To the University of Wyoming:

The members of the Committee approve the dissertation of Emmett M. Padway presented on September 11, 2020.

Professor Dimitri J. Mavriplis, Chairperson

Professor Victor Ginting, Outside Member

Professor Michael Stoellinger

Professor Ray Fertig III

Professor Siva Nadarajah, External Examiner, McGill University

APPROVED:

Professor Carl Frick, Head, Department of Mechanical Engineering

Professor Cameron H. G. Wright, Interim Dean, College of Engineering and Applied Science

Padway, Emmett M., Tangent and Adjoint Problems in Partially Converged Flows, Ph.D., Department of Mechanical Engineering, September, 2020.

Adjoint methods see widespread use in computational fluid dynamics (CFD) in two important domains of the field: shape optimization and output-based refinement. CFD has become more widespread as computing power has become more available and algorithms have become both more advanced and efficient. This allows for the use of cheap, low-fidelity methods in the conceptual design stage, and more accurate high-fidelity methods later in the detailed design stage. The high fidelity methods coupled with optimization toolboxes for automated design have a significant place in the detailed design process, mainly to better inform the use of expensive wind tunnel testing. Furthermore, the use of adjoint methods in shape optimization allows for cheaper sensitivity evaluation for gradient-based design approaches, allowing for scaling independent of the number of design variables. As a result highly refined/parameterized design optimizations are possible with negligible cost increases.

Adjoint methods are also utilized in the context of adaptive mesh refinement as they can be used to create output-based error estimates. Adjoint-based methods form an error estimate that calculates error in the output-of-interest, and are desirable for their efficacy in tailoring/adapting meshes for maximum accuracy in the engineering quantity of interest. This can allow for coarsening meshes in areas of little engineering interest and refining in areas of high interest, thus resulting in a maximally useful mesh for engineering quantities with a given number of degrees of freedom.

The adjoint systems used in these analyses require that the nonlinear problem be solved to machine precision –that the discretized form of the governing equations be satisfied. However, as the field has attempted more difficult simulations either due to the accuracy of spatial discretization of the simulation, the geometry of the model, or the increasing push to unsteady flow, it has become increasingly difficult to satisfy this constraint. Some members of the field have demonstrated that this can lead to difficult to converge adjoint problems that provide sensitivities highly dependent on the state at which the simulation is terminated. Additionally, this can lead to issues with the error estimation process by returning inaccurate error estimates and poor refinement patterns.

This work develops a novel methodology for adjoint based optimization and error estimation for unconverged simulations through linearization of the nonlinear solution process. It contains results for various different nonlinear solvers and applications of the adjoint system to design optimization and error estimation. As CFD has been used for more complex simulations (unsteady, high-order, complex geometry) the ability to get a useful adjoint solution that is guaranteed to not diverge is tremendously desirable. The goal of this work is to show a hierarchy of different linearizations with different approximations and apply them to these partially converged problems. This work presents tangent and adjoint formulations for each of the various nonlinear solution strategies used in the test cases, including the solution strategies required to avoid flow divergence for problems with strong shocks. It also shows the varying linear solver technologies, used

in the nonlinear process, the mesh deformation process, and the tangent and adjoint processes. Included also are three different error estimation techniques for the steady state adjoint and their analogues for the pseudo-time accurate adjoint. These error estimation techniques are used to drive mesh adaptation and the algorithms for the adaptive mesh refinement package are included as well. A detailed error analysis of approximate linearization of the nonlinear solution process for both the tangent and adjoint modes is shown as well. The end result is a series of methods and algorithms guaranteed to provide either adjoint based sensitivities or adjoint based error estimates that are insensitive to the specific state of termination of the solution process and can be useful for simulations which do not fully converge, where the calculation of accurate sensitivities has been a major stumbling block to date.

TANGENT AND ADJOINT PROBLEMS IN PARTIALLY CONVERGED FLOWS

by

Emmett M. Padway, B.S.M.E.

A dissertation submitted to the
Department of Mechanical Engineering
and the
University of Wyoming
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY
in
MECHANICAL ENGINEERING

Laramie, Wyoming
September 2020

To my beloved partner Larisa, my family, friends, and the great outdoors. I learned more here than I ever thought possible and enjoyed my time here beyond my wildest dreams.

Contents

List of Figures

List of Tables

Acknowledgments

This thesis would not have been possible without the support and advice of my committee members, colleagues, friends, and family. First, my sincerest gratitude to my advisor, Professor Dimitri Mavriplis, who gave me one of the most interesting projects I ever could have dreamed of and gave me the freedom to pursue it. I tried idea after idea, with his advice to help me discard the worst ones and pursue the better ones that make up this dissertation. He was an excellent resource and very patient throughout our exploration of this topic. He encouraged me to start from scratch and I got the education in CFD that I had hoped for when I decided to come to the University of Wyoming. My time here has been incredibly rewarding and educational, and that is largely due to his influence and perspective on research. My thanks to Professor Siva Nadarajah who gave me my introduction to fluid mechanics and CFD through both classes and his time advising me when I worked in his lab; he was an excellent advisor and teacher. I asked him where to go to obtain my PhD; I wanted to go somewhere I could start a project from scratch, work on adjoint method development, and be close to a beach – he suggested Wyoming (2/3 isn't bad, and he understood the primacy of the research for the path ahead). I am grateful to Professor Stoellinger for his enlightening course on turbulence, and his discussions on it which I hope to use in my research going forward. I would also like to thank Professor Fertig with whom I took two excellent finite element courses and enjoyed discussing hunting with throughout my time here. I would like to thank Professor Ginting as well for his feedback, I had heard excellent things about him as a professor, and am regretful I never had the opportunity to take a class with him. I am grateful to the members of my committee who read my dissertation and provided valuable feedback on it as well as instruction through the various classes I took throughout my time here.

I am very appreciative of the more experienced researchers who helped me tremendously when I first arrived: Mike, Behzad, Andrew, and Enrico; as well as those who were newer and I shared many memories with: Sudeh and Donya. I'd like to thank my friends (Ayoub, Cory, Jan and Jan) and mentors (Mike Aftosmis, Marian Nemec and Marsha Berger) at NASA Ames who made both my summers there so educational and worthwhile. My time at Ames directed me toward an interest in error estimation and showed me an example of experimental attitude necessary to do high quality research while having a large user base for a production code. I also appreciated their emphasis on the importance of communicating the research in a digestible manner to me and the scientific community at large. I'd also like to thank Mike Park at NASA Langley for

spending time helping me work with his Refine package, which made the error estimation portion of this work feasible. This work was made possible by NASA through NASA Grant NNX16AT23H and the NASA Graduate Aeronautics Scholars Program, and by the Dassault Systémes U.S. Foundation through grant for: "Inter-University Collaborative Design Project Using Multidisciplinary Design Optimization Technologies". Computing time was provided by NASA Advanced Supercomputing (NAS) on the Pleiades supercomputer and by the Advanced Research Computing Center (ARCC) on the Teton supercomputer.

I am grateful to all my friends and family, especially my parents, for their support throughout this challenging and lengthy process. I am thankful for my family's support and for putting me in position to pursue this graduate education. I am also so appreciative for my friends those from back home who came out to visit or just kept me company over the phone, and new ones I made throughout my time here. Last, but most certainly not least, thank you to my partner Larisa, who has planned such wonderful excursions and activities for us throughout this journey, who supported me ceaselessly, and who encouraged me when I had doubts as to the feasibility of the project.

Emmett M. Padway

University of Wyoming
September 2020

Chapter 1

Background and Motivation

The design problem in the field of partial differential equation (PDE) constrained optimization begins from an objective function L to be minimized, subject to a constraint $R = 0$. In the field of the computational fluid dynamics (CFD), L is often an integrated quantity, such as lift or drag, and it is expressed as $L = L(U(D), D)$, where U is the conservative variable vector, and D is the design variables. The constraint that the discretized form of the governing equations is satisfied is written as $R(U(D), D) = 0$, and indicates that the discretized form of the PDE has been solved to machine zero. To obtain the sensitivity vectors ($\frac{dL}{dD}$) to drive the optimization one may use the tangent method or the adjoint method. Beginning from $L = L(U(D), D)$, the derivative is:

$$\frac{dL}{dD} = \frac{\partial L}{\partial D} + \frac{\partial L}{\partial U}\frac{dU}{dD} \tag{1.1}$$

By differentiating the PDE constraint it is possible to compute the values of $\frac{dU}{dD}$ by solving the below system of linear equations, which scales with the dimension of D.

$$\frac{dR}{dD} = 0 = \frac{\partial R}{\partial D} + \frac{\partial R}{\partial U}\frac{dU}{dD} \tag{1.2}$$

Substituting in the expression for $\frac{dU}{dD}$ obtained from the equation above returns the following expression.

$$\frac{dL}{dD} = \frac{\partial L}{\partial D} - \frac{\partial L}{\partial U}\left[\frac{\partial R}{\partial U}\right]^{-1}\frac{\partial R}{\partial D} \tag{1.3}$$

This also allows for a simple definition of the adjoint equation; where Λ^T is the adjoint variable vector:

$$\Lambda^T = -\frac{\partial L}{\partial U}\left[\frac{\partial R}{\partial U}\right]^{-1} \tag{1.4}$$

the adjoint equation scales with L rather than D, and can be used to compute the sensitivities.

$$\frac{dL}{dD} = \frac{\partial L}{\partial D} + \Lambda^T\frac{\partial R}{\partial D} \tag{1.5}$$

Using the constraint that the residual is equal to zero, while necessary for both formulations, is not always feasible or possible – cases with complex geometries or high-order discretizations are difficult to converge to

machine zero. Often times the nonlinear flow equations are solved using some form pseudo-time integration. Consequently, this work is focused on the development of a novel pseudo-time accurate approach to the tangent and adjoint formulation for design optimization and error estimation which would therefore not make use of such a constraint on the residual.

Today in the field of computational fluid dynamics (CFD), the most commonly used methods are second-order accurate finite-volume codes with adjoint capabilities for design and/or error estimation. Some research groups use higher order accurate discretizations with adjoint capabilities for error estimation, but not often for design. The biggest use case as the field of CFD has progressed is multidisciplinary design optimization (MDO). As part of this increased use of MDO, the field has seen an increased desire for a totally automatic design process with minimal user input that provides for a robust and efficient design process with a minimum amount of physical testing (due to the prohibitive expense of such tests). To make this goal – as well as current design optimizations – computationally tractable, most MDO is done using gradient driven optimization as this allows for far fewer function evaluations when compared to gradient-free methods, such as genetic or particle swarm algorithms. This is necessary when the function evaluations are as expensive as they are for many CFD simulations. The optimization toolboxes used for these purposes (SNopt [1, 2], DAKOTA [3], etc.) are indifferent to the source of the gradient calculation. The finite-difference approach to compute sensitivities is trivial to implement, and would therefore appear to be a natural method by which to obtain the gradients. However, the finite difference method is both susceptible to round-off error and prohibitively expensive as it requires $n + 1$ function evaluations for n design variables. If one simply wanted to address the round off error one could use complex-step finite-differences [4]; however, this will be more expensive than the typical finite-difference approach due to the expense from the complex arithmetic, and would still scale with the number of design variables. Therefore researchers aim to provide sensitivities through the more efficient tangent or adjoint methods. The last two methods [5] are more accurate than the traditional finite-difference method (providing they are properly implemented), and, as shown above, are developed through conditions on convergence of the nonlinear problem to generate mathematical equations to solve for the sensitivities. The tangent formulation scales similarly to the finite-difference method, in that it generates a linear system which scales with the number of design variables; the adjoint transposes the system and scales independently of the number of design variables, it scales with the number of objective functions. For many aerospace applications the number of design variables is one to two orders of magnitude higher than the number of objective functions and the adjoint method is a powerful and preferred technique to obtain sensitivities. Also, the adjoint solution is well suited for adaptive mesh refinement (AMR) as a tool to estimate the error in a functional and refine a computational mesh to increase accuracy in the output of interest [6, 7].

The first attempts to use computational methods for design were by Lighthill using conformal mapping for design in subsonic flow [8]. McFadden later extended this method to compressible flow through use of

artificial viscosity and thus allowing for design even in the presence of shock waves [9]. In the 1970s many groups started looking into the use of transonic small disturbance theory and other potential flow methods for design [10, 11]. Jameson also began developing transonic Euler codes for structured and unstructured meshes [12, 13]. Lions et al. [14]and Pironneau et al. [15] later developed the first formulations of elliptic design problems through use of the adjoint. Jameson then applied adjoint methods to the Euler equations using the conformal mapping approach for the airfoil design [16]. Subsequently, Hicks and Henne used finite-differences to tackle partial differential equation (PDE) constrained optimization with shape parameterization to ensure smooth perturbations of the baseline geometry [17]. Jameson then applied the adjoint method in the context of PDE constrained optimization for shape optimization [16]. In such cases, where there are many design variables required to obtain a sufficiently fine design space, the adjoint allows for less costly sensitivity calculation as discussed previously. As the adjoint has become more prevalent, the field has seen three dimensional design optimization as well as multidisciplinary optimization methodologies become possible. Nadarajah et al. developed optimal design for unsteady – also referred to as time-accurate – flows [5]. This was followed by Mani et al. [18] developing the unsteady adjoint for design in multidiscplinary problems. Today the field has multiple codes that can run unsteady multidiscplinary design cases. Zhang et al. [19] presented results from a monolithic aerostructural Reynolds Averaged Navier Stokes (RANS) code for unsteady design/optimization. Fabiano et al. [20], Anderson et al. [21]. and Kamali et al. [22], show results for a loosely coupled aero-thermo-elastic-acoustic solver with design capabilities.

The history of AMR shows the development and effect of adjoint methods as well. AMR has become a larger part of the expanding field of CFD as computers and algorithms have developed. As the field has matured and seen an explosion in computing power and the development of stronger and more efficient methods, more complicated and expensive simulations are being undertaken. In order to make these simulations more computationally tractable the field has moved towards mesh refinement methodologies, which can now be found in many cutting edge research codes as well as some industrial codes. The earliest refinement methods were pioneered by Berger and Oliger in 1982 [23]. These were then moved by Berger first to multidimensional Euler flows [24] and then to arbitrary complex geometries [25]. These utilized patch based methodologies and form the basis of many packages used today. The other prominent methodology used is local refinement, first displayed in performant codes by Lohner in 1987 [26]. The simplest mesh refinement technology is a preprocessing refinement, that takes the initial mesh and refines in an area the user believes will have important flow features; this suffers from being non-adaptive. To compensate for this lack of adaptivity, these methods may be used in conjunction with an adaptive method. An example of such a code is the well validated and widely used Cart3D code which has an a priori refinement tool combined with an output-based error estimate to drive the adaptive refinement technology [6, 27]. When looking at adaptive refinement, the two main branches of adaptive methodologies are feature-based and error/output-based refinement. Feature based refinement refines the mesh in areas of high flow gradients and coarsens in smooth areas; the idea

being that areas of high flow gradients are crucial to resolve accurately in order to obtain accuracy in the outputs of engineering interest. The Refine package from NASA can calculate the Mach Hessian for the user from a flow field and can use that to drive the adaptation process on simplex meshes [28]. Feature-based methods are popular due to ease and low cost of implementation. Many large scale codes use feature-based adaptation when error estimates are difficult or expensive to implement or compute; one such example is refining based on q-criterion in exascale simulations of wind turbines [29].

The error-based or output-based refinement techniques utilize measures of error in the PDE discretization or the output of interest to drive the mesh adaptation. In error based refinement, without a specific output of interest, the error is computed in one of the two following ways. The first way is a more general approach that can be used in both Finite Volume (FVM) and Finite Element Method (FEM) codes; the mesh is uniformly refined and the converged flow state is interpolated onto the refined mesh and the residual is calculated on the fine mesh, then the residual from the fine mesh is restricted back to the coarse mesh. While the residual on the coarse mesh may be machine zero, the coarse mesh flow field interpolated onto the fine mesh will lead to a nonzero residual evaluation as these are different discretizations of the PDE with different solutions. The second method is often used in FEM codes; the residual operator for a higher order discretization than was utilized in the analysis is calculated using the converged flow state and the magnitude of the residual in each element becomes the refinement criterion and drives the mesh refinement module [30]. Similarly, the higher-order discretization will have a different converged flow state than the lower order one and therefore the residual will be nonzero. The issue with error-based approaches that do not factor in the output is that the mesh may be refined in places to better satisfy the governing equations that have no impact on the output of engineering interest. Output-based error estimates have seen the most development in recent years due to their usefulness in engineering applications, and this is where the adjoint formulation enters the picture. These began with the formulation by Becker and Rannacher [31] for FEM solvers, these use the adjoint to weight the residual from a higher order discretization to drive the refinement process, this is referred to as the dual weighted residual (DWR) error estimate. These were then moved to FVM solvers by Venditti and Darmofal including functional corrections [7]. Mani and Mavriplis [32] then applied these error estimation techniques to unsteady aerostructural problems refining the temporal mesh.

Whether one uses an industry code to solve a complicated geometry or a high-order research code to solve a simple geometry the same issue can occur: a lack of convergence of the residual. In typical engineering analysis cases one might simply check that the integrated quantities (such as lift and drag) do not vary with further iterations, and assume that this is sufficient for engineering purposes and design; this is not the case. As discussed previously, the adjoint and tangent systems require that the primal problem be fully converged, indicating that the governing equations have been satisfied to machine precision. Therefore the commonly used heuristic of convergence of engineering forces is no longer sufficient. However, as CFD has developed and matured and the field has tackled more difficult problems (higher-order formulations, blunt

geometries, or time-accurate simulations) this constraint has become more difficult or impractical to obey, and numerous design or mesh refinement cycles have been performed using adjoint systems linearized about partially converged primal solutions. This defect shows itself in less robust and more difficult to solve adjoint systems that are also sensitive to the specific state of the primal problem about which they are linearized when the convergence of the primal simulation is terminated prematurely [33,34]. For AMR, this can lead to refining the mesh in areas that may not contribute to the output of interest, and coarsening in areas where the output of interest is affected, which can impact the accuracy of the primal problem as the mesh adaptation proceeds. In the realm of design optimization, inaccurate adjoint vectors can lead to inaccurate sensitivities, which can change the course of the design cycle and lead to stagnation (as the Karush-Kuhn-Tucker (KKT) conditions, which govern termination, require that the gradient vanish at a local extremum) [2].

Krakos and Darmofal [33] illustrate that for a nonconvergent case, the state about which the adjoint is linearized can notably affect the sensitivity calculations. The authors show that by running the non-convergent steady-state case as a time-accurate case and applying the unsteady adjoint to the time-accurate case returns useful and accurate adjoint computed sensitivities for the time-averaged lift. The authors suggest that for steady-state cases which can be solved by strong solvers, but which may show physical unsteadiness, the time-accurate approach is in fact the proper analysis framework to use as otherwise CFD practitioners may risk obtaining unphysical and non-useful sensitivity vectors. Krakos et al. follow their previous work with an investigation of statistical and windowing techniques to allow only partial time integration for periodic primal flows with time-averaged outputs of interest [35]. The authors demonstrate that with proper windowing techniques only partial time integration is required to obtain accurate sensitivities. However even for time-accurate formulations, Mishra et al. [36] have demonstrated growing error in the sensitivity vector throughout the adjoint reverse time-integration due to partial convergence of the primal problem at each implicit time step, which is a common practice in applied CFD problems. Mishra et al. applied the unsteady adjoint to a helicopter blade optimization and compared the sensitivities of the tangent and adjoint linearizations to those of the complex-step finite-difference approach for both a rigid and a flexible structure. Figure 1, taken with permission from the work of Mishra et al., shows growing error in the sensitivity through the time integration for a flexible rotor case with the aforementioned partial convergence at each implicit time step. The left plot shows the absolute error growing as the tangent is integrated forward in time and the adjoint is integrated backwards in time for one rotor revolution. The right plot shows the relative error for the two methods over the same window with the adjoint plotted as it integrates backwards in time.

It is clear that the discrepancy between the complex-step finite-difference sensitivities and the adjoint computed sensitivities increases steadily over time for this rotorcraft simulation. Therefore, it seems to be necessary to develop a method that would be accurate without this reliance on full convergence of the implicit time step. Furthermore, this raises the question as to how partial convergence of the steady-state nonlinear problem affects the accuracy of the tangent and adjoint computed sensitivities. Luers et al. [37] illustrate

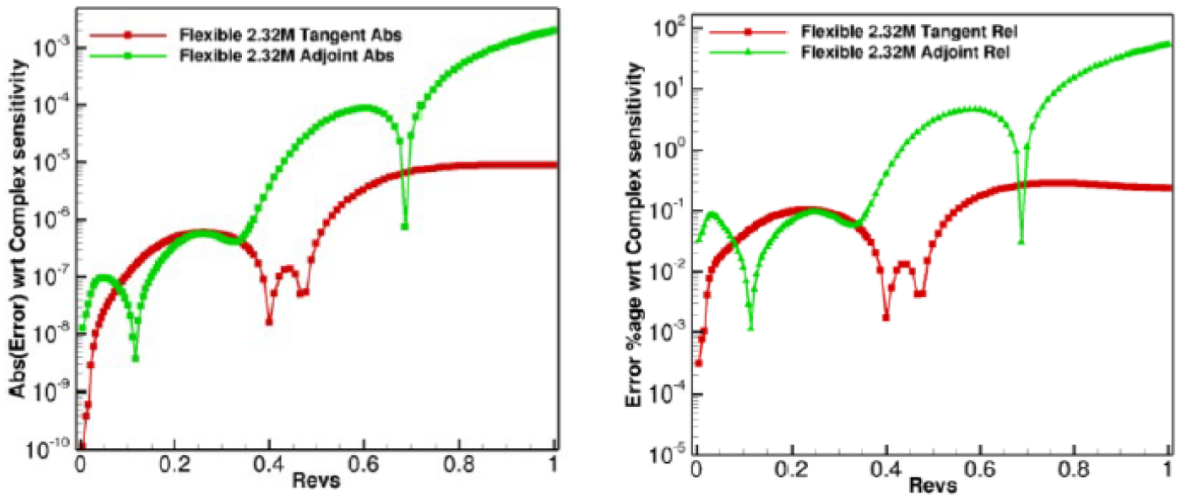


Figure 1.1: Complex vs. adjoint sensitivities for partially converged unsteady rotorcraft flows

the importance of accurate gradients in well converging cases that have not reached deep convergence. In their paper they present steady-state optimizations for a CRESCENDO turbine cascade, and contrast the optimization for finite-difference computed gradients to that driven by adjoint computed gradients in a case with a 5 order drop in the nonlinear residual. They ran this partially converged analysis problem and compared the adjoint to the finite-difference computed sensitivities and showed very good qualitative correspondence and took this as verification of the implementation of the adjoint and the ability to only partially converge the analysis and still obtain useful sensitivities. They then show that by using the finite-difference provided gradients the optimized objective function is lower than that obtained by using the adjoint provided gradients, computed through linearization about the partially converged state. Brown and Nadarajah [38] investigate an upper bound for the error in the adjoint computed sensitivities arising from partial convergence of the analysis problem for problems that are smoothly converging to the steady-state solution. From these error estimates, the authors can focus computational resources on the more crucial stages of the design cycle, i.e., get a mostly directionally correct sensitivity vector early on in the design cycle and obtain more accurate vectors as the optimizer moves closer to the minimum where accuracy in the sensitivity vector is of utmost importance. The work of Brown and Nadarajah is a natural solution to the issues demonstrated by Luers et al. [37] in their volumetric optimization cases.

The development of an approach for computing tangent and adjoint problems in nonconvergent steady-state problems is the thrust of this thesis. The proposed approach linearizes the fixed point iteration used to solve the steady-state problem to develop the pseudo-time accurate tangent and adjoint systems [34, 39]. The pseudo-time accurate formulation of the tangent and adjoint systems is used with a pseudo-time averaged objective functional to compute the adjoint and tangent sensitivities for a system in limit-cycle oscillations. These methods will provides sensitivities that correspond exactly to the sensitivities obtained

through the complex step differentiation of the solution process. The argument is that these sensitivities, which correspond to the linearization of the simulation itself are the ones to use for optimization rather than sensitivities provided by an adjoint linearized about an unconverged state. In this work it is shown that the pseudo-time accurate approach yields benefits not only for the truncated cases like those of Luers et al. [37], but also (and primarily) for cases that cannot and will not converge. The pseudo-time accurate formulations of the adjoint problem will be applied to design optimization and error estimation and adaptive mesh refinement, showing the efficacy of such techniques.

Chapter 2

Implementation Details: Aerodynamic Analysis, Sensitivities, and Parallelism

2.1 Governing Equations

The governing equations for this work are the steady Euler equations. denoted by:

$$\nabla \cdot F(U) = 0 \tag{2.1}$$

where U is the conservative variable vector and $F(U)$ is the conservative variable flux. The conservative variable vector U is written as:

$$U = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e \end{bmatrix} \tag{2.2}$$

where ρ is the density, u and v are the velocity components in the x and y directions respectively and e is the energy. The flux operator has two components, in the x and y directions:

$$\mathbf{F_x}(\mathbf{U}) = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (\rho e + p)u \end{bmatrix}, \mathbf{F_y}(\mathbf{U}) = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (\rho e + p)v \end{bmatrix} \tag{2.3}$$

P is the fluid pressure obtained by:

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

where $\gamma = 1.4$ is the specific heat ratio. The flux at a face with normal (n_x, n_y) is given as:

$$\mathbf{F}(\mathbf{U}) = \mathbf{F_x}(\mathbf{U})\mathbf{n_x} + \mathbf{F_y}(\mathbf{U})\mathbf{n_y} \tag{2.5}$$

2.2 Boundary Conditions

This work uses dual consistent formulations of the boundary conditions to allow for better refinement patterns as shown in other works [40].

2.2.1 No Penetration Boundary Condition (Slip Wall)

The no penetration boundary condition enforces the condition on the boundary wall (denoted by Γ_{wall}):

$$\vec{u} \cdot \vec{n} = 0, \vec{x} \in \Gamma_{wall} \tag{2.6}$$

This is enforced by taking the state of the boundary element (u^+) and calculating a boundary state (u^b) from it. The flux across the face is then calculated using the exact flux F where $F = F(u^b(u^+))$. In order to impose the no-penetration boundary condition/slip-wall, the normal component of the velocity is enforced to be equal to zero and the tangent component is unchanged across the boundary. This gives the following equations:

$$\begin{aligned} \vec{u^b} \cdot \vec{n} &= 0 \\ \vec{u^b} \cdot \vec{t} &= \vec{u^+} \cdot \vec{t} \end{aligned} \tag{2.7}$$

where $\vec{t}$ is the tangent vector to the boundary.

$$\mathbf{u}^+ = \begin{bmatrix} \rho^+ \\ \rho^+ u^b \\ \rho^+ v^b \\ \rho^+ e^+ \end{bmatrix} \tag{2.8}$$

2.2.2 Characteristic Boundary Condition (Inflow/Outflow Boundary)

Here again it is used that the boundary cell (u^b) is a function of the boundary element (u^+) and the flux is computed across the face using the same exact flux F. First, the Riemann invariants – denoted by

R_p and R_m – must be defined so that the boundary state can be computed.

$$\begin{aligned} w_1 &= s \\ w_2 &= \vec{u} \cdot \vec{t} \\ w_3 &= R_p = \vec{u} \cdot \vec{n} + \frac{2c}{\gamma - 1} \\ w_4 &= R_m = \vec{u} \cdot \vec{n} - \frac{2c}{\gamma - 1} \end{aligned} \tag{2.9}$$

The code is nondimensionalized such that the speed of sound, c, is equal to unity. The above quantities must stay constant across the boundary and whether the information is propagated from the outside onto the boundary or from the inside out is dependent on the flow regime. In subsonic flow the characteristics will flow in and out of the domain, and the boundary state will depend on both the freestream values and the values inside the domain. For a subsonic inflow boundary ($\vec{u} \cdot \vec{n} < 0$) the invariants are calculated as follows:

$$\begin{aligned} w_1 &= s = \frac{P_\infty}{\rho_\infty^\gamma} \\ w_2 &= \vec{u} \cdot \vec{t} = -u_\infty n_y + v_\infty n_x \\ w_3 &= R_p = \vec{u} \cdot \vec{n}^+ + \frac{2c^+}{\gamma - 1} \\ w_4 &= R_m = \vec{u} \cdot \vec{n}_\infty - \frac{2c_\infty}{\gamma - 1} \end{aligned} \tag{2.10}$$

For a subsonic outflow boundary ($\vec{u} \cdot \vec{n} > 0$) the invariants are calculated as follows:

$$\begin{aligned} w_1 &= s = \frac{P^+}{(\rho^+)^\gamma} \\ w_2 &= \vec{u} \cdot \vec{t} = -u^+ n_y + v^+ n_x w_1 \\ w_3 &= R_p = \vec{u} \cdot \vec{n}^+ + \frac{2c^+}{\gamma - 1} \\ w_4 &= R_m = \vec{u} \cdot \vec{n}_\infty - \frac{2c_\infty}{\gamma - 1} \end{aligned} \tag{2.11}$$

The Riemann invariants are used to compute the boundary state and therefore the boundary flux. Supersonic flow is much simpler as all the characteristics point downstream, and so the upstream state is not impacted by the downstream one. For inflow the boundary state is given by:

$$\mathbf{u^b} = \begin{bmatrix} \rho_\infty \\ \rho_\infty u_\infty \\ \rho_\infty v_\infty \\ \rho_\infty e_\infty \end{bmatrix} \tag{2.12}$$

and for the outflow it is given by:

$$\mathbf{u^b} = \begin{bmatrix} \rho^+ \\ \rho^+ u^+ \\ \rho^+ v^+ \\ \rho^+ E^+ \end{bmatrix} \tag{2.13}$$

2.3 Spatial Discretization

The solver uses a finite-volume cell-centered approach that will be expanded on in the following sections. The residual about the closed control volume is:

$$R = \int_{dB} [F(U)] \cdot n, \mathrm{dB} = \sum_{i=1}^{n_{edge}} F_{e_i}^{\perp}(U, n_{e_i}) B_{e_i} \tag{2.14}$$

In a first-order finite-volume discretization the conservative variable values in the cell are held to be piecewise constant and the cell values are used to evaluate the numerical fluxes at the edge midpoints. This leads to a nearest neighbors stencil and the flow Jacobian is computed by linearizing the flux at each face with respect to the left and right states and storing these values in a sparse edge based structure of diagonals and off diagonals. The matrix vector product $\left[\frac{\partial R}{\partial u}\right]_1 v$ is evaluated by looping over the edges.

2.3.1 Numerical Flux

In this work three flux schemes are implemented for the interior numerical fluxes: Lax-Friedrichs/Rusanov scheme [41], Van Leer flux splitting [42], and Roe flux [43]. The Lax-Friedrichs scheme is written as:

$$F = \frac{1}{2}(F_L + F_R + \alpha(U_L - U_R)) \tag{2.15}$$

where α is the max eigenvalue at the interface:

$$\alpha = \max_{L,R}(|\vec{u} \cdot \vec{n}| + c) \tag{2.16}$$

The Van-Leer flux splitting scheme, which is predominantly used in this work, is implemented as:

$$f_x = \pm\frac{\rho}{4c}(u \pm c)^2 \begin{bmatrix} 1 \\ \frac{(\gamma-1)u \pm 2c}{\gamma} \\ v \\ \frac{v^2}{2} + \frac{[(\gamma-1)u \pm 2c]^2}{2(\gamma^2-1)} \end{bmatrix}, \; f_y = \pm\frac{\rho}{4c}(v \pm c)^2 \begin{bmatrix} 1 \\ u \\ \frac{(\gamma-1)v \pm 2c}{\gamma} \\ \frac{u^2}{2} + \frac{[(\gamma-1)v \pm 2c]^2}{2(\gamma^2-1)} \end{bmatrix} \tag{2.17}$$

The numerical flux at the face is computed by:

$$f(u) = f_{xL}(u)n_x + f_{yL}(u)n_y + f_{xR}(u)n_x + f_{yR}(u)n_y \tag{2.18}$$

The Roe flux is calculated as follows:

$$F = \frac{1}{2}(F(u_L) + F(u_R)) + \frac{1}{2}\left|\bar{A}\right|(u_L - u_R) \tag{2.19}$$

where the $\left|\bar{A}\right|$ matrix is the linearization of the flux about the Roe state, which is created through the use of Roe averaging. This can be rewritten:

$$F = \frac{1}{2}(F(u_L) + F(u_R)) + T|\Lambda|T^{-1}(u_L - u_R) \tag{2.20}$$

where the $|\Lambda|$ matrix is a diagonal matrix containing the eigenvalues corresponding to the characteristics: $u - c$, u, $u + c$. Oftentimes this can require the use of an entropy fix, although the Harten-Hyman entropy fix [44] was implemented it was not used, and instead the following is used [45]:

$$u = sign(u)max(|u|, \delta(|u| + c)) \tag{2.21}$$

$$u + c = sign(u - c)max(|u + c|, \delta(|u| + c)) \tag{2.22}$$

$$u - c = sign(u - c)max(|u - c|, \delta(|u| + c)) \tag{2.23}$$

where δ is a small limiting parameter between zero and one that prevents the eigenvalues from approaching zero and thus the dissipation from disappearing.

2.3.2 Extension to Second-Order Spatial Accuracy

The code developed for this work is spatially second-order accurate and uses gradient reconstruction to extend beyond first-order accuracy. Gradient reconstruction works by taking each cell to have piecewise linear conservative variable values using a gradient reconstruction method to reconstruct from the cell center to the face midpoint. The equation below shows the state reconstruction, where the conservative variable reconstruction at the face is the centroid value (u_i) plus the dot product of the gradient and the distance from the centroid of cell i to the center/Gauss point of the shared face k ($\vec{r}_{ik}$).

$$u_{rc_{ik}} = u + \nabla u_i \cdot \vec{r}_{ik} \tag{2.24}$$

The flux at the face separating cells i and j is calculated by using the numerical flux functions outlined previously with inputs being the reconstructed states. That is to say, $F = F(u_L, u_R)$ where, as above, $u_L = u_i + \nabla u_i \cdot \vec{r}_{ik}$ and $u_R = u_j + \nabla u_j \cdot \vec{r}_{jk}$. This reconstruction expands the stencil of the discretization to a neighbors of neighbors stencil; the first-order flux functions use a nearest neighbors approach and the reconstructed states extend this to the neighbors of neighbors stencil. This changes the Jacobian calculation and storage requirements. The first-order Jacobian is expressed as $\left[\frac{\partial R}{\partial u}\right]_1$ which consists of diagonal entries for each cell and off diagonal entries for each edge. When moving to the second-order accurate Jacobian, expressed as $\left[\frac{\partial R}{\partial u}\right]_2$, there is now a neighbors of neighbors stencil which takes up 2.5 times more memory than the first-order accurate one. Rather than storing the full matrix it is factored and any matrix vector products are formed in a staged approach. The matrix is computed as below:

$$\left[\frac{\partial R}{\partial u}\right]_2 = \frac{\partial R}{\partial u_L}\frac{\partial u_L}{\partial U} + \frac{\partial R}{\partial u_R}\frac{\partial u_R}{\partial U} \tag{2.25}$$

In order to compute the matrix vector product of $\left[\frac{\partial R}{\partial u}\right]_2 \delta U$, it is calculated as:

$$\begin{aligned} \delta u_L &= \frac{\partial u_L}{\partial U}\delta U \\ \delta u_R &= \frac{\partial u_R}{\partial U}\delta U \\ \left[\frac{\partial R}{\partial u}\right]_2 &= \frac{\partial R}{\partial u_L}\delta u_L + \frac{\partial R}{\partial u_R}\delta u_R \end{aligned} \tag{2.26}$$

Where the components of the matrix $\frac{\partial R}{\partial u_L}$,$\frac{\partial u_L}{\partial U}$ and $\frac{\partial u_R}{\partial U}$ are all stored in a sparse storage format. Breaking down the matrix vector product allows for easy verification of the linearization. By using a random vector perturbation δU to U it is possible to first check the linearization of the left side reconstructed state using the identity below for the complex-step method.

$$\frac{\partial u_L}{\partial U}\delta U = \frac{Imag(u_L(U + i\epsilon\delta U))}{\epsilon} \tag{2.27}$$

The linearization of the right constructed state can be verified similarly.

$$\frac{\partial u_R}{\partial U}\delta U = \frac{Imag(u_R(U + i\epsilon\delta U))}{\epsilon} \tag{2.28}$$

The linearization of the residual with respect to the left and right reconstructed states can be checked by combining the two routines above using two random perturbation vectors δu_L and δu_R.

$$\frac{\partial R}{\partial u_L} + \frac{\partial R}{\partial u_R} = \frac{Imag(R(u_L + i\epsilon\delta u_L, u_R + i\epsilon\delta u_R))}{\epsilon} \tag{2.29}$$

This linearization uses the linearized flux functions used in the first-order accurate spatial residual linearization but substitutes in the second-order accurate conservative variable values. The gradient reconstruction used in this work is the weighted least squares gradient reconstruction.

This was implemented by using Cramer's rule to solve the following system of linear equations:

$$\begin{aligned} a_i u_x + b_i u_y &= d_i \\ b_i u_x + c_i u_y &= e_i \end{aligned} \tag{2.30}$$

where the left hand side is defined below.

$$\begin{aligned} a_i &= \sum_{k=1}^{N} w_{ik}^2 dx_{ik}^2 \\ b_i &= \sum_{k=1}^{N} w_{ik}^2 dx_{ik} dy_{ik} \\ c_i &= \sum_{k=1}^{N} w_{ik}^2 dy_{ik}^2 \\ d_i &= \sum_{k=1}^{N} w_{ik}^2 du_{ik} dx_{ik} \\ e_i &= \sum_{k=1}^{N} w_{ik}^2 du_{ik} dy_{ik} \end{aligned} \tag{2.31}$$

w is the distance weights between the cell centroids and the d terms are the differences in coordinates and conservative variable values between the cell centroids.

$$\begin{aligned} dx_{ik} &= x_k - x_i \\ dy_{ik} &= y_k - y_i \\ du_{ik} &= u_k - u_i \\ w_{ik} &= \frac{1}{\sqrt{dx_{ik}^2 + dy_{ik}^2}} \end{aligned} \tag{2.32}$$

Since this work uses a cell-centered finite volume discretization there will often be meshes that contain elements with two boundary faces, as such the least squares system includes the boundary states and distances as part of the least squares system. It should be noted from the boundary formulation given above that the boundary states computed are created at the boundary Gauss points, and that therefore the distance in the least squares system is from the cell centroid to the Gauss point (the edge midpoint) rather than to a ghost centroid reflected across the boundary face. When this is implemented incorrectly this can lead to a loss of accuracy across the boundary.

The gradient reconstruction routines used in the residual evaluations are also used to calculate the objective functions for design optimization. The quantities of engineering interest, such as lift and drag, are computed on the airfoil boundary, but the conservative variable values are computed and stored at the cell centers. For the first-order spatially accurate solver, the cell center values are used at the boundary as the conservative variable values are taken to be constant through the cells. For the second-order spatially accurate solver the option is to either use the cell center values, or to reconstruct from the cell center to the boundary and evaluate the forces based off those reconstructed values. While this method usually leads to greater accuracy, the cell gradients are only first-order accurate on arbitrary meshes and their accuracy can suffer on highly stretched or anisotropic meshes, which can in turn affect the quantities of engineering interest.

2.3.3 Limiting Algorithm

It should be noted that for second-order discretizations in transonic or hypersonic flow, often a gradient/slope limiting method is required to assist in stability of the flow solver; this limits the solution to being first-order accurate in the vicinity of shocks to keep the total variation diminishing property necessary for stability. This work uses a modified Venkatakrishnan's Limiter towards that end, shown in algorithm 2. Venkatakrishnan's limiter is itself a smooth modification of the Barth-Jespersen limiter. The original Barth-Jespersen limiter is shown in algorithm 1 and is written as follows for each cell:

1. Find the largest negative and positive differences between the cell average solution in the neighbors and the current control volume

2. Compute the unlimited reconstructed value at the face midpoint

   In the implementation in this work the reconstruction is performed at the intersection of the vector between the two cell centers and the face separating them, this prevents spurious limiting

3. Compute the maximum allowable value of the limiter of each field variable and each face by enforcing that no extremum is created at each face

   This is done by reconstructing the the midpoint of each face and checking that the reconstructed value is not larger than the cell center values of the cells that share the face

4. Take the minimum value at each face to get the most conservative limiter for each field variable

In algorithm form:

Algorithm 1 Barth-Jespersen Limiter

1: **procedure** BARTH-JESPERSEN
2: **for** $i = 1, ..., fields$ **do**
3: $\delta u_i^{min} = \infty$, $\delta u_i^{max} = 0$
4: **for** $j = 1, ..., neighbors$ **do**
5: $\delta u_i^{min} = \min(\delta u_i^{min}, \bar{u_i} - \bar{u_i^j})$
6: $\delta u_i^{max} = \max(\delta u_i^{max}, \bar{u_i} - u_i^j)$
7: **for** $j = 1, ..., neighbors$ **do**
8: $u_{rc_j} = u_i + \nabla \bar{u_i} \cdot \vec{r_j}$
9: $$\Phi_{ij} = \begin{cases} \min(1, \frac{\delta u_i^{max}}{u_{rc_j} - \bar{u_i}}), & if\, u_{ij} - \bar{u_i} > 0 \\ \min(1, \frac{\delta u_i^{min}}{u_{rc_j} - \bar{u_i}}), & if\, u_{ij} - \bar{u_i} < 0 \\ 1, & if u_{ij} - \bar{u_i} = 0 \end{cases}$$
10: $\Phi_i = min(\Phi_{ij})$

To aid convergence and differentiability, Venkatakrishnan's limiter replaces the min in step 9 with a smooth approximation of the form:

$$\phi(y) = \frac{y^2 + 2y}{y^2 + y + 2} \tag{2.33}$$

where $y = \frac{\Delta_+}{\Delta_-}$,

$$\Delta_+ = \begin{cases} \delta u_i^{max}, & if\, u_{ij} - \bar{u_i} > 0 \\ \delta u_i^{min}, & if\, u_{ij} - \bar{u_i} < 0 \end{cases} \tag{2.34}$$

and Δ_- is written below.

$$\Delta_- = \nabla u_i \cdot \vec{r}_{ij} \tag{2.35}$$

This expression in terms of y can be modified to prevent activation in smooth regions, as will be shown in the algorithm of the implementation used in this work. In this work Venkatakrishnan's limiter is modified in a few important ways.

