamics, D.A. Caughey and M.M. Hafez, eds., Wiley, 1994.

[92] Frew, K., and Zingg, D.W., "On Artificial Dissipation Models for Viscous Airfoil Computations." AIAA Paper 96-1970, June 1996.

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

[94] Pulliam, T.H., "Implicit Methods In CFD." The Institute of Mathematics and Its Publications Conference Series, Proceeding of the ICFD 1988 Conference on Numerical Methods for Fluid Dynamics, June 1988.

[95] Saleem, M., Pulliam, T.H., and Cheer, A.Y., "Acceleration of Convergence and Spectrum Transformation of Implicit Finite Difference Operators Associated with Navier-Stokes Equations," *J. Comp. Phys.*, no. 104, pp. 1–13, 1993.

[96] Mulder, W.A., and Van Leer, B., "Implicit Upwind Methods for the Euler Equations." AIAA Paper 83-1930, June 1983.

[97] Pueyo, A. and Zingg, D.W., "Airfoil Computations Using a Newton-GMRES Method." CFD96, Proceedings of the Fourth Annual Conference of the CFD Society of Canada, June 1996.

[98] Ajmani, K., Ng, W., and Liou, M., "Preconditioned Conjugate Gradient Methods for the Navier-Stokes Equations," *J. Comp. Phys.*, vol. 110, pp. 68–81, 1994.

[99] Page, M., "Étude de Méthodes Itératives pour la Résolution des Équations de Navier-Stokes." Thèse de doctorat (Ph.D.), École Polytechnique de Montréal, Québec, Canada, June 1995.

[100] Saad, Y., "Krylov Subspace Techniques, Conjugate Gradients, Preconditioning and Sparse Matrix Solvers." 1994-05, von Karman Institute for Fluid Dynamics, March 1994.

[101] Duff, I.S., and Meurant, G.A., "The Effect of Ordering on Preconditioned Conjugate Gradients," *BIT*, vol. 29, pp. 635–657, 1989.

[102] Chow, E., and Saad, Y., "Experimental Study of ILU Preconditioners for Indefinite Matrices," Tech. Rep. UMSI 97/95, University of Minnesota Supercomputing Institute Research, June 1997.

[103] Saad, Y., "A Flexible Inner-outer preconditioned GMRES Algorithm," *SIAM J. Sci. Stat. Comp.*, vol. 14, pp. 461–469, 1993.

[104] Van der Vorst, H.A., and Vuik, C., "GMRESR: a Family of Nested GMRES Methods," *Numerical Linear Algebra with Applications*, vol. 1, pp. 1–7, 1993.

[105] Axelsson, O., "A survey of preconditioned iterative methods for linear systems of algebraic equations," *BIT*, vol. 25, pp. 166–187, 1985.

[106] Saad, Y., "Preconditioning techniques for indefinite and nonsymmetric linear systems," *Journal of Computational and Applied Mathematics*, vol. 24, pp. 89–105, 1988.

[107] Elman, H.C., "A Stability Analysis of Incomplete LU Factorizations," *Math. Comp.*, no. 47, pp. 191–217, 1986.

[108] Bruaset,, A.M., Tveito, A., and Winther, R., "On the Stability of Relaxed Incomplete LU Factorizations," *Math. Comp.*, no. 54, pp. 701–719, 1990.

[109] Saad, Y., "ILUT: A Dual Threshold Incomplete LU Factorization," Tech. Rep. UMSI 92/38, University of Minnesota Supercomputer Institute, March 1992.

[110] Saad, Y., "SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations, version 2," June 1994.

[111] Cuthill, E.H., and McKee, J.M., "Reducing the bandwidth of sparse symmetric matrices." Proc. 24th National Conference of the Association for Computing Machinery, Brondon Press.

[112] Martin, G., "Méthodes des préconditionnement par factorisation incomplète." Mémoire de Maîtrise, Université de Laval, Québec, Canada, 1991.

[113] George, A., and Liu, J.W.H., "The Evolution of the Minimum Degree Ordering Algorithm," *SIAM Rev.*, no. 31, pp. 1–19, 1989.

[114] Ollivier-Gooch, C.F., "Improved Asymptotic Convergence Rates For An Unstructured Multigrid Solver." submitted to *AIAA J.*, 1996.

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

[116] Brown, P.N., and Saad, Y., "Hybrid Krylov methods for nonlinear systems of equations," *SIAM J. Sci. Stat. Comput.*, no. 11, pp. 450–481, 1990.

[117] Dembo, R.S., and Steihaug, T., "Truncated Newton Algorithms for Large-Scale Optimizations," *Math. Programming*, vol. 26, pp. 190–212, 1983.

[118] Cai, X., Gropp, W.D., Keyes, D.E., and Tidriri, M.D., "Newton-Krylov-Schwarz methods in CFD." Proceedings of the International Workshop on the Navier-Stokes Equations, R. Rannacher, ed., Notes in Numerical Fluid Mechanics, Braunschweig, 1994.

[119] Pueyo, A., and Zingg, D.W., "Progress in Newton-Krylov Methods for Aerodynamic Calculations." AIAA Paper 97-0877, January 1997.

[120] Pueyo, A., and Zingg, D.W., "An Efficient Newton-GMRES Solver for Aerodynamic Computations." AIAA Paper 97-1955, June 1997.

[121] Godin, P., Zingg, D.W., and Nelson, T.E., "High-Lift Aerodynamic Computations with One- and Two-Equation Turbulence Models," *AIAA J.*, vol. 35, pp. 16–32, February 1997.

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

[123] Menter, F.R., "Zonal Two-Equation $k-\omega$ Models for Aerodynamic Flows." AIAA Paper 93-2906, July 1993.