1. All max and min functions use smooth max and min functions to allow differentiability

2. All switches in the code dependent on the flow state are done in a smooth and blended manner using a sine shut-off function to aid differentiability

3. The limiter is augmented with a stagnation point fix that turns off the limiter entirely if the local Mach number is below a certain value (again done in a smooth manner)

4. Finally, the limiter contains a realizability check that if violated (pressure, energy, or density are reconstructed to below 5% of the cell center value) the cell is knocked down to first-order

Algorithm 2 Augmented Venkatakrishnan Limiter

procedure AUGMENTED VENKATAKRISHNAN LIMITER
  $M_1 = .80, M_2 = .85, \epsilon_{lim} = 1e-5, \epsilon_{TFO_s} = -.95, \epsilon_{TFO_e} = -.9$
  $\epsilon = \sqrt{\kappa(r_1)}^3$, where r_1 is the radius of the circumscribed circle of the triangular cell
  $M_{max} = \bar{M}$
  for $i = 1, ..., fields$ **do**
    $\Phi_i = 1$
    $\delta u_i^{min} = \infty$, $\delta u_i^{max} = 0$
    for $j = 1, ..., neighbors$ **do**
      $\delta u_i^{min} = \min(\delta u_i^{min}, \bar{u_i} - \bar{u_i^j})$
      $\delta u_i^{max} = \max(\delta u_i^{max}, \bar{u_i} - \bar{u_i^j})$
      $M_{max} = \max(M_{max}, M_i)$
    $\sigma_{Mach} = 1 - SSO(M_{max}, M_1, M_2)$
    for $j = 1, ..., neighbors$ **do**
      $\Delta_- = \nabla \bar{u_i} \cdot \vec{r_j}$
      $s = SSO(\Delta_-, 0, \epsilon_{lim})$
      $\Delta_+ = (1-s)(u_i^{min} - \bar{u}_i) + s(u_i^{max} - \bar{u}_i)$
      $\phi = \frac{\Delta_+^2 + \epsilon^2 + 2\Delta_+\Delta_-}{\Delta_+^2 + \epsilon^2 + 2\Delta_+\Delta_- + 2\Delta_-^2}$
      Stagnation point fix
      $\phi = \sigma_{Mach} + (1 - \sigma_{Mach})\phi$
      $\Phi_i = \min(\Phi_i, \phi)$
  Begin realizability check
  for $j = 1, ..., neighbors$ **do**
    $u_{rc_j} = u_i + \Phi \nabla \bar{u_i} \cdot \vec{r_j}$
    $\delta_\rho = \rho_{rc_j} - \bar{\rho}$
    $\delta_P = P_{rc_j} - \bar{P}$
    $\delta_E = E_{rc_j} - \bar{E}$
    $s_e = SSO(\delta_e, \epsilon_{TFO_s}, \epsilon_{TFO_e})$
    $s_p = SSO(\delta_p, \epsilon_{TFO_s}, \epsilon_{TFO_e})$
    $s_\rho = SSO(\delta_\rho, \epsilon_{TFO_s}, \epsilon_{TFO_e})$
    $s = \min(s_\rho, s_e, s_p)$
    $\Phi = s\Phi$

This limiter in algorithm (2) provides better convergence properties than the standard Venkatakrishnan Limiter, and is used in this work. It can also be noted that due to the use of smooth max and min functions, as well as a smoothly blending shut-off (SSO) function outlined in equation (2.40) [46], every step in this limiter is differentiable. This makes the linearization of the limiters possible and much easier to code, as multiple branching statements are avoided.

2.3.4 Smooth Function Implementations

With the goal of making this code smooth and differentiable to assist in convergence and adjoint linearization, 5 smooth functions to approximate sign, absolute value, max, min, and a state-dependent branching statement have been implemented. The branching statement uses the smooth shut-off function as in the limiting algorithm.

$$sign(x) = tanh(100x) \tag{2.36}$$

$$|x| = \frac{x^2}{sign(x) + 1e - 13} \tag{2.37}$$

$$\max(x, y) = \frac{1}{2}(x + y + |x - y|) \tag{2.38}$$

$$\min(x, y) = \frac{1}{2}(x + y - |x - y|) \tag{2.39}$$

$$SSO(x, x_s, x_e) = \begin{cases} 0 & if\ x < x_s \\ \frac{1}{2}\left(sin\left(\frac{\pi}{2}\frac{2x-(x_e+x_e)}{x_e-x_s}\right) + 1\right) & if\ x_s < x < x_e \\ 1 & if\ x > x_e \end{cases} \tag{2.40}$$

where θ is defined by:

$$\theta = \frac{\pi}{2}\frac{2x - (x_e + x_e)}{x_e - x_s} \tag{2.41}$$

2.4 Nonlinear Solvers

The solver technology implemented in this work consists of either explicit time stepping through pseudo-time using a forward Euler scheme or a low storage five stage Runge-Kutta scheme; or implicit time stepping through Newton's Method. The forward Euler scheme is written as follows.

$$u^k = u^{k-1} + CFL\Delta t(u^{k-1}, D)R(u(D), D) \tag{2.42}$$

The five stage Runge-Kutta shown below is used because of its increased stability over the forward Euler method. It is as follows, iterating k through pseudo-time/nonlinear iterations and l through stages one to five at each pseudo-time iteration:

$$u^{k,l} = u^{k,0} + CFL\alpha^{l-1}\Delta t(u^{k-1,0})R(u^{k-1,l-1}) \tag{2.43}$$

with the end of the sub-stage time-stepping being governed as follows.

$$u^{k,0} = u^{k-1,5} \tag{2.44}$$

Newton's method was implemented using pseudo-transient continuation (PTC) with a first-order Backward Difference (BDF1) scheme in the context of a quasi-Newton method. For Newton's method the time-stepping procedure is written as:

$$u^k = u^{k-1} + \Delta u \tag{2.45}$$

where Δu is computed by solving the following system of linear equations.

$$[P]\,\Delta u = -R(u) \tag{2.46}$$

Δu can be substituted into the time-stepping equation (2.45) to obtain the final form of this equation.

$$u^k = u^{k-1} - [P_{k-1}]^{-1} R \tag{2.47}$$

Where $[P_{k-1}]$ is a first or second-order accurate Jacobian of the spatial residual augmented with a diagonal term to ensure that it is diagonally dominant. In this work, typically the preconditioner matrix is the first-order accurate Jacobian:

$$[P_{k-1}] = \frac{\partial R}{\partial u^{k-1}} + \frac{vol}{\Delta t CFL} \tag{2.48}$$

vol is the area of the cell, and the CFL is scaled either with a simple ramping coefficient (β) and a maximum allowable CFL:

$$CFL = min(\beta * CFL, CFL_{max}) \tag{2.49}$$

or with a line search and CFL controller, which seeks to minimize the L_2 norm of the pseudo-unsteady residual. The pseudo-unsteady residual is defined below.

$$R_t(u + \alpha\Delta u) = \frac{vol}{\Delta t}\alpha\Delta u^k + R(u + \alpha\Delta u^k) \tag{2.50}$$

When the temporal residual decreases, this is considered to be a satisfactory value for α and the CFL is changed accordingly. Algorithm 3 shows the actual process for the line search and CFL controller.

Algorithm 3 CFL controller

procedure LINE SEARCH AND CFL CONTROLLER
  $r_{t0} = \|R_t(u + \Delta u)\|_2$
  $r_{s0} = \|R(u)\|_2$
  $r_{s1} = \|R(u + \Delta u)\|_2$
  if $r_{s0} < r_{s1}$ **then**
    for $k = 1, ..., itermax$ **do**
      $\alpha = c\alpha$
      $r_{t1} = \|R_t(u + \alpha\Delta u)\|_2$
      if $r_{t1} < r_{t0}$) **then** exit
  if $alpha < alpha_{l1}$ **then**
    $\alpha = 0$
    $CFL = \beta_1 CFL$
  else if $alpha_{l1} < \alpha < alpha_{l2}$ **then**
    $CFL = CFL$
  else if $\alpha < alpha_{l2}$ **then**
    $CFL = min(\beta_2 CFL, CFL_{max})$

Where the parameters $itermax$, c, α_{l1}, α_{l2}, β_1, and β_2 are all user defined input values, defaulted to 30, .9, .1, .75, .1, and 1.5 respectively.

2.5 Linear Solvers

The need for linear solvers implemented in this work is driven first by the need to solve a linear system in the Newton-Krylov nonlinear solver shown above in equation 2.47 as well as the requirement of solving the tangent and adjoint systems. Generally a linear system is defined as below, with A being a matrix, and x and b being vectors of unknowns and knowns respectively.

$$Ax = b \tag{2.51}$$

The linear residual of the system is written as follows.

$$r = b - Ax \tag{2.52}$$

In order to solve the linear system in this work either element-implicit Jacobi or Gauss-Seidel smoothers are used, shown in algorithms 4 and 5 respectively. They are used either as linear solvers or as preconditioners for BiCGStab [47] or FGMRES [48] linear solvers. This is done by lagging the diagonal components, denoted as $[D]$, with the right hand side being the linear residual of the original system. If the discretization is first-order accurate, then the residual and Jacobian will both be first-order accurate. If the discretization is second-order accurate, the residual is second-order accurate, and the Jacobian can be either first or second-order accurate. If the Jacobian is first-order accurate this is a defect-correct or inexact Newton solver. If the Jacobian is second-order accurate then the smoother can use either the first-order or second-order accurate Jacobian on the left hand side; if the Jacobian is second-order accurate in both the linear system and the smoother then the smoother is under relaxed. The smoothers used in this work often use the first-order accurate left hand side for simplicity and computational expense.

Algorithm 4 Element-Implicit Jacobi

1: **procedure** JACOBI
2: $k = 0$
3: $v = Ax$
4: $r = b - v$
5: **for** $k < itermax$ **do**
6: $i = 0$
7: **for** $i < cellnum$ **do**
8: $\Delta x_i = -[D_i]^{-1} r_i$
$x = x + \omega \Delta x$
9: **if** $(\|r\| < lintol)$ **then** exit

The smoothing iteration for the analysis code as:

$$[D]_1 \Delta u = -R - \left[\frac{\partial R}{\partial u}\right]_1 \Delta u \tag{2.53}$$

Typically this would hamper the nonlinear solution process as it would only allow for the inexact Newton solver, however, using second-order matrix vector products and first-order preconditioning is an effective

Algorithm 5 Element-Implicit Gauss-Seidel

procedure GAUSS-SEIDEL
  $k = 0$
  for $k < itermax$ **do**
    $i = 0$
    for $i < cellnum$ **do**
      for $k < nneighbors(i)$ **do**
        $v_i = v_i + A_{i,k} x_{i,k}$
      $r_i = b_i - v_i$
      $\Delta x_i = -\left[D_i\right]^{-1} r_i$
    $x = x + \omega \Delta x$
    if $(\|r\| < lintol)$ **then** exit

and widely used technique to implement GMRES or other Krylov solvers and this allows for a second-order accurate Jacobian in the Newton-Krylov solver used for the primal problem. The flexible GMRES linear solver is also implemented to solve the stiff steady state tangent and adjoint systems. For preconditioning it uses either of the point-implicit linear smoothers.

Algorithm 6 Flexible Restarted GMRES

procedure FLEXIBLE GMRES
  for $k = 1, ..., ncycles$ **do**
    $r_0 = b - Ax_0, \beta = \|r_0\|, v_1 = r_0/\beta$
    for $j = 1, ..., m$ **do**
      $z_j = M^{-1} v_j$
      $v_{j+1} = A z_j$
      for $i = 1, ..., j$ **do**
        $h_{i,j} = (v_{j+1}, v_i)$
        $v_{j+1} = v_{j+1} - h_{i,j} v_i$
      $h_{j+1,j} = \|v_{j+1}\|, v_{j+1} = v_{j+1}/h_{j+1,j}$
      Define $Z_m = [z_1, ..., z_m], \bar{H}_m = [h_{i,j}]_{1<i<j+1, 1<j<m}$
    Solve least squares problem for y_m
    $x_0 = x_0 + Z_m y_m$

2.6 The Design Problem

The bulk of this work focuses on gradient based design for aerodynamic shape optimization, and it is assumed that these cases begin with a baseline geometry and mesh [49]. A flowchart showing the optimization process is in Figure 2.1.

First, the design variables smoothly perturb the geometry nodes. Second, the perturbed geometry moves the volume nodes of the mesh through use of a mesh motion algorithm. The flow solution problem is solved on the perturbed geometry and mesh. The objective function is then solved using the computed flow field. Finally the sensitivities are computed (often using the adjoint method) and the objective and sensitivities are given to an optimizer to provide new values of the design variables and the sequence is repeated. The

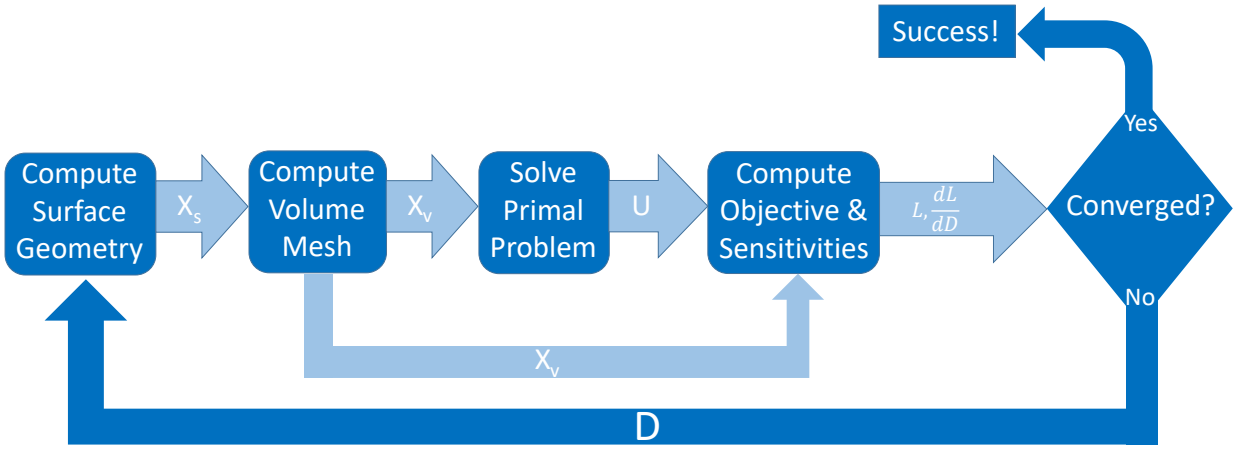


Figure 2.1: Design Process Flow Chart

below sequence outlines the design problem where D is the design variable vector, x_{surf} is the geometry coordinate vector, x_v is the volume mesh point coordinate vector, U is the conservative variable vector, and L is the objective function.

$$x_{surf} = x_{surf}(D) \tag{2.54}$$

$$x_v = x_v(x_{surf}) \tag{2.55}$$

$$U = U(x_v) \tag{2.56}$$

$$L = L(U, x_v) \tag{2.57}$$

The functional dependence of the objective function is written as:

$$L = L(U(x_v(x_{surf}(D))), x_v(x_{surf}(D))) \tag{2.58}$$

An example of a design case is optimizing the airfoil shape for given inflow conditions with an objective that would be minimized at a target lift coefficient (c_{LT}) with minimal drag coefficient; such an objective function is given below.

$$J = \omega_{cl}(c_L - C_{LT})^2 + \omega_{cd}c_D^2 \tag{2.59}$$

For this work, the code is coupled with SNopt [1, 2] as the optimizer and allows use of quadratic penalty composite objective functions to weight and target lift, drag, and entropy.

2.6.1 Design Variables

The design optimization portion of this code is done using Hicks-Henne bump functions [17] – the simplicity of implementation makes them a desirable approach – to smoothly perturb the airfoil coordinates. The equation is:

$$\delta x_s(x) = a \cdot sin^4(\pi x^{m_i}) \tag{2.60}$$

where:

$$m_i = \frac{ln(.5)}{ln(x_{M_i})} \tag{2.61}$$

where x_{M_i} is the bump center, x_s is the surface node under the bump functions' influences, x is the location of the surface node in question, and a is the amplitude of the bump function as well as the design variable. The bump functions are spread out evenly along the chord for all cases.

2.6.2 Mesh Deformation

There are two separate mesh deformation schemes used in this work to compute the mesh motion for design optimization. The first is a local method based off of Batina's spring analogy [50], where a system of linear equations is solved for mesh deformations based off the surface deformations.

$$[K]\,\Delta x_v = \Delta x_s \tag{2.62}$$

$[K]$ is the sparse spring stiffness matrix which is only non-zero where the nodes are connected by an edge. In those entries, the stiffness value is:

$$k_{ik} = -\frac{1}{r_{ik}^2} \tag{2.63}$$

and the diagonal elements of the matrix are the sum of the edge stiffnesses going into each node:

$$k_{ii} = \sum_{k=1}^{n_{con}} \frac{1}{r_{ik}^2} \tag{2.64}$$

where r_{ik} is the distance between nodes i and k. This equation is solved using point Jacobi. This method is slow without multigrid, and as such in this work most of the results use the second mesh deformation scheme which is more robust and faster when compared to the spring analogy approach.

The second mesh deformation scheme is a global inverse distance weighted method [51]. The displacement at each node in the mesh is determined as follows:

$$\Delta x_k = \frac{\sum w_{ik}(\vec{r})\vec{\delta}_i(\vec{r})}{\sum w_{ik}(\vec{r})} \tag{2.65}$$

where the w_{ik} function is the weight of surface point i on interior node k, and the delta function is the surface points displacement. The weight function is calculated as follows:

$$w_{ik}(\vec{r}) = A_i \left[\left(\frac{L_{def}}{\|\vec{r_k} - \vec{r_i}\|} \right)^a + \left(\frac{\alpha L_{def}}{\|\vec{r_k} - \vec{r_i}\|} \right)^b \right] \tag{2.66}$$

where L_{def} is the characteristic length of the body, a and b are experimentally derived constants, and α is a fraction of L_{def} to reserve for stiffer deformation.

2.7 Sensitivity Computation Methods

2.7.1 Finite Differences

The finite difference method begins from an objective function $L(U(D), D)$, depending on the conservative variable vector (U) and the design variable vector (D). Each design variable is perturbed independently and the primal system (analysis) solved, with the difference in the objective function recorded and divided by the perturbation:

$$\frac{dL}{dD_i} = \frac{L(U(D + \delta D_i), D + \delta D_i) - L(U, D)}{\delta D_i} \tag{2.67}$$

This method is clearly expensive as it requires n+1 analysis runs for n design variables. Furthermore, the finite difference approach suffers from round off error for $\epsilon << 1$ [4]. This approach can be improved using the complex step method. This is done for scalar functions by using a complex perturbation $\epsilon \delta D_i$ to the inputs:

$$\frac{dL}{dD_i} = Imag\left(\frac{L(U(D + i\epsilon\delta D_i), D + i\epsilon\delta D_i) - L(U, D)}{\epsilon \delta D_i}\right) \tag{2.68}$$

This can be extended to vector valued functions, where this corresponds to the Frechét derivative. Beginning with a function $f(u)$, where f and u are both vector valued, the Frechét derivative below is the derivative of f with respect to u along the vector v.

$$\frac{\partial f}{\partial u} v = Imag\left(\frac{f(u + i\epsilon v) - f(u)}{\epsilon}\right) \tag{2.69}$$

This identity is used frequently for Jacobian free Newton Krylov (JFNK) solvers so as to avoid having to implement the full linearization of the residual operator; this can also be used to compute the tangent system, or the forward linearization. This method cannot be used for the adjoint system, as a result some research groups use coloring techniques to form the full Jacobian matrix in the minimum number of residual evaluations [52] and then transpose the Jacobian in order to solve the adjoint system that is shown later.

2.7.2 Tangent Formulation

For an aerodynamic optimization problem, consider an objective functional $L(u(D), x(D))$, for example lift or drag, where u is the conservative flow variable vector, and x is the mesh coordinate vector. In order to obtain an expression for the sensitivities take the derivative of the objective functional. [53]:

$$\frac{dL}{dD} = \frac{\partial L}{\partial x_v}\frac{dx_v}{dD} + \frac{\partial L}{\partial u}\frac{du}{dx_v}\frac{dx_v}{dD} \tag{2.70}$$

For the above expression $\frac{\partial L}{\partial x_v}$ and $\frac{\partial L}{\partial u}$ can be directly obtained by differentiating the subroutine that calculates the objective function. If $\frac{dx_v}{dD}$ is calculated by solving an implicit mesh deformation scheme, like Batina's spring analogy shown earlier [50], the mesh sensitivity equation is:

$$[K]\frac{dx_v}{dD_i} = \frac{dx_s}{dD_i} \tag{2.71}$$

where $[K]$ is the global stiffness matrix, x_v is the interior mesh coordinate vector and x_s is the vector of coordinates on the geometry. For an explicit mesh motion method like the global inverse distance weighted method used in this work [51], the mesh sensitivities are computed explicitly:

$$\frac{dx_v}{dD_i} = [K]\frac{dx_s}{dD_i} \tag{2.72}$$

where $[K]$ is the matrix formed from the explicit method. For both methods, $\frac{dx_s}{dD_j}$ is calculated through differentiating the shape design variables.

It is not possible to obtain $\frac{du}{dD}$ through linearization of the subroutines in the code without linearizing the entire primal solution process, as will be covered in later sections. In order to solve for this term one can instead use the constraint that for a fully converged flow, the discretized form of the governing equations is satisfied and the residual is equal to zero – denoted by $R(u(D), x(D)) = 0$. By taking the derivative of the residual operator one may obtain the equation below.

$$\left[\frac{\partial R}{\partial x_v}\right]\frac{dx_v}{dD} + \left[\frac{\partial R}{\partial u}\right]\frac{du}{dD} = 0 \tag{2.73}$$

The sensitivity of the residual to the design variables must be isolated to obtain the tangent system.

$$\left[\frac{\partial R}{\partial u}\right]\frac{du}{dD} = -\left[\frac{\partial R}{\partial x_v}\right]\frac{dx_v}{dD} \tag{2.74}$$

$$\frac{dL}{dD} = \frac{\partial L}{\partial x_v}\frac{dx_v}{dD} - \frac{\partial L}{\partial u}\left[\frac{\partial R}{\partial u}\right]^{-1}\left[\frac{\partial R}{\partial x_v}\right]\frac{dx_v}{dD} \tag{2.75}$$

The $\left[\frac{\partial R}{\partial x_v}\right]$ on the right hand side of the equation above is calculated as follows:

$$\left[\frac{\partial R}{\partial x_v}\right]_2 = \left[\frac{\partial R}{\partial x_v} + \frac{\partial R}{\partial u_L}\frac{\partial u_L}{\partial x_v} + \frac{\partial R}{\partial u_R}\frac{\partial u_R}{\partial x_v}\right] \tag{2.76}$$

In order to compute the matrix vector product of $\left[\frac{\partial R}{\partial u}\right]_2 \delta x_v$, it is calculated as:

$$\begin{aligned}
\delta R &= \frac{\partial R}{\partial x_v}\delta x_v \\
\delta u_L &= \frac{\partial u_L}{\partial x_v}\delta x_v \\
\delta u_R &= \frac{\partial u_R}{\partial x_v}\delta x_v \\
\frac{dR}{dx_v} &= \delta R + \frac{\partial R}{\partial u_L}\delta u_L + \frac{\partial R}{\partial u_R}\delta u_R
\end{aligned} \tag{2.77}$$

As before, this break down of the matrix vector product allows for easy verification of the linearization. By using a random vector perturbation δx_v added to the nodal coordinate vector x_v the linearization of the residual operator with respect to the nodal coordinate can be checked through perturbation of the nodes while holding the reconstructed states to be unchanged:

$$\frac{\partial R}{\partial x_v}\delta x_v = \frac{Imag(R(u_L, u_R, x_v + i\epsilon\delta x_v))}{\epsilon} \tag{2.78}$$

the linearization of left side reconstructed state with respect to the nodal coordinates can be checked using the identity below:

$$\frac{\partial u_L}{\partial x_v}\delta x_v = \frac{Imag(u_L(U, x_v + i\epsilon\delta x_v))}{\epsilon} \tag{2.79}$$

The linearization of the right constructed state can be verified similarly.

$$\frac{\partial u_R}{\partial x_v}\delta x_v = \frac{Imag(u_R(U, x_v + i\epsilon\delta x_v))}{\epsilon} \tag{2.80}$$

Finally, the linearization of the entire residual operator with respect to the nodal coordinate vector can be verified as below.

$$\frac{dR}{dx_v}\delta x_v = \frac{R(u_L(U, x_v + i\epsilon\delta x_v), u_R(U, x_v + i\epsilon\delta x_v), x_v + i\epsilon\delta x_v)}{\epsilon} \tag{2.81}$$

The entire chain from the residual to the design variable can be verified using perturbations of the design variables:

$$\frac{dR}{dx_v}\frac{dx_v}{dx_s}\frac{dx_s}{dD}\delta D = \frac{R(u_L(U, x_v(x_s(D + i\epsilon\delta D))), u_R(U, x_v(x_s(D + i\epsilon\delta D))), x_v(x_s(D + i\epsilon\delta D)))}{\epsilon} \tag{2.82}$$

combining this verification with the verification of the flow Jacobian allows the researcher to move on to the tangent and adjoint systems confident that the manual linearizations are correct.

In this work, the linear system in equation 2.74 is solved using linear solvers discussed previously. The linear system is created using hand differentiated subroutines to provide the left hand matrix $\left[\frac{\partial R}{\partial u}\right]$ and the right hand side $\left[\frac{\partial R}{\partial x_v}\right]\frac{dx_v}{dD}$, which scales with the design variables. It is then possible to substitute $\frac{du}{dD}$ into equation (2.70) to obtain the final sensitivities. The tangent system is also called the forward linearization, and when looking at the flow of information in the tangent the reason for this terminology becomes clear; The tangent follows the forward linearization process of equation 2.58. The tangent begins from a perturbation to the design variables and moves that perturbation to the airfoil geometry and then onto the volume mesh. The change in the volume mesh is then passed onto a change in the conservative variables, and finally the sensitivity of the objective is computed.

2.7.3 Discrete Adjoint Formulation

This section discusses the discrete adjoint and its application to the design process and error estimation. In this work the discrete adjoint is computed used, it is implemented by first discretizing the nonlinear problem and then by linearizing the PDE to obtain the adjoint [54]. In the continuous form, the PDE

itself is linearized and then discretized to obtain the adjoint. The discrete adjoint has become widespread due to the ease of implementation, the low barrier to understanding (it falls out naturally from the chain rule), and the fact that it can be easily verified (the discrete adjoint sensitivities should match the tangent provided ones and the complex ones for a simulation that is converged to machine zero). In the limit of an infinitesimally fine mesh, the continuous and discrete adjoint solutions should correspond exactly to one another for a dually consistent discretization. The adjoint formulation begins with the same sensitivity equation as in the tangent equation:

$$\frac{dL}{dD} = \frac{\partial L}{\partial x_v}\frac{dx_v}{dx_s}\frac{dx_s}{dD} + \frac{\partial L}{\partial u}\frac{du}{dx_v}\frac{dx_v}{dx_s}\frac{dx_s}{dD} \tag{2.83}$$

Using the condition $R(u(D), D) = 0$, one can return to equation (2.74) and pre-multiply both sides of the equation by the inverse Jacobian matrix to obtain:

$$\frac{\partial u}{\partial D} = -\left[\frac{\partial R}{\partial u}\right]^{-1}\left[\frac{\partial R}{\partial x_v}\right]\frac{dx_v}{dx_s}\frac{dx}{dD} \tag{2.84}$$

Substituting the above expression into the sensitivity equation yields:

$$\frac{dL}{dD} = \frac{\partial L}{\partial x_v}\frac{dx_v}{dx_s}\frac{dx_s}{dD} - \frac{\partial L}{\partial u}\left[\frac{\partial R}{\partial u}\right]^{-1}\left[\frac{\partial R}{\partial x_v}\right]\frac{dx_v}{dx_s}\frac{dx_s}{dD} \tag{2.85}$$

An adjoint variable $\mathbf{\Lambda}$ is defined such that:

$$\mathbf{\Lambda}^T = -\frac{\partial L}{\partial u}\left[\frac{\partial R}{\partial u}\right]^{-1} \tag{2.86}$$

which gives an equation for the adjoint variable:

$$\left[\frac{\partial R}{\partial u}\right]^T \mathbf{\Lambda} = -\left[\frac{\partial L}{\partial u}\right]^T \tag{2.87}$$

This linear system can be solved and the sensitivities for the objective function can therefore be obtained as follows:

$$\frac{dL}{dD} = \left[\frac{\partial L}{\partial x_v} + \mathbf{\Lambda}^T\frac{\partial R}{\partial x_v}\right]\frac{dx_v}{dx_s}\frac{dx_s}{dD} \tag{2.88}$$

It is advisable to then define a mesh adjoint variable $\mathbf{\Lambda}_{\mathbf{x_s}}$, to keep the adjoint's scaling properties and avoid solving many mesh motion systems. For the implicit mesh motion system the mesh adjoint is computed by solving the equation below.

$$[K]^T\,\mathbf{\Lambda}_{\mathbf{x_s}} = \left[\frac{\partial L}{\partial x_v}\right]^T + \left[\frac{\partial R}{\partial x_v}\right]^T\mathbf{\Lambda} \tag{2.89}$$

Similarly, for an explicit mesh motion system the mesh adjoint is computed as follows:

$$\mathbf{\Lambda}_{\mathbf{x_s}} = [K]^T\left[\left[\frac{\partial L}{\partial x_v}\right]^T + \left[\frac{\partial R}{\partial x_v}\right]^T\mathbf{\Lambda}\right] \tag{2.90}$$

and the final expression for sensitivity is given as:

$$\frac{dL}{dD} = \mathbf{\Lambda}_{\mathbf{x_s}}{}^T\frac{dx_s}{dD} \tag{2.91}$$

The adjoint system is of interest, because as mentioned previously, it results in an equation for the sensitivity that does not scale with the number of design variables. The adjoint is called the reverse linearization as it reverses the flow of information from the tangent and will proceed in the reverse of the behavior shown in 2.58. It begins with a perturbation to the objective that results in a perturbation of the residual operator, the perturbation in the residual operator is then transferred to the volume mesh. The volume mesh perturbations are then propagated onto the surface mesh nodes, and from there they are moved onto the design variables; thus completing the reverse propagation from the objective function to the design variables.

2.8 Parallelism

When moving to more expensive problems, parallelization become necessary in order to get results within a reasonable time frame; this code was parallelized using OpenMP directives in Fortran due to the ease of implementation into existing codes. This was done using a cell and edge coloring scheme, and since this is a cell-centered finite volume code, many properties of the parallelization are guaranteed, which makes the cell coloring simpler. It is guaranteed that there will be at most 4 colors for the cell coloring algorithm shown in algorithm 7, with the colored mesh shown in Figure 2.2.

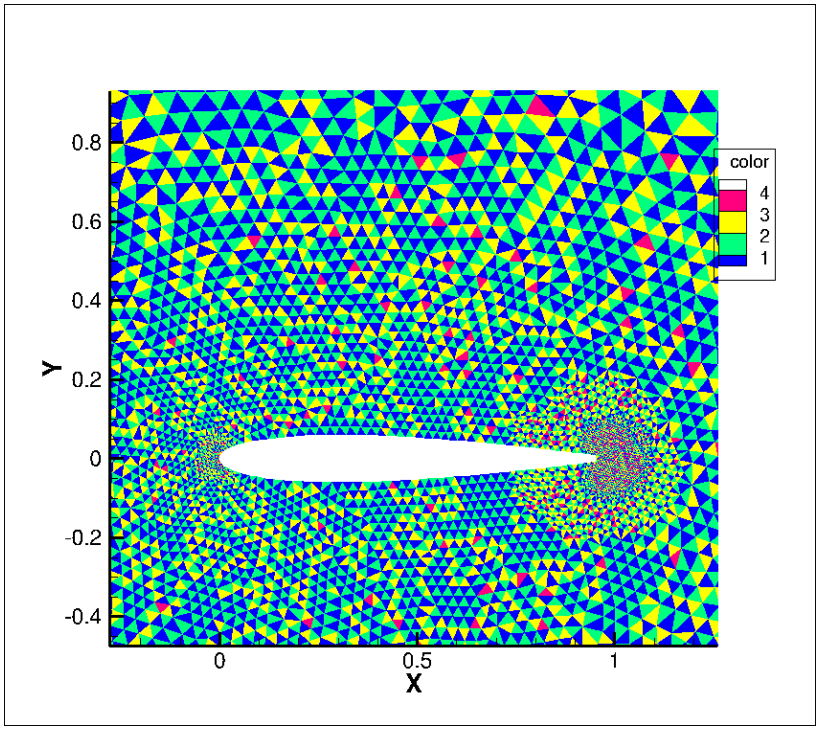


Figure 2.2: Cell coloring for arbitrary unstructured mesh

Even though an arbitrary unstructured triangular mesh is guaranteed to have no more than 4 colors for an optimal edge coloring algorithm with switching [55], algorithm 8 shows the greedy algorithm used in this

Algorithm 7 Cell Coloring Algorithm

```
1: procedure Color Cells
2:     for cellnum = 1, ..., nCells do
3:         for col = 1, ..., 4 do
4:             tf ← FALSE
5:             for neighbor = 1, .., 3 do
6:                 coln ← cellArray(neighbor)
7:                 if col = coln then
8:                     tf ← TRUE
9:                     exit
10:            if tf == FALSE then
11:                cellArray(cellnum) ← col
12:                exit
13:    Reorder cells by color
```

work which is guaranteed to have no more than 5 colors. While this is suboptimal, optimal algorithms of parallelization and coloring is not the thrust of this work.

Algorithm 8 Edge Coloring Algorithm

```
1: procedure Color Edges
2:     for edgenum = 1, ..., nEdges do
3:         for col = 1, ..., 5 do
4:             tf ← FALSE
5:             if interioredge, neNeighbors ← 4
6:             if boundaryedge, neNeighbors ← 2
7:             for neighbor = 1, .., neNighbors do
8:                 coln ← edgeArray(neighbor)
9:                 if col = coln then
10:                    tf ← TRUE
11:                    exit
12:            if tf == FALSE then
13:                edgeArray(edgenum) ← col
14:                exit
15:    Reorder edges by color
```

Figure 2.3 shows the scaling of the different parts of this framework by showing the speedup compared to the serial code. The residual evaluations scale poorly and as such scalable solvers should have fewer residual evaluations and spend more time in the more scalable parts of the code. The analysis mode scales better than the residual evaluations as this was run with a Newton-Krylov solver and the linear solver portion scales better, as is shown for the tangent and the adjoint modes. The steady state tangent and adjoint systems scale quite well, as the bulk of the computational expense is easily parallelized as it is primarily work done with the linear solver. The pseudo-time accurate adjoint scales similarly to the analysis, which accords with intuition as it is the linearization of the analysis and requires more of the less scalable Jacobian and residual evaluations rather than spending more time on the scalable linear solver. One benefit of using the first-order accurate Jacobian as the smoother – as shown in the linear solver section– is ease of parallelism and coloring.

Because of the nearest neighbors stencil, the triangular element coloring for parallelization is limited to four colors rather than ten, and it requires no additional subroutines to compute the coloring.

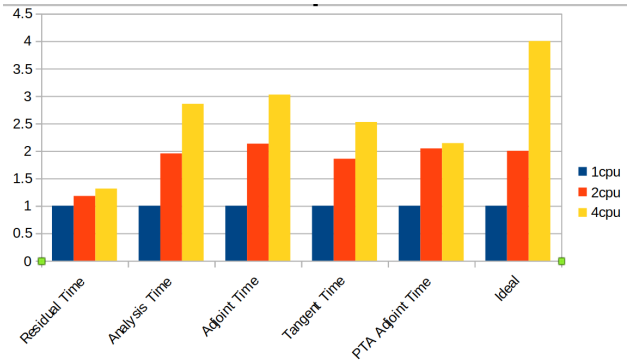


Figure 2.3: Summary of OpenMP scaling

Chapter 3

Pseudo-time Accurate Approaches for Design for the Tangent and Adjoint Problems for Simulations at Partial Convergence

One important aspect to note about the following derivations is that due to the theoretical overlap between the one shot adjoint approach [56] and the methods shown here to compute sensitivities at partial convergence is the similar condition on convergence. The one shot method guarantees convergence of the adjoint problem by the linearization of the given fixed point iteration, the method presented here linearizes the entire history of the fixed point iteration. The one shot method uses the fact that the fixed point iteration is contractive to say that the transpose linearization can converge, similarly it can be said that the full linearization of the fixed point linearization history can converge. The motivation behind the pseudo-time accurate tangent and adjoint formulations rests in switching the constraint from $R = 0$, which is only true at the state of a fully converged primal problem, to a constraint that is true from iteration to iteration through linearization of the fixed-point iteration used to solve the nonlinear problem; by definition this formulation is solver dependent.

3.1 Pseudo-time Accurate Tangent Problem for Design Optimization

3.1.1 Explicit Solve (Forward Euler)

The pseudo-time accurate tangent is first developed for a forward Euler (explicit) solver, this is the simplest linearization shown in this work. The fixed-point iteration is written below.

$$u^{k+1} = u^k + CFL\Delta t R \tag{3.1}$$

Where the CFL is a user defined parameter chosen to assist with stability as described previously in the section on nonlinear solvers. The equation for the local explicit time step limit Δt is given as:

$$\Delta t_i = \frac{r_i}{\sqrt{(u^2 + v^2)} + c} \tag{3.2}$$

where r_i is the circumference of the inscribed circle for mesh cell i, u and v are the horizontal and vertical velocity components respectively, and c is the speed of sound in the triangular element.

Simplifying the notation from the steady state adjoint derivation will make the following derivations easier to follow and so the convention is established that $\frac{\partial}{\partial D} = \frac{\partial}{\partial x_v}\frac{dx_v}{dx_s}\frac{dx_s}{dD}$ and $\frac{d}{dD} = \frac{d}{dx_v}\frac{dx_v}{dx_s}\frac{dx_s}{dD}$.Taking the derivative of each side of the equation (3.1) returns:

$$\frac{du^{k+1}}{dD} = \frac{du^k}{dD} + CFL\Delta t \left[\frac{\partial R}{\partial D} + \left[\frac{\partial R}{\partial u}\right]_2 \frac{du}{dD}^k\right] \tag{3.3}$$

which gives a simple expression for a pseudo-time-accurate sensitivity of the conservative variable vector to the design variable vector. This expression was derived using the assumption of constant time-steps, and can be extended to varying time steps by including a term for the varying time-step:

$$\frac{du^{k+1}}{dD} = \frac{du^k}{dD} + CFL\Delta t \left[\frac{\partial R}{\partial D} + \left[\frac{\partial R}{\partial u}\right]_2 \frac{du}{dD}^k\right] + CFL\left[\frac{\partial \Delta t}{\partial D} + \frac{\partial \Delta t}{\partial u^k}\frac{du^k}{dD}\right] R \tag{3.4}$$

3.1.2 Low Storage Explicit Runge-Kutta (LSERK45) Solver

This nonlinear solver/fixed point iteration is explicit, like the forward Euler scheme, but is a multi-stage solver. The pseudo-time evolution is written as follows:

$$u^{k,l} = u^{k,0} + CFL\alpha^{l-1}\Delta t R(u^{k,l-1}) \tag{3.5}$$

where Δt is computed based off the stage at the initial stage and held frozen throughout the sub-stages.

$$\Delta t = \Delta t(u^{k,0}) \tag{3.6}$$

Taking the derivative of each side returns the below expression for the accumulation of the sensitivities of the conservative variable vector to the design variable vector.

$$\frac{du^{k,l+1}}{dD} = \frac{du^{k,0}}{dD} + CFL\alpha^l \left[\left(\frac{\partial \Delta t}{\partial u^{k,0}} \frac{du^{k,0}}{dD} + \frac{\partial \Delta t}{\partial D} \right) R(u^{k,l}) + \Delta t \left(\frac{\partial R}{\partial D} + \left[\frac{\partial R}{\partial u} \right]_2 \frac{du^{k,l}}{dD} \right) \right] \tag{3.7}$$

3.1.3 Newton Solver

This section shows three different methods to handle the linearization of the inexact-Newton solver, and it will discuss the different assumptions that go into each method and the practicality of implementation thereof. The assumptions relate to how the linearization of the approximate inverse of the left hand matrix is computed/estimated. Newton's method was implemented using pseudo-transient continuation (PTC) with a BDF1 scheme in the context of a quasi-Newton method. This is specifically not a full Newton method as it typically uses an approximation to the Jacobian matrix which is the Jacobian of the first-order spatially accurate discretization, and the resulting linear system is only approximately solved. For Newton's method the time-stepping procedure is written as first shown in equation (2.45) and reproduced in equation 3.8.

$$u^k = u^{k-1} + \Delta u \tag{3.8}$$

Δu is computed by solving the following system of linear equations.

$$[P]\,\Delta u = -R(u) \tag{3.9}$$

Substituting in the expression for Δu into the fixed point iteration in equation (3.8) returns the final form of this equation.

$$u^k = u^{k-1} - [P_{k-1}]^{-1} R \tag{3.10}$$

Here $[P_{k-1}]$ is a first-order spatially accurate Jacobian augmented with a diagonal term to ensure that it is diagonally dominant, shown in equation (3.11).

$$[P_{k-1}] = \left[\frac{\partial R}{\partial u^{k-1}} \right]_1 + \frac{vol}{\Delta t CFL} \tag{3.11}$$

The subscript on the Jacobian above denotes that it is a first-order accurate Jacobian; in this work the subscripts of 1 and 2 will denote first and second-order spatially accurate Jacobian matrices respectively. The derivatives of the nonlinear solution process are written below.

$$\frac{du}{dD}^k = \frac{du}{dD}^{k-1} - [P_{k-1}]^{-1} \left[\frac{\partial R}{\partial D} + \left[\frac{\partial R}{\partial u^{k-1}} \right]_2 \frac{du}{dD}^k \right] - R(u^{k-1}) \left[\frac{\partial [P_{k-1}]^{-1}}{\partial D} + \frac{\partial [P_{k-1}]^{-1}}{\partial u} \frac{du}{dD}^{k-1} \right] \tag{3.12}$$

Equation 3.12 shows the evolution of the sensitivities of the conservative variable vector to the design variable vector through pseudo-time, assuming that it is possible to linearize the approximate matrix inverses commonly used in Newton-Krylov solvers.

3.1.3.1 Tangent System for Newton-Chord Method

If one chooses to neglect the change in the preconditioner due to the flow conditions and the design variables this can simplify equation 3.12 a great deal. By freezing the preconditioner matrix inverse and thereby avoiding taking derivatives of the approximate inverse preconditioner matrix, the second term on the right-hand side of 3.13 is set to zero and the results is Equation 3.13. Since the preconditioner shown in 3.11 is an approximate Jacobian, this corresponds to using a frozen Jacobian, which is also known as a Newton-Chord method. For these methods, where the Jacobian is frozen in pseudo-time and the approximate inversion uses a smoothing operation that is independent of the right hand side and the flow state, this is the exact linearization of the primal solution process. It could also be argued by ergodicity that, for partially converged cases, oscillations about some mean are independent of the initial state, even if the preconditioner in the primal is not frozen in these limit cycle oscillations. Furthermore, the residual in these limit cycle oscillations are very small and will be multiplying this small in magnitude derivative of the preconditioner. Therefore neglecting the derivative of the preconditioner matrix inverse could lead to negligible contributions from this term overall. It could also be investigated whether running for a long enough period of pseudo-time in the limit cycle oscillation zone gives sufficient information to provide a suitable approximation of the tangent system and allows a close approximation of the sensitivities obtained using the complex-step method. Proceeding with the simplified equation:

$$\frac{du}{dD}^{k} = \frac{du}{dD}^{k-1} - [P_{k-1}]^{-1}\left[\frac{\partial R}{\partial D} + \left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 \frac{du}{dD}^{k-1}\right] \tag{3.13}$$

This can then be rewritten as a pseudo-time evolution equation for the sensitivities of the conservative variable vector with respect to the design variable vector.

$$\frac{du}{dD}^{k} = \frac{du}{dD}^{k-1} + \Delta\left(\frac{du}{dD}\right) \tag{3.14}$$

and therefore it is necessary to solve the following linear system:

$$[P_{k-1}]\,\Delta\left(\frac{du}{dD}\right) = -\left[\frac{\partial R}{\partial D} + \left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 \frac{du}{dD}^{k-1}\right] \tag{3.15}$$

It is important to note that the $[P_{k-1}]^{-1}$ in the equation above is not the exact inverse of the preconditioner matrix $[P_{k-1}]$. Rather the algorithm performs the same number of steps to invert it as it did in the primal problem at each nonlinear iteration, as this formulation is drawn directly off the primal solution process. If this iterative process is run at the converged state then it corresponds to the direct differentiation; its adjoint presented later is the dual of that process [57, 58].

3.1.3.2 Tangent System for quasi-Newton Method with Inverse Approximation

Rather than neglecting the change in the preconditioner this section uses a definition of the derivative of the matrix inverse defined as:

$$\frac{d\left[K\right]^{-1}}{dx} = -\left[K\right]^{-1}\left[\frac{dK}{dx}\right]\left[K\right]^{-1} \tag{3.16}$$

This assumption will not be exact for any case in which the linear system solve is not exact to machine precision. However, it will allow for a clearly defined source of error in these computations and makes possible investigations as to how much the tolerance of the linear system affects the sensitivity computation. This also allows for research as to the behavior in near-ergodic limit cycle oscillations. By computing the total derivative of the nonlinear solver– as in equation 3.12– and substituting in the expression from 3.16 into equation 3.12, the below expression is obtained, where $\frac{d[P_k]}{dD}$ is left unexpanded.

$$\frac{du^k}{dD} = \frac{du^{k-1}}{dD} - \left[P_{k-1}\right]^{-1}\left[\left[\frac{\partial R}{\partial u^{k-1}}\right]_2\frac{du^{k-1}}{dD} + \left[\frac{\partial R}{\partial x}\right]\frac{dx}{dD}\right] + \left[P_{k-1}\right]^{-1}\frac{d\left[P_{k-1}\right]}{dD}\left[P_{k-1}\right]^{-1}R(u^{k-1}, D) \tag{3.17}$$

Expanding the total derivative $\frac{d[P_k]}{dD}$ reveals that the derivative of the preconditioner matrix in the right most term is in fact the sum of two matrix vector products as shown below.

$$\begin{aligned}\frac{du^k}{dD} = \frac{du^{k-1}}{dD} &- \left[P_{k-1}\right]^{-1}\left[\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2\frac{du^{k-1}}{dD} + \frac{\partial R}{\partial D}\right] \\ &+ \left[P_{k-1}\right]^{-1}\left[\frac{\partial P_{k-1}}{\partial u^{k-1}}\frac{du^{k-1}}{dD} + \frac{\partial P_{k-1}}{\partial D}\right]\left[P_{k-1}\right]^{-1}R(u^{k-1}, D)\end{aligned} \tag{3.18}$$

The linearization of the preconditioner matrix is an undesirable item to compute as it is the Hessian of the first-order accurate residual operator augmented with the derivative of the mass matrix. Fortunately, this formulation only requires two Hessian vector products that can be obtained through complex perturbations to the conservative variable vector and mesh coordinate vector, and subsequent evaluation of the preconditioner matrix; thus avoiding the need for the full Hessian computation. Furthermore, one of the matrix inverses from equation 3.18 can be removed by reusing the computation of Δu and substituting that value for $\left[P_{k-1}\right]^{-1}R(u^{k-1}, D)$. This returns the equation below.

$$\begin{aligned}\frac{du^k}{dD} = \frac{du^{k-1}}{dD} &- \left[P_{k-1}\right]^{-1}\left[\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2\frac{du^{k-1}}{dD} + \left[\frac{\partial R}{\partial x}\right]\frac{dx}{dD}\right] \\ &+ \left[P_{k-1}\right]^{-1}\left[\frac{\partial P_{k-1}}{\partial u^{k-1}}\frac{du^{k-1}}{dD} + \frac{\partial P_{k-1}}{\partial D}\right]\Delta u\end{aligned} \tag{3.19}$$

This allows for the definition of a pseudo-time accurate evolution equation for the sensitivities of the conservative variables with respect to the design variables.

$$\frac{du}{dD}^{k} = \frac{du}{dD}^{k-1} + \Delta\left(\frac{du}{dD}\right) \tag{3.20}$$

and the linear system is solved as shown below.

$$\left[P_{k-1}\right]\Delta\left(\frac{du}{dD}\right) = -\left[\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2\frac{du^{k-1}}{dD} + \frac{\partial R}{\partial D}\right] + \left[\frac{\partial P_{k-1}}{\partial u^{k-1}}\frac{du^{k-1}}{dD} + \frac{\partial P_{k-1}}{\partial D}\right]\Delta u \tag{3.21}$$

It is important to note that this expression is only exact for exact solution of the linear system, but for those cases this algorithm will provide exact correspondence between this formulation and the complex-step sensitivities. This obviates the need to differentiate the linear system solver, which is intractable for many cases for the forward mode, and even more difficult for the reverse or adjoint mode. The reverse differentiation of a Krylov solver would be an onerous task that would yield little gain. One additional point is that with this method there are no conditions on exact duals of the linear solver, and one can use different linear solvers for each of the primal, tangent and adjoint modes. Please note, that where in the initial formulation– in equation (3.18)– the equation presents 3 approximate linear solves; when grouping like terms and using the already stored information from the primal solve, this algorithm requires only one approximate linear solve as shown in equation 3.21, the same as in the primal problem nonlinear solver.

3.1.3.3 Exact Tangent System for quasi-Newton Method

This section uses Frechét derivatives to compute the linearization of the approximate matrix inverse. Grouping the derivative terms in equation 3.12 returns the following equation.

$$\frac{du}{dD}^{k} = \frac{du}{dD}^{k-1} - \left[\frac{\partial([P_{k-1}]^{-1}R)}{\partial D} + \frac{\partial([P_{k-1}]^{-1}R)}{\partial u^{k-1}}\frac{du^{k-1}}{dD}\right] \tag{3.22}$$

Using complex perturbations to the conservative variable vector of $\frac{du^{k-1}}{dD}$ and to the mesh coordinate vector of $\frac{dx}{dD}$ allows for the Frechét differentiation of the linear solver and the forward in pseudo-time accumulation of the sensitivity of the conservative variables to the design variables.

3.1.4 General Sensitivity Convergence Proof for Approximate Tangent Linearization of the Fixed Point Iteration

Since the tangent method presented in 3.1.3.2 contains the inverse identity approximation from equation 3.16 that is not exact except in the limit of machine precision linear system tolerance, there is an interest in quantifying the error introduced by this approximate linearization of the fixed point iteration. The following proof is for the more general case of linearizing a fixed point iteration that multiplies the residual operator by some operator A where the residual is exactly linearized with respect to its inputs, but the linearization of A has some general approximation. This begin withs a fixed point iteration:

$$u^{k+1} = N(u^k, D) = u^k + H(u^k, D) = u^k + A(u^k, D)R(u^k, D) \tag{3.23}$$

where A is some operator dependent on the pseudo-temporal discretization and R is the appropriate residual operator for the spatial discretization. For an exact linearization of the solution process this results in the expression below:

$$\frac{du^{k+1}}{dD} = \frac{dN(u^k)}{dD} = \frac{du^k}{dD} + \frac{dA}{dD}R + A\frac{dR}{dD} \tag{3.24}$$

For an inexact linearization A is inexactly linearized as $\widetilde{\frac{dA}{dD}}$, with $\tilde{x}$ denoting the inexact value of x due to the errors in the linearization of A. The computed linearization of the fixed point iteration with the approximate linearization of A is below:

$$\widetilde{\frac{du^{k+1}}{dD}} = \widetilde{\frac{dN(u^k)}{dD}} = \widetilde{\frac{du^k}{dD}} + \widetilde{\frac{dA}{dD}}R + A\widetilde{\frac{dR}{dD}} \tag{3.25}$$

The equation above shows an error in the total derivative of the residual operator, because while although the partial linearizations of the residual are exact, the total derivative include the inexact linearization of the conservative variables with respect to the design variables.

$$\widetilde{\frac{dR}{dD}} = \frac{\partial R}{\partial D} + \frac{\partial R}{\partial u}\widetilde{\frac{du}{dD}} \tag{3.26}$$

To get the error in the conservative variable linearization it is necessary to subtract the two expressions from one another:

$$\frac{du^{k+1}}{dD} - \widetilde{\frac{du^{k+1}}{dD}} = \frac{du^k}{dD} - \widetilde{\frac{du^k}{dD}} + \frac{dA}{dD}R - \widetilde{\frac{dA}{dD}}R + A\frac{dR}{dD} - A\widetilde{\frac{dR}{dD}} \tag{3.27}$$

$\frac{d\epsilon}{dD}$ is defined as the error in $\frac{du}{dD}$ and used to group like terms:

$$\frac{d\epsilon^{k+1}}{dD} = \frac{d\epsilon^k}{dD} + \left[\frac{dA}{dD} - \widetilde{\frac{dA}{dD}}\right]R + A\left[\frac{dR}{dD} - \widetilde{\frac{dR}{dD}}\right] \tag{3.28}$$

The error in $\frac{dA}{dD}$ is expanded to obtain the below expression.

$$\begin{aligned}\frac{d\epsilon^{k+1}}{dD} = \frac{d\epsilon^k}{dD} &+ \left[\frac{\partial A}{\partial D} + \frac{\partial A}{\partial u^k}\frac{du^k}{dD} - \widetilde{\frac{\partial A}{\partial D}} - \widetilde{\frac{\partial A}{\partial u^k}}\widetilde{\frac{du^k}{dD}}\right]R \\ &+ A\left[\frac{\partial R}{\partial D} + \frac{\partial R}{\partial u^k}\frac{du^k}{dD} - \frac{\partial R}{\partial D} - \frac{\partial R}{\partial u^k}\widetilde{\frac{du^k}{dD}}\right]\end{aligned} \tag{3.29}$$

In the third term on the right hand side the $\frac{\partial R}{\partial D}$ terms cancel out and the following expression is obtained.

$$\begin{aligned}\frac{d\epsilon^{k+1}}{dD} = \frac{d\epsilon^k}{dD} &+ \left[\frac{\partial A}{\partial D} + \frac{\partial A}{\partial u^k}\frac{du^k}{dD} - \widetilde{\frac{\partial A}{\partial D}} - \widetilde{\frac{\partial A}{\partial u^k}}\widetilde{\frac{du^k}{dD}}\right]R \\ &+ A\left[\frac{\partial R}{\partial u^k}\frac{du^k}{dD} - \frac{\partial R}{\partial u^k}\widetilde{\frac{du^k}{dD}}\right]\end{aligned} \tag{3.30}$$

By grouping like terms and using the definition of $\frac{d\epsilon}{dD}$ one obtains the below expression.

$$\begin{aligned}\frac{d\epsilon^{k+1}}{dD} = \frac{d\epsilon^k}{dD} &+ \left[\left(\frac{\partial A}{\partial D} - \widetilde{\frac{\partial A}{\partial D}}\right) + \left(\frac{\partial A}{\partial u^k} - \widetilde{\frac{\partial A}{\partial u^k}}\right)\frac{du^k}{dD} + \frac{\partial A}{\partial u^k}\frac{d\epsilon^k}{dD}\right]R \\ &+ A\left[\frac{\partial R}{\partial u^k}\frac{d\epsilon^k}{dD}\right]\end{aligned} \tag{3.31}$$

By rearranging these terms the below expression is obtained, where the first three terms on the right hand side are the linearization of the fixed point iteration multiplied by $\frac{d\epsilon}{dD}$ and the final term is the error in the approximate linearization multiplied by the residual of the nonlinear problem.

$$\frac{d\epsilon^{k+1}}{dD} = \frac{d\epsilon^k}{dD} + A\left[\frac{\partial R}{\partial u^k}\frac{d\epsilon^k}{dD}\right] + \frac{\partial A}{\partial u^k}\frac{d\epsilon^k}{dD}R + \left[\left(\frac{\partial A}{\partial D} - \widetilde{\frac{\partial A}{\partial D}}\right) + \left(\frac{\partial A}{\partial u^k} - \widetilde{\frac{\partial A}{\partial u^k}}\right)\frac{du^k}{dD}\right]R \tag{3.32}$$

Finally by grouping the first three terms on the right hand side into B^k where $B^k = \frac{\partial N}{\partial u^k}$, the use of the Cauchy-Schwarz inequality returns the inequality below.

$$\left\|\frac{d\epsilon^k}{dD} + A\left[\frac{\partial R}{\partial u^k}\frac{d\epsilon^k}{dD}\right] + \frac{\partial A}{\partial u^k}\frac{d\epsilon^k}{dD}R\right\| < \left\|B^k\right\|\left\|\frac{d\epsilon^k}{dD}\right\| \tag{3.33}$$

B is the derivative of the contractive fixed point iteration, therefore $\|B\| < 1$. As a result, the error in the sensitivities expressed by $\frac{d\epsilon}{dD}$ decreases as the residual decreases and the contractivity of the fixed point iteration progresses through the primal solution process. The triangle inequality is used to show the pseudo-temporal evolution of the error as governed by the equation below.

$$\left\|\frac{d\epsilon^{k+1}}{dD}\right\| < \|B\|\left\|\frac{d\epsilon^k}{dD}\right\| + \left\|\left[\frac{dA}{dD} - \widetilde{\frac{dA}{dD}}\right]R\right\| \tag{3.34}$$

This shows that once the residual is much lower than the sensitivity error, the sensitivity converges as the contractivity of the fixed-point iteration of the nonlinear problem. One can also note, that in cases where A approaches $-\frac{\partial R}{\partial u}^{-1}$ there is no dependence on contractivity of the fixed point in the derivative and the error will be multiplied by the residual in all terms.

3.2 Pseudo-time Accurate Adjoint Problem for Design Optimization

The pseudo-time accurate (PTA) adjoint method is drawn from the derivation of the unsteady adjoint. In this method, the derivation begins from viewing each pseudo-time step as a time step and working backwards through pseudo-time to get the adjoint based off the linearization of the nonlinear solution process. In this derivation the objective function is a pseudo-time averaged functional averaged over the last m steps for a simulation that ran through n pseudo-time steps.

$$L = L(u^n, u^{n-1}, ..., u^{n-m}, D) \tag{3.35}$$

where u^n is the conservative variable vector at the final time step and D is the design variable vector. For the constraint, since $R(u, D) = 0$ cannot be used, as that is not true at each pseudo-time step, the constraint is instead based off the pseudo-time evolution of the solution, the k^{th} constraint will be referred to as G^k, which is the shifted fixed-point at iteration k. Based on the nonlinear solution process used in this work the constraint is dependent only on the old time-step, the new time-step, and the design variables, expressed as below. This constraint is solver dependent and will be different for each nonlinear solution strategy.

$$G = G(u^k, u^{k-1}, D) = 0 \tag{3.36}$$

To begin one must define an augmented objective function with n constraints and n Lagrange multipliers:

$$\begin{aligned} J(D, u^n, u^{n-1}, u^{n-2}, ..., \Lambda^n, \Lambda^{n-1}, ...) = & L(u^n, u^{n-1}, ..., u^{n-m}, D) \\ & + \Lambda^{nT} G^n(u^n(D), u^{n-1}(D), D) \\ & + \Lambda^{n-1T} G^{n-1}(u^{n-1}(D), u^{n-2}(D), D) \\ & + ... \\ & + \Lambda^{1T} G^1(u^1(D), u^0(D), D) \end{aligned} \tag{3.37}$$

Then the derivative of the augmented objective function is taken with respect to the Lagrange multipliers, knowing that each derivative must be 0 at an optimum that satisfies the constraints. This gives:

$$\begin{aligned} \frac{\partial J}{\partial \Lambda^n} &= G^n(u^n(D), u^{n-1}(D), D) = 0 \\ \frac{\partial J}{\partial \Lambda^{n-1}} &= G^{n-1}(u^{n-1}(D), u^{n-2}(D), D) = 0 \\ &... \\ \frac{\partial J}{\partial \Lambda^1} &= G^1(u^1(D), u^0(D), D) = 0 \end{aligned} \tag{3.38}$$

where the initial prescribed state u^0 that is not dependent on the design variables. If the derivative of the augmented objective function is taken with respect to the design variables it returns the sensitivity equation below.

$$\frac{\partial J}{\partial D} = \frac{\partial L}{\partial D} + \Lambda^{nT} \frac{\partial G^n}{\partial D} + \Lambda^{n-1T} \frac{\partial G^{n-1}}{\partial D} + \Lambda^{n-2T} \frac{\partial G^{n-2}}{\partial D} + ... \tag{3.39}$$

In order to get an expression for the adjoint, the derivative of the augmented objective function is taken with respect to the conservative variable vector at different pseudo-time steps.

$$\begin{aligned} \frac{\partial J}{\partial u^n} &= \frac{\partial L}{\partial u^n} + \Lambda^{nT} \frac{\partial G^n}{\partial u^n} = 0 \\ \frac{\partial J}{\partial u^{n-1}} &= \frac{\partial L}{\partial u^{n-1}} + \Lambda^{nT} \frac{\partial G^n}{\partial u^{n-1}} + \Lambda^{n-1T} \frac{\partial G^{n-1}}{\partial u^{n-1}} = 0 \\ \frac{\partial J}{\partial u^{n-2}} &= \frac{\partial L}{\partial u^{n-2}} + \Lambda^{n-1T} \frac{\partial G^{n-1}}{\partial u^{n-2}} + \Lambda^{n-2T} \frac{\partial G^{n-2}}{\partial u^{n-2}} = 0 \\ &... \\ \frac{\partial J}{\partial u^1} &= \frac{\partial L}{\partial u^1} + \Lambda^{2T} \frac{\partial G^2}{\partial u^1} + \Lambda^{1T} \frac{\partial G^1}{\partial u^1} = 0 \end{aligned} \tag{3.40}$$

Using that $L = L(u^n, u^{n-1}, ..., u^{n-m}, D)$ returns the below expression where the source term $\frac{\partial L}{\partial u^k}$ only

appears for the last m pseudo-time steps.

$$\begin{aligned}
\frac{\partial J}{\partial u^n} &= \frac{\partial L}{\partial u^n} + \Lambda^{nT}\frac{\partial G^n}{\partial u^n} = 0 \\
\frac{\partial J}{\partial u^{n-1}} &= \frac{\partial L}{\partial u^{n-1}} + \Lambda^{nT}\frac{\partial G^n}{\partial u^{n-1}} + \Lambda^{n-1T}\frac{\partial G^{n-1}}{\partial u^{n-1}} = 0 \\
&... \\
\frac{\partial J}{\partial u^{n-m}} &= \frac{\partial L}{\partial u^{n-m}} + \Lambda^{n-(m-1)T}\frac{\partial G^{n-(m-1)}}{\partial u^{n-m}} + \Lambda^{n-mT}\frac{\partial G^{n-m}}{\partial u^{n-m}} = 0 \\
\frac{\partial J}{\partial u^{n-(m+1)}} &= \Lambda^{n-mT}\frac{\partial G^{n-m}}{\partial u^{n-(m+1)}} + \Lambda^{n-(m+1)T}\frac{\partial G^{n-(m+1)}}{\partial u^{n-(m+1)}} = 0 \\
&... \\
\frac{\partial J}{\partial u^1} &= \Lambda^{2T}\frac{\partial G^2}{\partial u^1} + \Lambda^{1T}\frac{\partial G^1}{\partial u^1} = 0
\end{aligned} \tag{3.41}$$

Using the equation for the adjoint at the final pseudo-time step returns the following expression for the final state adjoint.

$$\left[\frac{\partial G^n}{\partial u^n}\right]^T \Lambda^n = -\left[\frac{\partial L}{\partial u^n}\right]^T \tag{3.42}$$

Using the other adjoint equations gives a recurrence relation for $k = m+1, ..., n-1$ with a source term of the linearization of the objective function.

$$\frac{\partial G^{k-1}}{\partial u^{k-1}}^T \Lambda^{k-1} = -\frac{\partial G^k}{\partial u^{k-1}}^T \Lambda^k - \left[\frac{\partial L}{\partial u^{k-1}}\right]^T \tag{3.43}$$

Outside of the functional averaging window ($k = 1, 2, ...m$), the source term vanishes, and the recurrence relation is simplified.

$$\frac{\partial G^{k-1}}{\partial u^{k-1}}^T \Lambda^{k-1} = -\frac{\partial G^k}{\partial u^{k-1}}^T \Lambda^k \tag{3.44}$$

Its important to note, that the same result must be possible without the use of Lagrange multipliers. Starting with the objective function:

$$L = L(u^n, D) = L(u^n(u^{n-1}, D), D) = L(u^n(u^{n-1}(u^{n-2}, D), D), D) = ... \tag{3.45}$$

The derivative of the objective function (the sensitivity equation) is written below.

$$\begin{aligned}
\frac{dL}{dD} = &\frac{\partial L}{\partial D} + \frac{\partial L}{\partial u^n}\frac{\partial u^n}{\partial D} \\
&+ \frac{\partial L}{\partial u^n}\frac{\partial u^n}{\partial u^{n-1}}\frac{\partial u^{n-1}}{\partial D} \\
&+ ... \\
&+ \frac{\partial L}{\partial u^n}\frac{\partial u^n}{\partial u^{n-1}}...\frac{\partial u^1}{\partial D} \\
&+ \frac{\partial L}{\partial u^n}\frac{\partial u^n}{\partial u^{n-1}}...\frac{\partial u^0}{\partial D}
\end{aligned} \tag{3.46}$$

Using that the initial condition has no dependence on the design variables it is possible to neglect the last term to get:

$$\frac{dL}{dD} = \frac{\partial L}{\partial D} + \frac{\partial L}{\partial u^n}\frac{\partial u^n}{\partial D} + \frac{\partial L}{\partial u^n}\frac{\partial u^n}{\partial u^{n-1}}\frac{\partial u^{n-1}}{\partial D} + ... + \frac{\partial L}{\partial u^n}\frac{\partial u^n}{\partial u^{n-1}}...\frac{\partial u^1}{\partial D} \tag{3.47}$$

which allows the definition of an adjoint recurrence relation as above.

$$\Lambda^{k-1} = \left[\frac{\partial u^k}{\partial u^{k-1}}\right]^T \Lambda^k \tag{3.48}$$

This returns the sensitivity equation:

$$\frac{dL}{dD} = \frac{\partial L}{\partial D} - \Lambda^{nT}\frac{\partial u^n}{\partial D} - \Lambda^{n-1T}\frac{\partial u^{n-1}}{\partial D} + ... + \Lambda^{1T}\frac{\partial u^1}{\partial D} \tag{3.49}$$

where $\frac{\partial u^n}{\partial D} = \frac{\partial G^n}{\partial D}$ The same can also be shown for a pseudo-time averaged objective function, but for simplicity that is not shown here.

3.2.1 Explicit Solver (forward Euler)

$$u^{k+1} = u^k + CFL\Delta t R \tag{3.50}$$

For this explicit solver the constraint formula for each pseudo-time step is written below. This use of the shifted fixed point as the constraint is consistent throughout this

$$G^k(u^k(D), u^{k-1}(D), D) = u^k - u^{k-1} - CFL\Delta t R(u^{k-1}) = 0 \tag{3.51}$$

Taking the derivative of this constraint returns the three equations below, linearized with respect to the old state (u^{k-1}) the new state (u^k) and the design variables.

$$\begin{aligned}
\frac{\partial G^k}{\partial u^k} &= I \\
\frac{\partial G^k}{\partial u^{k-1}} &= -I - CFL\Delta t\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 \\
\frac{\partial G^k}{\partial D} &= -CFL\Delta t\frac{\partial R(u^{k-1})}{\partial D}
\end{aligned} \tag{3.52}$$

Using the equation for the adjoint at the final pseudo-time step with the constraint derivatives returns:

$$[I]\,\Lambda^n = -\left[\frac{\partial L}{\partial u^n}\right]^T \tag{3.53}$$

By plugging in the constraint derivatives into equation 3.44 returns an adjoint recurrence relation of the form shown in the general derivation.

$$[I]\,\Lambda^{k-1} = -\left[-I - CFL\Delta t\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2\right]^T \Lambda^k - \frac{\partial L}{\partial u^k} \tag{3.54}$$

Starting from the augmented objective function as displayed below and in equation 3.37:

$$\begin{aligned} J(D, u^n, u^{n-1}, u^{n-2}, ..., \Lambda^n, \Lambda^{n-1}, ...) = & L(u^n, u^{n-1}, ..., u^{n-m}, D) \\ & + \Lambda^{nT} G^n(u^n(D), u^{n-1}(D), D) \\ & + \Lambda^{n-1T} G^{n-1}(u^{n-1}(D), u^{n-2}(D), D) \\ & + ... \\ & + \Lambda^{1T} G^1(u^1(D), u^0(D), D) \end{aligned} \tag{3.55}$$

This returns the sensitivity equation:

$$\frac{\partial J}{\partial D} = \frac{\partial L}{\partial D} + \Lambda^{nT} \frac{\partial G^n}{\partial D} + \Lambda^{n-1T} \frac{\partial G^{n-1}}{\partial D} + ... + \Lambda^{1T} \frac{\partial G^1}{\partial D} \tag{3.56}$$

substituting in the constraint derivatives gives the expression for the sensitivities:

$$\begin{aligned} \frac{\partial J}{\partial D} = \frac{\partial L}{\partial D} & - \Lambda^{nT} \left[CFL\Delta t \frac{\partial R(u^{k-1})}{\partial D} \right] \\ & - \Lambda^{n-1T} \left[CFL\Delta t \frac{\partial R(u^{k-2})}{\partial D} \right] \\ & - ... \\ & - \Lambda^{1T} \left[CFL\Delta t \frac{\partial R(u^0)}{\partial D} \right] \end{aligned} \tag{3.57}$$

3.2.1.1 Forward Euler with variable time steps

For variable time steps the equations change as follows: First the constraint derivatives change to include the sensitivity of the variable time step:

$$\begin{aligned} \frac{\partial G^k}{\partial u^k} &= I \\ \frac{\partial G^k}{\partial u^{k-1}} &= -I - CFL\Delta t^{k-1} \left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}} \right]_2 - CFL \frac{\partial \Delta t^{k-1}}{\partial u^{k-1}} R \\ \frac{\partial G^k}{\partial D} &= -CFL\Delta t^{k-1} \frac{\partial R(u^{k-1})}{\partial D} - CFL \frac{\partial \Delta t^{k-1}}{\partial D} R \end{aligned} \tag{3.58}$$

Using the equation for the adjoint at the final pseudo-time step with the constraint derivatives returns as before:

$$[I]\, \Lambda^n = - \left[\frac{\partial L}{\partial u^n} \right]^T \tag{3.59}$$

By plugging in the constraint derivatives into equation 3.44 the adjoint recurrence relation is changed to include the linearization of the pseudo-time step:

$$[I]\, \Lambda^{k-1} = - \left[-I - CFL\Delta t^{k-1} \left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}} \right]_2 - CFL \frac{\partial \Delta t}{\partial u^{k-1}} R \right]^T \Lambda^k - \frac{\partial L}{\partial u^{k-1}} \tag{3.60}$$

This can be applied to sensitivity equation 3.39 where the constraint derivative is expanded to get a final expression for the sensitivity equation:

$$\begin{aligned}\frac{\partial J}{\partial D} = \frac{\partial L}{\partial D} &- \Lambda^{nT}\left[CFL\Delta t^{n-1}\frac{\partial R(u^{n-1})}{\partial D} + CFL\frac{\partial \Delta t}{\partial D}R\right] \\ &- \Lambda^{n-1T}\left[CFL\Delta t^{n-2}\frac{\partial R(u^{n-2})}{\partial D} + CFL\frac{\partial \Delta t}{\partial D}R\right] \\ &- ... \\ &- \Lambda^{1T}\left[CFL\Delta t^{0}\frac{\partial R(u^{0})}{\partial D} + CFL\frac{\partial \Delta t}{\partial D}R\right]\end{aligned} \tag{3.61}$$

Looking at the expression for the sensitivities for the forward Euler with constant time-step (equation 3.57) and with variable time-step (equation 3.61), it is clear that neither expression requires solving a linear system with the Jacobian as the A matrix.

3.2.2 Low Storage Explicit Runge-Kutta Solver

This Runge-Kutta solver gives a more complicated fixed-point iteration as shown in the tangent derivation. The first portion occurs within the time step k and iterates for l = 1 through 5:

$$u^{k,l} = u^{k,0} + CFL\Delta t\alpha^{l-1}R(u^{k,l-1}) \tag{3.62}$$

And the second portion defines the new state:

$$u^{k,0} = u^{k-1,5} \tag{3.63}$$

This leads to two separate constraint equations, the first of which is based of the sub-stage iteration.

$$G^{k,l}(u^{k,l}, u^{k,l-1}, u^{k,0}, D) = u^{k,l} - u^{k,0} - CFL\alpha^{k}\Delta t R(u^{k,l-1}) = 0 \tag{3.64}$$

This has the following derivatives:

$$\begin{aligned}\frac{\partial G^{k,l}}{\partial u^{k,l}} &= I \\ \frac{\partial G^{k,l}}{\partial u^{k,l-1}} &= -CFL\alpha^{k}\Delta t\left[\frac{\partial R(u^{k,l-1})}{\partial u^{k,l-1}}\right]_2 \\ \frac{\partial G^{k,l}}{\partial u^{k,0}} &= -I - CFL\alpha^{k}\frac{\partial \Delta t}{\partial u^{k,0}}R(u^{k,l-1}) \\ \frac{\partial G^{k,l}}{\partial D} &= -CFL\alpha^{k}\left[\frac{\partial \Delta t}{\partial D}R + \Delta t\frac{\partial R}{\partial D}\right]\end{aligned} \tag{3.65}$$

The second constraint is based off the pseudo-time update that assigns a value to $u^{k,0}$ from $u^{k-1,5}$.

$$G^{k,0}(u^{k,0}(D), u^{n-1,5}(D)) = u^{k,0} - u^{k-1,5} = 0 \tag{3.66}$$

This has the following derivatives, which are simple by nature of the simplicity of the update.

$$\begin{aligned}\frac{\partial G^{k,0}}{\partial u^{k,0}} &= I \\ \frac{\partial G^{k,0}}{\partial u^{k-1,5}} &= -I \\ \frac{\partial G^{k,0}}{\partial D} &= 0\end{aligned} \tag{3.67}$$

It is therefore necessary to modify 3.37 and get the expression for the sensitivity equation for a Runge-Kutta solver and take into account the sub-iteration behavior.

$$\begin{aligned}&J(D, u^n, u^{n-1,5}, ..., u^{n-1,0}, ..., \Lambda^n, \Lambda^{n-1,5}, ..., \Lambda^{n-1,0}, ...) \\ &= \\ &L(u^n, D) + \Lambda^{nT} G^{k,0}(u^n(D), u^{n-1,5}(D), D) \\ &+ \Lambda^{n-1,5T} G^{n-1,5}(u^{n-1,5}(D), u^{n-1,4}(D), u^{n-1,0}(D), D) \\ &+ ... \\ &+ \Lambda^{n-1,1T} G^{n-1,1}(u^{n-1,1}(D), u^{n-1,0}(D), D) \\ &+ \Lambda^{n-1,0T} G^{n-1,0}(u^{n-1,0}(D), u^{n-2,5}(D), D) \\ &+ ... \\ &+ \Lambda^{0,1T} G^{0,1}(u^{0,1}(D), u^0(D), D)\end{aligned} \tag{3.68}$$

By taking the derivatives with respect to the states the recurrence relations are obtained. Beginning with $L = L(u^n, ..., u^{n-m}, D)$ returns the below expression:

$$\begin{aligned}\frac{\partial J}{\partial u^n} &= \frac{\partial L}{\partial u^n} + \Lambda^{nT} \frac{\partial G^n}{\partial u^n} = 0 \\ \frac{\partial J}{\partial u^{n-1,5}} &= \Lambda^{nT} \frac{\partial G^{k,0}}{\partial u^{n-1,5}} + \Lambda^{n-1,5T} \frac{\partial G^{n-1,5}}{\partial u^{n-1,5}} = 0 \\ \frac{\partial J}{\partial u^{n-1,4}} &= \Lambda^{n-1,5T} \frac{\partial G^{n-1,5}}{\partial u^{n-1,4}} + \Lambda^{n-1,4T} \frac{\partial G^{n-1,4}}{\partial u^{n-1,4}} = 0 \\ &... \\ \frac{\partial J}{\partial u^{n-1,0}} &= \frac{\partial L}{\partial u^{n-1,0}} + \Lambda^{n-1,5T} \frac{\partial G^{n-1,5}}{\partial u^{n-1,0}} + \Lambda^{n-1,4T} \frac{\partial G^{n-1,4}}{\partial u^{n-1,0}} + \Lambda^{n-1,3T} \frac{\partial G^{n-1,3}}{\partial u^{n-1,0}} \\ &\quad + \Lambda^{n-1,2T} \frac{\partial G^{n-1,2}}{\partial u^{n-1,0}} + \Lambda^{n-1,1T} \frac{\partial G^{n-1,1}}{\partial u^{n-1,0}} + \Lambda^{n-1,0T} \frac{\partial G^{n-1,0}}{\partial u^{n-1,0}} = 0 \\ &... \\ \frac{\partial J}{\partial u^{n-m,0}} &= \frac{\partial L}{\partial u^{n-m,0}} + \Lambda^{n-m,5T} \frac{\partial G^{n-m,5}}{\partial u^{n-m,0}} + \Lambda^{n-m,4T} \frac{\partial G^{n-m,4}}{\partial u^{n-m,0}} + \Lambda^{n-m,3T} \frac{\partial G^{n-m,3}}{\partial u^{n-m,0}} \\ &... \\ \frac{\partial J}{\partial u^1} &= \Lambda^{2T} \frac{\partial G^2}{\partial u^1} + \Lambda^{1T} \frac{\partial G^1}{\partial u^1} = 0\end{aligned} \tag{3.69}$$

Using the equation for the adjoint at the final pseudo-time step returns:

$$\left[\frac{\partial G^n}{\partial u^n}\right]^T \Lambda^n = -\left[\frac{\partial L}{\partial u^n}\right]^T \tag{3.70}$$

The adjoint recurrence relation equation for the pseudo-time update in 3.63 returns the adjoint equation below for $k = 1, 2, ..., n-1$.

$$\left[\frac{\partial G^{k-1,5}}{\partial u^{k-1,5}}\right]^T \Lambda^{k-1,5} = -\left[\frac{\partial G^{k,0}}{\partial u^{k-1,5}}\right]^T \Lambda^{k,0} \tag{3.71}$$

The sub-stage fixed point iteration returns an adjoint expression for $k = 1, 2, ..., n-1$ and $l = 1, 2, 3, 4$:

$$\left[\frac{\partial G^{k,l-1}}{\partial u^{k,l-1}}\right]^T \Lambda^{k,l-1} = -\left[\frac{\partial G^{k,l}}{\partial u^{k,l-1}}\right]^T \Lambda^{k,l} \tag{3.72}$$

The final adjoint equation is applied for $k = 1, 2, ..., n-1$ and $l = 0$, and is complicated due to the presence of the 0 state in all 5 steps of the Runge-Kutta fixed-point iteration:

$$\begin{aligned}\left[\frac{\partial G^{k,0}}{\partial u^{k,0}}\right]^T \Lambda^{k,0} = &-\frac{\partial L}{\partial u^k} - \left[\frac{\partial G^{k,1}}{\partial u^{k,0}}\right]^T \Lambda^{k,1} - \left[\frac{\partial G^{k,2}}{\partial u^{k,0}}\right]^T \Lambda^{k,2} - \left[\frac{\partial G^{k,3}}{\partial u^{k,0}}\right]^T \Lambda^{k,3} \\ &- \left[\frac{\partial G^{k,4}}{\partial u^{k,0}}\right]^T \Lambda^{k,4} - \left[\frac{\partial G^{k,5}}{\partial u^{k,0}}\right]^T \Lambda^{k,5}\end{aligned} \tag{3.73}$$

The specific definitions of the derivatives are substituted into the recurrence relations to get the following equations. The adjoint for the last time step is:

$$[I]^T \Lambda^n = -\left[\frac{\partial L}{\partial u^n}\right]^T \tag{3.74}$$

Substituting in the constraint derivatives into 3.71 returns the adjoint equation to switch from the 5 index to the next state.

$$\Lambda^{m-1,5} = \Lambda^{k,0} \tag{3.75}$$

Substituting the constraint derivatives into equation 3.72 shows that the adjoint relation has no dependence on the time step, because the pseudo-time step depends only on the 0 state.

$$[I]^T \Lambda^{k,l-1} = -\left[CFL\Delta t^k \alpha^{l-1} \left[\frac{\partial R(u^{k,l-1})}{\partial u^{k,l-1}}\right]_2\right]^T \Lambda^{k,l} \tag{3.76}$$

Finally, the constraint for the base state in the Runge-Kutta solver is shown below by substituting in the appropriate derivatives into 3.73, where the derivatives of the pseudo-time step appear.

$$\begin{aligned}[I]^T \Lambda^{k,0} = &-\frac{\partial L}{\partial u^k} - \left[-I - CFL\alpha^0 \frac{\partial \Delta t}{\partial u^{k,0}} R(u^{k,0}) - CFL\alpha^0 \Delta t \frac{\partial R(u^{k,0})}{\partial u^{k,0}}\right]^T \Lambda^{k,1} \\ &- \left[-I - CFL\alpha^1 \frac{\partial \Delta t}{\partial u^{k,0}} R(u^{k,1})\right]^T \Lambda^{k,2} \\ &- \left[-I - CFL\alpha^2 \frac{\partial \Delta t}{\partial u^{k,0}} R(u^{k,2})\right]^T \Lambda^{k,3} \\ &- \left[-I - CFL\alpha^3 \frac{\partial \Delta t}{\partial u^{k,0}} R(u^{k,3})\right]^T \Lambda^{k,4} \\ &- \left[-I - CFL\alpha^4 \frac{\partial \Delta t}{\partial u^{k,0}} R(u^{k,4})\right]^T \Lambda^{k,5}\end{aligned} \tag{3.77}$$

The equation below is the result of taking the partial derivative of equation 3.68.

$$\begin{aligned}\frac{\partial J}{\partial D} = L(u^n, ..., u^{n-m}, D) &+ \Lambda^{nT}\frac{\partial G^{k,0}}{\partial D} \\ &+ \Lambda^{n-1,5T}\frac{\partial G^{n-1,5}}{\partial D} \\ &+ ... \\ &+ \Lambda^{n-1,1T}\frac{\partial G^{n-1,1}}{\partial D} \\ &+ \Lambda^{n-1,0T}\frac{\partial G^{n-1,0}}{\partial D} \\ &+ ... \\ &+ \Lambda^{0,1T}\frac{\partial G^{0,1}}{\partial D}\end{aligned} \tag{3.78}$$

Substituting in the definitions for the constraint derivatives returns:

$$\begin{aligned}\frac{\partial J}{\partial D} = L(u^n, ..., u^{n-m}D) &- \Lambda^{n-1,5T}CFL\alpha^k\left[\frac{\partial \Delta t^{n-1}}{\partial D}R(u^{n-1,4}) + \Delta t^{n-1}\frac{\partial R(u^{n-1,4})}{\partial D}\right] \\ &- ... \\ &- \Lambda^{n-1,1T}CFL\alpha^k\left[\frac{\partial \Delta t^{n-1}}{\partial D}R(U^{n-1,0}) + \Delta t^{n-1}\frac{\partial R(u^{n-1,0})}{\partial D}\right] \\ &- ... \\ &- \Lambda^{0,5T}CFL\alpha^k\left[\frac{\partial \Delta t^{0}}{\partial D}R(u^{0,4}) + \Delta t\frac{\partial R(u^{0,4})}{\partial D}\right] \\ &- ... \\ &- \Lambda^{0,1T}CFL\alpha^k\left[\frac{\partial \Delta t^{0}}{\partial D}R(u^{0,0}) + \Delta t\frac{\partial R(u^{0,0})}{\partial D}\right]\end{aligned} \tag{3.79}$$

3.2.3 Newton Solver

3.2.3.1 Adjoint Computed Sensitivities for Newton-Chord Method

It is necessary to refer once again to equation (3.10)

$$u^k = u^{k-1} - [P_{k-1}]^{-1} R \tag{3.80}$$

To proceed all terms are moved to one side and the following equation is obtained as the constraint, through use of the shifted fixed point iteration.

$$G^k(u^k(D), u^{k-1}(D), D) = u^k - u^{k-1} + [P_{k-1}]^{-1} R(u^{k-1}) = 0 \tag{3.81}$$

The preconditioner matrix is defined as seen previously in (3.11):

$$[P_k] = \left[\frac{\partial R}{\partial u^k}\right]_1 + \frac{vol}{\Delta t^k CFL^k} \tag{3.82}$$

The next step is to take the derivatives of the constraint equations. These are written as follows.

$$\begin{aligned}
\frac{\partial G^k}{\partial u^k} &= I \\
\frac{\partial G^k}{\partial u^{k-1}} &= -I + \frac{\partial \left[P_{k-1}\right]^{-1} R(u^{k-1})}{\partial u^{k-1}} = -I + \left[P_{k-1}\right]^{-1} \left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 + \frac{\partial \left[P_{k-1}\right]^{-1}}{\partial u^{k-1}} R(u^{k-1}) \\
\frac{\partial G^k}{\partial D} &= -I + \frac{\partial \left[P_{k-1}\right]^{-1} R(u^{k-1})}{\partial D} = \left[P_{k-1}\right]^{-1} \frac{\partial R(u^{k-1})}{\partial D} + \frac{\partial \left[P_{k-1}\right]^{-1}}{\partial D} R(u^{k-1})
\end{aligned} \tag{3.83}$$

By using the same logic as in the pseudo-time accurate Newton-Chord tangent method, the preconditioner is frozen and its derivatives are set to 0.

$$\begin{aligned}
\frac{\partial G^k}{\partial u^k} &= I \\
\frac{\partial G^k}{\partial u^{k-1}} &= -I + \left[P_{k-1}\right]^{-1} \left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 \\
\frac{\partial G^k}{\partial D} &= \left[P_{k-1}\right]^{-1} \frac{\partial R(u^{k-1})}{\partial D}
\end{aligned} \tag{3.84}$$

For a Newton-Chord solver, this is the exact linearization. However, for an inexact quasi-Newton scheme, as was discussed in the tangent formulation, this is only an approximation of the exact sensitivity equations, with the possibility being to run long enough in the oscillatory portion of the convergence history to get a good enough adjoint approximation to compute sensitivities, or to freeze the preconditioner inverse for exact sensitivities. Using the equation for the adjoint at the final pseudo-time step with the constraint derivatives returns the same initial source term.

$$\left[I\right] \Lambda^n = -\left[\frac{\partial L}{\partial u^n}\right]^T \tag{3.85}$$

Substituting in the constraint derivatives into equation (3.44) returns the following recurrence relation

$$\left[I\right] \Lambda^{k-1} = -\left[-I + \left[P_{k-1}\right]^{-1} \left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2\right]^T \Lambda^k \tag{3.86}$$

This recurrence relation can be written in delta form so that it more closely follows the primal problem. To this end, a $\Delta\Lambda$ is defined such that $\Lambda^{k-1} = \Lambda^k + \Delta\Lambda$, which gives the recurrence relation as:

$$\Delta\Lambda = -\left[\left[P_{k-1}\right]^{-1} \left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2\right]^T \Lambda^k \tag{3.87}$$

distributing the transpose allows the rewriting of the equation.