[124] Nelson, T. E., "Numerical Solution of the Navier-Stokes Equations for High-Lift Airfoil Configurations." Ph.D. thesis, Institute for Aerospace Studies, University of Toronto, 1994.

[125] Dutto, L.C., Habashi, W.G., Robichaud, M.P., and Fortin, M., "A Method for Finite Element Parallel Viscous Compressible Flow Calculations," *International Journal for Numerical Methods in Fluids*, vol. 19, pp. 275–294, 1994.

Appendix A

Flux Jacobians for the thin-layer Navier-Stokes equations

The inviscid flux Jacobians are given by (from Ref. [16]):

$$\begin{bmatrix} 0 & \kappa_x & \kappa_y & 0 \\ -u\theta + \kappa_x\phi^2 & \theta - (\gamma-2)\kappa_x u & \kappa_y u - (\gamma-1)\kappa_x v & (\gamma-1)\kappa_x \\ -v\theta + \kappa_y\phi^2 & \kappa_x v - (\gamma-1)\kappa_y u & \theta - (\gamma-2)\kappa_y v & (\gamma-1)\kappa_y \\ \theta\left(\phi^2 - a_1\right) & \kappa_x a_1 - (\gamma-1)u\theta & \kappa_y a_1 - (\gamma-1)v\theta & \gamma\theta \end{bmatrix} \tag{A.1}$$

where $\kappa = \xi$ for $\hat{E}$ and $\kappa = \eta$ for $\hat{F}$. The other entries are

$$\begin{aligned} a_1 &= \gamma\frac{e}{\rho} - \phi^2 \\ \theta &= \kappa_x u + \kappa_y v \\ \phi^2 &= \tfrac{1}{2}(\gamma-1)\left(u^2+v^2\right) \end{aligned}$$

The viscous flux Jacobian is

$$\frac{\partial\hat{S}}{\partial\hat{Q}} = J^{-1}\begin{bmatrix} 0 & 0 & 0 & 0 \\ m_{21} & \alpha_1\partial_\eta\left(\rho^{-1}\right) & \alpha_2\partial_\eta\left(\rho^{-1}\right) & 0 \\ m_{31} & \alpha_2\partial_\eta\left(\rho^{-1}\right) & \alpha_3\partial_\eta\left(\rho^{-1}\right) & 0 \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \tag{A.2}$$

with

$$
\begin{aligned}
m_{21} &= -\alpha_1 \partial_\eta \left(u/\rho\right) - \alpha_2 \partial_\eta \left(v/\rho\right) \\
m_{31} &= -\alpha_2 \partial_\eta \left(u/\rho\right) - \alpha_3 \partial_\eta \left(v/\rho\right) \\
m_{41} &= \alpha_4 \partial_\eta \left[-\left(e/\rho^2\right) + \left(u^2+v^2\right)/\rho\right] - \alpha_1 \partial_\eta \left(u^2/\rho\right) \\
&\qquad\qquad -2\alpha_2 \partial_\eta \left(uv/\rho\right) - \alpha_3 \partial_\eta \left(v^2/\rho\right) \\
m_{42} &= -\alpha_4 \partial_\eta \left(u/\rho\right) - m_{21} \\
m_{43} &= -\alpha_4 \partial_\eta \left(v/\rho\right) - m_{31} \\
m_{44} &= \alpha_4 \partial_\eta \left(\rho^{-1}\right)
\end{aligned}
$$

and the α parameters given by

$$
\begin{aligned}
\alpha_1 &= \left(\mu+\mu_t\right)\left[(4/3)\eta_x^2+\eta_y^2\right] \\
\alpha_2 &= \tfrac{1}{3}\left(\mu+\mu_t\right)\eta_x\eta_y \\
\alpha_3 &= \left(\mu+\mu_t\right)\left[\eta_x^2+(4/3)\eta_y^2\right] \\
\alpha_4 &= \gamma\left(\mu\mathcal{P}r^{-1}+\mu_t\mathcal{P}r_t^{-1}\right)\left(\eta_x^2+\eta_y^2\right)
\end{aligned}
$$

Appendix B

Flow solution for the eight cases

This appendix contains some flowfield results for the eight cases. In Table B.1 we show lift, drag and pitching moment coefficients. Figures B.1 to B.8 show the Mach number contours and the pressure coefficient distribution over the airfoil's surface.

case	airfoil	Ma	α	Re	C_l	C_d	C_m
1	NACA 0012	0.63	2.00	invisc	0.33760	0.000217	-0.001773
2	NACA 0012	0.80	1.25	invisc	0.37598	0.023825	-0.043987
3	NACA 0012	0.80	5.00	5.00e2	0.24444	0.214965	0.013698
4	NACA 0012	0.30	0.00	2.88e6	-0.00033	0.008991	0.000044
5	NACA 0012	0.30	6.00	2.88e6	0.67951	0.011759	0.007525
6	NACA 0012	0.70	1.49	9.00e6	0.25194	0.008816	0.004662
7	NACA 0012	0.16	12.00	2.88e6	1.28294	0.019518	0.013881
8	RAE 2822	0.729	2.31	6.50e6	0.77305	0.015351	-0.098772

Table B.1: Lift, drag and pitching moment coefficients for the eight cases described in Table 4.1.

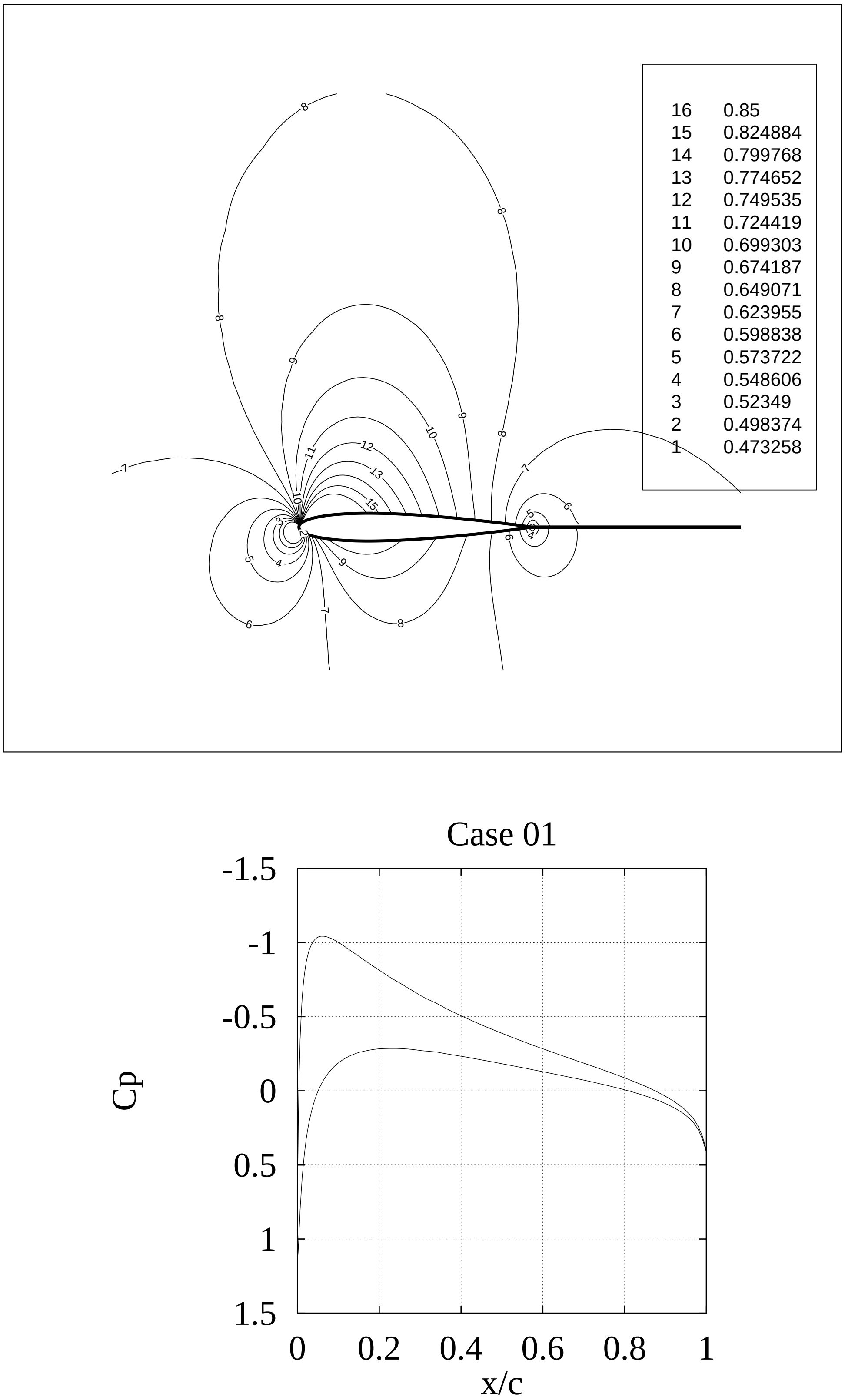


Figure B.1: Mach contours and C_p plot for case 1, described in Table 4.1.