$$\Delta\Lambda = -\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2^T \left[P_{k-1}\right]^{-T} \Lambda^k \tag{3.88}$$

This motivates the definition of a secondary adjoint variable for each recurrence relation which requires the solution of a system of linear equations at each iteration which follows the behavior of the primal problem and the tangent lineatization.

$$\left[P_{k-1}\right]^T \psi^k = \Lambda^k \tag{3.89}$$

Using the secondary adjoint variable in the delta form of the adjoint recurrence relation returns the equation below.

$$\Delta\Lambda = \left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2^T \psi^k \tag{3.90}$$

It is important to note that, as in the tangent mode, to obtain exact correspondence, these linear systems must be solved as the exact dual of the primal solve, as this algorithm transposes exactly the primal solve. Substituting in the appropriate constraint derivatives into equation 3.39 returns the following equation for the sensitivities.

$$\frac{dJ}{dD} = \frac{\partial L}{\partial D} + \Lambda^{nT}\left[P_{n-1}\right]^{-1}\frac{\partial R(u^{n-1})}{\partial D} + \Lambda^{n-1T}\left[P_{n-2}\right]^{-1}\frac{\partial R(u^{n-2})}{\partial D} + ... + \Lambda^{1T}\left[P_0\right]^{-1}\frac{\partial R(u^0)}{\partial D} \tag{3.91}$$

The sensitivity equation above has an additional linear solve at each nonlinear iteration for each design variable. This scales as the tangent and is highly undesirable. The secondary adjoint variable can be used in the sensitivity equation (3.39) and returns an expression without any additional linear solves.

$$\frac{dJ}{dD} = \frac{\partial L}{\partial D} + \psi^{nT}\frac{\partial R(u^{n-1})}{\partial D} + \psi^{n-1T}\frac{\partial R(u^{n-2})}{\partial D} + ... + \psi^{1T}\frac{\partial R(u^0)}{\partial D} \tag{3.92}$$

3.2.3.2 Adjoint Computed Sensitivities for quasi-Newton Method with Inverse Identity

This derivation begins with the derivatives of the constraint equations as shown in equation (3.83), before the simplification shown for the Newton-Chord method. These are written as follows.

$$\begin{aligned}
\frac{\partial G^k}{\partial u^k} &= I \\
\frac{\partial G^k}{\partial u^{k-1}} &= -I + \left[P_{k-1}\right]^{-1}\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 + \frac{\partial\left[P_{k-1}\right]^{-1}}{\partial u^{k-1}}R(u^{k-1}) \\
\frac{\partial G^k}{\partial D} &= \left[P_{k-1}\right]^{-1}\frac{\partial R(u^{k-1})}{\partial D} + \frac{\partial\left[P_{k-1}\right]^{-1}}{\partial D}R(u^{k-1})
\end{aligned} \tag{3.93}$$

By using the differentiation of matrix inverse first shown in equation (3.16) and reproduced below

$$\frac{d\left[K\right]^{-1}}{dx} = -\left[K\right]^{-1}\left[\frac{dK}{dx}\right]\left[K\right]^{-1} \tag{3.94}$$

the derivative of the constraint term is obtained and shown below.

$$\begin{aligned}
\frac{\partial G^k}{\partial u^k} &= I \\
\frac{\partial G^k}{\partial u^{k-1}} &= -I + \left[P_{k-1}\right]^{-1}\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 - \left[P_{k-1}\right]^{-1}\frac{\partial\left[P_{k-1}\right]}{\partial u^{k-1}}\left[P_{k-1}\right]^{-1}R(u^{k-1}) \\
\frac{\partial G^k}{\partial D} &= \left[P_{k-1}\right]^{-1}\frac{\partial R(u^{k-1})}{\partial D} - \left[P_{k-1}\right]^{-1}\frac{\partial\left[P_{k-1}\right]}{\partial D}\left[P_{k-1}\right]^{-1}R(u^{k-1})
\end{aligned} \tag{3.95}$$

The definition of the nonlinear solver increment allows simplification of the above equation to the below one, where Δu replaces $\left[P_{k-1}\right]^{-1}R(u^{k-1})$.

$$\begin{aligned}\frac{\partial G^k}{\partial u^k} &= I \\ \frac{\partial G^k}{\partial u^{k-1}} &= -I + [P_{k-1}]^{-1}\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 - \frac{\partial [P_{k-1}]}{\partial u^{k-1}}\Delta u \\ \frac{\partial G^k}{\partial D} &= [P_{k-1}]^{-1}\left[\frac{\partial R(u^{k-1})}{\partial D} - \frac{\partial [P_{k-1}]}{\partial D}\Delta u\right]\end{aligned} \tag{3.96}$$

Using the equation for the adjoint at the final pseudo-time step with the constraint derivatives gives the same initial source term as in the Newton-Chord formulation.

$$[I]\,\Lambda^n = -\left[\frac{\partial L}{\partial u^n}\right]^T \tag{3.97}$$

Substituting in the constraint derivatives from equation (3.96) into equation (3.44) returns:

$$[I]\,\Lambda^{k-1} = -\left[-I + [P_{k-1}]^{-1}\left[\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 - \frac{\partial [P_{k-1}]}{\partial u^{k-1}}\Delta u\right]\right]^T \Lambda^k \tag{3.98}$$

This recurrence relation can also be rewritten in delta form as in the Newton-Chord formulation.

$$\Delta\Lambda = -\left[[P_{k-1}]^{-1}\left[\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 - \frac{\partial [P_{k-1}]}{\partial u^{k-1}}\Delta u\right]\right]^T \Lambda^k \tag{3.99}$$

Distributing the transpose returns the following equation.

$$\Delta\Lambda = -\left[\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 - \frac{\partial [P_{k-1}]}{\partial u^{k-1}}\Delta u\right]^T [P_{k-1}]^{-T}\,\Lambda^k \tag{3.100}$$

This motivates the definition of a secondary adjoint variable for each recurrence relation, the same one as in the Newton-Chord formulation:

$$[P_{k-1}]^T\,\psi^k = \Lambda^k \tag{3.101}$$

Using the secondary adjoint variable gives the final form of the recurrence relation as shown below.

$$\Delta\Lambda = -\left[\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 - \frac{\partial [P_{k-1}]}{\partial u^{k-1}}\Delta u\right]^T \psi^k \tag{3.102}$$

It is important to note that, as noted for the tangent, that this algorithm does not need exact correspondence here between the dual solver and the forward one, this will recover machine precision in the limit of the linear system being solved to machine precision regardless of the linear solver used in any mode of the algorithm. The next step has the substitution of the constraint derivatives from equation (3.96) into the sensitivity equation (3.39) and returns the expression below.

$$\frac{dJ}{dD} = \frac{\partial L}{\partial D} + \Lambda^{nT}\,[P_{n-1}]^{-1}\left[\frac{\partial R(u^{n-1})}{\partial D} - \frac{\partial [P_{n-1}]}{\partial D}\Delta u\right] + \ldots + \Lambda^{1T}\,[P_0]^{-1}\left[\frac{\partial R(u^0)}{\partial D} - \frac{\partial [P_0]}{\partial D}\Delta u\right] \tag{3.103}$$

Referring back to the definition of the secondary adjoint variable allows further simplification of this equation. It removes an additional linear solve per pseudo-time step and returns the equation below:

$$\frac{dJ}{dD} = \frac{\partial L}{\partial D} + \psi^{nT}\left[\frac{\partial R(u^{n-1})}{\partial D} - \frac{\partial\left[P_{n-1}\right]}{\partial D}\Delta u\right] + ... + \psi^{1T}\left[\frac{\partial R(u^{0})}{\partial D} - \frac{\partial\left[P_{0}\right]}{\partial D}\Delta u\right] \tag{3.104}$$

note that the adjoint sensitivity formulation requires only one approximate linear solution per nonlinear step, as shown in equation 3.101, which parallels the behavior of the nonlinear and tangent solvers.

3.2.3.3 Adjoint Computed Sensitivities for quasi-Newton Method with Exact Linearization through Frechét Differentiation

This section begins its derivation with the derivatives of the constraint equations as shown in equation (3.83), before the simplifications shown for either of the two previous formulations. These are written as follows.

$$\begin{aligned}\frac{\partial G^k}{\partial u^k} &= I \\ \frac{\partial G^k}{\partial u^{k-1}} &= -I + \frac{\partial\left[P_{k-1}\right]^{-1} R(u^{k-1})}{\partial u^{k-1}} \\ \frac{\partial G^k}{\partial D} &= \frac{\partial\left[P_{k-1}\right]^{-1} R(u^{k-1})}{\partial D}\end{aligned} \tag{3.105}$$

This derivation proceeds by assuming that it is possible to differentiate the linear solver with respect to the design variables and the state variables. The assumption is that this can be done using the Frechét derivative as was done for the tangent exact linearization. Using the equation for the adjoint at the final pseudo-time step with the constraint derivatives returns the same initial source term as in the previous formulations.

$$\left[I\right]\Lambda^n = -\left[\frac{\partial L}{\partial u^n}\right]^T \tag{3.106}$$

Substituting in the constraint derivatives from equation (3.83) into equation (3.44) returns:

$$\left[I\right]\Lambda^{k-1} = -\left[-I + \frac{\partial\left[P_{k-1}\right]^{-1} R(u^{k-1})}{\partial u^{k-1}}\right]^T \Lambda^k \tag{3.107}$$

as before this recurrence relation can be rewritten in delta form.

$$\Delta\Lambda = -\left[\frac{\partial\left[P_{k-1}\right]^{-1} R(u^{k-1})}{\partial u^{k-1}}\right]^T \Lambda^k \tag{3.108}$$

It is clear that the above equation requires a transpose Frechét derivative, which is impossible to implement without creating the full tensor using forward products and then doing the transpose matrix vector product. This would be very expensive and impractical. Should the derivation be continued it is possible to obtain the sensitivity equation which is also untenably expensive.

$$\frac{dJ}{dD} = \frac{\partial L}{\partial D} + \Lambda^{nT}\frac{\partial\left[P_{n-1}\right]^{-1} R(u^{n-1})}{\partial D} + ... + \Lambda^{1T}\frac{\partial\left[P_{0}\right]^{-1} R(u^{0})}{\partial D} \tag{3.109}$$

This equation will scale, like the tangent, with the number of design variables, and requires many Frechét differentiations of the linear solver at each nonlinear step, this is because of the lack of the secondary adjoint that was previously present through manipulation of the linear solves and made this possible.

3.2.3.4 Adjoint Computed Sensitivities for quasi-Newton Method with Exact Linearization through Manual Differentiation

As above this section begins with the derivatives of the constraint equations as shown in equation (3.83), as in the previous formulation. These are written as follows.

$$\begin{aligned}\frac{\partial G^k}{\partial u^k} &= I \\ \frac{\partial G^k}{\partial u^{k-1}} &= -I + \frac{\partial \left[P_{k-1}\right]^{-1} R(u^{k-1})}{\partial u^{k-1}} \\ \frac{\partial G^k}{\partial D} &= \frac{\partial \left[P_{k-1}\right]^{-1} R(u^{k-1})}{\partial D}\end{aligned} \tag{3.110}$$

This derivation assumes that linearization of the linear solver with respect to the inputs, the design variables, the state variables, and the right hand side is possible; while relaxation methods are right hand side independent, Krylov solvers are path dependent and right hand side dependent, and therefore the linear solver must be differentiated with respect to the right hand side. The assumption is that this differentiation can be done manually and returns the following expression for the derivatives of the constraints.

$$\begin{aligned}\frac{\partial G^k}{\partial u^k} &= I \\ \frac{\partial G^k}{\partial u^{k-1}} &= -I + \frac{\partial \left[P_{k-1}\right]^{-1}}{\partial u^{k-1}} R(u^{k-1}) + \frac{\partial \left[P_{k-1}\right]^{-1}}{\partial R(u^{k-1})} \left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 R(u^{k-1}) + \left[P_{k-1}\right]^{-1} \left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 \\ \frac{\partial G^k}{\partial D} &= \frac{\partial \left[P_{k-1}\right]^{-1}}{\partial D} R(u^{k-1}) + \frac{\partial \left[P_{k-1}\right]^{-1}}{\partial R(u^{k-1})} \frac{\partial R(u^{k-1})}{\partial D} R(u^{k-1}) + \left[P_{k-1}\right]^{-1} \frac{\partial R(u^{k-1})}{\partial D}\end{aligned} \tag{3.111}$$

Using the equation for the adjoint at the final pseudo-time step with the constraint derivatives returns the same initial source term as in the previous formulations.

$$\left[I\right] \Lambda^n = -\left[\frac{\partial L}{\partial u^n}\right]^T \tag{3.112}$$

Substituting in the constraint derivatives from equation (3.111) into equation (3.44) returns:

$$\begin{aligned}\left[I\right] \Lambda^{k-1} = -\Bigg[-I + \frac{\partial \left[P_{k-1}\right]^{-1}}{\partial u^{k-1}} R(u^{k-1}) + \frac{\partial \left[P_{k-1}\right]^{-1}}{\partial R} \left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 R(u^{k-1}) + \left[P_{k-1}\right]^{-1} \left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 \Bigg]^T \Lambda^k \\ - \frac{\partial L}{\partial u^{k-1}}\end{aligned} \tag{3.113}$$

This recurrence relation can be rewritten in delta form as in the previous formulations.

$$\Delta\Lambda^{k-1} = -\left[\frac{\partial\left[P_{k-1}\right]^{-1}}{\partial u^{k-1}}R(u^{k-1}) + \frac{\partial\left[P_{k-1}\right]^{-1}}{\partial R}\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 R(u^{k-1}) + \left[P_{k-1}\right]^{-1}\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2\right]^T \Lambda^k - \frac{\partial L}{\partial u^{k-1}} \tag{3.114}$$

Distributing the transpose and grouping like terms returns the below expression:

$$\Delta\Lambda = -\left[\frac{\partial\left[P_{k-1}\right]^{-1}}{\partial u^{k-1}}R(u^{k-1}) + \frac{\partial\left[P_{k-1}\right]^{-1}}{\partial R}\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 R(u^{k-1})\right]^T \Lambda^k - \left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 \left[P_{k-1}\right]^{-T}\Lambda^k - \frac{\partial L}{\partial u^{k-1}} \tag{3.115}$$

This motivates the definition of a secondary adjoint variable for each recurrence relation, as in the previous formulations.

$$\left[P_{k-1}\right]^T \psi^k = \Lambda^k \tag{3.116}$$

Substituting in the secondary adjoint variable shows that this method requires only one linear solve per nonlinear iteration as in the analysis, but this requires some very involved derivatives. In order to obtain the below expression it is necessary to linearize the approximate linear solver with respect to the full conservative variable vector and the residual vector.

$$\Delta\Lambda = -\left[\frac{\partial\left[P_{k-1}\right]^{-1}}{\partial u^{k-1}}R(u^{k-1}) + \frac{\partial\left[P_{k-1}\right]^{-1}}{\partial R(u^{k-1})}\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 R(u^{k-1})\right]^T \Lambda^k - \left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 \psi^k \tag{3.117}$$

Clearly the above equation has such involved derivatives that computation through the use of hand differentiation is not practical. The linearization of a GMRES algorithm is untenable even in the forward mode, much less in the reverse. There are also further issues that will be addressed at the end of the derivation. Now that an expression for the pseudo-time adjoint has been obtained the next step is to move on to the sensitivity equation and plug in the appropriate constraint derivatives.

$$\begin{aligned}\frac{dJ}{dD} = \frac{\partial L}{\partial D} &+ \Lambda^{nT}\left[\frac{\partial\left[P_{n-1}\right]^{-1}}{\partial D}R(u^{n-1}) + \frac{\partial\left[P_{n-1}\right]^{-1}}{\partial R(u^{n-1})}\frac{\partial R(u^{n-1})}{\partial D}R(u^{n-1}) + \left[P_{n-1}\right]^{-1}\frac{\partial R(u^{n-1})}{\partial D}\right] \\ &+ ... \\ &+ \Lambda^{1T}\left[\frac{\partial\left[P_0\right]^{-1}}{\partial D}R(u^0) + \frac{\partial\left[P_0\right]^{-1}}{\partial R(u^0)}\frac{\partial R(u^0)}{\partial D}R(u^0) + \left[P_0\right]^{-1}\frac{\partial R(u^0)}{\partial D}\right]\end{aligned} \tag{3.118}$$

It looks as though this formulation requires the solution as many linear systems per nonlinear iterations as there are design variables but,through the reuse of the secondary adjoint variable it is possible to do away

with this additional cost as in previous sections.

$$\begin{aligned}\frac{dJ}{dD} = \frac{\partial L}{\partial D} + \Lambda^{nT}\left[\frac{\partial\left[P_{n-1}\right]^{-1}}{\partial D}R(u^{n-1}) + \frac{\partial\left[P_{n-1}\right]^{-1}}{\partial R(u^{n-1})}\frac{\partial R(u^{n-1})}{\partial D}R(u^{n-1})\right] + \psi^{nT}\frac{\partial R(u^{n-1})}{\partial D} \\ + ... \\ + \Lambda^{1T}\left[\frac{\partial\left[P_0\right]^{-1}}{\partial D}R(u^0) + \frac{\partial\left[P_0\right]^{-1}}{\partial R(u^0)}\frac{\partial R(u^0)}{\partial D}R(u^0)\right] + \psi^{1T}\frac{\partial R(u^0)}{\partial D}\end{aligned} \tag{3.119}$$

For additional insight the partial derivatives of the design variables are expanded:

$$\begin{aligned}\frac{dJ}{dD} = \frac{\partial L}{\partial x}\frac{\partial x}{\partial D} + \Lambda^{nT}\left[\frac{\partial\left[P_{n-1}\right]^{-1}}{\partial x}\frac{\partial x}{\partial D}R(u^{n-1}) + \frac{\partial\left[P_{n-1}\right]^{-1}}{\partial R(u^{n-1})}\frac{\partial R(u^{n-1})}{\partial x}\frac{\partial x}{\partial D}R(u^{n-1})\right] + \psi^{nT}\frac{\partial R(u^{n-1})}{\partial x}\frac{\partial x}{\partial D} \\ + ... \\ + \Lambda^{1T}\left[\frac{\partial\left[P_0\right]^{-1}}{\partial x}\frac{\partial x}{\partial D}R(u^0) + \frac{\partial\left[P_0\right]^{-1}}{\partial R(u^0)}\frac{\partial R(u^0)}{\partial x}\frac{\partial x}{\partial D}R(u^0)\right] + \psi^{1T}\frac{\partial R(u^0)}{\partial x}\frac{\partial x}{\partial D}\end{aligned} \tag{3.120}$$

which reveals that in order to evaluate this expression the linearization of the GMRES solver with respect to the mesh coordinate variables is necessary for shape optimization. In summation, this algorithm requires that the developer linearize the iterative solver with respect to the nodal coordinates, the conservative variables, and the residual/right hand side; furthermore it actually requires not just the derivative of the linear solver, but the derivative of the approximate inverse it represents which is never explicitly computed or stored. This is an impractical algorithm and so it is not used going forward. Therefore, in this work the inexact quasi-Newton method with the inverse identity is used for the bulk of the results. It is the simplest to implement for the level of accuracy it provides and it allows for investigation of the effects of partial convergence of the linear system on the accuracy of the sensitivities going forward.

3.2.4 General Sensitivity Convergence Proof for Approximate Adjoint Linearization of the Fixed Point Iteration

As in the tangent section, it is desirable to quantify the error introduced to the sensitivity calculation by the approximate linearization of the matrix inverse through the inverse identity approximation. It is known that for a properly implemented linearization and transposition that the adjoint and tangent return the same sensitivity values. Therefore the error in the sensitivities goes to zero when computed through the adjoint as the nonlinear problem is converged, as was proven to be the case for the tangent sensitivities. To begin the more rigorous proof, it is necessary to begin from the more general version of the solver as a fixed point iteration that multiplies the residual operator by some A operator. It is important to note that by definition of the fixed point, A is not orthogonal to R and it is bounded away from 0, lest the fixed point terminate at a state that does not satisfy the discretized form of the governing equations (signified by $R = 0$).

$$u^{k+1} = N(u^k, D) = u^k + H(u^k, D) = u^k + A(u^k, D)R(u^k, D) \tag{3.121}$$

The fixed point iteration derivatives are expressed below, where the linearization of the residual is exact, but the linearization of the A matrix is approximate in the actual implementation:

$$\frac{\partial H^{k+1}}{\partial u^k} = \frac{\partial A(u^k, D)}{\partial u^k} R(u^k, D) + A(u^k, D) \frac{\partial R(u^k, D)}{\partial u^k} \tag{3.122}$$

The sensitivity equation first shown in equation (3.39) in the initial pseudo-time accurate adjoint derivations are reproduced below, where G is the shifted fixed point iteration, $G = u - N(u)$.

$$\frac{dL}{dD} = \frac{\partial L}{\partial D} + \Lambda^{nT} \frac{\partial G^n}{\partial D} + \Lambda^{n-1T} \frac{\partial G^{n-1}}{\partial D} + \Lambda^{n-2T} \frac{\partial G^{n-2}}{\partial D} + ... + \Lambda^{1T} \frac{\partial G^1}{\partial D} \tag{3.123}$$

Substituting in the approximate terms into 3.39 returns the expression below, where the approximation in a given variable x denoted by $\tilde{x}$:

$$\widetilde{\frac{dL}{dD}} = \frac{\partial L}{\partial D} + \tilde{\Lambda}^{nT} \widetilde{\frac{\partial G^n}{\partial D}} + \tilde{\Lambda}^{n-1T} \widetilde{\frac{\partial G^{n-1}}{\partial D}} + \tilde{\Lambda}^{n-2T} \widetilde{\frac{\partial G^{n-2}}{\partial D}} + ... + \tilde{\Lambda}^{1T} \widetilde{\frac{\partial G^1}{\partial D}} \tag{3.124}$$

By subtracting the exact sensitivity equation from the approximate one, the error equation below is obtained. In it there is error in the pseudo-temporal adjoint vectors due to the linearization error of the fixed point with respect to the flow variables ($\tilde{\Lambda}$), and there is explicit error from inexact linearization of the fixed point ($\widetilde{\frac{\partial G^k}{\partial D}}$).

$$\begin{aligned}
\frac{dL}{dD} - \widetilde{\frac{dL}{dD}} = & \left(\frac{\partial L}{\partial D} - \frac{\partial L}{\partial D} \right) \\
& + \left(\Lambda^{nT} \frac{\partial G^n}{\partial D} - \tilde{\Lambda}^{nT} \widetilde{\frac{\partial G^n}{\partial D}} \right) \\
& + \left(\Lambda^{n-1T} \frac{\partial G^{n-1}}{\partial D} - \tilde{\Lambda}^{n-1T} \widetilde{\frac{\partial G^{n-1}}{\partial D}} \right) \\
& + \left(\Lambda^{n-2T} \frac{\partial G^{n-2}}{\partial D} - \tilde{\Lambda}^{n-2T} \widetilde{\frac{\partial G^{n-2}}{\partial D}} \right) \\
& + ... \\
& + \left(\Lambda^{1T} \frac{\partial G^1}{\partial D} - \tilde{\Lambda}^{1T} \widetilde{\frac{\partial G^1}{\partial D}} \right)
\end{aligned} \tag{3.125}$$

It is necessary to define three error terms for the errors at each nonlinear iteration, the error in: the pseudo-time adjoint, the linearization of the nonlinear iteration with respect to the state variable and the linearization with respect to the design variables. These are denoted by $\epsilon_\Lambda^k, \epsilon_u^k, \epsilon_D^k$ respectively.

$$\begin{aligned}
\epsilon_\Lambda^k &= \Lambda^k - \tilde{\Lambda}^k \\
\epsilon_u^k &= \frac{\partial G^k}{\partial u^{k-1}} - \widetilde{\frac{\partial G^k}{\partial u^{k-1}}} \\
\epsilon_D^k &= \frac{\partial G^k}{\partial D} - \widetilde{\frac{\partial G^k}{\partial D}}
\end{aligned} \tag{3.126}$$

Using these epsilon definitions makes it possible to simplify equation 3.125 into the below equation with the epsilon terms as the unknowns.

$$\begin{aligned}\frac{dL}{dD} - \widetilde{\frac{dL}{dD}} = \epsilon_\Lambda^{nT}\frac{\partial G^n}{\partial D} + \Lambda^{nT}\epsilon_D^n - \epsilon_\Lambda^{nT}\epsilon_D^n \\ + \epsilon_\Lambda^{n-1T}\frac{\partial G^{n-1}}{\partial D} + \Lambda^{n-1T}\epsilon_D^{n-1} - \epsilon_\Lambda^{n-1T}\epsilon_D^{n-1} \\ + \epsilon_\Lambda^{n-2T}\frac{\partial G^{n-2}}{\partial D} + \Lambda^{n-2T}\epsilon_D^{n-2} - \epsilon_\Lambda^{n-2T}\epsilon_D^{n-2} \\ + ... \\ + \epsilon_\Lambda^{1T}\frac{\partial G^1}{\partial D} + \Lambda^{1T}\epsilon_D^1 - \epsilon_\Lambda^{1T}\epsilon_D^1\end{aligned} \tag{3.127}$$

This proof cannot proceed any further without some expressions for the epsilon error terms, and it is therefore necessary to proceed using the fact that only the $\frac{\partial A}{\partial ()}$ term has an approximation to get the following two identities:

$$\begin{aligned}\epsilon_u^k &= \frac{\partial G^k}{\partial u^{k-1}} - \widetilde{\frac{\partial G^k}{\partial u^{k-1}}} = \frac{\partial A}{\partial u}R + A\frac{\partial R}{\partial u} - \widetilde{\frac{\partial A}{\partial u}}R - A\frac{\partial R}{\partial u} = \left(\frac{\partial A}{\partial u} - \widetilde{\frac{\partial A}{\partial u}}\right)R \\ \epsilon_D^k &= \frac{\partial G^k}{\partial D} - \widetilde{\frac{\partial G^k}{\partial D}} = \frac{\partial A}{\partial D}R + A\frac{\partial R}{\partial D} - \widetilde{\frac{\partial A}{\partial D}}R - A\frac{\partial R}{\partial D} = \left(\frac{\partial A}{\partial D} - \widetilde{\frac{\partial A}{\partial D}}\right)R\end{aligned} \tag{3.128}$$

The two error terms above have a scaling with the residual that is important to keep in mind. Using the definitions of the error terms allows for an expression of the difference between the sensitivities with exact and approximate linearizations. Using equation 3.44 gives an error recurrence relationship:

$$\epsilon_\Lambda^{k-1} = -\frac{\partial G^k}{\partial u^{k-1}}^T\Lambda^k + \widetilde{\frac{\partial G^k}{\partial u^{k-1}}}^T\tilde{\Lambda}^k \tag{3.129}$$

Since $\Lambda^n = \frac{\partial L}{\partial u^n}$ which has no approximation error, $\epsilon_\Lambda^n = 0$. It is possible to substitute in the epsilon identities into the equation above to obtain the one below.

$$\begin{aligned}\epsilon_\Lambda^{k-1} &= \left(\frac{\partial G^k}{\partial u^{k-1}}^T - \epsilon_u^k\right)\left(\Lambda^k - \epsilon_\Lambda^k\right) - \frac{\partial G^k}{\partial u^{k-1}}^T\Lambda^k \\ &= -\epsilon_u^{kT}\Lambda^k + \epsilon_u^{kT}\epsilon_\Lambda^k - \frac{\partial G^k}{\partial u^{k-1}}\epsilon_\Lambda^k\end{aligned} \tag{3.130}$$

Applying this epsilon recurrence relation to the second to last iteration:

$$\epsilon_\Lambda^{n-1} = -\epsilon_u^{nT}\Lambda^n + \epsilon_u^{nT}\epsilon_\Lambda^n - \frac{\partial G^n}{\partial u^{n-1}}\epsilon_\Lambda^n \tag{3.131}$$

then using $\epsilon_\Lambda^n = 0$ and substituting in the expression for ϵ_u^{kT} returns.

$$\epsilon_\Lambda^{n-1} = -\left(\frac{\partial A}{\partial u} - \widetilde{\frac{\partial A}{\partial u}}\right)R^n\Lambda^n \tag{3.132}$$

It is therefore clear that in order to show that the error goes to zero it must be shown that $\left\|R^k\Lambda^k\right\| \approx 0$ for all iterations. It is important here to refer back to the definition of the fixed point iteration; A is bounded away from 0 and A is not orthogonal to R (lest the fixed point terminate at a state that does not satisfy the discretized form of the PDE). Therefore $\|A\| \neq 0$ and $\|R^n\Lambda^n\| \approx 0$ Since the pseudo-temporal adjoint converges at the reverse of the analysis process (through linearizing the fixed point iteration and transposing the derivative), $\left\|R^k\Lambda^k\right\| = \|R^n\Lambda^n\|$, which for a converged simulation is on the order of machine zero. Therefore the errors in the linearization of the A operator (denoted by ϵ_u) do not contribute to the sensitivity error as the ϵ_Λ they contribute to is itself equal to zero. The reverse convergence of the adjoint as compared to the analysis mode is confirmed by the results in [39], where the adjoint is shown to converge to its final value in the reverse of the analysis problem.

$$\epsilon_\Lambda^{k-1} = -\left(\frac{\partial A}{\partial u} - \widetilde{\frac{\partial A}{\partial u}}\right) R^k\Lambda^k \approx 0 \tag{3.133}$$

Otherwise, for a stalled or truncated simulation, the error at every iteration is scaled by the magnitude of the residual at the final state. Using that $\epsilon_\Lambda^k = 0$ for a converged simulation returns the following expression:

$$\frac{dL}{dD} - \widetilde{\frac{dL}{dD}} = \Lambda^{nT}\epsilon_D^n + \Lambda^{n-1T}\epsilon_D^{n-1} + \Lambda^{n-2T}\epsilon_D^{n-2} + ... + \Lambda^{1T}\epsilon_D^1 \tag{3.134}$$

Substituting in the expression for the error in the linearization of the nonlinear iteration with respect to the design variable returns the following equation for the error in the sensitivities.

$$\begin{aligned}\frac{dL}{dD} - \widetilde{\frac{dL}{dD}} = &-\Lambda^{nT}\left(\frac{\partial A}{\partial u}^n - \widetilde{\frac{\partial A}{\partial u}}^n\right) R^n \\ &- \Lambda^{n-1T}\left(\frac{\partial A}{\partial u}^{n-1} - \widetilde{\frac{\partial A}{\partial u}}^{n-1}\right) R^{n-1} \\ &- \Lambda^{n-2T}\left(\frac{\partial A}{\partial u}^{n-2} - \widetilde{\frac{\partial A}{\partial u}}^{n-2}\right) R^{n-2} \\ &- ... \\ &- \Lambda^{1T}\left(\frac{\partial A}{\partial u}^1 - \widetilde{\frac{\partial A}{\partial u}}^1\right) R^1\end{aligned} \tag{3.135}$$

Cauchy-Schwarz inequality allows for the identity at each pseudo-temporal iteration:

$$\Lambda^{kT}\left(\frac{\partial A}{\partial u}^k - \widetilde{\frac{\partial A}{\partial u}}^k\right) R^k \leq \left\|\frac{\partial A}{\partial u}^k - \widetilde{\frac{\partial A}{\partial u}}^k\right\| \left\|R^k\Lambda^k\right\| \tag{3.136}$$

This means that the error added at each iteration in the backwards-in-pseudo-time integration is upper bounded by $\left\|\frac{\partial A}{\partial u}^k - \widetilde{\frac{\partial A}{\partial u}}^k\right\| \left\|R^k\Lambda^k\right\|$ Using the same argument as above that $\left\|R^k\Lambda^k\right\| \approx \|\Lambda^n R^n\| \approx 0$ returns that for a converged flow the error in the sensitivities goes to 0. Again it is shown that the convergence of the nonlinear problem leads to less error in the sensitivities even in unconverged flows. The expressions

above show that the error in the sensitivities is dependent on the error in the A operator and the residual of the nonlinear problem. For quasi-Newton solvers like the ones that are the thrust of this work, this indicates the convergence is a multiple of the tolerance of the linear system and the convergence of the non-linear problem, and this behavior is borne out in the results section.

Chapter 4

Verification of Implementation

4.1 Verification of Analysis Order of Accuracy and Steady State Tangent and Adjoint Sensitivity Computation

This code is verified in three parts: first, the order of accuracy is verified through a Gaussian bump case and a uniformly refined mesh. Then the code is compared to a well validated code on a benchmarked test case to ensure proper implementation of the physics modules. Finally the steady state adjoint and tangent are verified through comparison to the sensitivities obtained through complex differentiation. The Gaussian bump case is a flat plate with a small Gaussian bump with an amplitude of .05 in subsonic ($M = .2$) flow. For an ideal case, the solution should have no drag due to the lack of viscosity or shock discontinuities, therefore, any drag is due to the discretization error. As the mesh is refined, it is expected to see the drag vanish at a second-order rate (that the drag decreases with the square root of the mesh sizing at a slope of 2). Figure 4.1 shows the converged flow on the finest mesh used. Figure 4.2 shows the expected convergence of the drag error and second order accuracy is confirmed.

To verify proper implementation of the physics the test case of a NACA0012 airfoil in Mach .85 flow with no angle of incidence was run. This case was chosen because it has been benchmarked as part of the ADODG workshop and the well known and well validated Cart3D code presents results for the baseline case that closely correspond to the results obtained by this in-house finite-volume code. This test case was run on a mesh generated by an in-house two-dimensional unstructured mesh generation [59,60] (shown in Figure 4.3a) and then refined with an in-house AMR package; the refined mesh is shown in Figure 4.3b. The flow field shown in Figure 4.4 is run on this refined mesh. This case shows close correspondence to the values obtained by Cart3D [61] which obtained a drag coefficient of 471.3 counts, this code obtained a value of 457.7 counts; a difference of only 2.8% between two codes that vary vastly in terms of implementation and discretization. The flow does not show exact symmetry but this can be attributed to the highly asymmetric

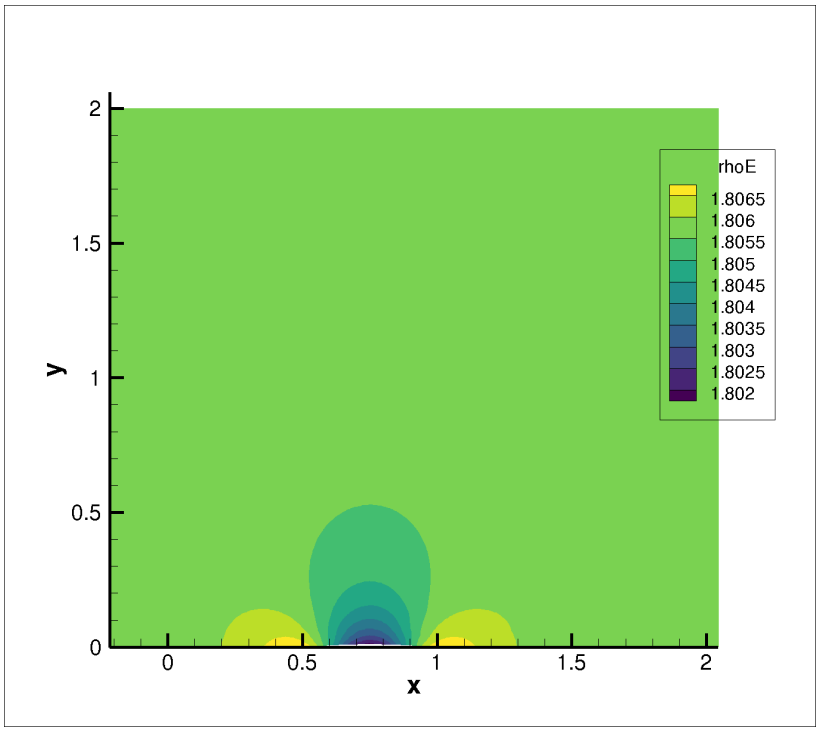


Figure 4.1: Plot of energy on the finest mesh for Gaussian bump

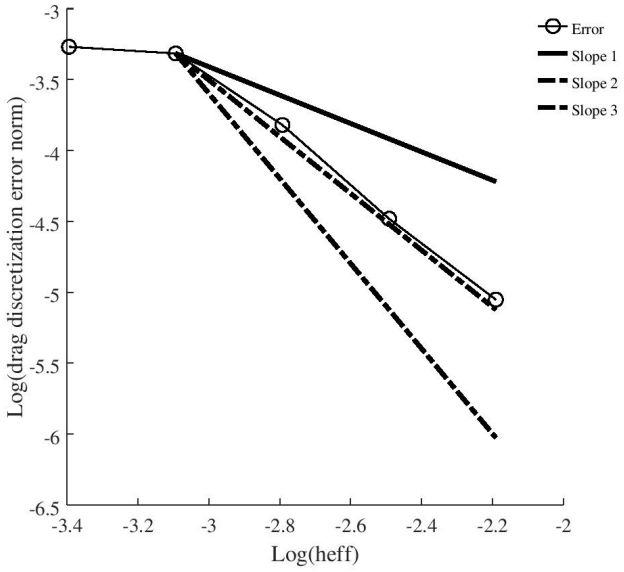
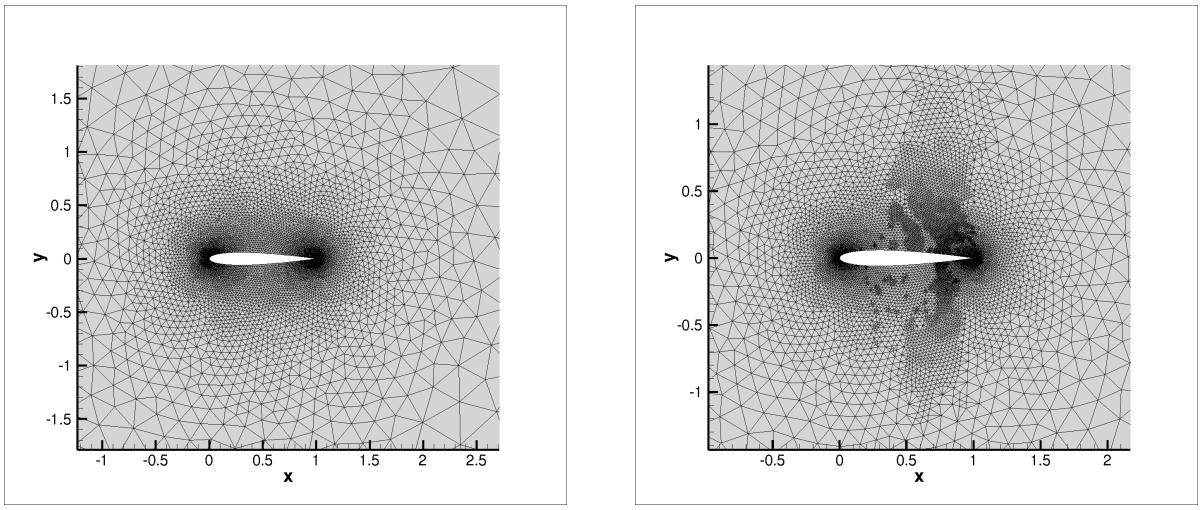


Figure 4.2: Order of accuracy verification plot

nature of the mesh.

Finally, the sensitivities computed through the classical/steady state tangent and adjoint methods are verified for a transonic flow test case. This was run on the UMESH2D mesh shown in Figure 4.3a with the



(a) Baseline Mesh (b) Adapted Mesh

Figure 4.3: Mesh comparison for NACA0012 verification case

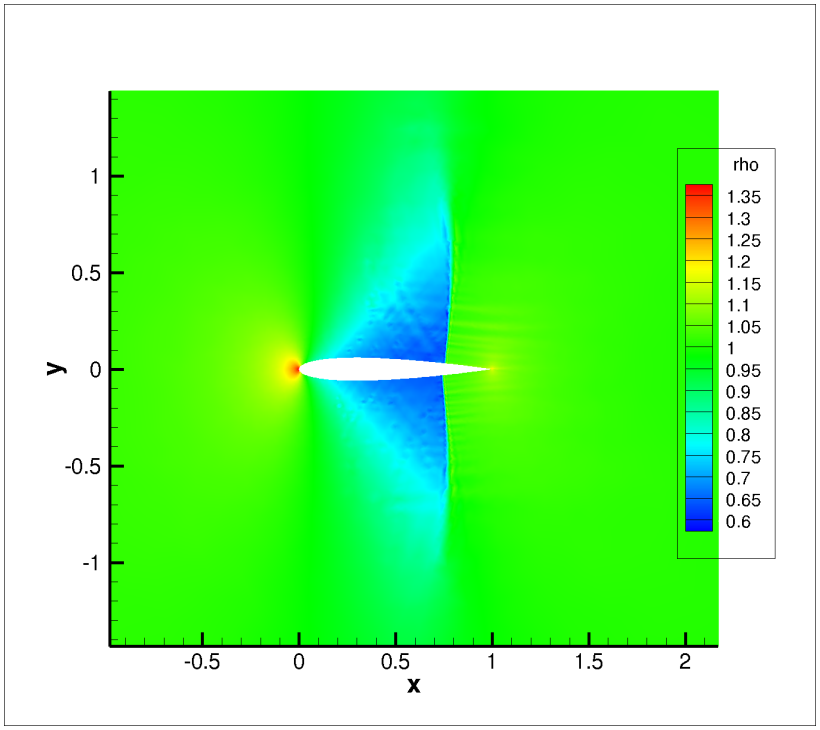
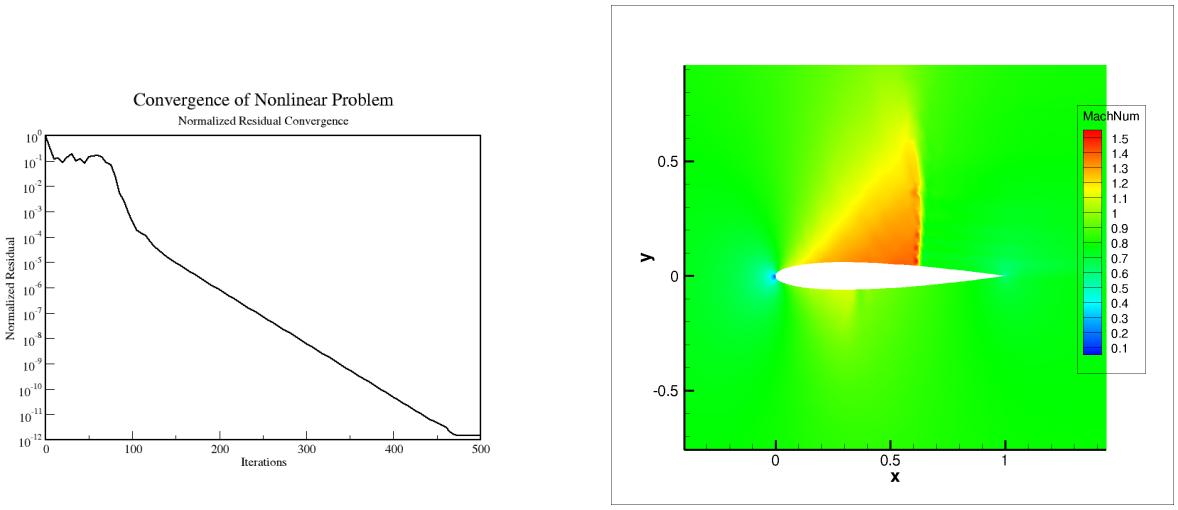


Figure 4.4: Mach flow field on adapted mesh for NACA0012 verification case

second-order objective function –calculated by reconstructing the solution variables from the cell center to the airfoil surface– with the objective function being $J = c_L^2 + c_D^2$. This case also used the new limiting algorithm to assist with stability, symmetric Hicks-Henne bump functions as the design variables, and the inverse distance weighting method for the mesh deformation. The convergence history and the flow field are

shown in tandem in Figure 4.5, which shows good convergence in the residual, as the non-linear problem has stalled at 12 orders of convergence with the nonlinear residual being on the order of $1e-15$. Figure 4.6 presents the adjoint plots for density, momentum and energy with the expected behavior. Table 4.1 shows the sensitivities obtained from the tangent and adjoint systems and compares the results to those obtained by the complex-step method (with the difference from the complex-step method sensitivities being in bold-face) to verify the linearization of the analysis code and that the tangent and adjoint have been properly implemented. For this case the absolute tolerance of the non-linear problem was set to 5e-16 and that of the linear problem was set to 5e-15. This work presents good agreement (but not perfect) for such inputs.