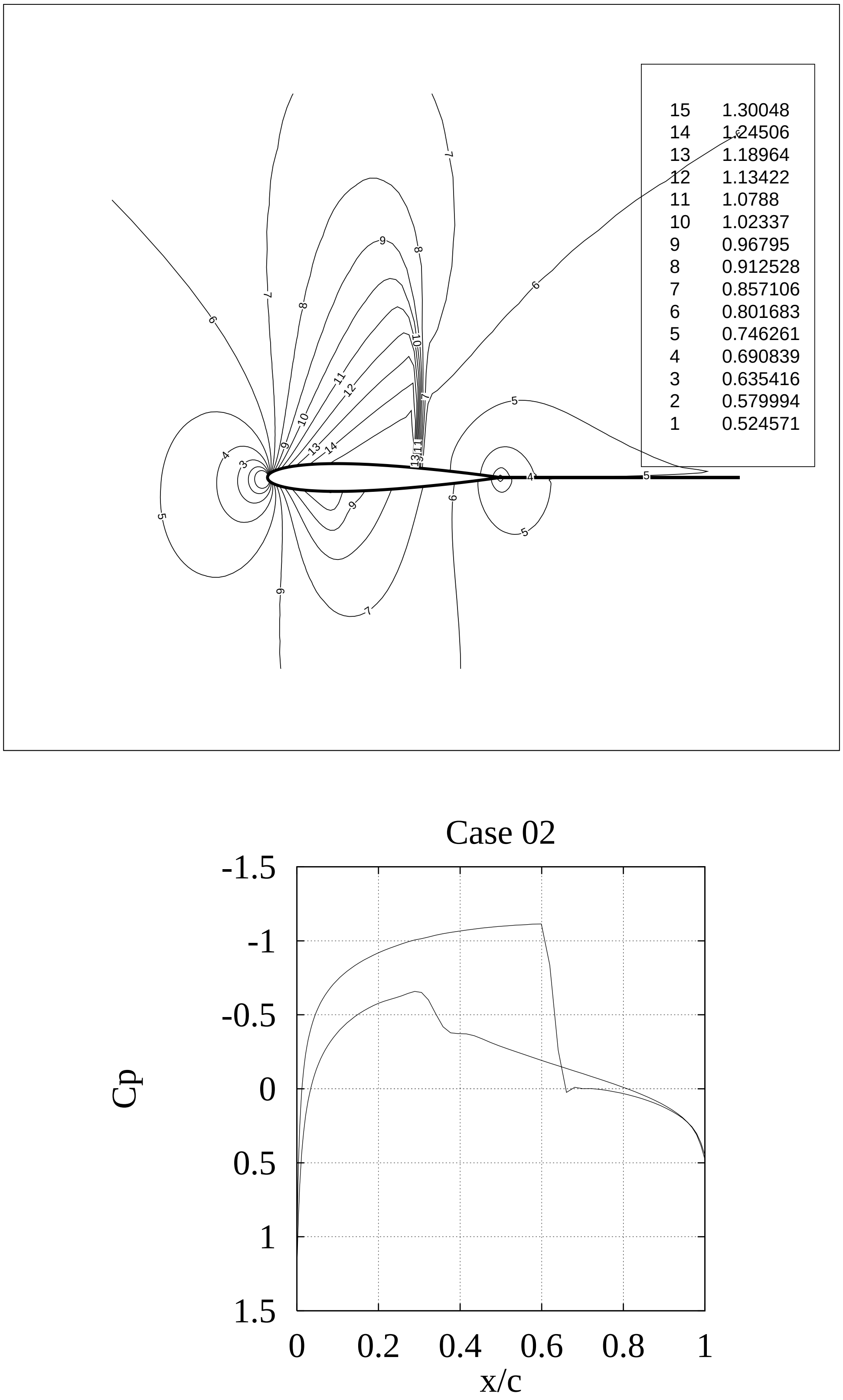
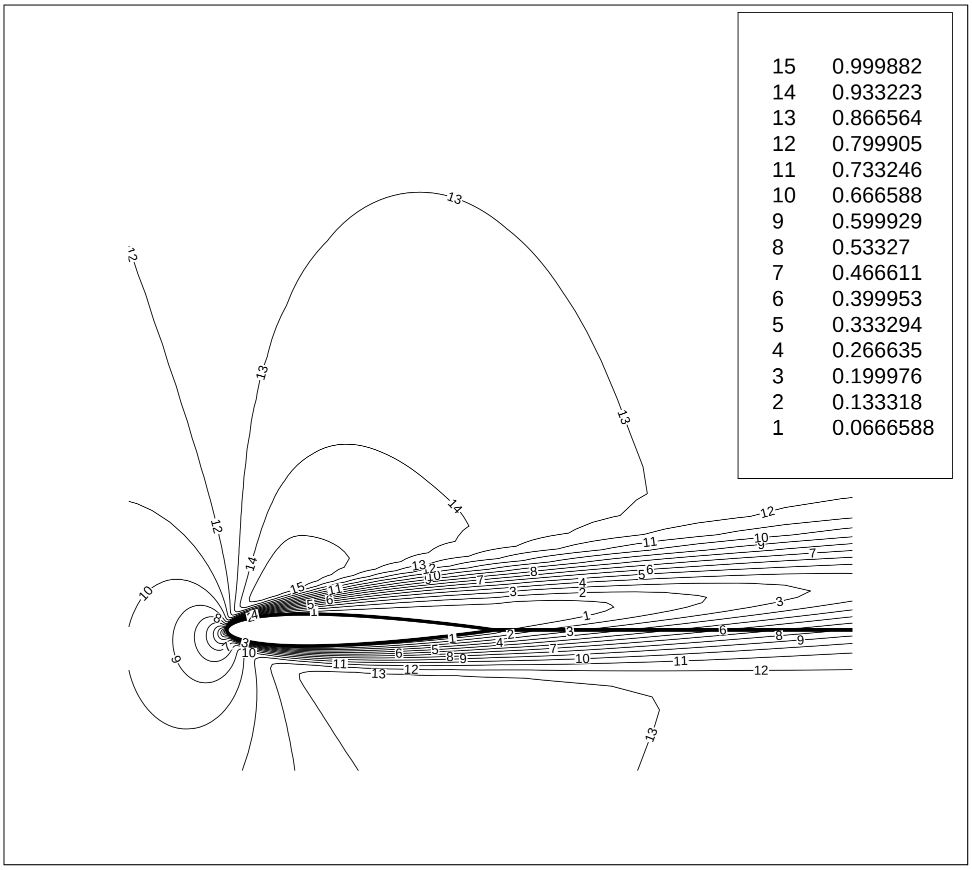


Figure B.2: Mach contours and C_p plot for case 2, described in Table 4.1.



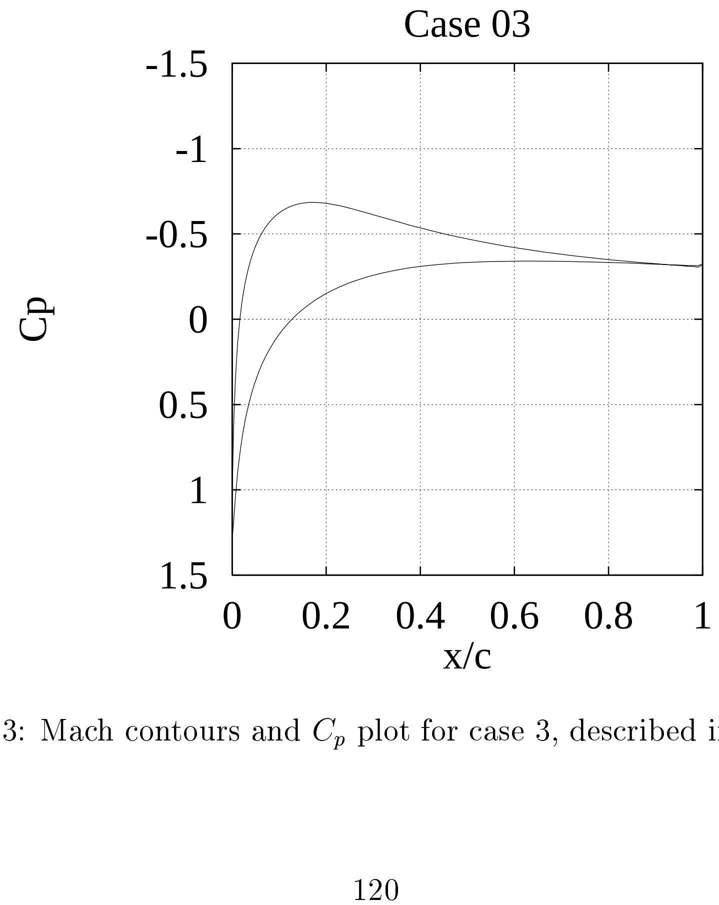
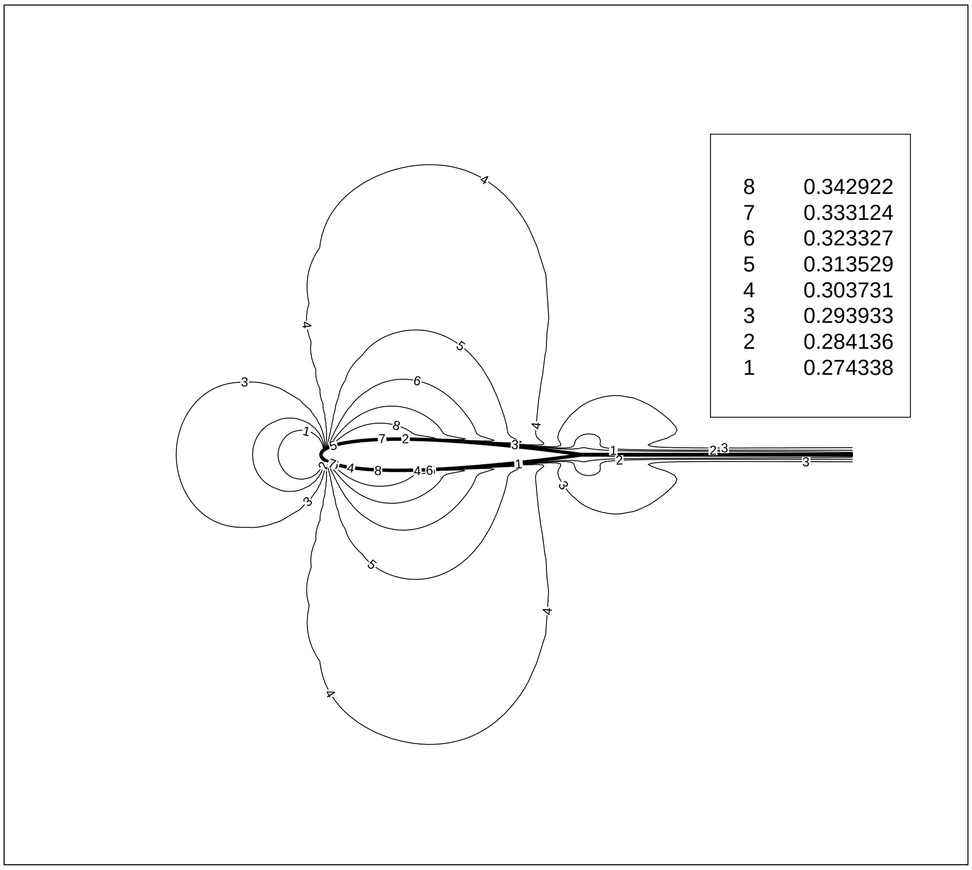


Figure B.3: Mach contours and C_p plot for case 3, described in Table 4.1.