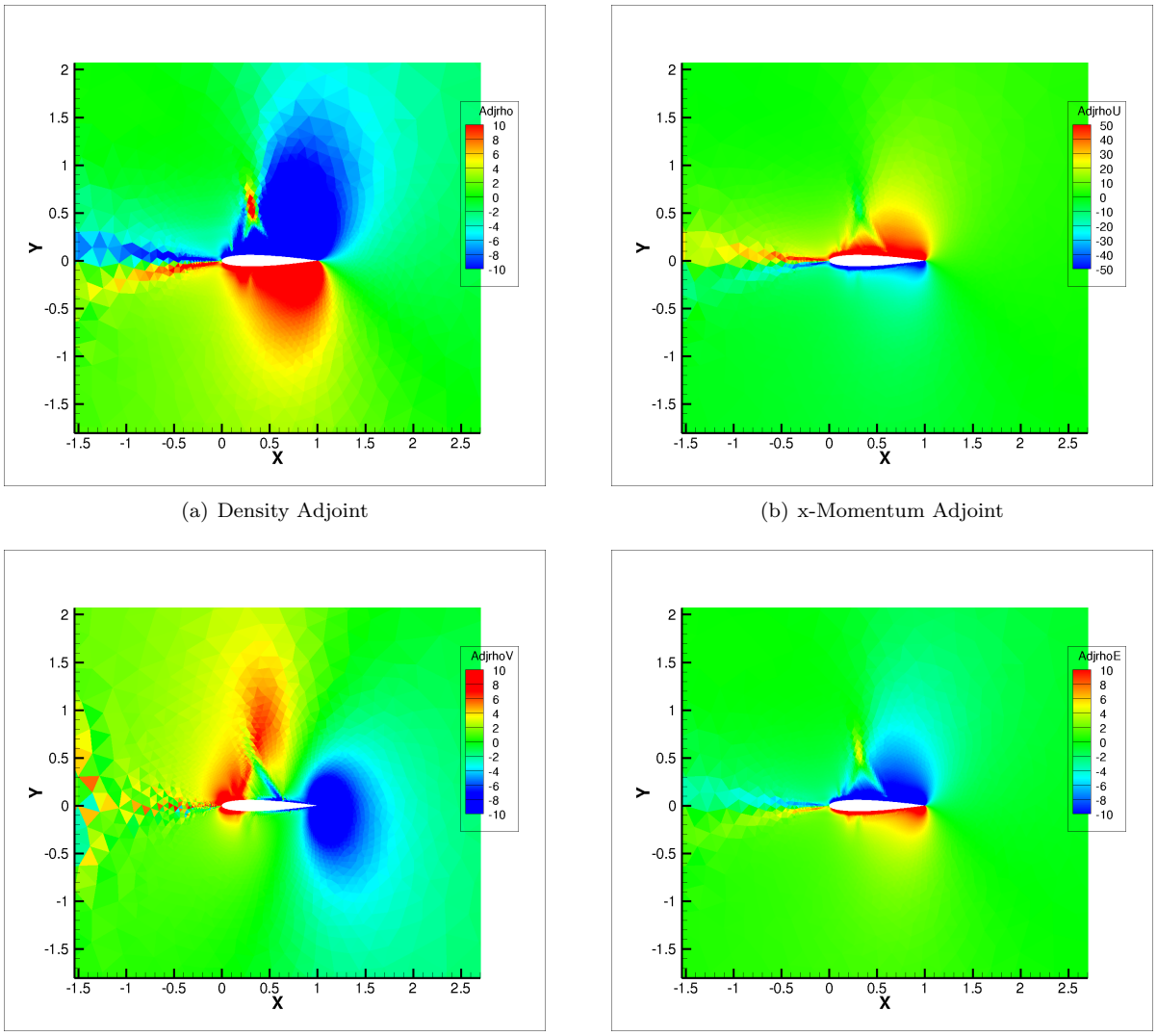
(a) Analysis Convergence

(b) Mach Flow Field

Figure 4.5: Analysis problem convergence and flow field for verification of sensitivities

Sensitivity Computation	Design Variable 1	Design Variable 2
Complex-Step Value	2.67138168470938	5.95205862248555
Adjoint Value	2.67138168470**456**	5.9520586224**7514**
Tangent Value	2.67138168470**465**	5.9520586224**7512**

Table 4.1: Verification of steady state adjoint and complex-step computed sensitivities



(a) Density Adjoint

(b) x-Momentum Adjoint

(c) y-Momentum Adjoint

(d) Energy Adjoint

Figure 4.6: Adjoint fields for verification case

4.2 Verification of the Pseudo-Time Accurate Tangent and Adjoint Sensitivities

The verification of the pseudo-time accurate formulations of the tangent and adjoint problems is done using the computed sensitivities of a combined weighted lift and drag objective functional ($L = \omega_L c_L + \omega_D c_D$) as proxies for the behavior of the tangent and adjoint systems and as a means to examine their behavior and convergence. The design variables are two Hicks-Henne bump functions located at one third and two thirds of the chord length respectively (denoted by the red circles in the figure) which symmetrically perturb the airfoil. The mesh sensitivities were calculated with the spring analogy outlined previously. The mesh is unstructured and consists of 4212 triangular elements shown in Figure 4.7; this mesh is used for all

verification cases as well as the investigations in the following chapter. All verification cases were run with $Mach = .6$ and $\alpha = 1^o$.

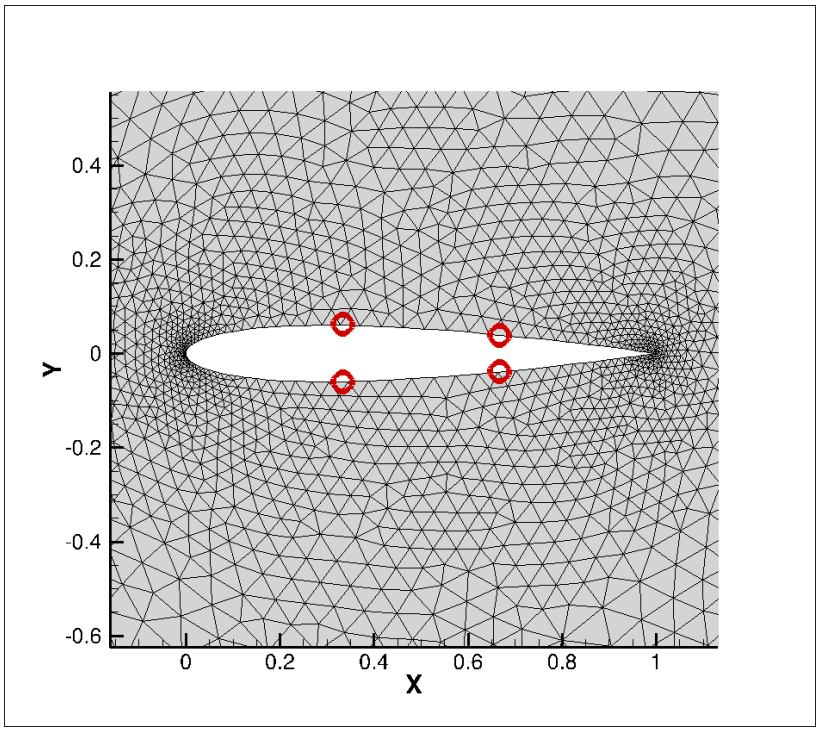


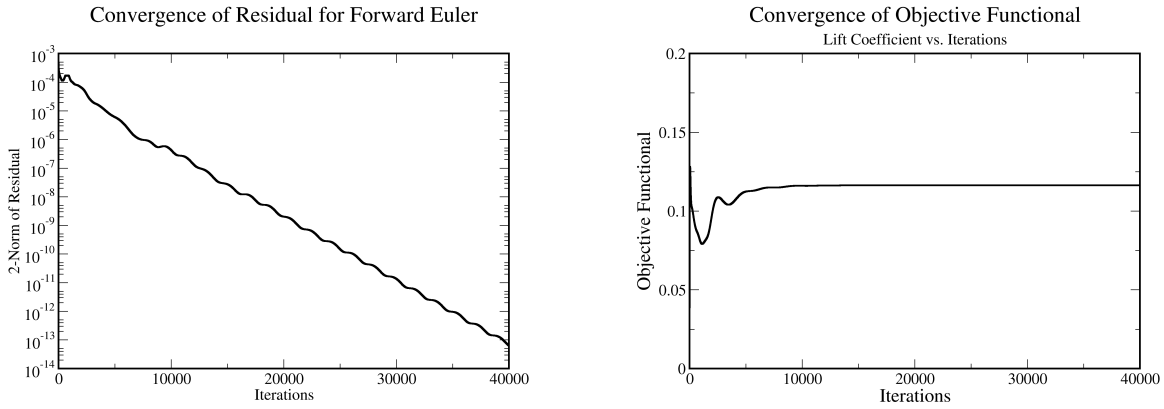
Figure 4.7: Computational mesh for NACA0012 airfoil in verification cases

4.2.1 Pseudo-time Accurate Tangent Verification

For the pseudo-time accurate tangent system verification, the complex step computed sensitivities are compared to those provided by the pseudo-time accurate tangent system. Since the pseudo-time accurate tangent formulation is designed to compute the exact derivative it is expected to see exact correspondence between the two methods, i.e. differences near the order of machine zero at every step of the nonlinear solution process.

4.2.1.1 Forward Euler Pseudo-Time Evolution

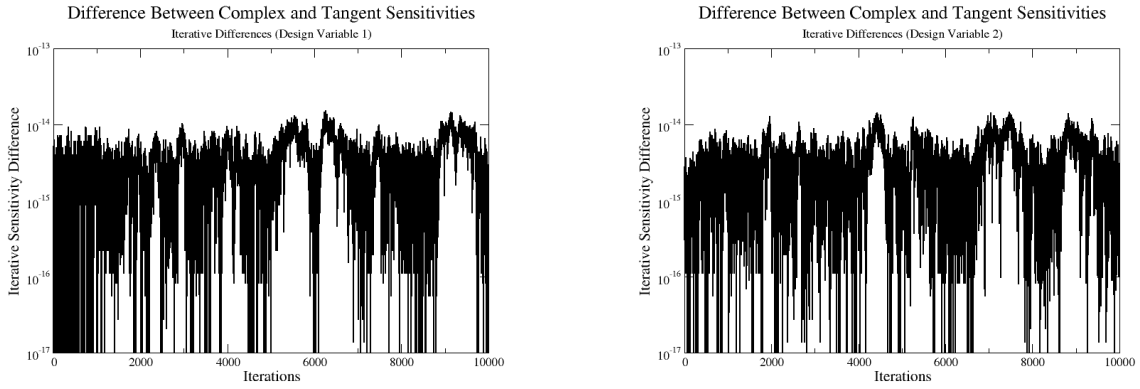
Figure 4.8 shows the convergence of the primal system residual and the convergence of the objective functional to its final value. Figure 4.9 shows verification of the implementation of the pseudo-time accurate tangent. This figure shows the difference between the pseudo-time accurate tangent and complex sensitivities is on the order of machine zero regardless of the residual value at the given pseudo-time step. The error is of order $1e-15$ for both design variables, and as such it is clear this method is verified.



(a) Residual

(b) Objective Functional

Figure 4.8: Convergence of spatial residual and objective function for forward Euler pseudo-time evolution



(a) Design Variable 1

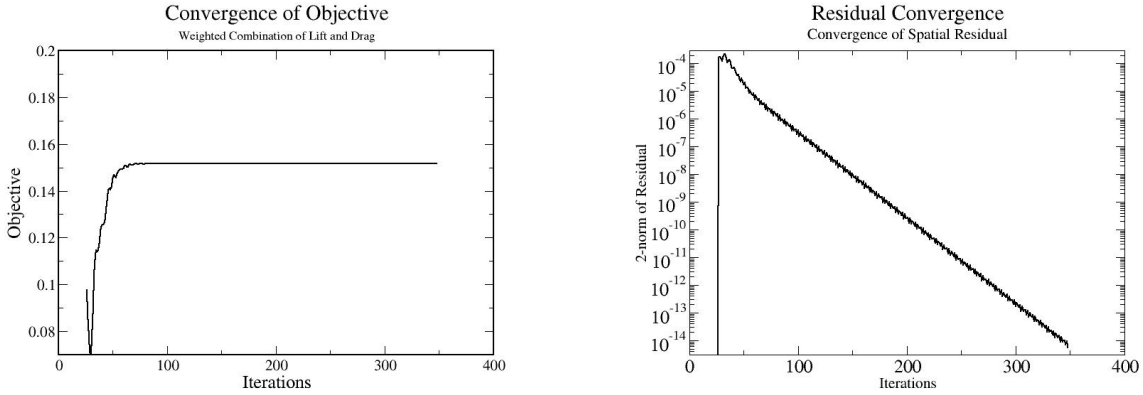
(b) Design Variable 2

Figure 4.9: Difference between pseudo-time accurate tangent sensitivities and complex sensitivities for forward Euler pseudo-time evolution

4.2.1.2 Newton-Chord Method

The verification of the Newton-Chord linearization is accomplished by starting the analysis problem at initial conditions and running a quasi-Newton algorithm for 25 steps, at which point the preconditioner is frozen and the algorithm continues solving the analysis problem with this frozen preconditioner. The complex-step finite-difference sensitivities were calculated by restarting at the 26th iteration with the frozen preconditioner and introducing a complex perturbation to the design variables, therefore ensuring the complex perturbation does not affect the preconditioner. To run the pseudo-time accurate tangent, the tangent problem was restarted at the 26th iteration. To parallel the Newton-Chord solution, the preconditioner matrix on the left hand side of equation (3.15) is the frozen Jacobian matrix, and the terms on the right hand side are the linearization of the spatial residual operator, which changes through pseudo-time. Since

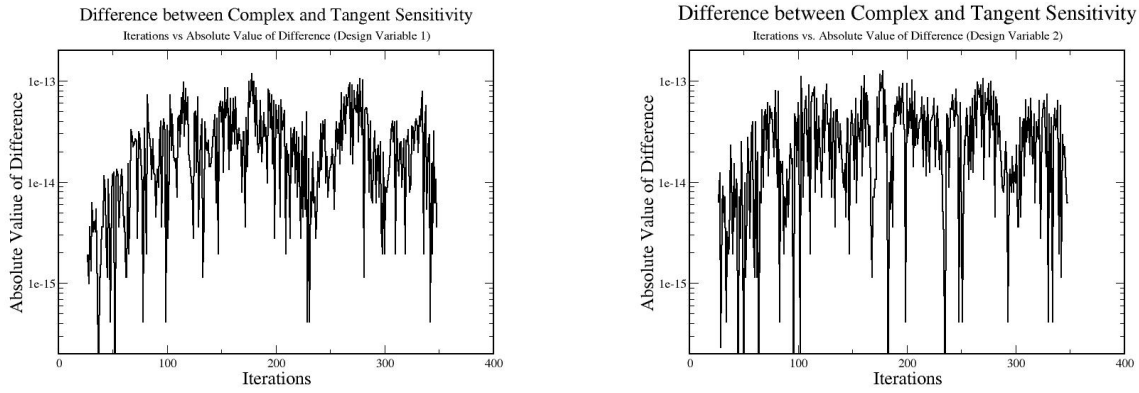
this is the exact derivative of the analysis solution process, it is expected that exact correspondence between the complex step derivative and the pseudo-time accurate tangent sensitivity is obtained. The functional and residual behavior are depicted in Figure (4.10). Figure (4.11) plots the difference between the complex and pseudo-time accurate tangent method sensitivities at each pseudo time step, and confirms that the two sensitivity methods agree to machine precision.



(a) Functional

(b) Residual

Figure 4.10: Convergence of objective function and residual for Newton-Chord Method



(a) Design Variable 1

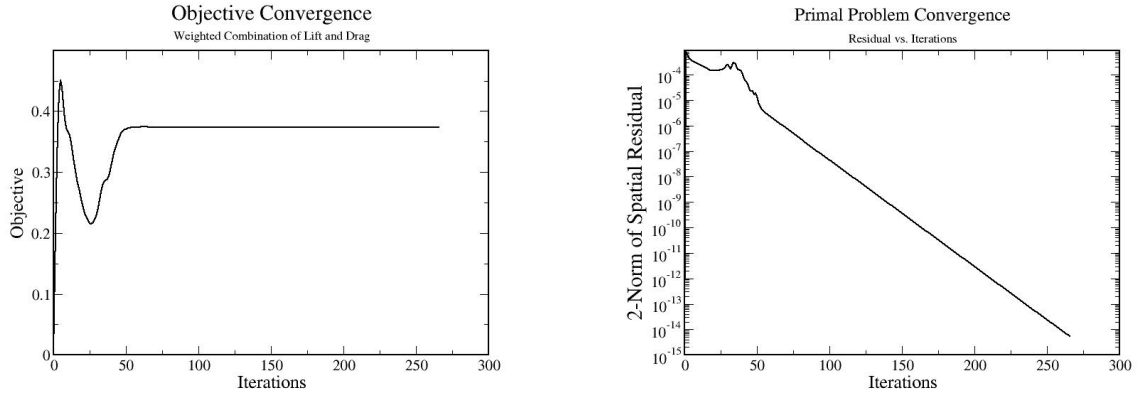
(b) Design Variable 2

Figure 4.11: Difference between pseudo-time accurate tangent computed sensitivities and complex sensitivities for Newton-Chord method at each pseudo-time step

4.2.1.3 Quasi-Newton Method

The verification in this section is done by using a very tight tolerance on the linear system of $1e-13$ at every non-linear iteration. The Hessian vector products are computed using complex-step perturbation to the conservative variable vectors and the mesh coordinate vector as stated previously. Figure (4.12) shows

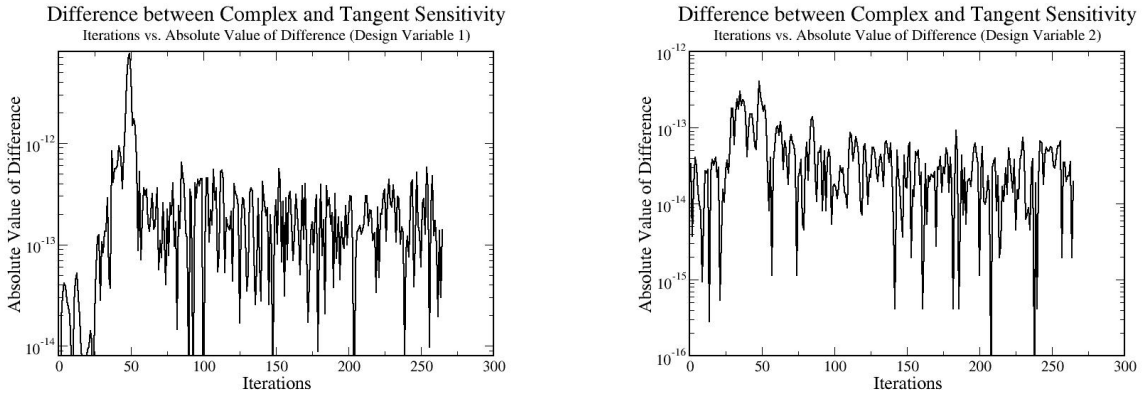
the convergence of the nonlinear residual and the objective of the analysis system. Figure (4.13) shows the difference between the complex and pseudo-time accurate tangent method sensitivities at each pseudo-time step. At each step machine level correspondence between the complex-step and tangent computed sensitivities is obtained; provided that the linear system is solved to machine precision. This is sufficient verification of the implementation.



(a) Functional

(b) Residual

Figure 4.12: Convergence of objective function and residual for quasi-Newton Method



(a) Design Variable 1

(b) Design Variable 2

Figure 4.13: Difference between pseudo-time accurate tangent computed sensitivities and complex sensitivities for quasi-Newton method at each pseudo-time step

4.2.2 Pseudo-time Accurate Adjoint Verification

Because the adjoint sensitivities are computed through reverse integration in pseudo-time, a comparison similar to the graphical representation in the tangent section is not practical. As such, Table 4.2 shows the difference between the complex and adjoint provided sensitivities on truncated runs for verification purposes.

It is clear that regardless of the convergence of the residual, which is marked by reporting the L_2-norm of the spatial residual at the final iteration in the rightmost column of the table, that exact correspondence between the sensitivities provided by the adjoint and complex-step methods is obtained. The table below that RK5 denotes the 5 stage low storage explicit Runge-Kutta scheme (of which forward Euler is a special case), NC denotes the frozen preconditioner Newton-Chord method, and QNII is the Quasi-Newton method using the identity for the derivative of an inverse matrix.

Scheme	Steps	Complex Sensitivity	Adjoint Sensitivity	$\|\|Residual\|\|_2$
RK5	50	2.582417731112833	2.582417731112833	0.2315064463096067E-03
RK5	200	1.8735681698763	1.8735681698761	0.1941186088514834E-03
RK5	25000	-36.6039235232560	-36.6039235232588	0.7237556735513925E-04
NC	10	-2.344315562026746	-2.344315562026735	0.3205858355993253E-03
NC	20	-5.11707061016280**4**	-5.11707061016280**7**	0.2522158472255535E-03
QNII	10	-3.482196439075767	-3.482196439075786	0.2982633094835574E-03
QNII	20	-2.264493935071775	-2.264493935071703	0.1691766297621063E-03

Table 4.2: Comparison of pseudo-time-accurate adjoint and complex-step computed sensitivities

4.3 Summary

This chapter presents verification of design order of accuracy as well as validation by comparison to the well verified Cart3D code from NASA Ames. The appropriate decrease of drag with mesh refinement was observed for a Gaussian bump case and the percentage difference in drag between the two codes was 2.8%. The steady state tangent and adjoint were then verified by comparing the computed sensitivities through those methods to those computed by the complex-step finite-difference method. Finally the pseudo-time accurate tangent and adjoint were verified by comparison to the results from the complex-step finite-difference method. The 5 stage explicit Runge-Kutta solver was linearized through the pseudo-time accurate tangent formulation and the sensitivities at each iteration were compared to those obtained through complex differentiation and showed machine zero level error at each iteration. Then the implicit Newton solver linearization was verified for both approximations of the linearization of the preconditioner matrix inverse. The Newton-Chord method was verified by running a simulation with a frozen preconditioner with a complex perturbation to the design variables that did not affect the preconditioner, thus making the Newton-Chord linearization an exact linearization of the solution process; the tangent sensitivities were compared to those obtained through complex differentiation and showed machine zero level error at each iteration. The quasi-Newton method linearization using the identity for the derivative of a matrix inverse was verified by converging the linear system to machine zero at each nonlinear iteration, a comparison between the tangent and complex linearization again showed machine zero level correspondence at every iteration, thus confirming the proper implementation. Finally, the adjoint linearizations of these solvers were verified by early termination of the simulations and full backwards-in-pseudo-time integration and comparison of the adjoint-computed sensitiv-

ities to the complex-step computed sensitivities. The sensitivities from the adjoint linearizations of all the nonlinear solvers show machine zero level difference with the complex-step computed sensitivities at these unconverged states, thus confirming proper implementation of the adjoint linearizations.

Chapter 5

Investigation into Tangent and Adjoint Computed Sensitivities

5.1 The Pseudo-Time Adjoint as a Green's Function

Referring back to 3.44 allows for a generalized Green's function of the fixed-point iterations. Starting with the adjoint recurrence relation at k:

$$\left[\frac{\partial G^{k-1}}{\partial u^{k-1}}\right]^T \Lambda^{k-1} = -\left[\frac{\partial G^k}{\partial u^{k-1}}\right]^T \Lambda^k - \left[\frac{\partial L}{\partial u^{k-1}}\right]^T \tag{5.1}$$

and turning this a perturbation form returns the equation below.

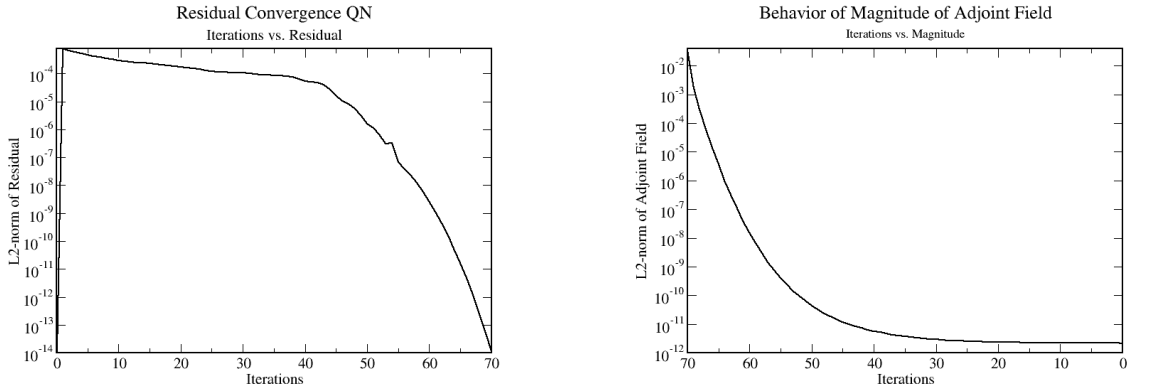
$$\delta L = -\Lambda^{k-1T}\delta G^{k-1} - \Lambda^{kT}\delta G^k \tag{5.2}$$

This corresponds to a generalized Green's function of the constraint operator (dependent on the time evolution method), where small perturbations in the time evolution operator produce a corresponding change in the objective at iterations where the output of interest depends on the flow field at that nonlinear iteration, otherwise it changes to the below equation.

$$0 = \Lambda^{k-1T}\delta G^{k-1} + \Lambda^{kT}\delta G^k \tag{5.3}$$

The generalized Green's function dictates how a perturbation in the constraint would change subsequent pseudo-time accurate flow adjoint vectors through the backwards-in-iteration-space integration. It is informative to look at the behavior of the adjoint for an exact Newton solver linearized in the adjoint mode with the inverse identity and measured by the magnitude of the L_2-norm of the adjoint field variables. Figure (5.1) shows that the magnitude of the norm of the adjoint field behaves as the inverse of the analysis problem convergence. The adjoint magnitude, plotted on a reversed x-axis, is greatest at convergence of

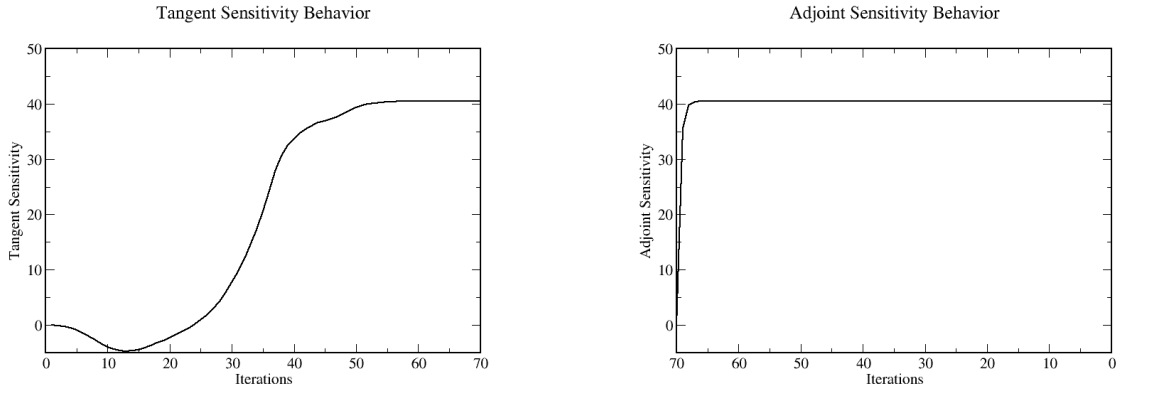
the analysis problem and near machine zero at initialization of the nonlinear problem. Furthermore, the adjoint magnitude is negligible during the region dominated by the pseudo-transient behavior early on in the analysis problem; it is only significant during the region dominated by the quadratic convergence behavior. Figure (5.2) shows that the adjoint very quickly integrates to near the final sensitivity as it integrates backwards through time; in contrast, the tangent system has very inaccurate gradients early on, and then only computes useful ones in the quadratic convergence region. This would indicate that full differentiation of the analysis solve is not necessary, rather the differentiation algorithm could be run only through the quadratic convergence region of the simulation.



(a) Residual Convergence

(b) Adjoint Magnitude Behavior (reversed x-axis)

Figure 5.1: Residual convergence and adjoint magnitude behavior for Quasi-Newton scheme



(a) Tangent Sensitivity Convergence

(b) Adjoint Sensitivity Convergence (reversed x-axis)

Figure 5.2: Sensitivity convergence for Quasi-Newton scheme

5.2 Sensitivity Behavior as a Function of Backwards-In-Iteration-Space Integration

In this section a comparison is done between the pseudo-time accurate adjoint and the classical – or steady state – adjoint. The first case is a well converging simulation that is truncated before the residual reaches machine zero, and compares the sensitivities from the steady state adjoint linearized about the final state to those from a pseudo-time accurate adjoint. Then there are two cases which fail to converge and enter limit cycle oscillations; these cases show the effect of increasing the averaging window of the objective functional on the pseudo-time accurate adjoint-computed sensitivities. Next, the sensitivities of the steady state adjoint linearized about the final state of the simulation as well as about the averaged state are shown and these sensitivities are compared to those provided by the pseudo-time accurate adjoint using the same functional averaging window as that used to generate the average state. Finally, we also consider the effect of averaging the computed sensitivities.

The objective functional for a simulation that runs for n pseudo-time steps is either a windowed combination of the coefficients of lift and drag, in which case the lift and drag coefficients are averaged over the last m pseudo-time steps, or an instantaneous combination at the final state, which can be calculated by setting $m = 1$. The windowed objective function is written below, where ω is the weight, in this work $\omega = 8$.

$$L = \frac{1}{m} \sum_{i=n-m+1}^{n} C_{L_i} + \omega C_{D_i} \tag{5.4}$$

The comparisons in this section will compare the values of the sensitivity vectors computed from the different methods to one another, as well as the directions of those sensitivity vectors. The sensitivity vector direction is typically used in the line search methods found in most gradient-based optimizers, and errors in the search direction can lead to suboptimal designs or outright failure of the design process. All cases to follow in this investigation on sensitivity behavior as a function of the backwards-in-iteration-space integration were run with the low storage explicit five stage Runge-Kutta scheme; the adjoint linearizations were the exact linearizations of this scheme.

5.2.1 Application of the Pseudo-Time Accurate Adjoint to a Truncated Simulation

This case shows that, for a well converging primal problem which is truncated before deep convergence and which has a well converging adjoint system, there is a significant difference in the values between the pseudo-time accurate adjoint and the steady state adjoint-computed sensitivities. For this case the primal problem was solved for a NACA0012 airfoil on the mesh shown for verification in Figure 4.7, at $Mach = .85$ and $\alpha = 3^o$. The simulation is terminated when the L_2-norm of the residual is less than $1e-6$. The resulting

density field is shown in Figure 5.3.

The next step was to solve the steady state adjoint problem linearized about the final state; the convergence plots of the nonlinear residual in the primal problem and the linear residual in the adjoint problem are shown in Figure 5.4. The design variables consisted of 4 Hicks-Henne bump functions equally spaced at one-fifth, two-fifths, three-fifths and four-fifths of the chord length of the airfoil. The mesh sensitivities were calculated from the spring analogy method.

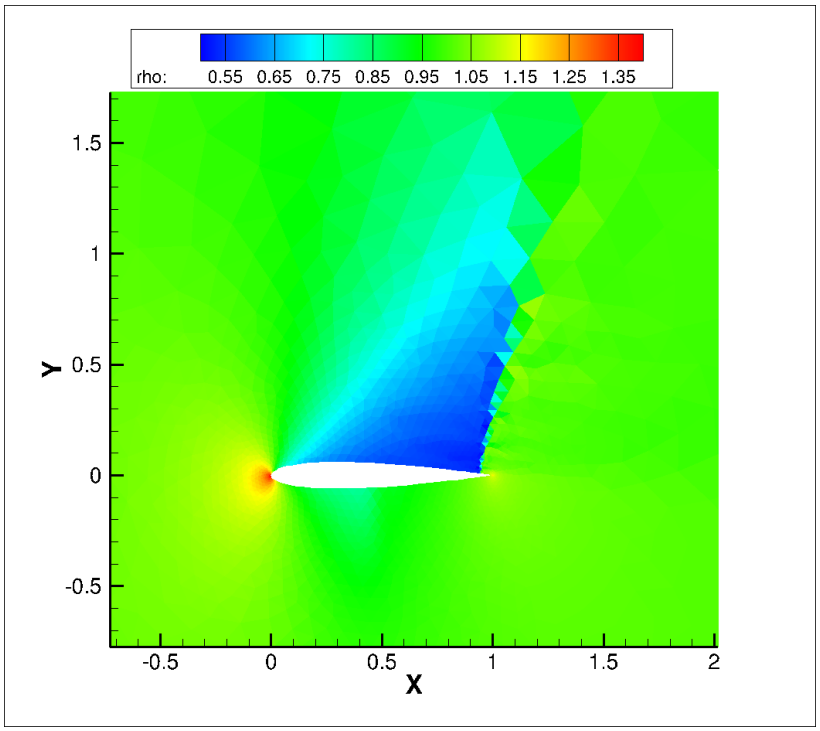
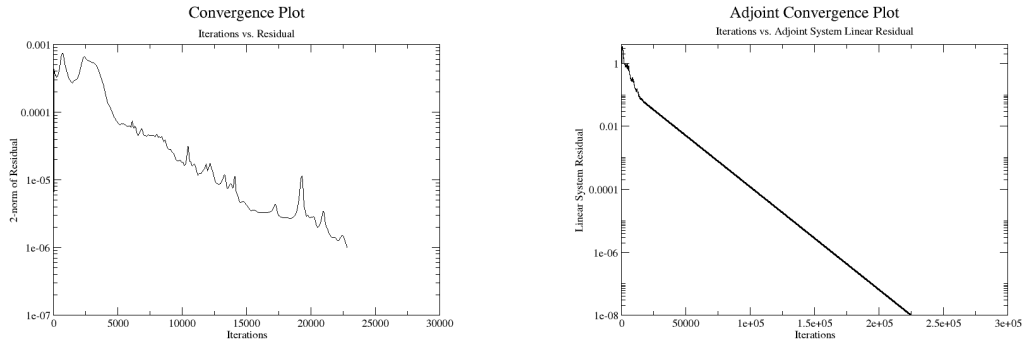


Figure 5.3: Density field for NACA0012 airfoil in $Mach = .85$, $\alpha = 3^o$

Table 5.1 shows the magnitude and percentage differences between the the steady state adjoint-computed sensitivities to the pseudo-time accurate adjoint-computed sensitivities. This table depicts that there are clearly large differences between the sensitivities obtained by the two approaches. It is instructive to then evaluate the angle between the two sensitivity vectors, as the direction is the most important part of the sensitivity vector for the optimization process (within reasonable magnitude errors). The angle θ is calculated as follows, with u and v being the sensitivity vectors.

$$a = \frac{u \cdot v}{\|u\|\|v\|} \tag{5.5}$$

$$\begin{aligned} \theta &= arccos(a), \theta \leq 180 \\ \theta &= 360 - arccos(a), \theta > 180 \end{aligned} \tag{5.6}$$



(a) Nonlinear Residual Convergence

(b) Adjoint Linear Convergence

Figure 5.4: Residual and steady state adjoint convergence for truncated simulation in $Mach = .85$, $\alpha = 3^o$

Design Variable	Steady State Value	Pseudo-Time Accurate Value	Percent Difference
1	-38.9168562768552	-34.07260014448312	14.2%
2	-58.8314866801896	-62.28918541979274	5.6%
3	-51.9374815863084	-70.15517817818902	26.0%
4	-69.3520141341324	-95.35571541527936	27.3%

Table 5.1: Comparison of pseudo-time accurate adjoint and steady state adjoint-computed sensitivities for truncated primal problem in $Mach = .85$, $\alpha = 3^o$

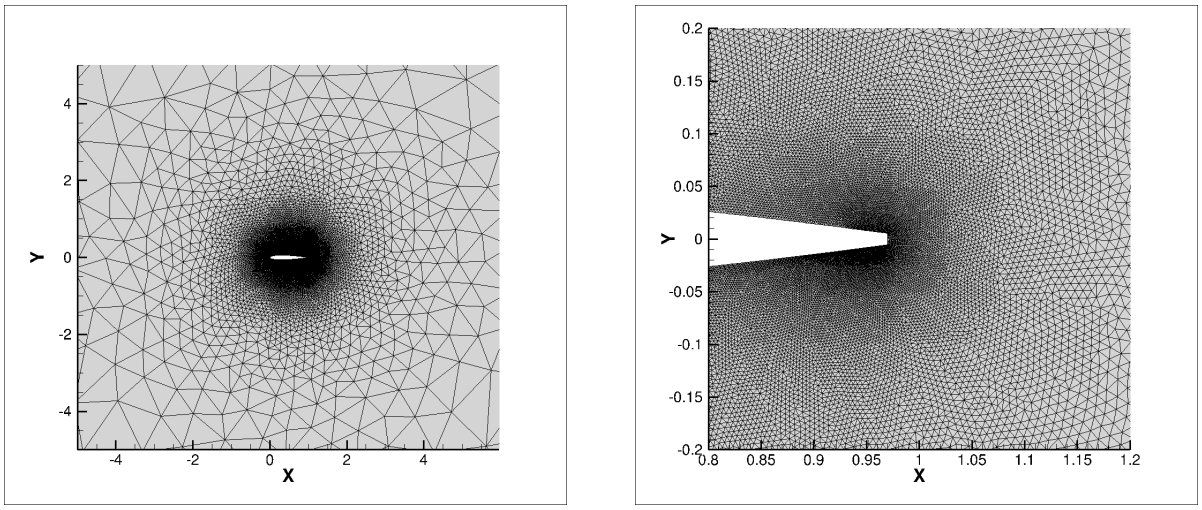
Using the values for the sensitivity vectors shown above in Table 5.1 it is possible to obtain a value of the angle between the two sensitivity vectors of $\theta = 8.652^o$. Between the angle value and difference in the magnitude values it is clear that there are significant discrepancies between the results of these two methods of sensitivity computation. Even in cases of well converging primal problems there are significant benefits from the use of the pseudo-time accurate adjoint as there is a significant difference between the steady state adjoint sensitivities and the exact sensitivities of the partially converged solution as obtained by the pseudo-time accurate adjoint formulation.

5.2.2 Application of the Pseudo-Time Accurate Adjoint to Non-converging Primal Problem

A primal problem for a cutoff airfoil (NACA0012 with a blunt trailing edge at 97% of the chord length) was solved using a fine mesh with 64398 triangular elements shown in Figure 5.5 in order to devise a non-convergent steady state problem.

5.2.2.1 Subsonic Case

The flow conditions for this test case are given by $Mach = .3$ and $\alpha = 3^o$. Figure 5.6 shows the convergence of the residual, lift coefficient and drag coefficient through 60000 iterations, at which point the residual stopped decreasing when it was run out to 80000 iterations.



(a) Mesh (b) Trailing Edge

Figure 5.5: Fine mesh for NACA0012 airfoil cut off at 97% chord length

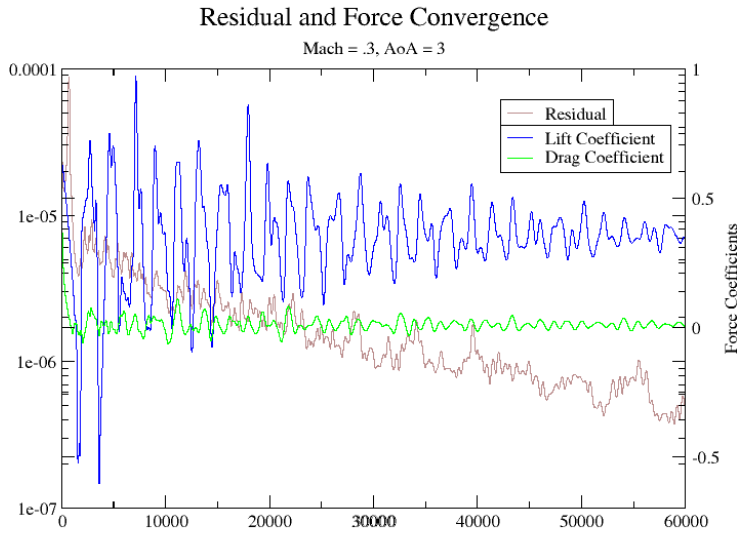
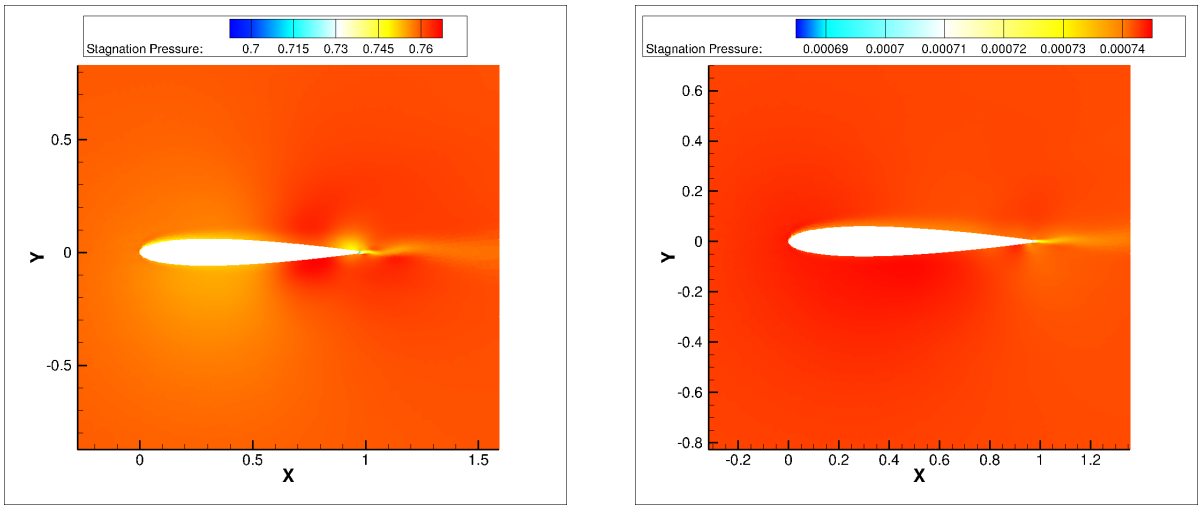


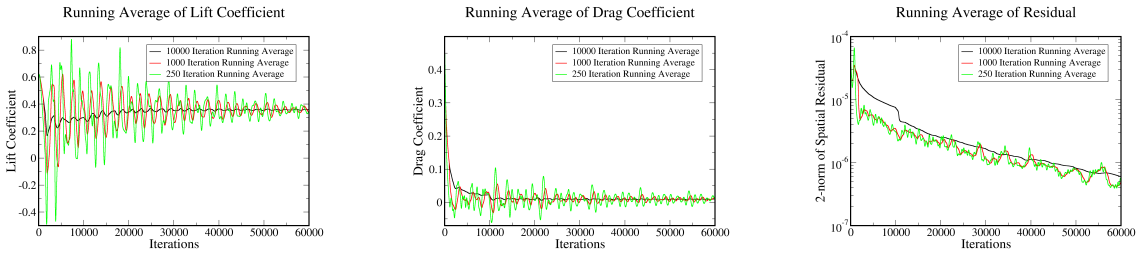
Figure 5.6: Primal problem convergence for $Mach = .3$, $\alpha = 3^o$

Figure 5.7 shows the stagnation pressure values for the final state and the averaged state. The figure shows that for the instantaneous final state there is shedding at the trailing edge of the airfoil, and the oscillations in the flow field are far stronger than those at the averaged state. Figure 5.8 shows the behavior of the coefficient of lift and drag over different averaging windows. It is clear that increasing the averaging window damps the oscillations and gives a more convergent behavior of the force coefficients, as the two smaller windows show large oscillations, with the largest window showing negligible oscillations. Averaging



(a) Final State (b) Average State

Figure 5.7: Stagnation pressure for final state and average state for $Mach = .3$, $\alpha = 3^o$

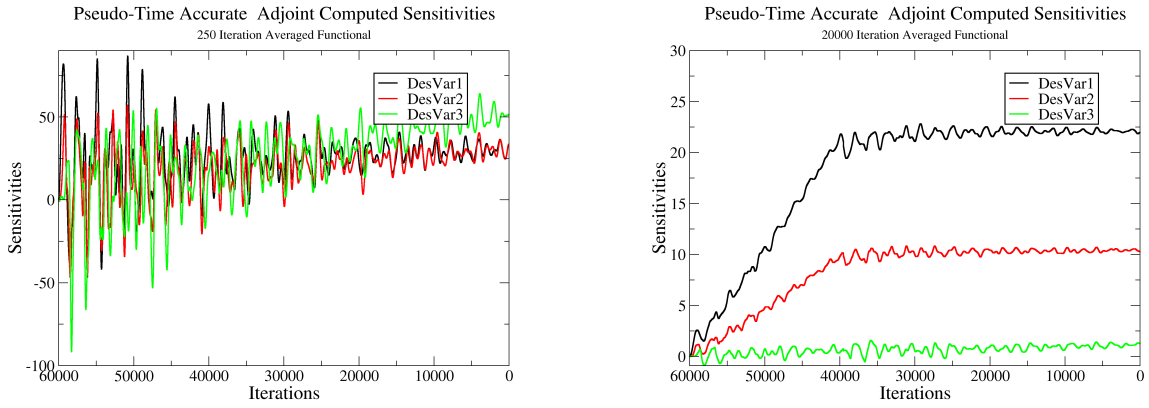


(a) Lift Coefficient (b) Drag Coefficient (c) Residual

Figure 5.8: Behavior of primal problem for different averaging windows for $Mach = .3$, $\alpha = 3^o$

the force coefficients leads to well behaved values over long intervals, which is a common practice when evaluating CFD results for engineering applications.

Figure 5.9 shows the convergence of the sensitivities back through pseudo-time on reversed x-axes when computed by the pseudo-time accurate adjoint of the Runge-Kutta solver. It is clear that by increasing the size of the objective functional averaging window, the magnitude of the sensitivities grows smaller. It is also clear that the sensitivities provided by the pseudo-time accurate adjoint for an objective functional averaged over a long window approach the final value after the adjoint integrates back through the averaging window. In contrast, the adjoint for a smaller averaging window objective functional does not show this behavior. It is clear that for the longer averaging window the oscillations in the sensitivities about this fully pseudo-time integrated value are smaller. In order to evaluate the use of the computed sensitivities for a gradient-based optimizer, it is helpful to evaluate the evolution of the angle between the partially (backwards-in-time) integrated adjoint sensitivity vector to the fully integrated adjoint sensitivity vector as the adjoint integrates back through pseudo-time. This is done to measure the feasibility of early termination

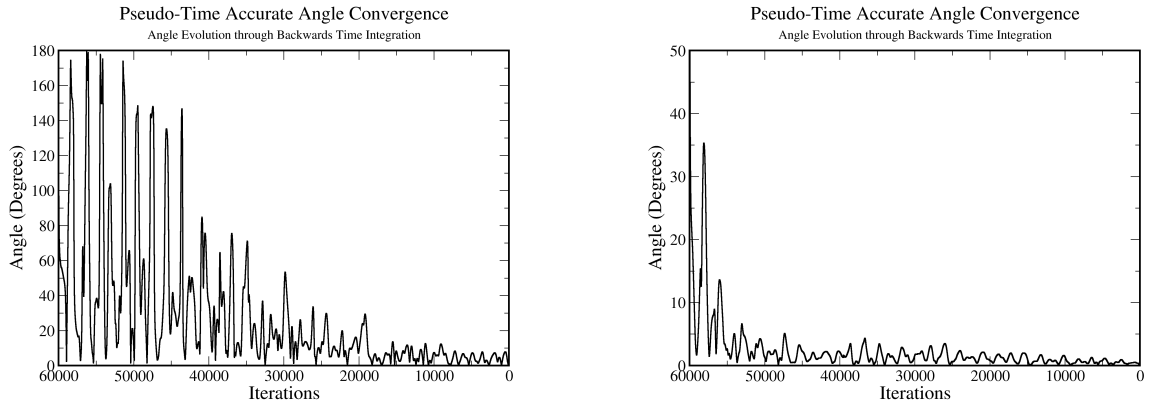


(a) 250 Iteration Window

(b) 20000 Iteration Window

Figure 5.9: Pseudo-Time accurate adjoint sensitivities for varying objective windows for $Mach = .3$, $\alpha = 3^o$

of the time integration. In fact, for the longer window objective function, the angle between the partially integrated versus the fully integrated adjoint-computed sensitivities is well behaved as is demonstrated in Figure 5.10. The fully pseudo-time integrated adjoint sensitivities are used as the comparison point as they represent the exact sensitivities and correspond to the complex-step finite-difference computed sensitivities. By integration back through pseudo-time to roughly twice the averaging window, for well behaved cases, the algorithm returns sensitivities that are very close to the exact sensitivities.



(a) 250 Iteration Window

(b) 20000 Iteration Window

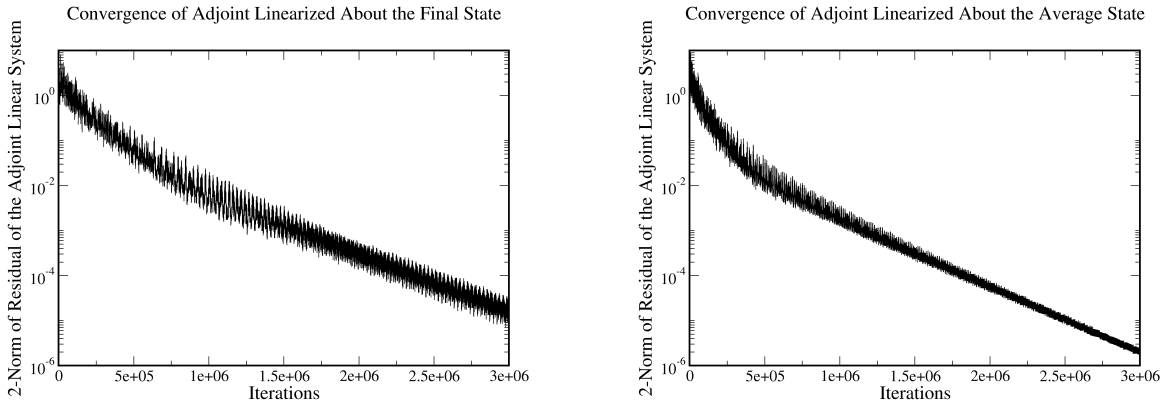
Figure 5.10: Angle between partially time-integrated and fully time-integrated sensitivities over iteration space to final sensitivity for $Mach = .3$, $\alpha = 3^o$

It is then enlightening to compare how the pseudo-time accurate adjoint-computed sensitivities compare to the values obtained by the steady state adjoint linearized about different states. The comparison is between the pseudo-time accurate adjoint using the objective functional average over the last 20000 iterations and

Design Variable	Steady State Value (Final State)	Pseudo-Time Accurate Value	Percent Difference
1	21.3443558908559	21.96809430870143	2.83%
2	9.15634414104837	10.26553487597339	10.8%
3	-2.88279274401136	1.239597303802120	332.56%

Table 5.2: Pseudo-Time accurate adjoint and steady state sensitivities computed at final state for non-converging primal problem for $Mach = .3$, $\alpha = 3^o$

the steady state adjoint evaluation at both the final state and an averaged state produced by averaging the conservative variables over the last 20000 iterations, the same window as the functional driving the pseudo-time accurate adjoint. Figure 5.11 depicts the plots of the convergence of the two steady state adjoint systems. It is good to note that it takes many iterations to converge these systems, as the adjoint linearized about a partially converged state has been found to be hard to converge [33]. The pseudo-time accurate adjoint does not suffer from this as it is the transpose linearization of the primal problem, and therefore does not require more advanced solver technology than the primal problem itself and converges at the same rate.



(a) Final State

(b) Average State

Figure 5.11: Steady state adjoint convergence for $Mach = .3, \alpha = 3^o$

Table 5.2 provides the computed sensitivity vectors and from this information the angle between the two sensitivity vectors is evaluated. The first vector is computed by the steady state adjoint linearized about the final state and the second is computed by the pseudo-time accurate adjoint. In this case the angle between the two vectors is obtained with a value of $\theta = 10.167^o$. This is a significant angle difference, and could cause problems for optimization with a gradient-based optimizer.

Table 5.3 compares two sensitivity vectors, calculated by the steady state adjoint linearized about the averaged state and the pseudo-time accurate adjoint respectively. In this case, the angle between these vectors is $\theta = 1.9396^o$. When the simulation has entered limit cycle oscillations, averaging the state greatly improves the performance of the steady state adjoint. However, although the angle is relatively small in this case the magnitudes of the individual sensitivities still differ substantially.

Design Variable	Steady State Value (Avg State)	Pseudo-Time Accurate Value	Percent Difference
1	22.6224747115315	21.96809430870143	2.97%
2	10.5354461565917	10.26553487597339	2.63%
3	0.430506286872907	1.239597303802120	65.27%

Table 5.3: Pseudo-Time accurate adjoint and steady state sensitivities computed at averaged state for non-converging primal problem for $Mach = .3$, $\alpha = 3^o$ flow

5.2.2.2 Transonic Case

In this section the flow conditions are changed to $Mach = .7$, $\alpha = 2^o$. Figure 5.12 shows the convergence of the residual, lift coefficient and drag coefficient, and the plots extend far into the limit-cycle oscillation region.

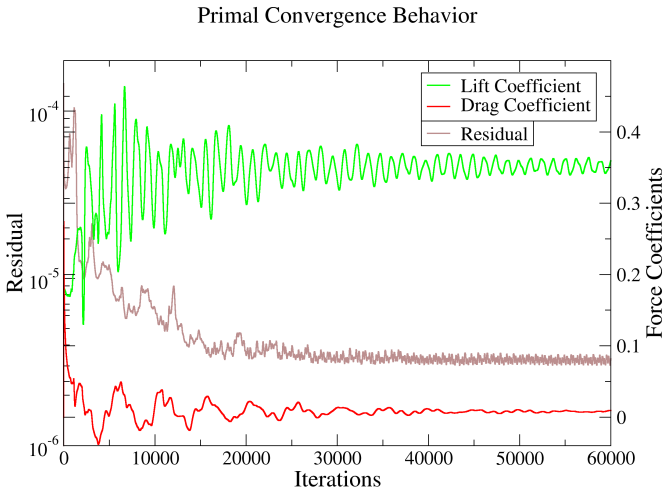
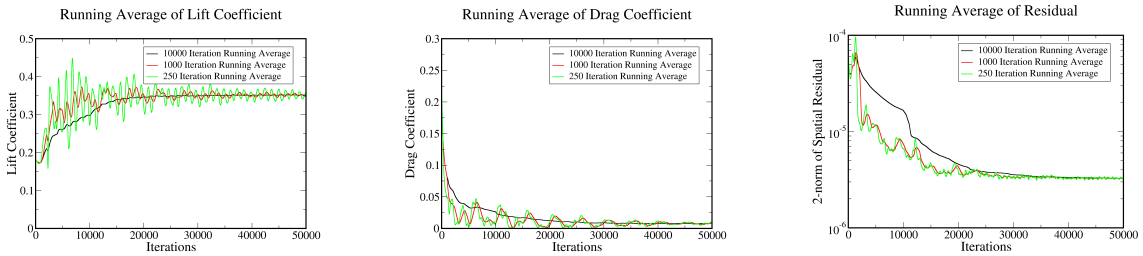


Figure 5.12: Primal problem convergence for cut-off NACA0012 airfoil at $Mach = .7$, $\alpha = 2^o$

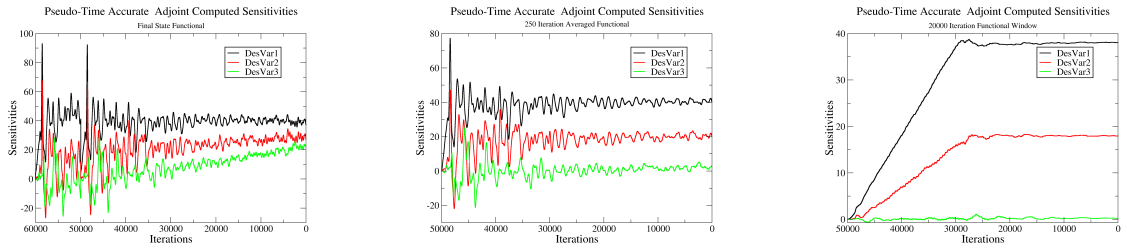


(a) Lift Coefficient (b) Drag Coefficient (c) Residual

Figure 5.13: Behavior of primal problem for different averaging windows for $Mach = .7$, $\alpha = 2^o$

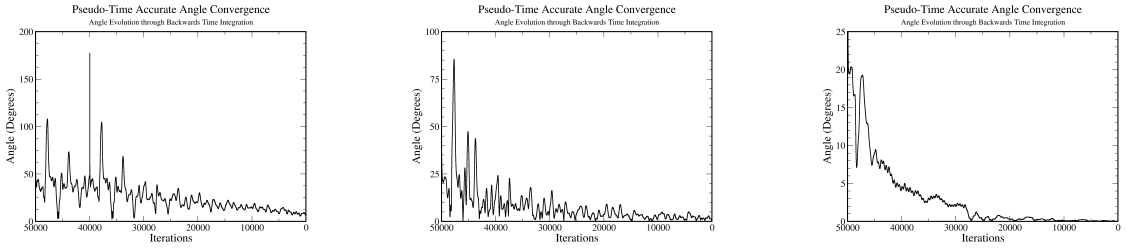
Figure 5.13 shows the behavior of the residual and the coefficients of lift and drag over different averaging windows. It is clear that the convergence behavior with averaging is better here than in the subsonic case, and

that for a 10000 iteration averaging window the average converges well. Figure 5.14 shows the convergence of the sensitivities back through time on reversed x-axes when computed by the pseudo-time accurate adjoint. As in the subsonic case, increasing the size of the averaging window for the objective functional gives smaller magnitude sensitivities. Additionally, by extending the simulation far into the region of limit cycle oscillations, very small oscillations in the values of the sensitivities are obtained. As before, it is helpful to look to the angle behavior in order to get a better sense of how truncating the backwards-in-pseudo-time integration may affect the line-search of a gradient-based optimizer.



(a) Final State (b) 250 Iteration Window (c) 20000 Iteration Window

Figure 5.14: Pseudo-Time accurate adjoint sensitivities for varying objective windows for $Mach = .7$, $\alpha = 2^o$



(a) Final State (b) 250 Iteration Window (c) 20000 Iteration Window

Figure 5.15: Angle convergence over iteration space to final sensitivity for $Mach = .7$, $\alpha = 2^o$

Figure 5.15 depicts the angle between the partially and fully integrated sensitivities for different objective functional average windows. It is clear that by increasing the functional averaging window better angular behavior is achieved when comparing the partially pseudo-time integrated sensitivity vector to the fully pseudo-time integrated sensitivity vector. Looking at the angle between the partially pseudo-time integrated and fully pseudo-time integrated sensitivity vectors as the algorithm integrates back through one and half times the averaging window, the angles are on the order of $.1^o$ or $.01^o$ for the largest functional averaging window.

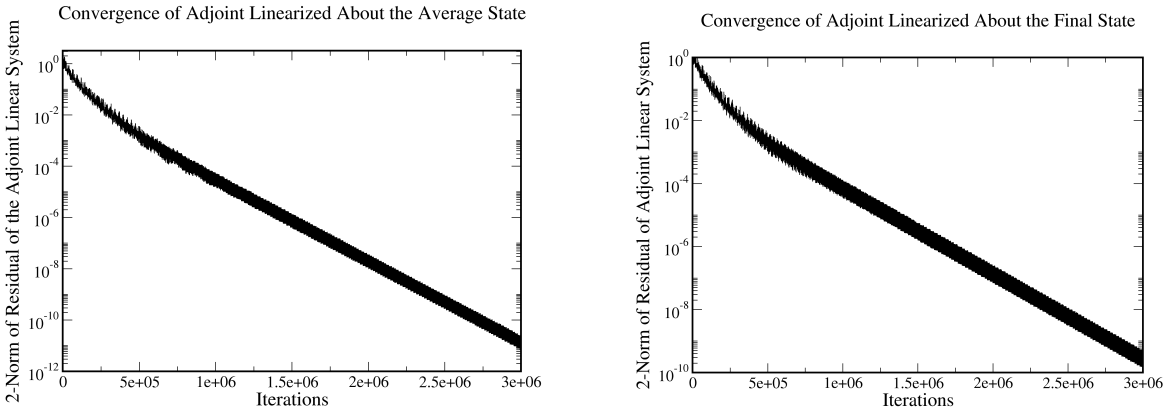
It is then instructive to compare the pseudo-time accurate adjoint sensitivities to those provided by the steady state adjoint. Figure 5.16 shows the convergence of the linear residual of the steady state adjoint systems, linearized about the final state and averaged state respectively. These plots show, as before, that

computing these adjoint systems is very expensive, more expensive than the pseudo-time accurate adjoint calculation.

Design Variable	Steady State Value (Final State)	Pseudo-Time Accurate Value	Percent Difference
1	44.5805974889098	38.03996816573513	17.19%
2	14.0224144011606	17.92859539270058	21.79%
3	-1.27443407446669	0.2080948770865852	712.43%

Table 5.4: Pseudo-Time accurate adjoint and steady state sensitivities for non-converging primal problem $Mach = .7$, $\alpha = 2^o$

As part of the comparison between the pseudo-time accurate adjoint-computed sensitivities and the final state steady state adjoint-computed sensitivities, the angle between the vectors is computed from the values in Table 5.4 and an angle of $\theta = 7.98972^o$ is obtained. As before the sign of the sensitivity of the third design variable in the steady state adjoint sensitivity vector is negative as opposed to the pseudo-time accurate sensitivity vector which has positive sign. This being an undesirable result for optimization, it is considered to be an unfeasible use of the steady state adjoint to linearize about the final state and it is advisable to move onto the steady state adjoint linearized about the averaged state, created by averaging the last 20000 iterations.



(a) Average State

(b) Final State

Figure 5.16: Steady State adjoint convergence for $Mach = .7$, $\alpha = 2^o$

Design Variable	Steady State Value (Avg State)	Pseudo-Time Accurate Value	Percent Difference
1	43.3642585762660	38.03996816573513	14.00%
2	16.9218981680917	17.92859539270058	5.62%
3	-9.717943674789181E-002	0.2080948770865852	146.70%

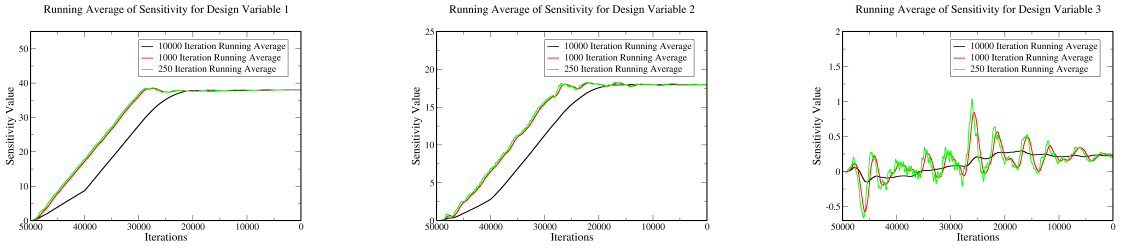
Table 5.5: Pseudo-Time accurate adjoint and steady state sensitivities for non-converging primal problem for $Mach = .7$, $\alpha = 2^o$

Table 5.5 shows a comparison between the pseudo-time accurate adjoint-computed sensitivities and the

averaged state steady state adjoint-computed sensitivities. From these values a smaller angle of $\theta = 3.93855^o$ is obtained– which is less than half the previous angle from the final state steady state adjoint– but once more the third design variable shows opposing signs for its sensitivity.

Given the significant magnitude differences and the angle between the steady state adjoint-computed sensitivity vectors and the pseudo-time accurate adjoint-computed sensitivity vectors, it is desirable to seek additional techniques for improving the accuracy of the partially integrated pseudo-time accurate adjoint. Since Figure 5.14c shows the pseudo-time accurate adjoint provided sensitivities are well behaved except for oscillations about the mean after the backwards-in-time integration is complete through the functional averaging window, the effects of averaging the sensitivity values themselves for the pseudo-time accurate adjoint problem when driven by a large functional averaging window are investigated.

Figure 5.17 depicts running averages of the sensitivities computed by the pseudo-time accurate adjoint for the largest objective functional averaging window. The instantaneous values of the sensitivities can be found for comparison in Figure 5.14c. The value of the running average of the sensitivity when the sensitivity averaging window is outside the functional averaging window shows a dampening of the oscillations. In this case, where the functional is averaged over the last 20000 iterations and the sensitivity over the last 10000,the value 30000 iterations through the backwards-in-pseudo-time integration is compared to the final sensitivity calculation for the non averaged – or instantaneous – sensitivities. It is clear that averaging over the largest window damps out the oscillations, but for smaller windows it is not as effective, especially in the averaging of the highly oscillatory third design variable sensitivity. Then comparing the partially integrated and averaged sensitivity vector to the fully time integrated and instantaneous sensitivity vector in Table 5.6 gives an angle of $.34481^o$ between the two vectors. This value is an order of magnitude smaller than the angle obtained when comparing the fully time integrated pseudo-time accurate adjoint-computed sensitivities to the steady state adjoint-computed sensitivities when linearized about the averaged state. Additionally, it is clear that the magnitude differences are far smaller. Further research was done into smaller sensitivity averaging windows, but these smaller windows led to more oscillatory average sensitivities. The results show that the use of a sensitivity averaging window of approximately 50% of the functional averaging window is a reasonable choice for these test cases.



(a) Design Variable 1 (b) Design Variable 2 (c) Design Variable 3

Figure 5.17: Effect of averaging sensitivities for pseudo-time accurate adjoint for $Mach = .7$, $\alpha = 2^o$

Design Variable	10000 Iteration Average Value	Instantaneous Value	Percent Difference
1	37.815963808090757	38.03996816573513	.5%
2	17.554182903957241	17.92859539270058	2%
3	.26808010969592244	0.2080948770865852	28%

Table 5.6: Comparison of partially integrated averaged pseudo-time accurate averaged sensitivities (computed at iteration 30000) to fully backwards integrated instantaneous pseudo-time accurate sensitivities for $Mach = .7$, $\alpha = 2^o$

5.3 Sensitivity as a Function of Accuracy of Approximation of Fixed Point Linearization

This section shows the effect of an approximate linearization on the accuracy of the sensitivity computation. All cases were run at $M = .7, \alpha = 2^o$ with an objective function of the form $J = c_L^2 + c_D^2$. The first set of results shows the impact of exactly linearizing the residual operator at every stage of a Runge-Kutta scheme when the gradients are only computed at the first stage and then frozen throughout the sub-stage iterations. The final two sets of results portray the effect of partial linear solves on the accuracy of the identity of the differentiation of the inverse matrix (shown in equation (3.16)) in the sensitivity computation.

5.3.1 Results for Inexactly Linearized Explicit Runge-Kutta Solver

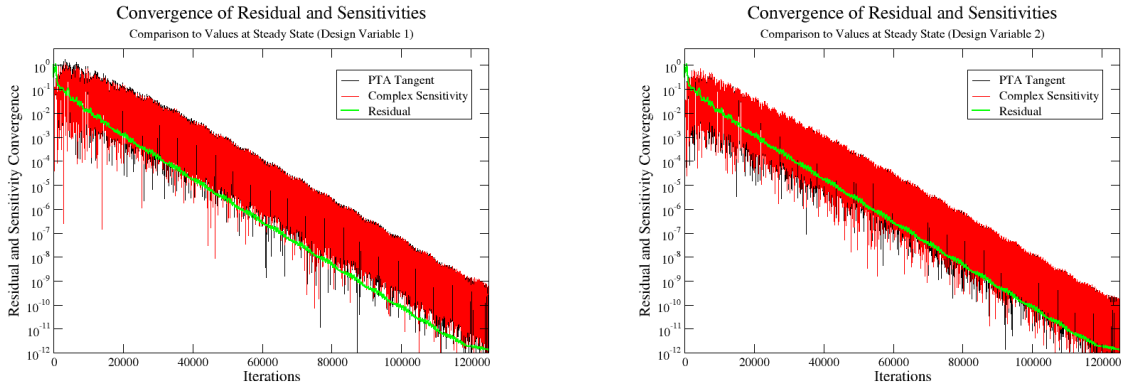
In this section an explicit Runge-Kutta solver is investigated. The solver, as expressed below, iterates k through pseudo-time (which Δt being a variable local time-step) and l through stages 1 to 5.

$$u^{k,l} = u^{k,0} + CFL\alpha^{l-1}\Delta t R(u^{k,l-1}) \tag{5.7}$$

with the end of the sub-stage time-stepping being governed as follows.

$$u^{k,0} = u^{k-1,5} \tag{5.8}$$

To check the effect of inexact linearization of the fixed point iterations for an explicit Runge-Kutta solver there is inaccuracy introduced through selective freezing and computation of the flow gradients. The flow gradients for the zero stage are calculated and and then held frozen through the fixed point iteration that solves the primal problem. For the linearization of the fixed point iteration the gradients are calculated at each stage and the Jacobian is computed and updated accordingly. This allows the Runge-Kutta scheme to be viewed as some A operator depending on the flow state and design variables that multiplies the 0 stage residual to get the final stage Δu, where the linearization of A is inexact due to the inconsistent handling of the gradients. The convergence of the flow sensitivities to their respective final values is shown in Figure



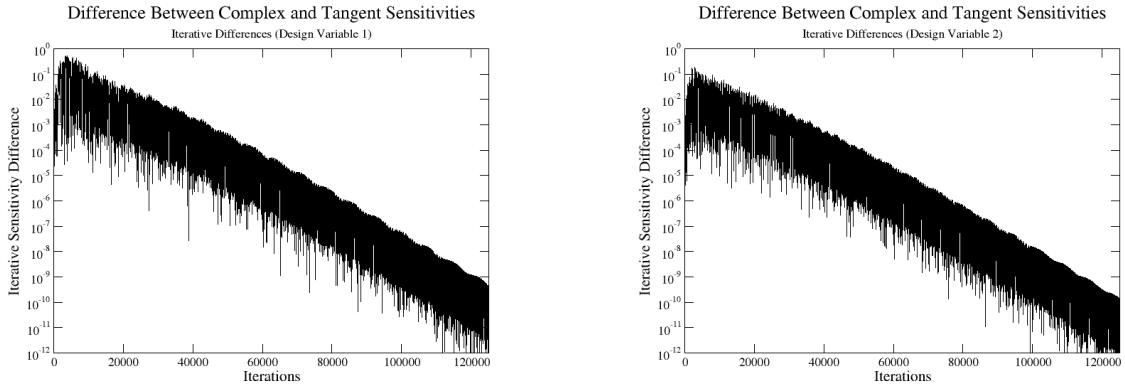
(a) Maximum Iterative Difference

(b) Minimum Iterative Difference

Figure 5.18: Runge Kutta Sensitivity Convergence

(5.18). It is clear that they converge to their respective final values as the primal problem converges, as expected.

The difference between the exact (complex) linearization and the approximate (tangent) linearization portrayed in Figure (5.19) shows the expected behavior. The error due to the inexact linearization of the Runge-Kutta scheme goes away at the rate of the primal problem convergence as derived in the proof section.



(a) Maximum Iterative Difference

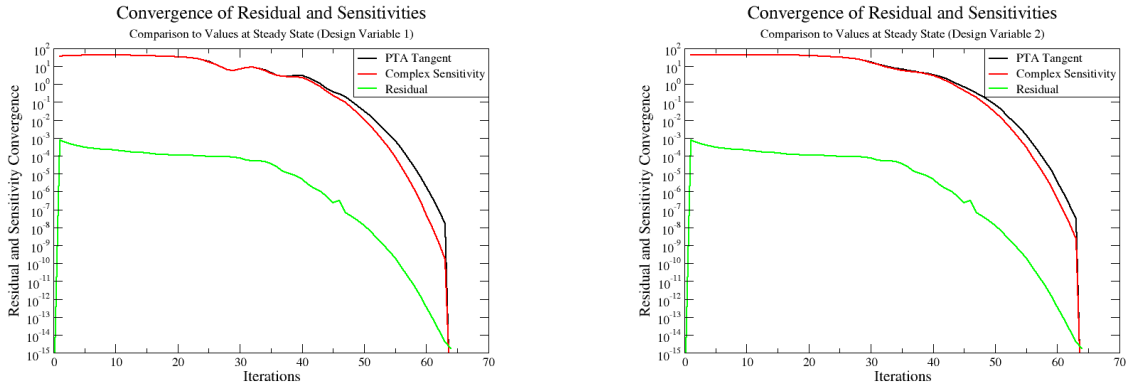
(b) Minimum Iterative Difference

Figure 5.19: Runge Kutta Sensitivity Difference

5.3.2 Results for an Exact Jacobian Augmented with a Mass Matrix

This section shows the results for an exact quasi-Newton solver with the tangent and adjoint linearizations being computed using the inverse identity method. Figure (5.20) shows that for the linear tolerance $1e-1$ that the complex and tangent sensitivities converge to their final values at the same rate as the analysis

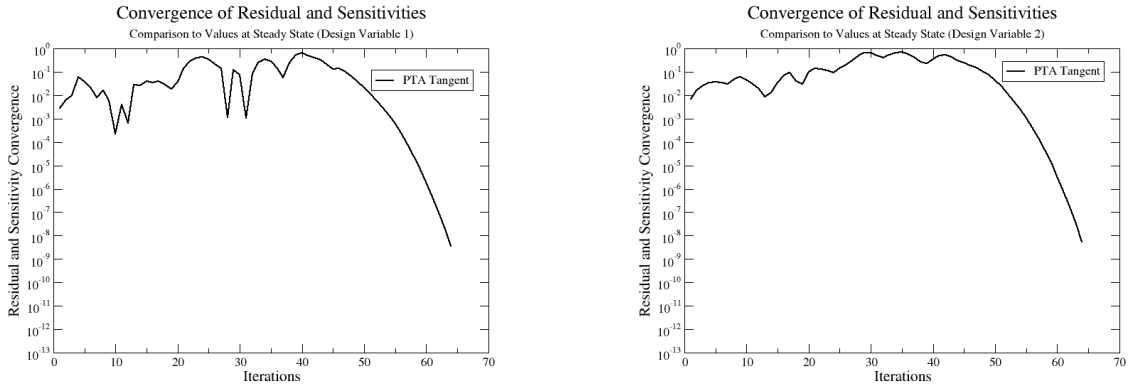
problem itself, which is expected based on the formulation. Figure (5.21) depicts the difference between the complex and tangent sensitivities over the iteration history of the analysis solution process, and it is clear that the maximum difference is of the order of the linear tolerance of the linear system. Furthermore, as the analysis problem converges, so do the complex and tangent sensitivities to each other despite the inexact differentiation. At full convergence of the analysis problem, the tangent and complex-step sensitivities correspond to each other to a high degree of precision and these also correspond to the steady state tangent and adjoint computed sensitivities linearized about the converged analysis state.



(a) Design Variable 1

(b) Design Variable 2

Figure 5.20: Sensitivity convergence for linear tolerance in a Newton solver, $1e-1$: difference between current and final sensitivity values



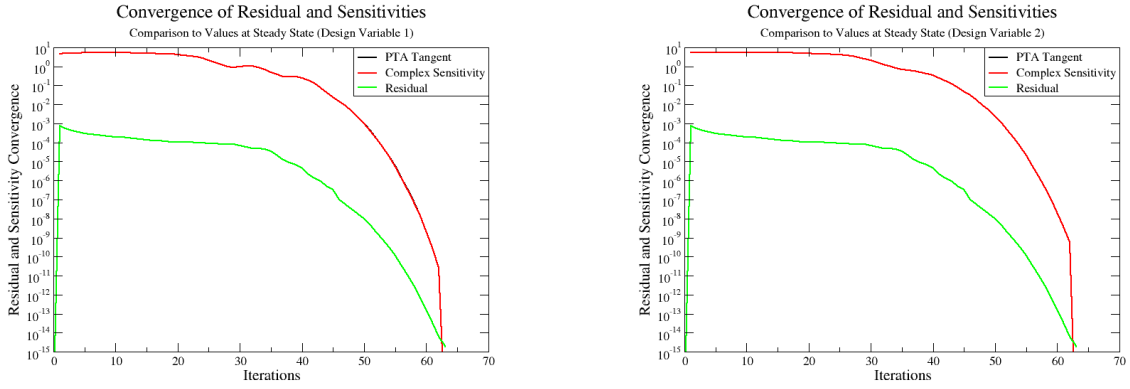
(a) Design Variable 1

(b) Design Variable 2

Figure 5.21: Iterative sensitivity difference for linear tolerance in a Newton solver, $1e-1$

Figure (5.22) contains the same information as Figure (5.20), and Figure (5.23) is the sister plot of Figure (5.21); the latter two plots show results with a tighter linear system tolerance of $1e-4$. As in the looser linear tolerance case, as the nonlinear problem converges so do the tangent and complex sensitivities to each

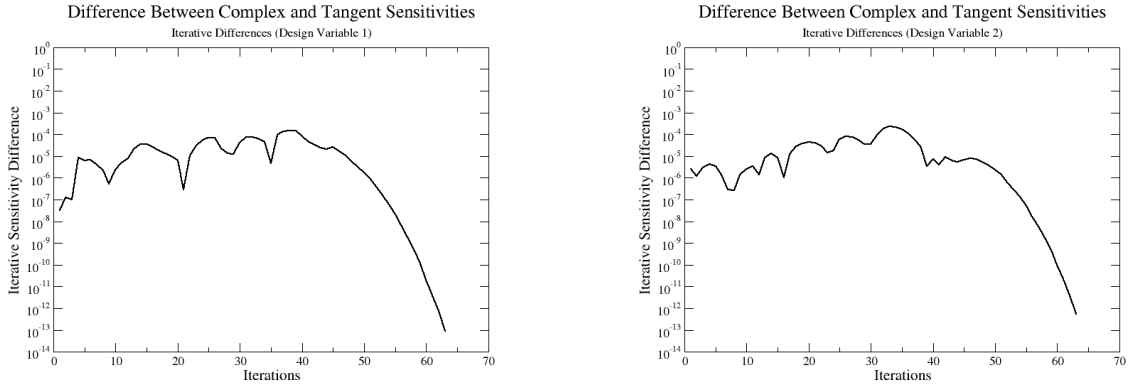
other, down to nearly machine precision correspondence. Furthermore, the maximum iterative difference is again on the order of the linear system tolerance.



(a) Design Variable 1

(b) Design Variable 2

Figure 5.22: Sensitivity convergence for linear tolerance in a Newton solver, $1e-4$: difference between current and final sensitivity values

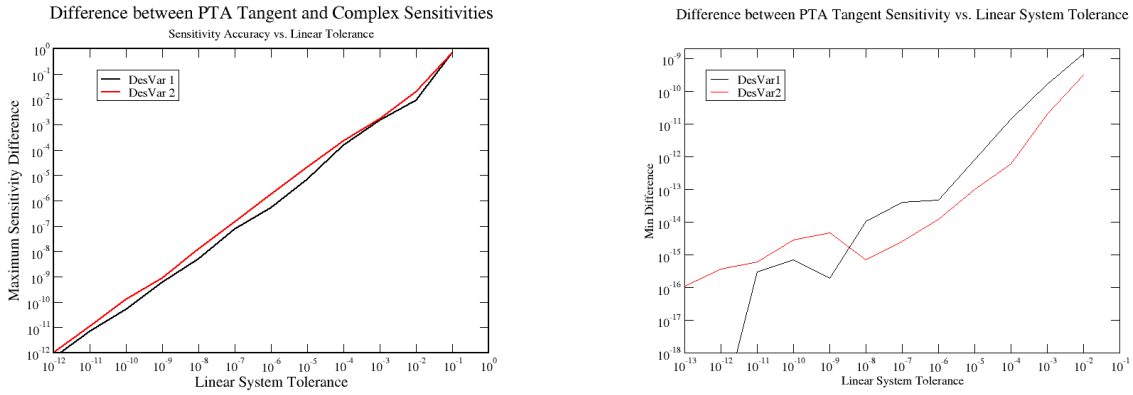


(a) Design Variable 1

(b) Design Variable 2

Figure 5.23: Iterative sensitivity difference for linear tolerance in a Newton solver, $1e-4$

Having seen the impact of the tighter linear system tolerance on the maximum iterative difference between the tangent and complex sensitivities, the analysis problem with linear system tolerances logarithmically spaced through the range $1e-1$ to $1e-13$ is simulated and the maximum iterative difference is plotted in Figure (5.24). As the figure shows, the maximum iterative difference is directly related to the linear system tolerance. This allows for good estimates of the maximum iterative error as a function of the linear system tolerance. The minimum iterative difference shows similar behavior, and converges as the product of the linear system tolerance and the normalized residual of the nonlinear problem.



(a) Maximum Iterative Difference

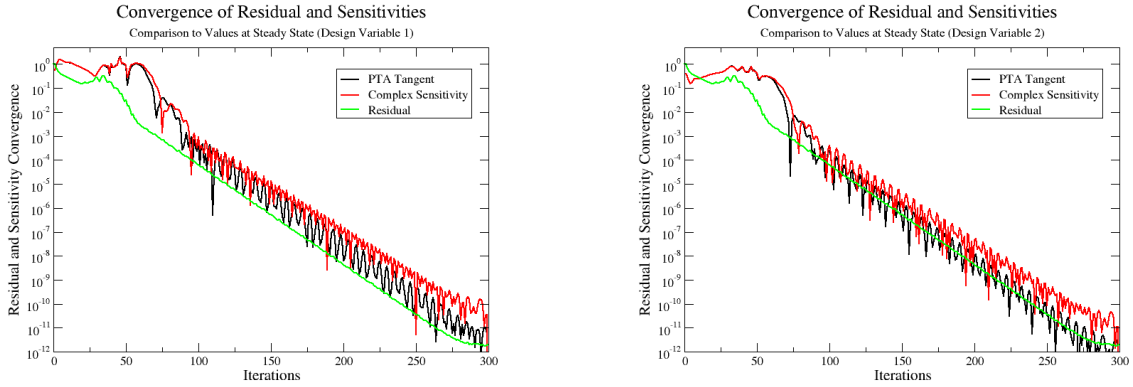
(b) Minimum Iterative Difference

Figure 5.24: Iterative difference vs. linear tolerance in a Newton solver

5.3.3 Results for an Inexact Jacobian Augmented with a Mass Matrix

Here the results are presented for an inexact-quasi-Newton solver, having a left hand side which is the first-order accurate linearization of the second-order accurate residual operator augmented with a suitable mass matrix. The tangent and adjoint linearizations are computed using the inverse identity method.The expectation is to obtain similar behavior to that seen in the previous section. The difference being, that rather than have quadratic convergence in the primal problem and quadratic convergence of the inexact linearization to the complex linearization, it is expected to see linear convergence between the two linearizations similar to that of the analysis for the inexact Newton solver. To demonstrate this, sister plots to those of the previous section – but for the inexact Newton runs – are shown. Figures (5.25, 5.26) show the behavior for a linear tolerance of $1e-1$ and Figures (5.27, 5.28) show the behavior for a tolerance of $1e-4$. These are the same tolerances portrayed in the previous section, and the same expected convergence of the inexactly linearized tangent to the complex linearization is achieved. As before, and as hypothesized by the error bounds in this work, the maximum iterative difference is again on the order of the linear system tolerance, and as the analysis converges so do the tangent and complex sensitivities to each other, down to nearly machine precision.

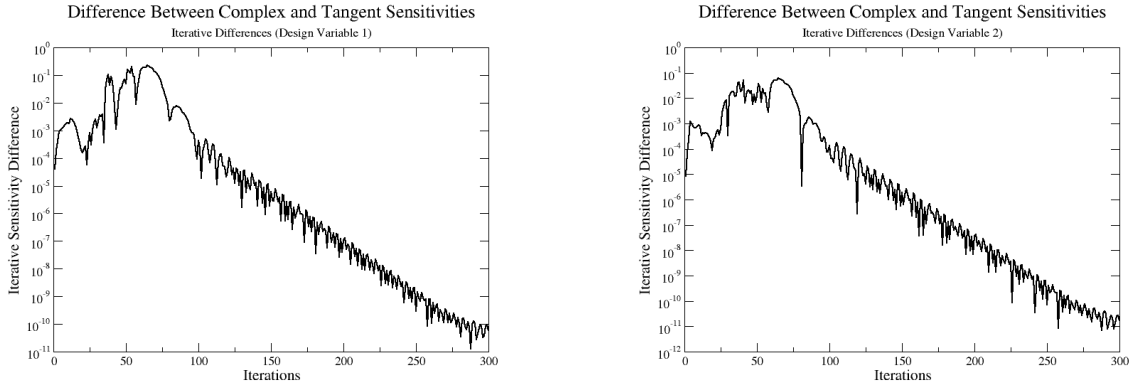
Having seen the impact of the tighter linear system tolerance on the maximum iterative difference between the tangent and complex sensitivities, and seeing the expected behavior as shown in the theoretical error bound and the previous section, the impact of linear tolerances is checked by simulating these tolerances from $1e-1$ to $1e-14$. Figure (5.29) depicts the maximum iterative difference with the expected behavior over that parameter sweep. This confirms the theoretical bound for a more general solver, one in which the left hand side is not an exact linearization of the right hand side.