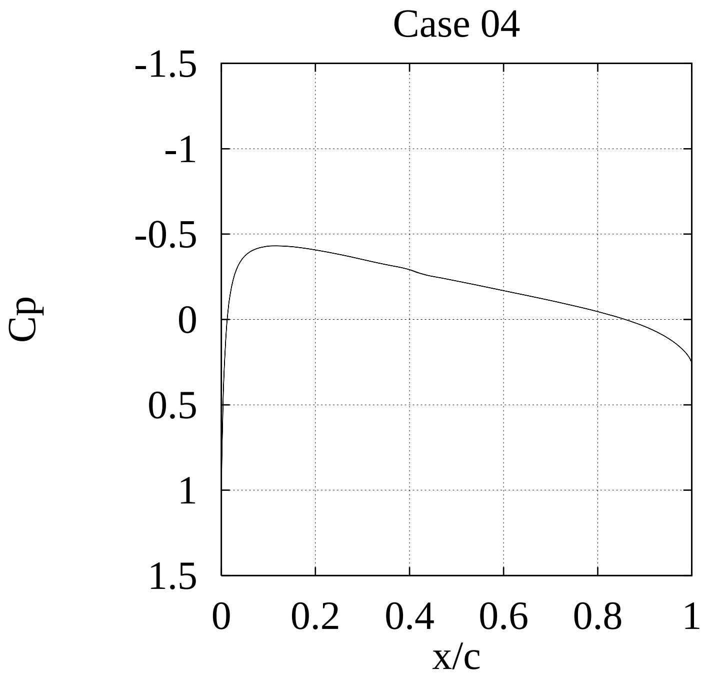
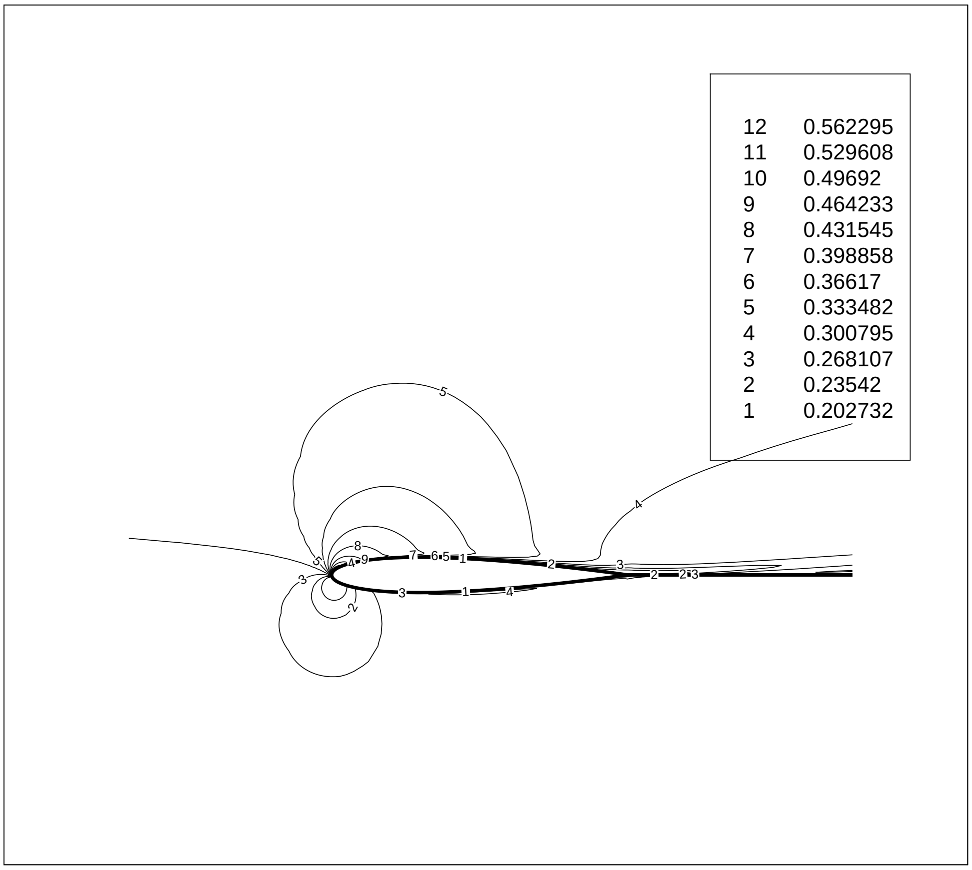


Figure B.4: Mach contours and C_p plot for case 4, described in Table 4.1.



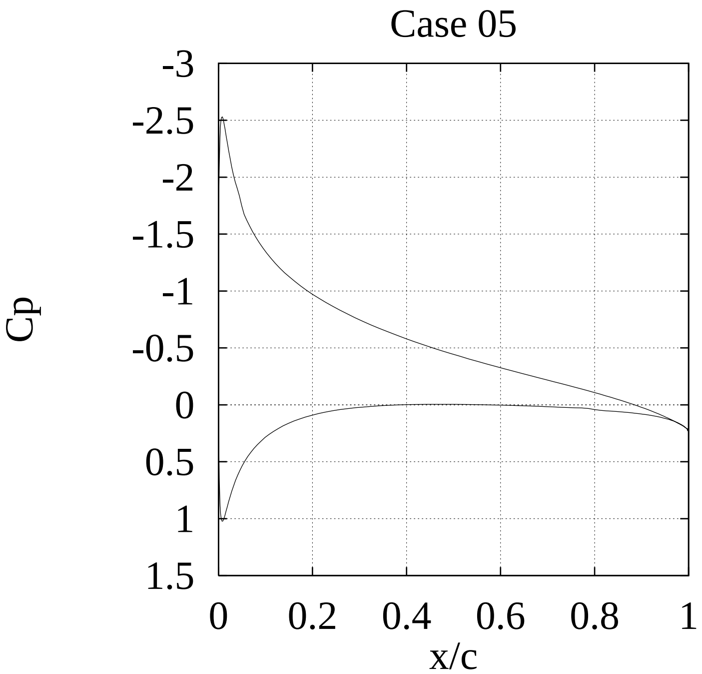
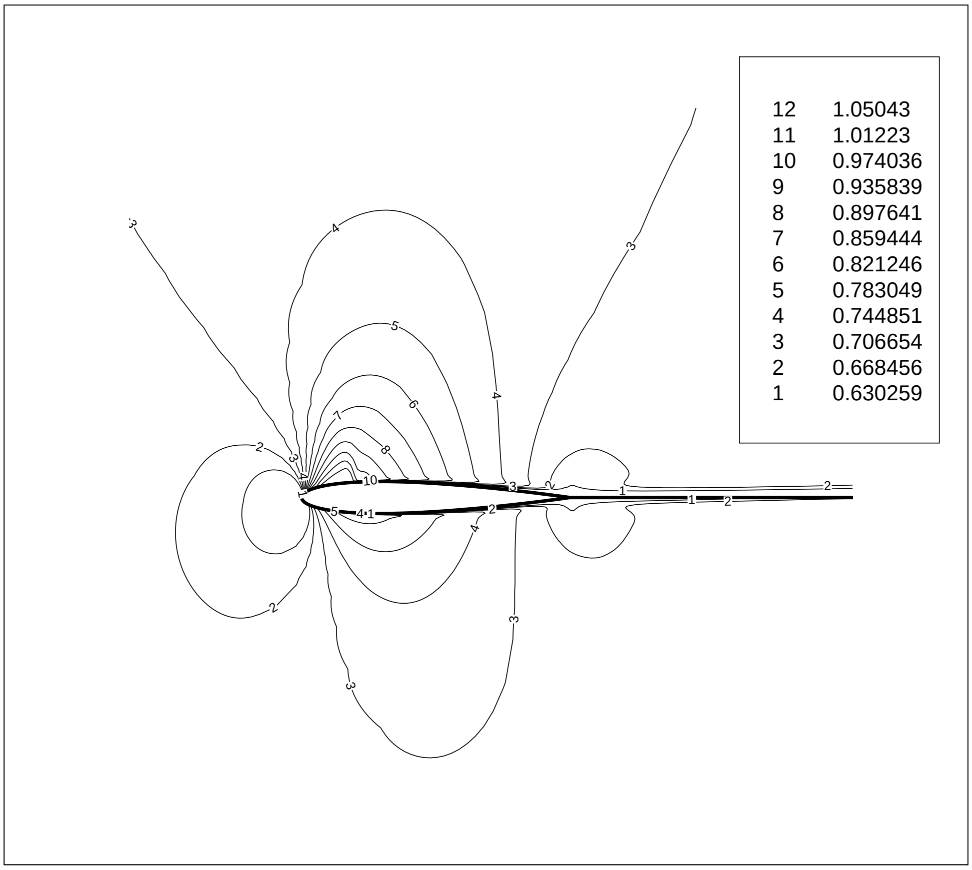