(a) Design Variable 1

(b) Design Variable 2

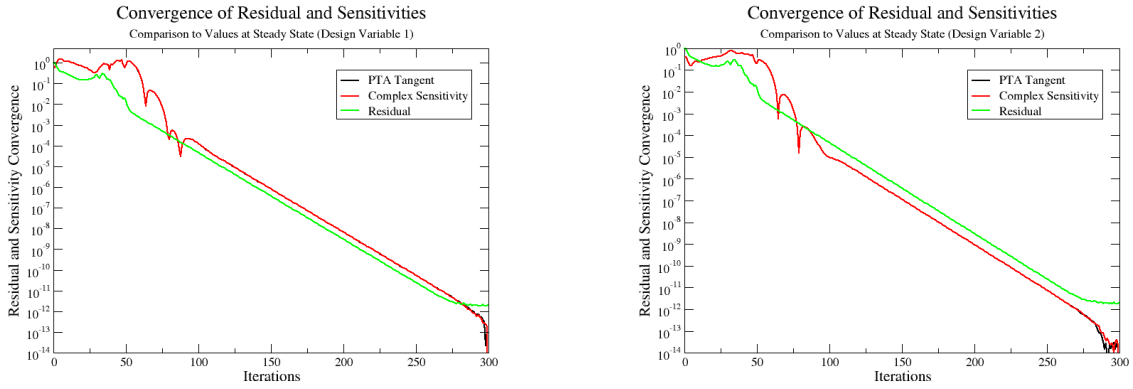
Figure 5.25: Sensitivity convergence for linear tolerance, $1e-1$: difference between current and final sensitivity values



(a) Design Variable 1

(b) Design Variable 2

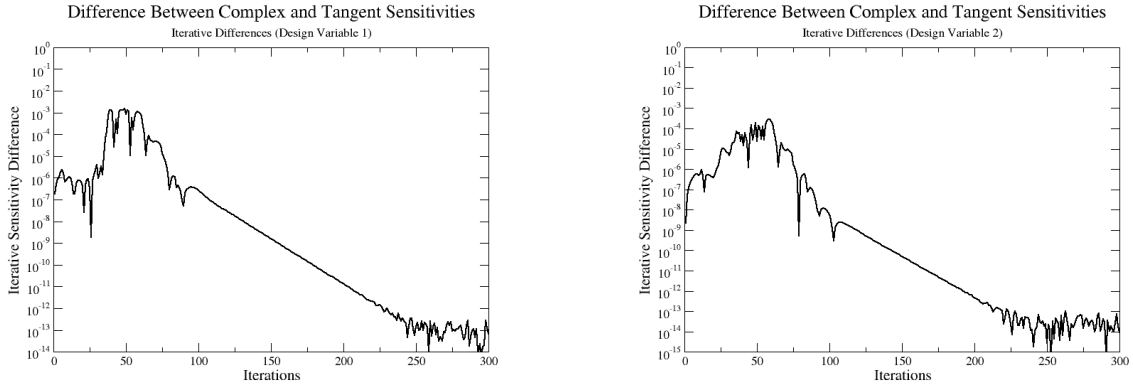
Figure 5.26: Iterative sensitivity difference for linear tolerance, $1e-1$



(a) Design Variable 1

(b) Design Variable 2

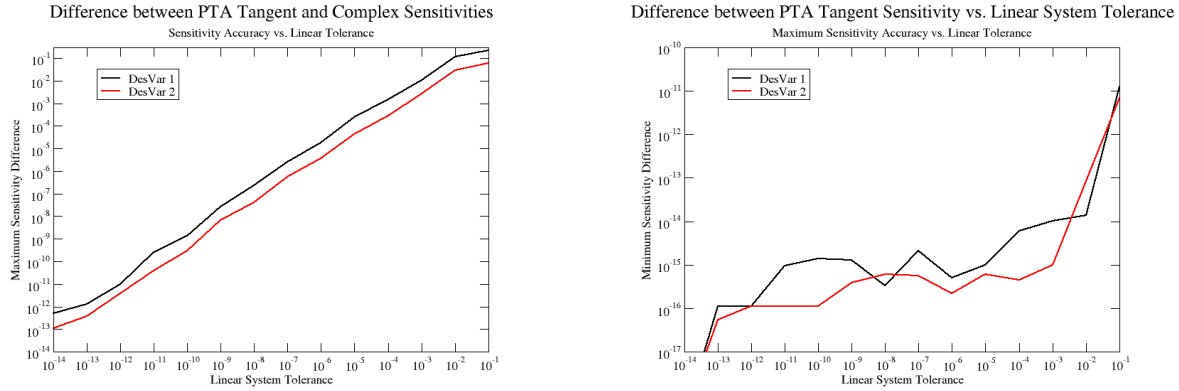
Figure 5.27: Sensitivity convergence for linear tolerance, $1e-4$: difference between current and final sensitivity values



(a) Design Variable 1

(b) Design Variable 2

Figure 5.28: Iterative sensitivity difference for linear tolerance, $1e-4$



(a) Maximum Iterative Difference

(b) Minimum Iterative Difference

Figure 5.29: Iterative difference vs. linear tolerance

5.4 Summary

This chapter discussed the pseudo-time accurate adjoint as a Green's function and showed that the adjoint problem integrated back in pseudo-time converges as the reverse of the primal problem. It was then shown that by pseudo-time averaging the objective function that the sensitivities obtained through the backwards-in-pseudo-time become well behaved as the averaging window is increased. The magnitude and sign differences between the partially and fully integrated sensitivity vectors were studied as well as the angles between the two vectors; for the long time averaging windows that damped out oscillations in the objective function the oscillations in the sensitivities were damped out as well. The ability to get good estimates of the final sensitivity vectors through a partial backwards-in-pseudo-time integration lends itself to similar integration in the design process allowing for much cheaper optimizations. Finally, an investigation was done into the behavior of the pseudo-time accurate tangent formulation for approximate linearization of the fixed point solvers. One explicit and two implicit solvers were studied. Error in the explicit solver was introduced by selective freezing and unfreezing of flow gradients in the analysis and not taking this into account for the tangent, linearization error in the implicit solvers was introduced by using identities assuming that the linear system was exactly solved while not doing so. The implicit solvers showed the expected error behavior and sensitivity convergence, as well as the expected dependence of the error on the linear system tolerance. All solvers showed decreased error as the nonlinear problem was solved as had been theorized and proved earlier in this work.

Chapter 6

Optimization Results

Previous chapters were devoted to development and investigations of the sensitivity computation techniques including quantifying the effect of varying averaging windows and the accuracy of various linearizations. This chapter applies these methods to aerodynamic shape optimization and the goal is to show better final designs in optimizations driven by the PTA adjoint than those driven by a steady-state adjoint. All cases in this chapter use an inexact-quasi-Newton-Krylov solver for the nonlinear problem and calculate the sensitivities with the approximate adjoint linearization by use of the inverse identity adjoint linearization. The optimization problem is solved using SNOPT [1], a limited-memory quasi-Newton solver for optimization, which will drive the Hicks-Henne bump function design variables.

6.1 Optimization of Symmetric Airfoil with Detached Bow Shock

The case presented in this section is a NACA0012 airfoil shown in Figure (4.7) with 20 symmetric Hicks-Henne design variables in $M = 1.25$ flow with $\alpha = 3^o$. The objective function here is an iteration averaged composite objective function of lift, drag, and entropy, where m is the size of the averaging window and $m = 75$.

$$L = \sum_{i=n-m}^{n} \omega_L (c_L - C_{L_T})_i^2 + \omega_D (c_D - C_{D_T})_i^2 + \omega_s (s - s_T)_i^2 \tag{6.1}$$

The targets for lift, drag, and entropy are denoted by C_{L_T}, C_{D_T}, and s_T respectively. The target lift coefficient is set to .211 which is the objective function value in this baseline simulation, the target drag coefficient and entropy are set to 0. The respective weights are set to 1.0 for all terms. This objective function will make the optimizer (SNopt) try to keep the lift constant, minimize the drag, and decrease the shock strength. The limiter used here to prevent divergence is the modified limiter presented in the beginning of this work in algorithm 2. The nonlinear convergence plot in Figure (6.1) shows that the convergence has stalled; for this case, the linear system in the inexact-quasi-Newton-Krylov solver is converged 5 orders of

magnitude at each nonlinear iteration. Figure (6.2) shows the accumulation of the sensitivities by using the

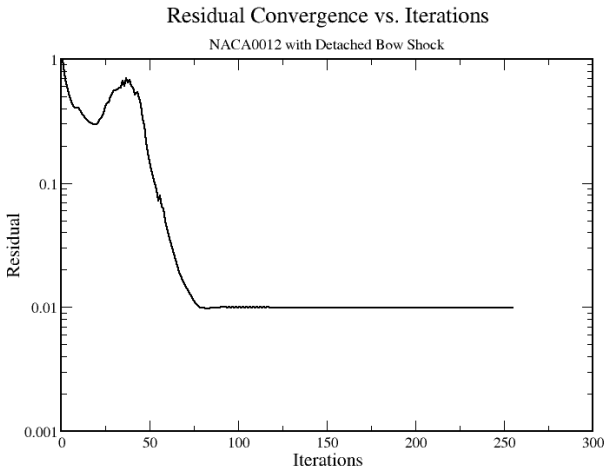


Figure 6.1: Analysis convergence plot for detached bow shock case

inverse identity adjoint formulation through the backwards-in-pseudo-time integration; it is clear that these sensitivities are very well behaved, as they do not change outside of the functional averaging window. This indicates that partially backwards-in-pseudo-time integrations are possible; and that using these sensitivities to drive an optimization is an attractive option. This assumption is used in this work, and the objective function is averaged over the last 75 iterations and the backwards integration is performed only through that averaging window to calculate the sensitivities. These assumptions were justified earlier in this work and examined at great length in section 5.2.

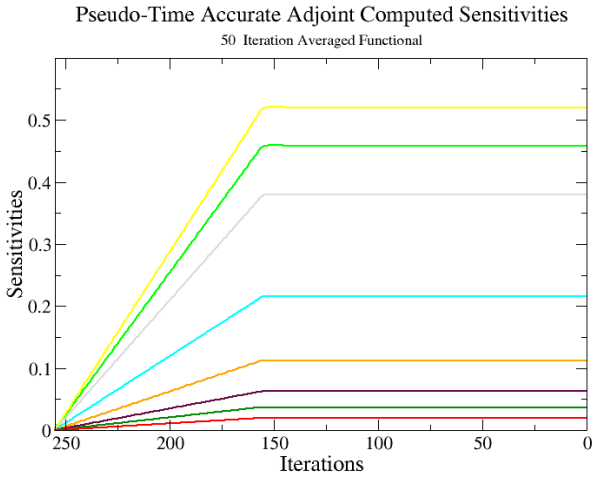
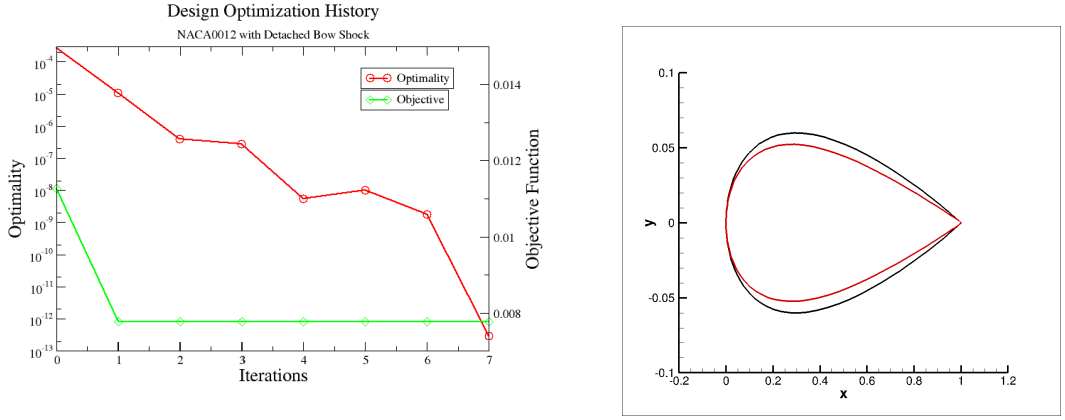


Figure 6.2: Backwards-in-iteration-space integration of sensitivities for detached bow shock case

The summary of the design cycle is in Figure (6.3), which shows a rapid convergence of the objective function and a convergence of the optimality condition to machine precision, this is despite the partial backwards in pseudo-time integration of the sensitivities, which were only integrated back through the function averaging window. This figure also shows that the optimized airfoil (in red) has shrunk at every coordinate along the chord when compared to the baseline airfoil (in black). This accords with the physical intuition for such a case. In this simulation (based off the Euler equations) the best way to decrease drag and entropy is to weaken the shock strength by lowering the airfoil thickness.



(a) Design Cycle Convergence

(b) Airfoil Comparison

Figure 6.3: Design cycle summary for detached bow shock

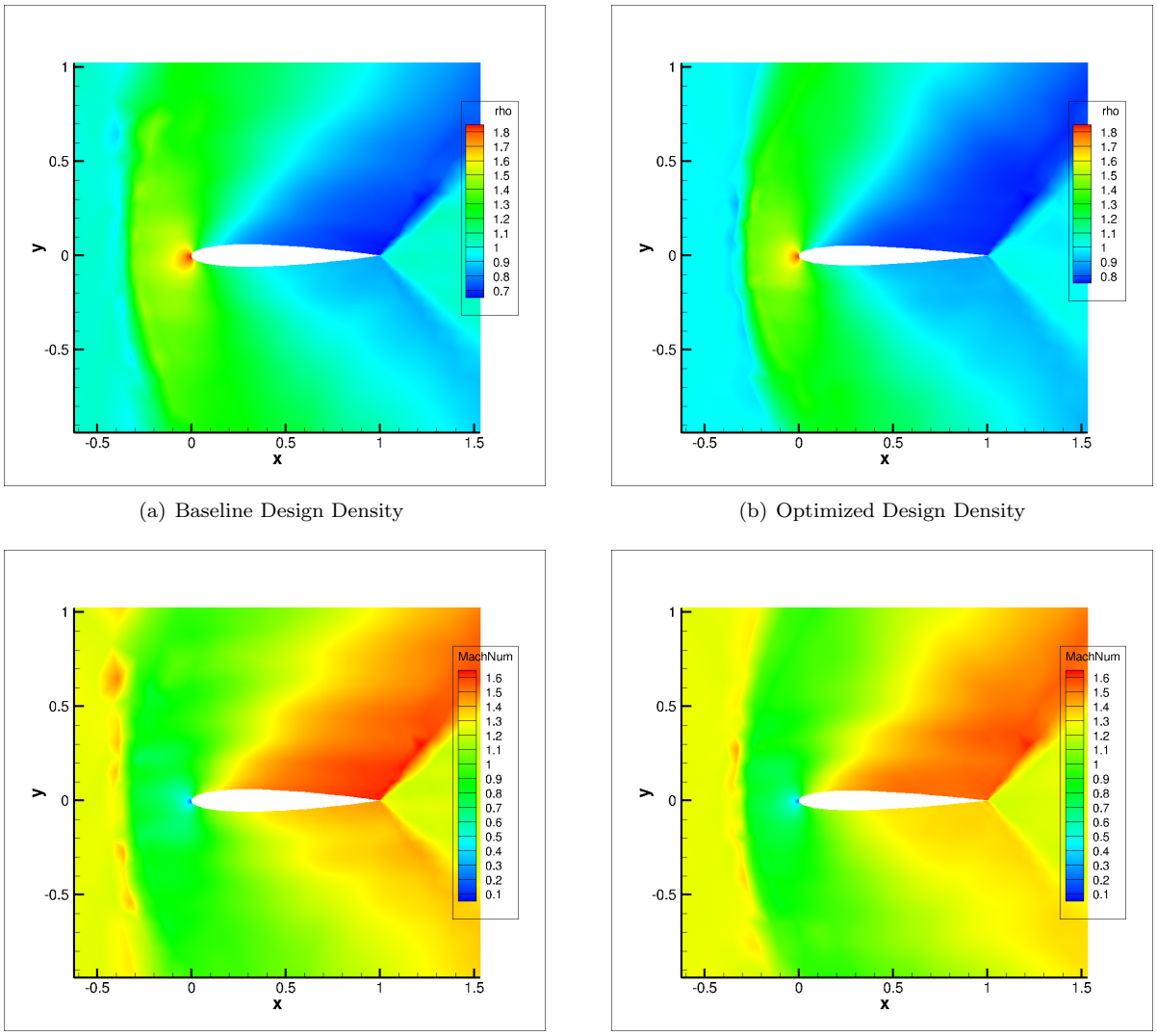
Figure (6.4) shows the density and Mach number fields; it is apparent that there is a lower flow acceleration on the back half of the suction side of the airfoil, this lowers the pressure differential and decreases the drag.

It is illustrative to compare this adjoint based optimization to one that uses complex-step sensitivities to drive the optimization. Since this formulation is based off computing the sensitivity of the process – and the complex-step sensitivities are these sensitivities – it is enlightening to compare the results of the optimizations using both the complex-step sensitivities (considered to be exact) and the adjoint sensitivities. This provides an estimate of how much a researcher is penalized by the inaccuracy in the sensitivities by using the approximation of the derivative of the linear system solve. The bounds refer to the minimum and maximum values of the amplitudes of the Hicks-Henne bump functions in these optimizations; the wide bounds are set to $-1e-2$ and $1e-2$ respectively, the narrow bounds are set to $-1e-3$ and $1e-3$ respectively.

Bounds	Complex Optimality	Adjoint Optimality	Complex Objective	Adjoint Objective
Narrow Bounds	2.9E-13	2.9E-13	7.7685583E-03	7.7685583E-03
Wide Bounds	2.6E-03	5.4E-03	2.**9609739E-03**	2.**7440594E-03**

Table 6.1: Comparison of adjoint and complex-step optimizations for detached bow shock case

Table (6.1) shows very close correspondence between the complex-step and adjoint optimizations; the



(a) Baseline Design Density

(b) Optimized Design Density

(c) Baseline Design Mach Number

(d) Optimized Design Mach Number

Figure 6.4: Flow field comparison for baseline and optimized detached bow shock case

difference between them shows the adjoint computed sensitivities returning a better final design. The natural concern with the narrow bound optimization is that the optimality is only machine zero because all the design variables are at the bounds; this is not the case. While the first 19 design variables are at the bounds, the 20th design variable (the aft-most one) is not at the bound but has a machine zero gradient; so the optimality condition is satisfied. This case was also run using a steady state adjoint to provide the sensitivities, and while the final designs were very similar, the optimizer stagnated for a long period of time before terminating due to numerical difficulties. This can be attributed to the noise in the sensitivity vector that comes from the steady state adjoint, even for this case which shows minimal unsteadiness in the solution output. One caveat to these cases is that all cases were run with a linear tolerance of $1e-5$ which is a far more restrictive tolerance than what is typically required, especially for inexact Newton solvers.

6.1.1 Investigation of Linear Tolerance on Design Optimization

The next step is examining the results for an optimization as a function of the linear tolerance and comparing these results to those of an optimization that uses the complex-step finite difference computed sensitivities.

Tolerance	Bounds	Complex Optimality	Adjoint Optimality	Complex Objective	Adjoint Objective
1e-2	Narrow	1.7E-12	2.9E-13	7.7685**83E-03**	7.7685**33E-03**
1e-3	Narrow	2.9E-12	3.0E-13	7.7685583E-03	7.7685583E-03
1e-4	Narrow	2.9E-13	7.8E-13	7.7685583E-03	7.7685533E-03

Table 6.2: Comparison of adjoint and complex-step optimizations for detached bow shock

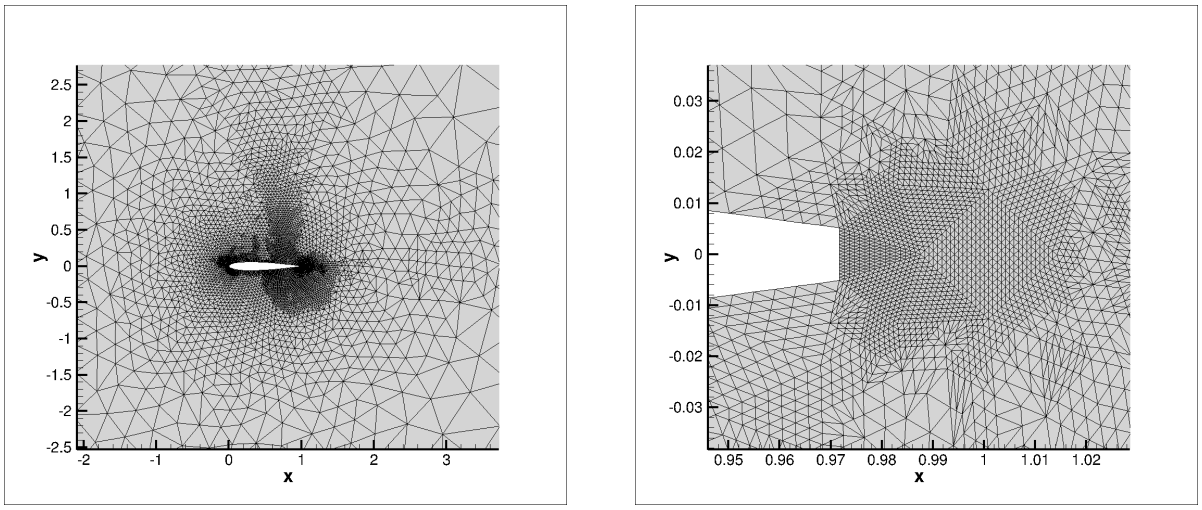
Table (6.2) shows that the linear tolerance has little effect on the adjoint sensitivities, as the adjoint sensitivity based optimizations were very similar to one another regardless of linear tolerance. This is an encouraging result, that allows further investigation into additional optimization cases with confidence. The overall behavior indicates that using the pseudo-time accurate adjoint-computed sensitivities is effective even with a loose linear tolerance, and that use of a partial backwards in time integration to calculate the sensitivities that drive the optimizer is feasible. The following results will show such optimizations.

6.2 Optimization of Symmetric Airfoil with Trailing Edge Unsteadiness

This is a case with trailing edge unsteadiness in transonic flow. This case is unsteady because the geometry is a NACA0012 with a truncated (at 97% of the chord) blunt trailing edge shown in Figure (6.5) which causes small scale shedding at the trailing edge. This mesh went through 4 refinement cycles (using the mesh refinement module developed for this work) to get sufficient fineness near the trailing edge and shock location without expending too much computational expense elsewhere. The case presented in this section is a NACA0012 airfoil with symmetric design variables in $M = 0.8$ flow with $\alpha = 3^o$. The objective function here is a composite objective function of lift and drag averaged over the last 200 iterations:

$$L = \sum_{i=n-m}^{n} \omega_L (c_L - C_{L_T})_i^2 + \omega_D (c_D - C_{D_T})_i^2 \tag{6.2}$$

where the targets for lift and drag are denoted by C_{L_T}, and C_{D_T} respectively. The target lift coefficient is set to .6 which is the objective function value in this baseline simulation and the target drag coefficient is set to 0. The respective weights are 2.0 for ω_L and 1.0 for ω_D. This objective function will make the optimizer try to keep the lift constant while minimizing the drag. The design variables are 20 symmetric Hick-Henne bump functions with the lower and upper bounds being $-1e-3$ and $1e-3$ respectively. The limiter used here to prevent divergence is the modified VK limiter presented in algorithm (2). Figure (6.6)

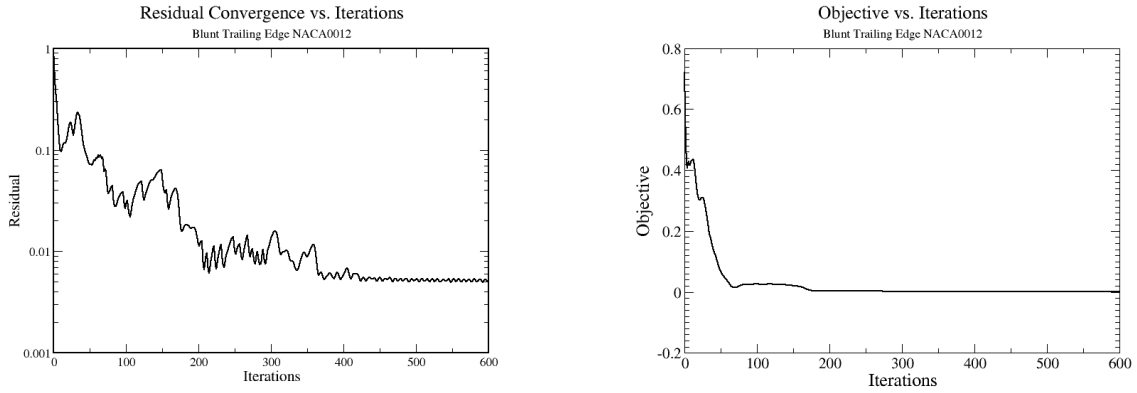


(a) Airfoil Mesh

(b) Airfoil Trailing Edge

Figure 6.5: Mesh for NACA0012 truncated at 97% of the chord

shows the nonlinear convergence and that the convergence has stalled and has small scale oscillations with negligible oscillations in the objective. The linear system from the Newton-Krylov solver is converged 4 orders of magnitude at each nonlinear iteration.



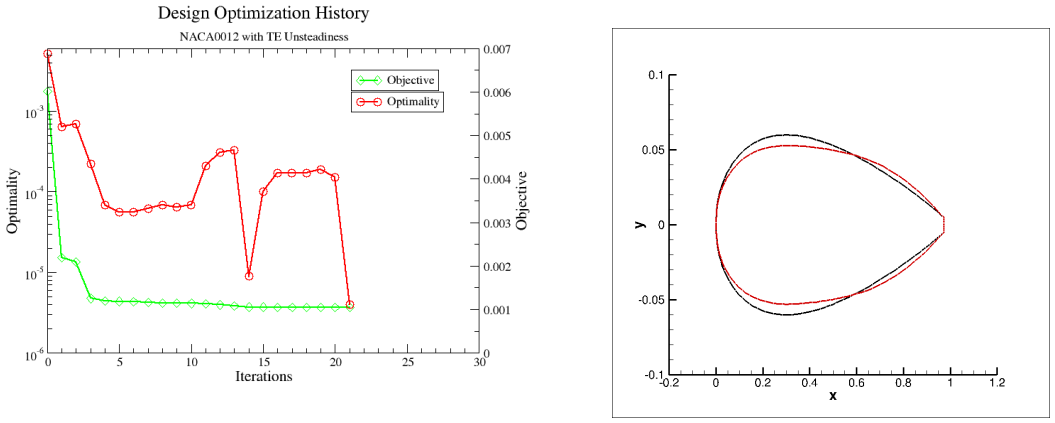
(a) Convergence History

(b) Objective History

Figure 6.6: Analysis behavior for NACA0012 truncated at 97% of the chord in transonic flow

Looking at the design cycle summary, it is clear that the design has achieved a large decrease in the functional even though the optimality condition – which measures the duality gap between a given iterate and the approximation of the dual problem– is not satisfied to machine precision. Figure (6.7) shows that the objective function is decreased to $\frac{1}{6}$ of the baseline value and the airfoil shows interesting behavior. The front 60% of the airfoil gets thinner, but the rear 37% gets thicker. It is hypothesized that this helps control the drag and shock behavior as the increase in thickness in the rear of the airfoil is similar to the behavior

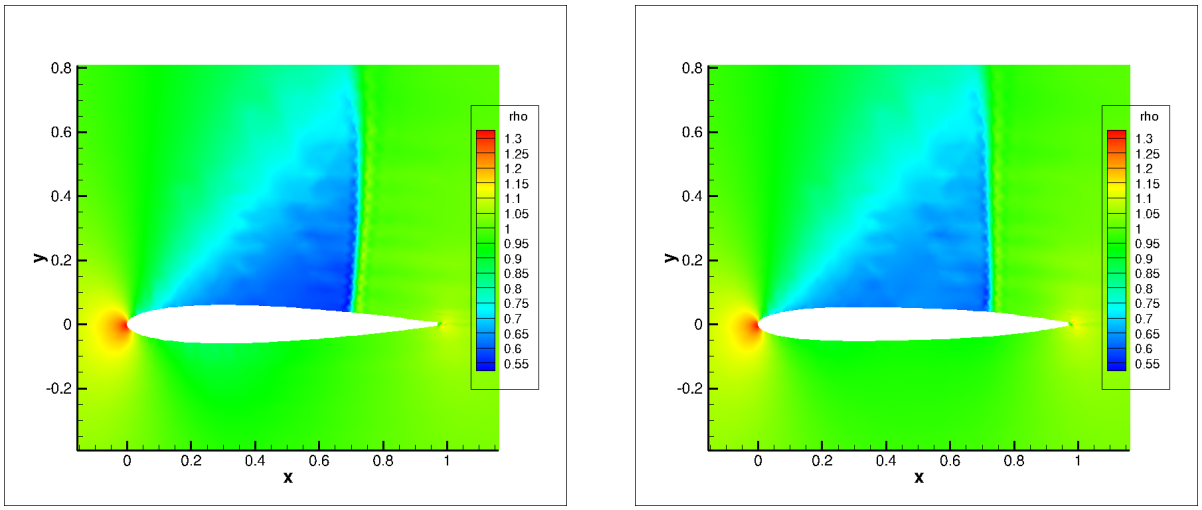
in the Aerodynamic Design Optimization Discussion Group (ADODG) NACA0012 case [62].



(a) Design Cycle Convergence (b) Airfoil Comparison

Figure 6.7: Design cycle summary for NACA0012 truncated at 97% of the chord in transonic flow

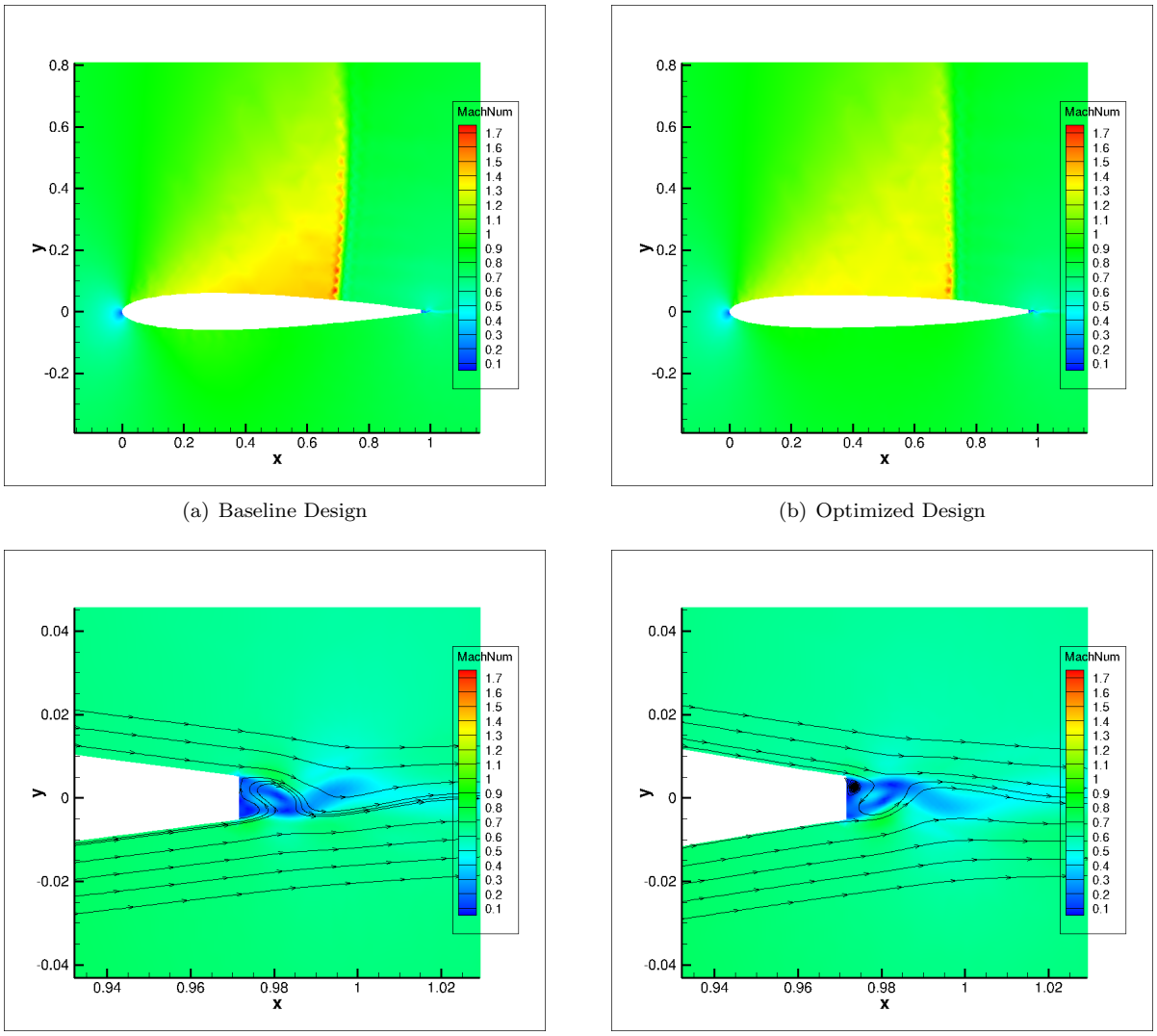
Figure (6.8) shows the density field plots at the final nonlinear iteration and compares the baseline to the optimized. It is clear the shock has gotten much weaker and moved further back along the airfoil.



(a) Baseline Design (b) Optimized Design

Figure 6.8: Density field comparison for NACA0012 truncated at 97% of the chord in transonic flow

The change in shock strength and location becomes clearer when looking at the Mach field in Figure (6.9). Although the streamlines and shedding appear to be stronger in the optimized case than the baseline case, the optimization is based off the last 200 nonlinear iterations, and over that window, the objective function average is far decreased, as in Figure (6.7).



(a) Baseline Design

(b) Optimized Design

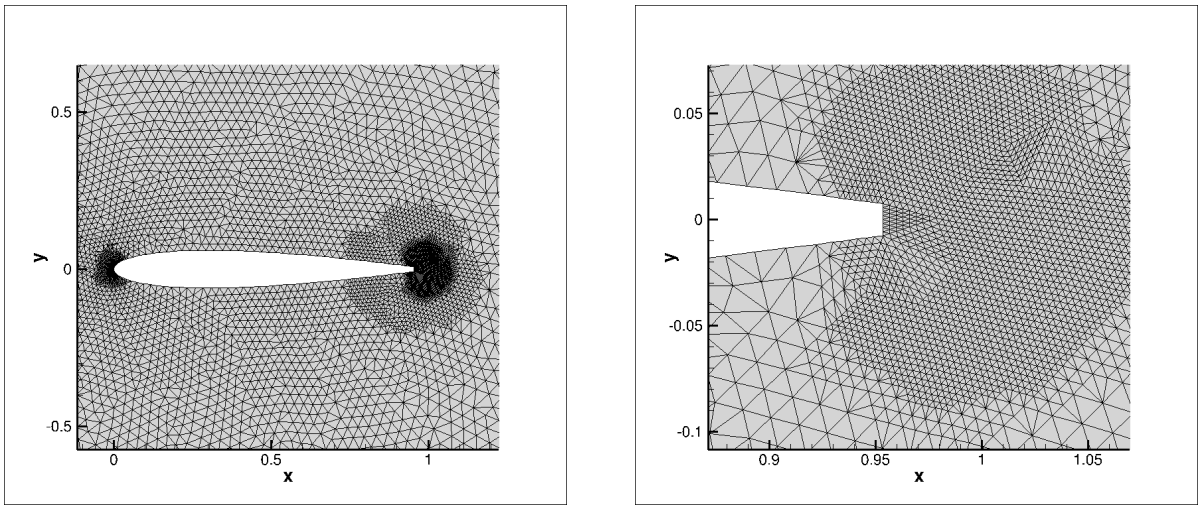
(c) Baseline Trailing Edge with Streamlines

(d) Optimized Trailing Edge with Streamlines

Figure 6.9: Final state Mach field comparison for NACA0012 truncated at 97% of the chord in transonic flow

6.3 Optimization of ADODG NACA0012 Airfoil with Trailing Edge Unsteadiness

This case has a similar set up to the previous ADODG NACA0012 airfoil [62] again with a trailing edge unsteadiness in transonic flow. This case begins with a NACA0012 (as in the previous case) with a truncated (at 95% of the chord rather than 97% as in the previous case) blunt trailing edge shown in Figure (6.10). This mesh went through 3 refinement cycles (using the same mesh refinement module) to get sufficient fineness near the trailing edge while minimizing overall computational expense. This airfoil is optimized with symmetric design variables in $M = 0.85$ flow with $\alpha = 0^o$ flow.



(a) Airfoil Mesh

(b) Airfoil Trailing Edge

Figure 6.10: Mesh for NACA0012 truncated at 95% of the chord

The objective function here is drag averaged over the last 200 iterations:

$$L = \sum_{i=n-m}^{n} c_{D_i} \tag{6.3}$$

This objective function will make the optimizer try to minimize the drag. On a symmetric structured mesh, lift would be equal to zero, in an unstructured case this is only the case in the limit of an infinitely fine mesh. The decision was made to not optimize with a target lift of 0 because that would ask the optimizer to optimize based off the error. The design variables are 20 symmetric Hick-Henne bump functions with the lower and upper bounds being $-1e-3$ and $1e-3$ respectively. The limiter used here to prevent divergence is again the modified VK limiter presented in algorithm (2). Figure (6.11) shows that convergence of the nonlinear problem has ceased as the analysis is stalled in limit cycle oscillations. The linear system from the Newton-Krylov solver is, as in previous cases, converged 4 orders of magnitude at each nonlinear iteration.

Figure (6.12) shows the design cycle summary and a large decrease in the functional even though the optimality condition is still not satisfied to machine precision. The objective function is decreased to a value of $\frac{1}{10}$ of the baseline and the airfoil shows interesting behavior. The front 60% of the airfoil gets thinner, but the rear 35% gets thicker, similar to the previous trailing edge unsteadiness case. It is hypothesized, as before, that this helps control the drag and shock behavior.

Figure (6.13) shows the density field at the final nonlinear iteration and compares the baseline geometry to the optimized one, which shows that the shock has again gotten much weaker and moved further back along the airfoil as in the previous case.

The change in shock strength and location becomes clearer in the Mach field shown in Figure (6.14). There is little difference in the streamlines and shedding between the optimized case and the baseline case, and the shock – which is the primary driver of the drag – has nearly disappeared.

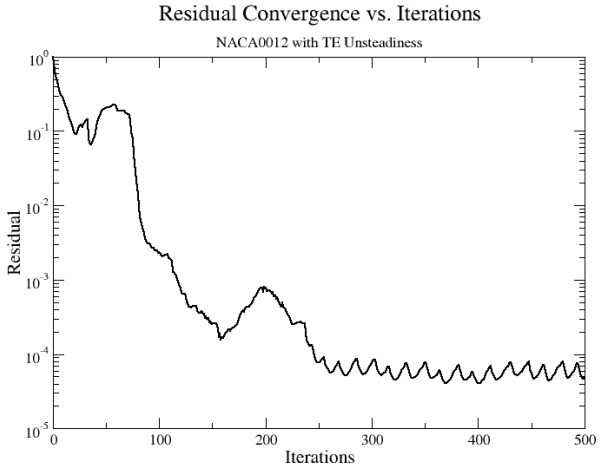
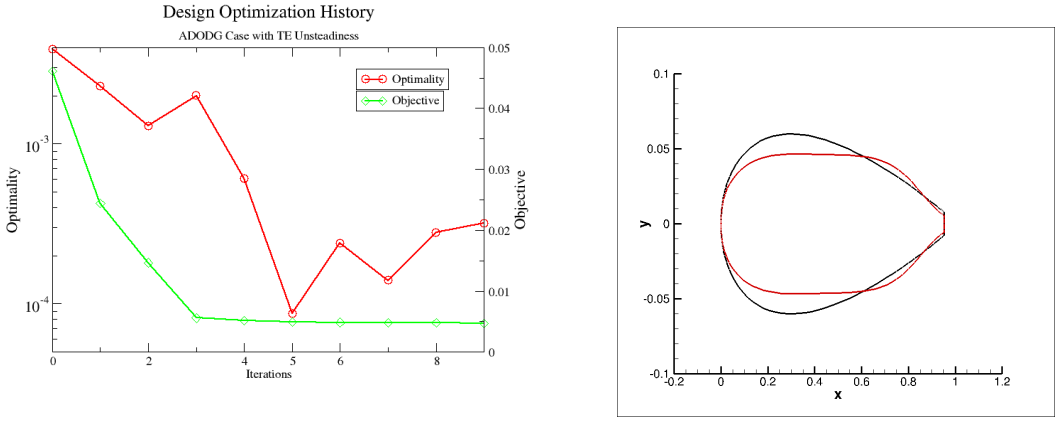


Figure 6.11: Analysis convergence plot for NACA0012 truncated at 95% of the chord in transonic flow



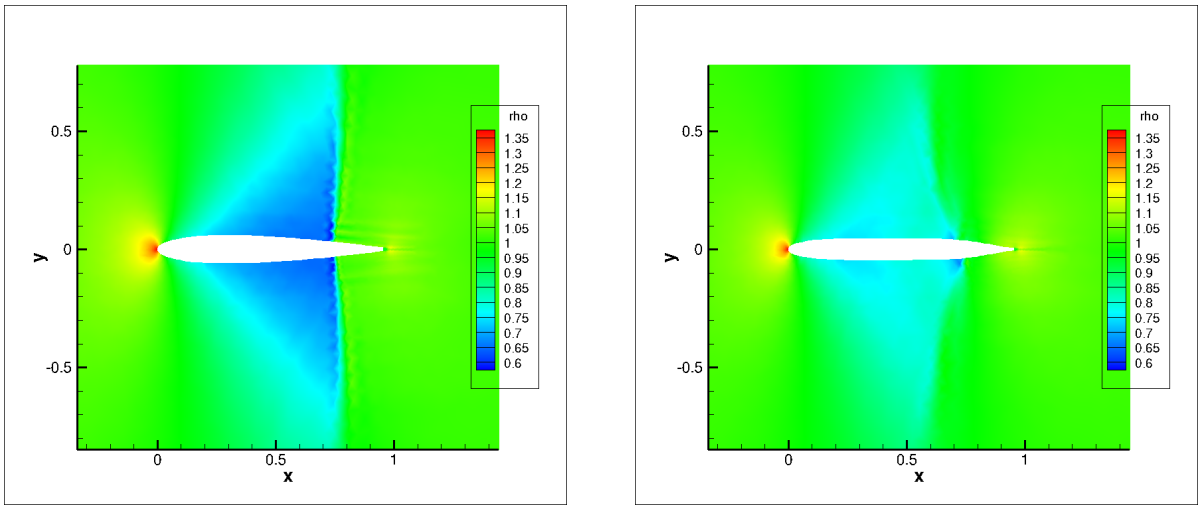
(a) Design Cycle Convergence

(b) Airfoil Comparison

Figure 6.12: Design cycle summary for NACA0012 truncated at 95% of the chord in transonic flow

This is clearly an encouraging result, so it is then desirable to compare this to design optimization when using a steady state adjoint to drive the design both using a final state objective function, and the objective function averaged over the same window as was used above.