Figure B.5: Mach contours and C_p plot for case 5, described in Table 4.1.



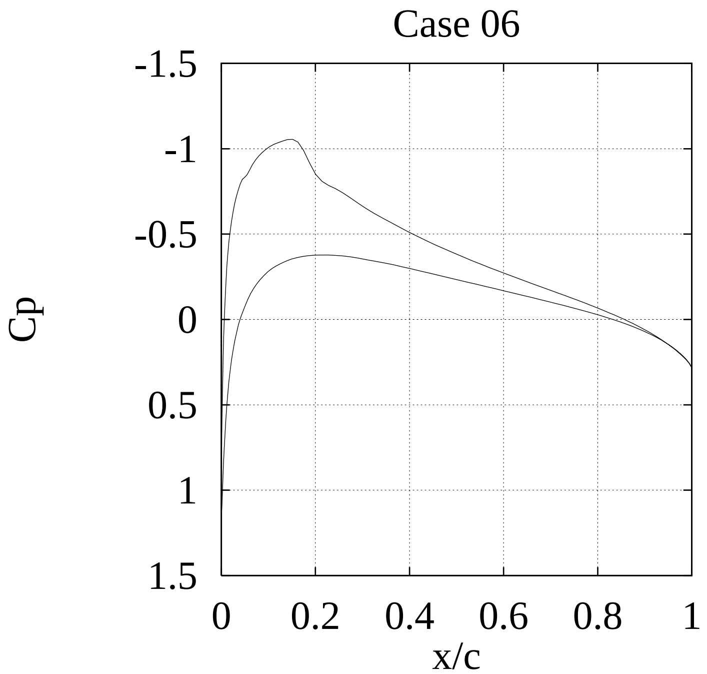
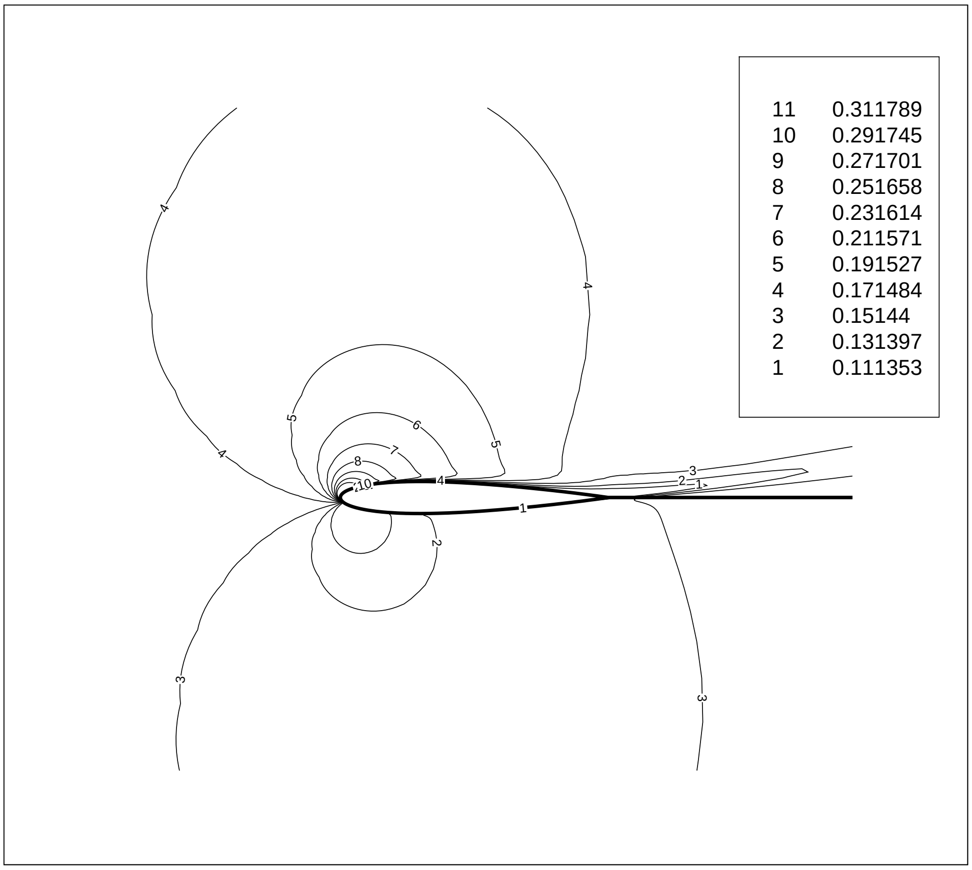


Figure B.6: Mach contours and C_p plot for case 6, described in Table 4.1.



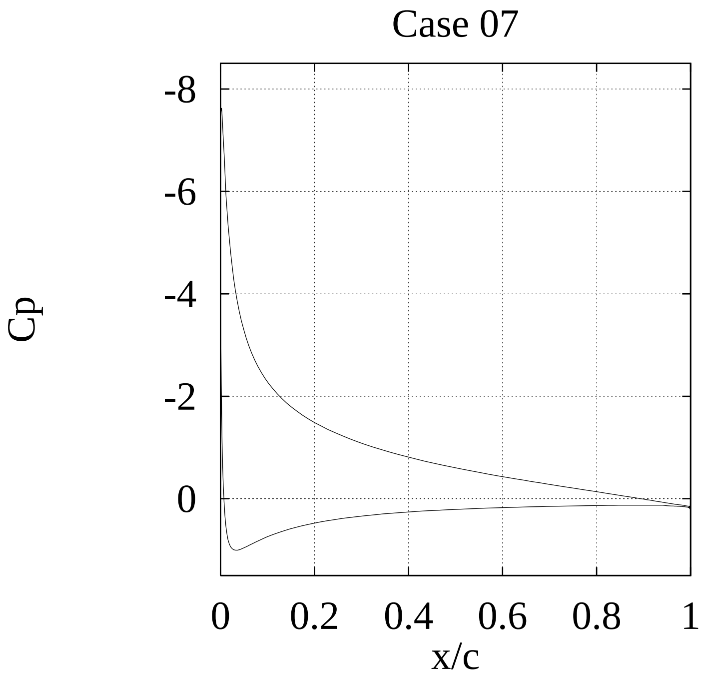


Figure B.7: Mach contours and C_p plot for case 7, described in Table 4.1.

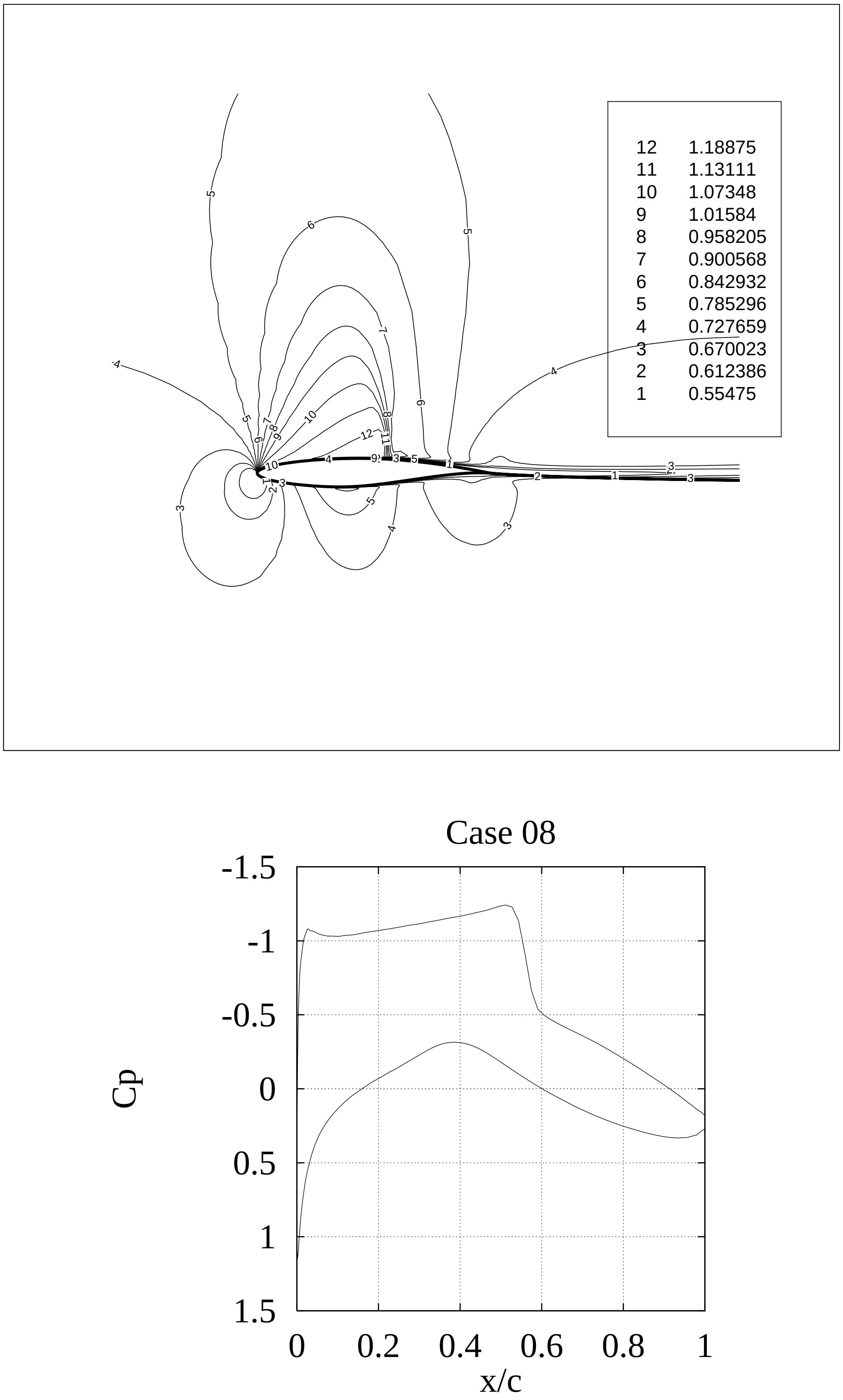


Figure B.8: Mach contours and C_p plot for case 8, described in Table 4.1.