Figure 6.15 shows the comparison between the optimizations using the different adjoint-based sensitivities. The steady state adjoint linearized about the final state has a final objective function that is approximately twice as high as that of the pseudo-time accurate adjoint or the steady state adjoint with an averaged objective function. Both steady state adjoint methods have trouble decreasing the optimality value, and the pseudo-time accurate adjoint has a final optimality value that is smoother and approximately half an order of magnitude lower than that provided by the design processes driven by the steady state adjoint-computed gradients. It is also clear that the pseudo-time accurate adjoint converges quicker through

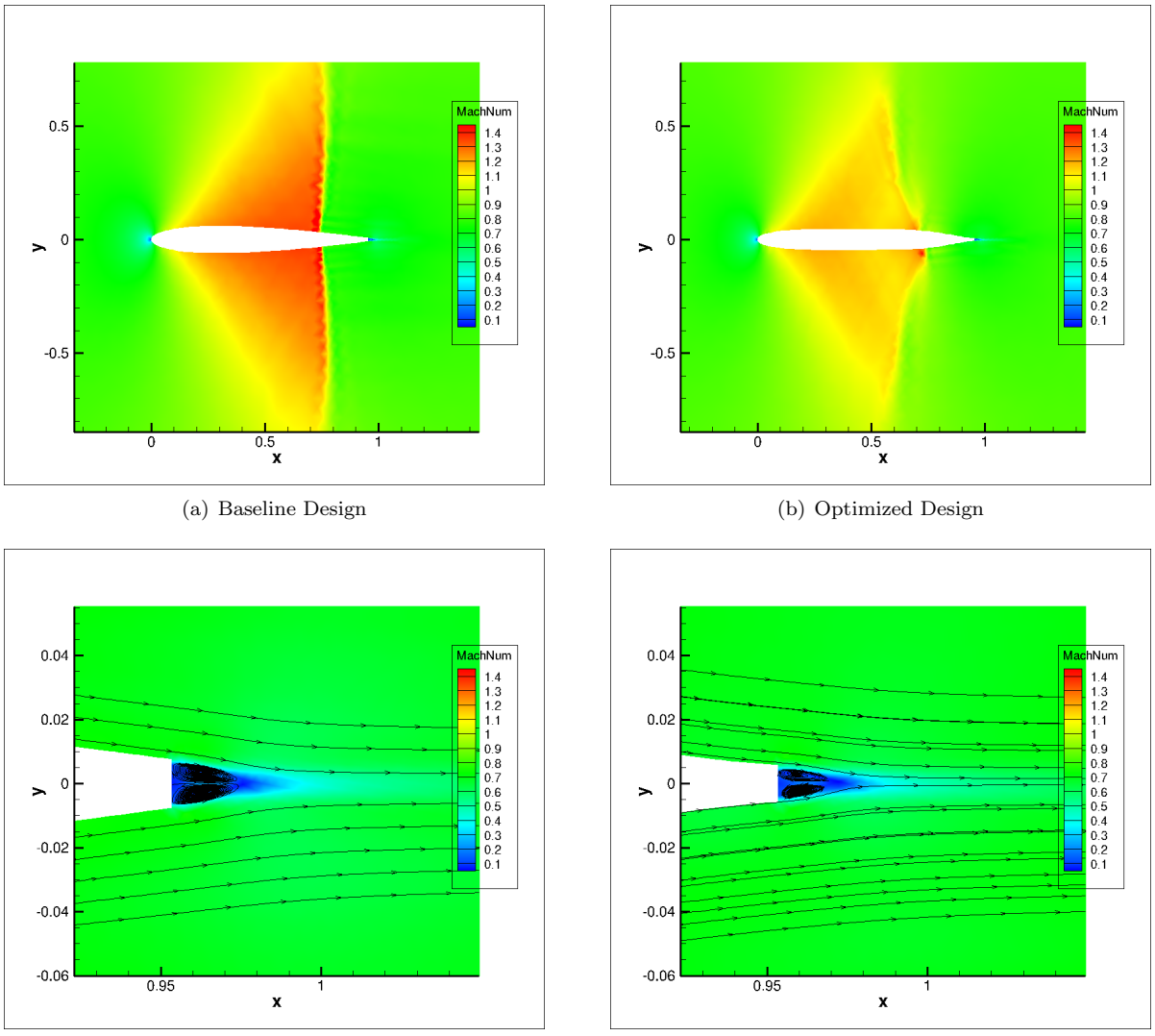


(a) Baseline Design

(b) Optimized Design

Figure 6.13: Density field comparison for NACA0012 truncated at 95% of the chord in transonic flow

the optimization and obtains a lower final value of the objective function. Finally, by inspecting Figure 6.16, it becomes clear that the final airfoil shapes are quite different, with the difference between the windowed steady state adjoint case and the pseudo-time accurate adjoint being of particular interest as compared to the steady state adjoint linearized about the final state. It is clear that the pseudo-time accurate adjoint finds a thinner optimal design than the windowed steady state adjoint does with generally the same features; while the steady state adjoint linearized about the final state shows a very different geometry from both of them.



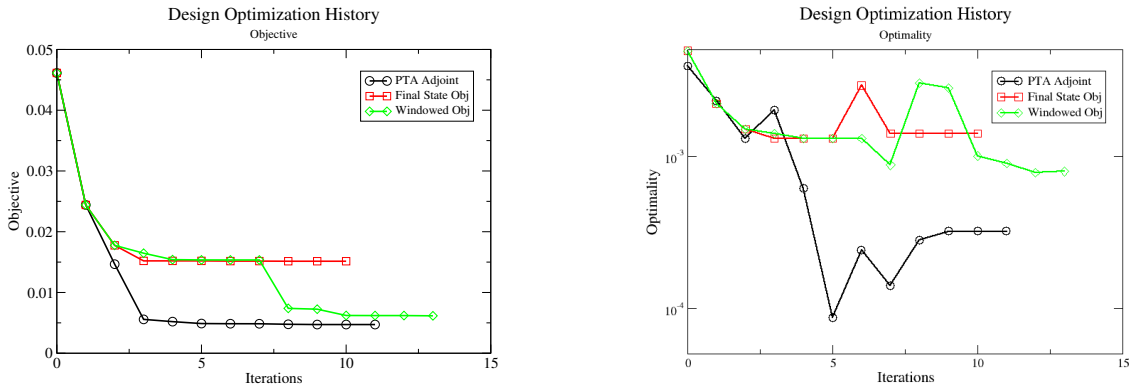
(a) Baseline Design

(b) Optimized Design

(c) Baseline Trailing Edge with Streamlines

(d) Optimized Trailing Edge with Streamlines

Figure 6.14: Mach field comparison for NACA0012 truncated at 95% of the chord in transonic flow



(a) Objective Function Convergence

(b) Optimality Convergence

Figure 6.15: Design cycle summary for NACA0012 truncated at 95% of the chord in transonic flow with steady state adjoint results

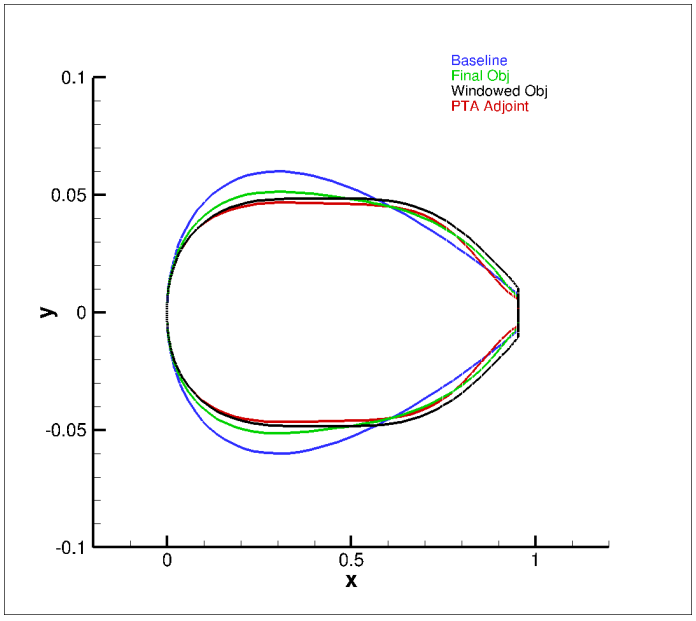


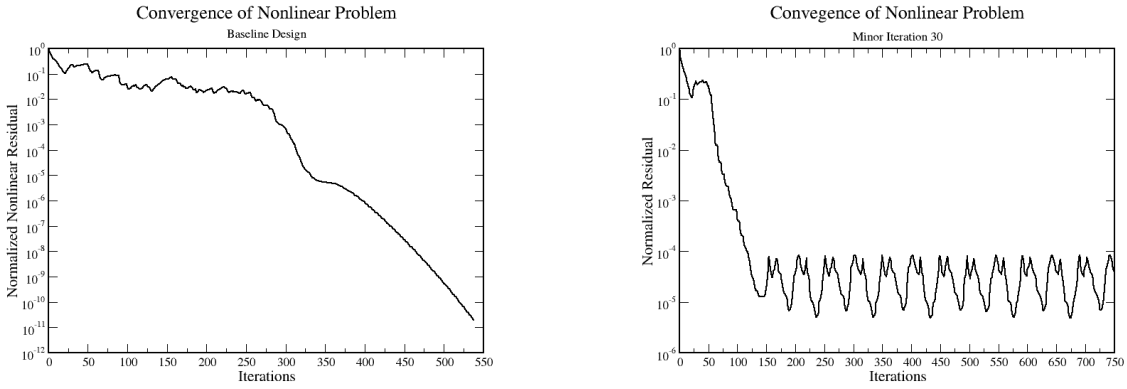
Figure 6.16: Baseline and optimized airfoils for optimization of NACA0012 airfoil with blunt trailing edge

6.4 Optimization of Truncated NACA0012 Airfoil in High Angle of Attack Flow

This case has a similar geometry to the previous case but with asymmetric design variables in $M = 0.7$ flow with $\alpha = 6^o$. The assumption is that due to the angled flow strong asymmetries in the final design would develop. The objective function here is a composite objective function with lift and drag targets averaged over the last 200 iterations:

$$L = \sum_{i=n-m}^{n} \omega_L (c_L - C_{L_T})_i^2 + \omega_D (c_D - C_{D_T})_i^2 \tag{6.4}$$

where the targets for lift and drag are denoted by C_{L_T}, and C_{D_T} respectively as previously. The design variables are 30 Hick-Henne bump functions equally spaced along the upper and lower surfaces with the lower and upper bounds being $-2.5e-3$ and $2.5e-3$ respectively.This case has some very interesting convergence behavior that is worth noting before proceeding to look at the optimization results. Figure 6.17 shows that while the baseline geometry has good convergence properties, once the baseline is perturbed the unsteadiness forms at the trailing edge of the geometry and this hampers the optimization process for the steady state adjoint.



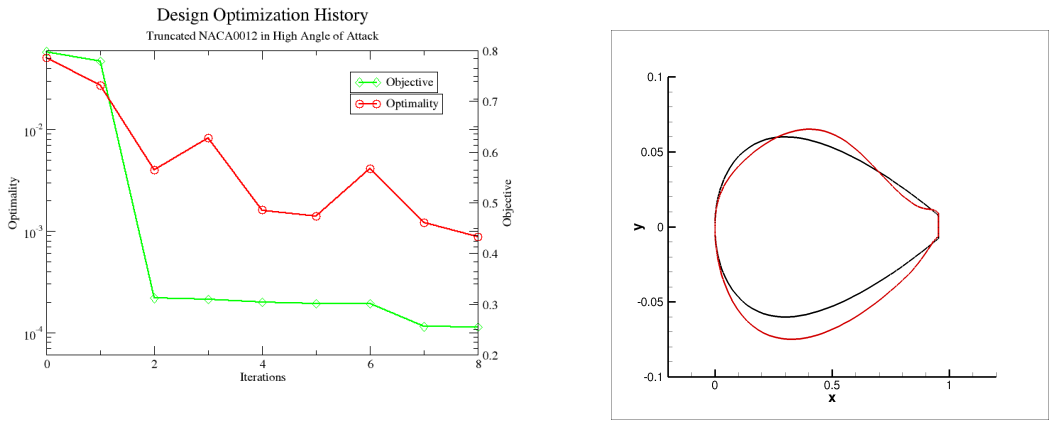
(a) Baseline Nonlinear Convergence

(b) Second Design Cycle Nonlinear Convergence

Figure 6.17: Analysis convergence plot for NACA0012 truncated at 95% of the chord in transonic flow with high angle of attack

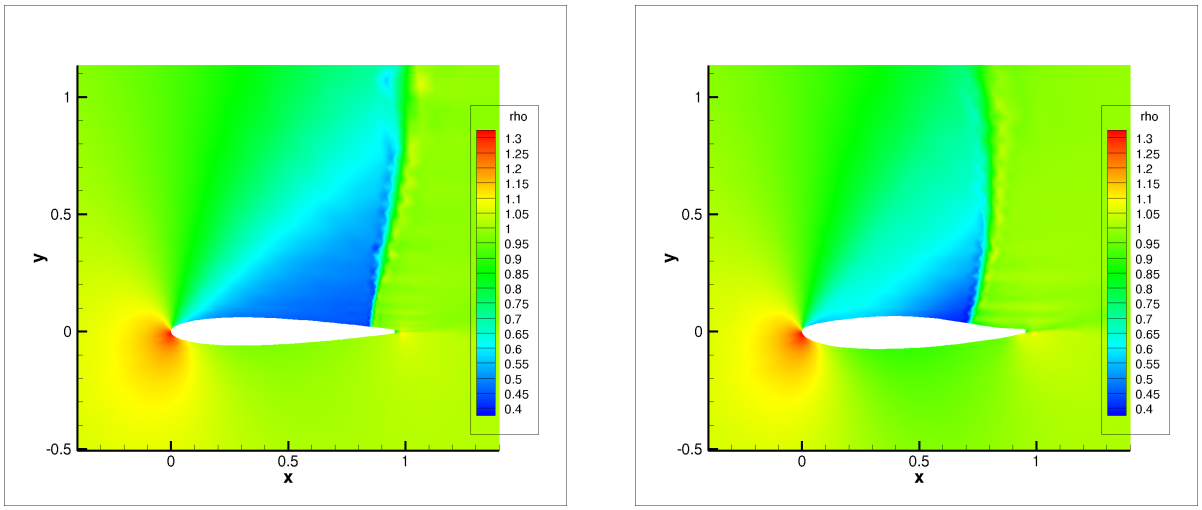
Figure (6.18) shows the design cycle summary with a large decrease in the functional even though the optimality condition is still not satisfied to machine precision. The objective function gets decreased to a value of $\frac{1}{3}$ its baseline value and the airfoil shows interesting behavior. The bottom of the optimized airfoil (in red) is made thicker than the baseline (in black) while the top is made thicker in the mid section and thinner both fore and aft. It is hypothesized, as before, that this helps control the drag and shock behavior. Figure (6.19) shows the density field at the final nonlinear iteration and compares the baseline shape to the

optimized one. It is clear that the shock has gotten much weaker and moved further back along the airfoil as in the previous cases, and the airfoil has gotten highly asymmetric in order to deal with the flow conditions of this case. The change in shock strength and location becomes clearer in the Mach field in Figure (6.20). There is little difference in the streamlines and shedding between the optimized case and the baseline case, and the shock – which is the primary driver of the drag – has nearly disappeared.



(a) Design Cycle Convergence (b) Airfoil Comparison

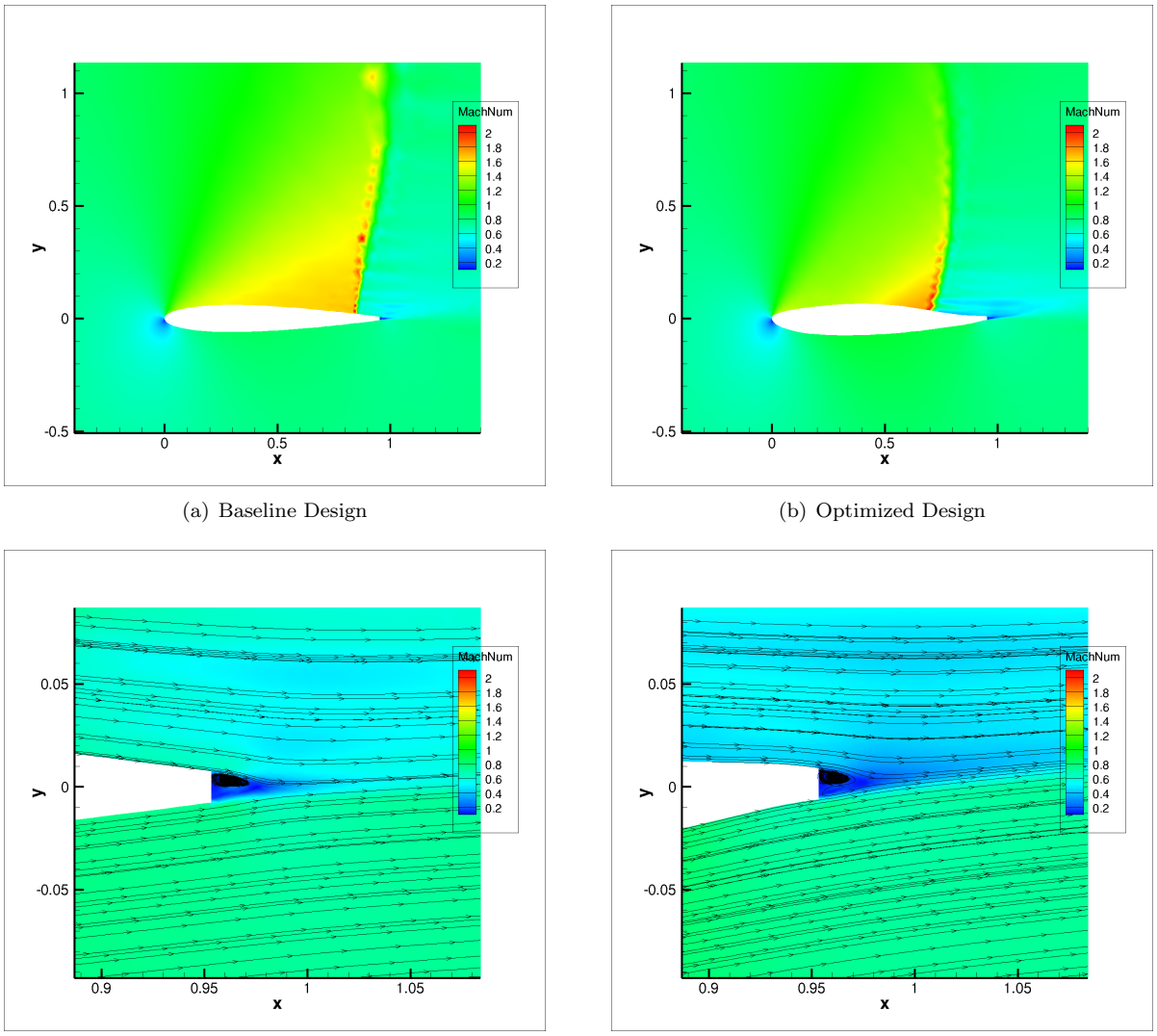
Figure 6.18: Design cycle summary for NACA0012 truncated at 95% of the chord in transonic flow with high angle of attack



(a) Baseline Design (b) Optimized Design

Figure 6.19: Density field comparison for NACA0012 truncated at 95% of the chord in transonic flow with high angle of attack

As before this is compared to design optimization when using steady state adjoint-computed sensitivities to drive the design both using a final state objective function, and the objective function averaged over the same window as was used to drive the pseudo-time accurate adjoint. Figure 6.21 shows the com-



(a) Baseline Design

(b) Optimized Design

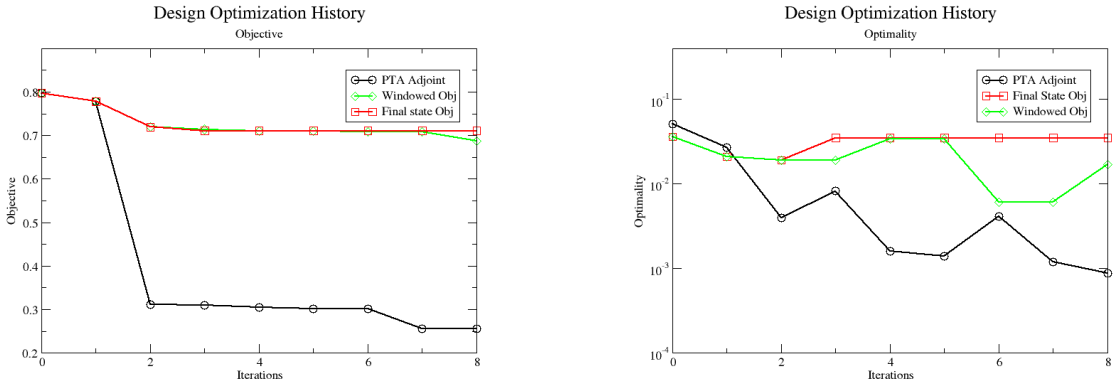
(c) Baseline Trailing Edge with Streamlines

(d) Optimized Trailing Edge with Streamlines

Figure 6.20: Mach field comparison for NACA0012 truncated at 95% of the chord in transonic flow with high angle of attack

parison between the optimizations based off the different adjoint-based sensitivities. Both optimizations using sensitivities computed from the steady state adjoint do not succeed in meaningfully optimizing the airfoil, with the optimization driven by the steady state adjoint using the averaged objective function doing only negligibly better. Both steady adjoint methods have trouble decreasing the optimality value, and the pseudo-time accurate adjoint has a final optimality value that is smoother and approximately one and a half orders of magnitude lower than that provided by the optimizations driven by the sensitivities computed by the steady state adjoints. It is also clear that the pseudo-time accurate adjoint converges quicker through the optimization and obtains a notably lower final value of the objective function. Finally, by inspecting Figure 6.22, it becomes clear that the steady state adjoint shapes are nearly identical to the baseline with

the pseudo-time accurate adjoint finding a thicker final design than the baseline.



(a) Objective Function Convergence

(b) Optimality Convergence

Figure 6.21: Design cycle summary for NACA0012 truncated at 95% of the chord in transonic flow with high angle attack: comparison to steady state adjoint results

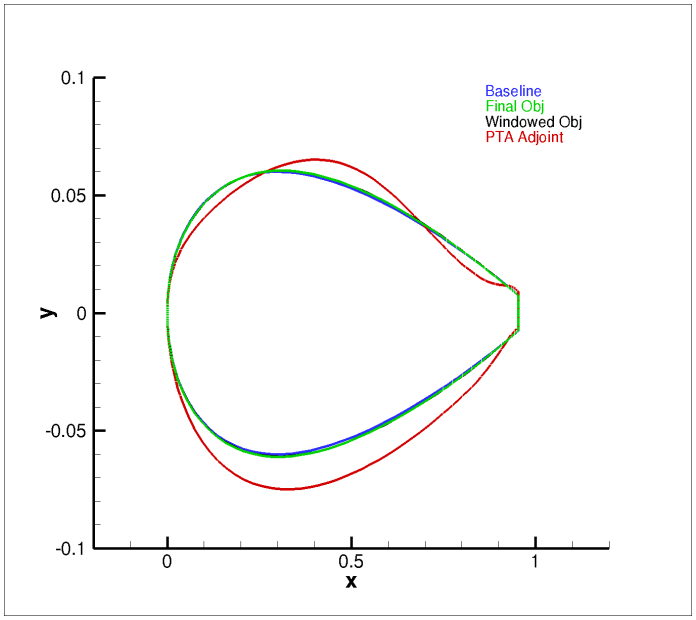


Figure 6.22: Baseline and optimized airfoils for high angle of attack optimization

6.5 Summary

This chapter presented design optimizations driven by the pseudo-time accurate adjoint-computed sensitivities and a study of the effect of different linear tolerances and a comparison to design cycles driven by the steady state adjoint-computed sensitivities. The first case presents the optimization of a NACA0012 airfoil

in supersonic flow with a detached bow shock. First, it is shown that for a case with minimal unsteadiness that this case can satisfy the optimality condition of the optimizer. The second case shows a NACA0012 airfoil in transonic flow with a blunt trailing edge that was created by truncating the airfoil at 97% of the chord. This case shows some small scale unsteadiness, and the optimization driven by the pseudo-time accurate adjoint-computed sensitivities succeeds in meaningfully decreasing the objective function. The final two cases use a similar geometry but for a NACA0012 airfoil truncated at 95% of the chord. The first of the two cases is in transonic flow with no angle of incidence, similar to the ADODG benchmark case [62]. This shows significant success for the optimization driven by the pseudo-time accurate adjoint-computed sensitivities which succeeds in meaningfully decreasing the objective function. When compared to the designs based off the steady state adjoint-computed sensitivities it shows meaningfully improved behavior and achieves a noticeably geometrically different design. The final case is the same geometry in slower transonic flow with a high angle of attack, and while the baseline nonlinear problem converges to machine zero, the nonlinear problems on the perturbed airfoils do not and instead enter limit cycle oscillations. Both optimizations driven by the steady state adjoint fail to noticeably decrease the objective function, while the one driven by the pseudo-time accurate adjoint-computed sensitivities obtains a significant reduction in the objective function. This chapter shows that the pseudo-time accurate adjoint can outperform the steady state adjoint in design optimization contexts, especially in the presence of small scale unsteadiness, which is the type of case this method was designed for. Improvements could be made to the optimization capabilities through employing the windowing regularization techniques that have worked so well in design in Unsteady Reynold Averaged Navier-Stokes (URANS) flows [63]. This could allow for quicker convergence of the objective function and better behavior of the sensitivities for a smaller averaging window as the sensitivity of the averaged objective function is not guaranteed to converge with a longer averaging window, unlike the windowing techniques applied in such cases.

Chapter 7

Pseudo-time Accurate Approaches to Error Estimation and Adaptive Mesh Refinement

This chapter seeks to adapt the pseudo-time accurate adjoint approach from optimization to the field of error estimation. To that end, it begins with an explanation of the dual-weighted residual method used in steady-state problems and three commonly used algorithms in that context and then shows the derivation of their analogues in the PTA adjoint context. These derivations are shown only for the inexact-quasi-Newton nonlinear solver using the inverse identity approximation for the adjoint linearization with an analogue for each of the three algorithms shown in the steady state context. Each nonlinear solver would have its own analogue for each of the three algorithms presented in the steady state section. Finally this section contains results for these methods for two different flow conditions demonstrating small-scale and large-scale unsteadiness respectively.

7.1 A Review of the Dual-Weighted Residual

A straightforward approach to error estimation would be to get an estimate of the fine mesh solution (u_h) using the coarse mesh solution (u_H), and calculate error based of the error in the solution. However, for engineering purposes, the practitioner's interest lies not in the actual values of the solution but instead in the values of the outputs of engineering interest (lift, drag, moment, etc.). To this purpose the dual weighted residual method was developed as it provides estimates of error in the output of interest and allows for automatic error control of simulations [6].

The method begins by approximating the output of interest (L_h) evaluated on the fine (embedded)

mesh (h) using the fine mesh solution (u_h) and performing a Taylor series expansion about it to calculate the output of interest based off the solution on the coarse (working) mesh (H) interpolated onto the fine mesh ($\tilde{u}_h$).

$$L_h(u_h) \approx L_h(\tilde{u}_h) + \frac{\partial L_h(\tilde{u}_h)}{\partial u_h}(u_h - \tilde{u}_h) \tag{7.1}$$

This would indicate that to get a good estimate of the fine mesh output of interest it is necessary to solve both the fine and coarse mesh problems, and take the difference between the coarse mesh solution prolongated onto the fine mesh and the fine mesh solution to multiply the derivative of the output of interest. This of course defeats the purpose of mesh refinement as it would require solution of the nonlinear problem on the more expensive embedded mesh. Instead a Taylor series expansion about the fine grid is used to move from the coarse grid onto the fine grid, which will have a nonzero residual for the coarse grid state interpolated onto the fine grid.

$$R_h(u_h) = 0 \approx R_h(\tilde{u}_h) + \frac{\partial R_h(\tilde{u}_h)}{\partial u_h}(u_h - \tilde{u}_h) \tag{7.2}$$

Combining the two equations returns the following.

$$L_h(u_h) \approx L_h(\tilde{u}_h) + \frac{\partial L_h(\tilde{u}_h)}{\partial u_h}\left[\frac{\partial R_h(\tilde{u}_h)}{\partial u_h}\right]^{-1} R_h(\tilde{u}_h) \tag{7.3}$$

Defining the adjoint on the embedded mesh, ψ_h, as:

$$\left[\frac{\partial R_h(\tilde{u}_h)}{\partial u_h}\right]^T \psi_h = \frac{\partial J_h(\tilde{u}_h)}{\partial u_h} \tag{7.4}$$

and substituting in ψ_h returns the below.

$$L_h(u_h) \approx L_h(\tilde{u}_h) + \psi_h^T R_h(\tilde{u}_h) \tag{7.5}$$

This requires an adjoint solve on the embedded mesh, which is not desirable due to the expense. To avoid computing the adjoint on the embedded mesh, the coarse mesh adjoint in reconstructed onto the embedded mesh using the same combined bilinear and biquadratic approximation used to reconstruct the conservative variables from the coarse mesh onto the fine mesh.

The first step is computing a bilinear approximation in each cell where f is the value of the variable of interest in this reconstruction – in this work this is one of the conservative or adjoint variables.

$$f = c_0 + c_1 x + c_2 y + c_3 xy \tag{7.6}$$

This is done by looping over the cells and reconstructing the conservative variables to each of the nodes and averaging the reconstructed conservative variable and gradient values at each node from the cells that it forms. Each cell then sets up an overconstrained linear least squares problem for each individual conservative variable. Having averaged the variable values and gradients to compute values at each node of the mesh, the values and gradients at each of the three nodes of the triangle are used to create a system with nine constraints (f, f_x, f_y at each of the three nodes) and, as can be seen in the approximation above, only four

unknowns (c_0, c_1, c_2, c_3). These bilinear approximations are then used to populate the elements of the right hand side corresponding to the conservative variables for the restricted biquadratic approximation.

$$f = c_0 + c_1 x + c_2 y + c_3 xy + c_4 x^2 + c_5 y^2 + c_6 x^2 y + c_7 xy^2 \tag{7.7}$$

This biquadratic approximation neglects the double quadratic term as it has been shown to add undesired oscillations into the solution and would also provide the ninth unknown which would entail no longer being an overconstrained system. The biquadratic approximation is then used to evaluate the conservative variable values in each of the child cells for the embedded mesh. The two predominant error estimates are the classical dual weighted residual and the dual weighted residual with computable correction. For the first method, the error due to the spatial discretization is written as follows, where the biquadratic interpolation of the adjoint is used as a proxy for the adjoint on the fine mesh.

$$\eta_H = |(\tilde{\psi}_h)^T R_h(\tilde{u}_h)| \tag{7.8}$$

A parent element error estimate vector can be calculated by looping over all the child elements of the larger parent cell as follows.

$$\eta_H = |\sum_{j \in V_{parent}} \psi_{BQ}^T R_h(\tilde{u}_h)_j| \tag{7.9}$$

The second technique is the dual weighted residual with computable correction method. To compute an adjoint-based computable correction it is necessary to split the adjoint into two terms. The approximate adjoint, which multiplies the residual and provides the computable correction in the objective and the error in the adjoint which will be dotted with the residual to obtain the error estimate.

$$L_h(u_h) \approx L_h(\tilde{u}_h) - (\tilde{\psi}_h)^T R_h(\tilde{u}_h) - (\psi_h - \tilde{\psi}_h)^T R_h(\tilde{u}_h) \tag{7.10}$$

The first term is the adjoint correction to the functional, and the second is the remaining error which is used to drive the mesh adaptation. In practice, again rather than computing the embedded mesh adjoint, a biquadratic interpolation of the adjoint is used, and a bilinear interpolation is used for for the prolongated coarse mesh adjoint (the approximate adjoint). Furthermore, the biquadratic interpolation (which is an estimate of ψ_h) is used for the adjoint correction even though the adjoint correction term actually asks for $\tilde{\psi}_h$.

$$L_h(u_h) \approx L_h(\tilde{u}_h) - (\psi_{BQ})^T R_h(\tilde{u}_h) - (\psi_{BQ} - \psi_{BL})^T R_h(\tilde{u}_h) \tag{7.11}$$

Others have dealt with the issue of solving the adjoint on the fine mesh by solving the adjoint on the coarse mesh, and then using that as an initial guess for the linear solver of the fine mesh adjoint problem, they then run a few smoothing passes to get a better approximation of the fine mesh adjoint at a low cost.

Implementation is achieved using the coarse adjoint prolongated onto the fine mesh as the approximate adjoint and the smoothed adjoint on the fine mesh is used as the the exact adjoint [30].

7.1.1 Virtual Mesh Method

The virtual mesh method allows for an error estimate and refinement without needing to actually create the embedded mesh. The virtual mesh uses differing order reconstructions of the conservative variables to produce estimates of the conservative variable and gradient values at the centroids of the virtual child cells and then uses those values to compute the residual lifted back to the coarse mesh.

The first step is computing a bilinear approximation of the conservative variables in each cell using the node aggregation and interpolation step shown previously. The biquadratic approximation is then used to evaluate the conservative variable values and gradients at each of the 3 exterior virtual children of the cell in question. These values and gradients are then used to reconstruct to the cell edge, and the flux across the child face (half of the original face) is computed.

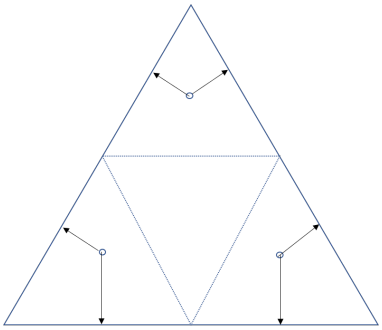


Figure 7.1: Virtual Mesh Residual Diagram

As can be seen in Figure 7.1, no fluxes are computed across the interior edges into the central child element of the virtual mesh as these fluxes will cancel out when they are lifted to the parent mesh. The exterior edge fluxes will be added to the residual vector of the parent mesh and this vector will then multiply the coarse mesh adjoint vector (this treats the coarse mesh adjoint as the fine mesh adjoint) to obtain a cell-wise error estimate. If one instead uses the biquadratic approximate to extrapolate to the edges, this could be viewed as calculating the fine mesh residual with a third order finite volume method with a non-k-exact least-squares reconstruction and without curved geometry and then weighting that higher accuracy residual with the coarse mesh adjoint. The virtual mesh method is only used for the error estimation without functional correction. The cellwise error estimate is expressed below:

$$\eta_H = |(\psi_H)^T R_H^h(\tilde{u}_h)| \tag{7.12}$$

where ψ_H is the coarse mesh adjoint, R_H^h is the fine mesh residual lifted back to the coarse mesh through use of the virtual mesh subroutines, and $\tilde{u}_h$ is the coarse mesh conservative variables interpolated onto the

virtual fine mesh using the interpolation routines previously outlined.

7.2 Mesh Refinement

For uniform mesh refinement, which is used in the embedded mesh error estimation subroutine, the driver simply marks all cells for a 4:1 refinement and all edges for a 2:1 refinement. It stores the parent child maps for the cells and uses those for the interpolations and computations required for the adjoint based error estimation. The residual is calculated on the fine mesh, and the adjoint is interpolated appropriately to form an error estimate.

For adaptive mesh refinement, the driver (as shown in algorithm 9) is more complex as the refinement is not uniform. The mesh refinement begins by ordering the elements by the magnitude of their respective error estimates. The driver then creates a mask marking all cells in the top $xx\%$ (a user defined value) for a 4:1 refinement that subdivides the triangular element into 4 similar elements –this is referred to as a fixed fraction approach to refinement. The algorithm then enters a loop where it calls a subroutine (shown in algorithm 10) which in turn passes over the edges marking and edge for refinement if it belongs to an element that will be refined 4:1. It will also mark all cells with a single neighbor marked for 4:1 refinement for a 2:1 refinement, and all cells with two or three neighbors marked for 4:1 refinement are marked for 4:1 refinement themselves. This subroutine is repeated until the algorithm ceases to tag new cells and edges for refinement.

The driver then passes over the edges and refines the edges appropriately; then the driver passes over the cells refining according to the tags calculated early on in the driver. This algorithm creates a mesh without hanging nodes. After refining the cells and creating the new interior edges to refine the mesh, the driver then recreates the cell-edge maps necessary for interpolation and residual computation on the refined mesh. For adaptively refined meshes the interpolation is used to create restart files. The driver then calls a boundary curvature correction based off a geometry parameterization.

Finally, for an adaptively refined mesh (the typical output of the AMR), the driver then loops over the edges performing incircle predicates [64], and edge swapping if the two triangles it forms are not the Delaunay triangulation, until every triangle in the mesh passes the incircle predicate test and the mesh is Delaunay. This step can fail on meshes created by halving the quads from a Cartesian mesh, in which case a non-swapped mesh is outputted.

The refinement module shown in this work, due to its hierarchical nature and inability to coarsen and redistribute mesh points, is poorly suited for the refinement required in the cases presented in this work. As such, the refinement module is used for uniform refinement in the error estimation algorithm to provide an elementwise error estimate. This is then paired with the Refine code developed at NASA Langley [28] to produce adaptively refined meshes through the refinement process. The Refine code uses a multiscale-metric

based refinement criteria where the metric at each node is defined by the error and the anisotropy of the mesh is determined through the gradient of the Mach number. The metric (Me) at each node is area averaged from the cells it comprises and it expressed as:

$$Me = \left(\frac{1}{\eta_g \eta_k}\right)^{\omega} \tag{7.13}$$

where ω is an under-relaxation parameter, and η_g and η_k are global and local error ratios. η_g is the integrated error in the mesh divided by the median error multiplied by the number of cells in the mesh. η_k is the cellwise ratio between the error in that cell and the median error in the mesh. This corresponds to targeting an error value in each subsequent mesh that is based off the median error in the previous mesh [65].

Algorithm 9 Adaptive Mesh Refinement Driver

```
1:  procedure MESH REFINEMENT
2:      Sort cells by error estimate
3:      Mark all cells in the top xx% with a 1 in mask, all others get 0
4:      Tag edges and cells for refinement
5:      Refine Edges
6:      Refine cells according to their tag
7:      Update edgelist
8:      Compute correction to surface nodes based of CST parameterization
9:      Compute mesh deformation due to surface perturbation
10:     Delaunay swap till mesh is delaunay
```

Algorithm 10 Tag Cells and Edges for Mesh Refinement

```
1:  procedure TAG CELLS
2:      tags = 0 and nprop = 0
3:      repeat
4:          for edgenum = 1, ..., nEdges do
5:              if Mask(lefttri) = 1 .AND. Mask(righttri) = 0 then
6:                  Refine edge and tag(righttri) + +
7:              else if Mask(righttri) = 1 .AND.Mask(lefttri) = 0 then
8:                  Refine edge and tag(lefttri) + +
9:              else if Mask(lefttri) = 1 .AND. Mask(righttri) = 1 then
10:                 Refine edge
11:         for cellnum = 1, ..., nCells do
12:             if tags(cellnum) = 2 .OR. (tags(cellnum) = 3 .AND. mask(cellnum) = 0) then
13:                 mask(cellnum) = 1 .AND. nprop + +
14:     until nProp = 0
```

7.2.1 CST Parameterization and Boundary Curvature Correction

The Class-Shape function Transformation (CST), introduced by Brenda Kulfan [66] is commonly used for aerodynamic shape optimization due to the smooth control of design points, and is instead used here to parameterize the airfoil and provide surface node curvature corrections. The class function portion of the

parameterization is written as:

$$C = \psi^{N_1}(1-\psi)^{N_2} \tag{7.14}$$

where $\psi = \frac{x}{ch}$ is the normalized x coordinate of the surface being parameterized, where ch is the chord length and the numbers N_1, N_2 are chosen by the user. The paper by Kulfan showed that the values of .5 and 1.0 work well for airfoils as they allow the shape function at the leading and trailing edges to correspond to design criteria. The shape function is written as such:

$$S = \sum_{i=0}^{n} K_{i,n} A_n \psi^i (1-\psi)^{n-i} \tag{7.15}$$

where the A_n coefficients are the design variables. The shape functions when combined with the class functions return $S(0,A) = A_0 = \sqrt{\frac{2R_{le}}{ch}}$ and $S(1,A) = A_n = tan\beta + \frac{\Delta z_{te}}{ch}$ where R_{le} is the leading edge radius, β is the trailing edge boat tail angle and $\frac{\Delta_{te}}{ch}$ is the trailing edge thickness. The y value at any particular x-coordinate is given by:

$$y_{cst}(\psi, A) = C(\psi)S(\psi, A) + \psi\frac{\Delta z_{te}}{c} \tag{7.16}$$

This allows the generation of an overconstrained least squares system that has one more unknown than the desired CST order (because of the trailing edge thickness unknown), and as many constraints as airfoil boundary nodes. For this system, the right hand side is the coarse airfoil surface coordinates, the unknowns are the CST A_n coefficients and the trailing edge thickness, and the matrix consists of the Bernstein polynomials multiplied by the class function evaluated at the normalized x-coordinate variables. This least squares system is solved and used to compute the locations of the new nodes created on the surface of the airfoil geometry. First, the new coordinate CST value is computed, this is done by looping over the subdivided edges on the coarse mesh, finding the normalized x-coordinate of the new node and calculating the CST value for that normalized x-coordinate on that given surface. The correction involves subtracting from that value the value given by linearly interpolating between the CST values of the two nodes on the coarse mesh that make the subdivided edge that created the node. Finally these corrections are fed into the mesh deformation scheme which moves the interior nodes due to the perturbations of the new nodes; this is done to prevent negative areas and increase mesh quality. This implementation will show a greater correction where the curvature is high (oftentimes near the leading edge) and almost no correction where the curvature is low (often near the trailing edge). Figure 7.2 shows a comparison of the initial mesh in red and a uniformly refined mesh with curvature correction in blue. It shows the high curvature at the leading edge, combined with the curvature correction and mesh motion, moves the points enough that the hierarchical nature of the mesh is not clear at first glance for elements near the airfoil. For elements further away from the geometry, the difference is negligible. Figure 7.3 shows the same comparison but away from the location of max curvature; it shows that this curvature correction affects primarily the new child cells that share the new node created by the subdivision of the edge on the geometry. Finally, Figure 7.4 shows almost no change

due to the correction as these nodes are located near the area of minimum curvature (near the trailing edge) and the hierarchical nature of the mesh is readily clear.

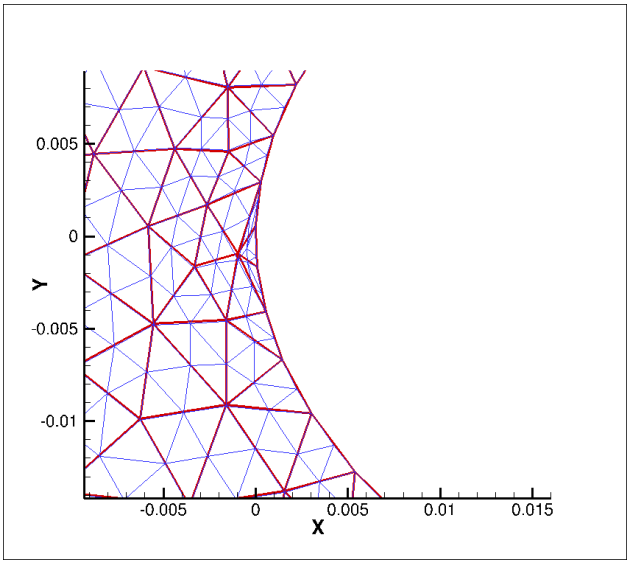


Figure 7.2: Curvature along a NACA0012 leading edge: contrast between initial mesh (red) and refined mesh with curvature correction(blue)

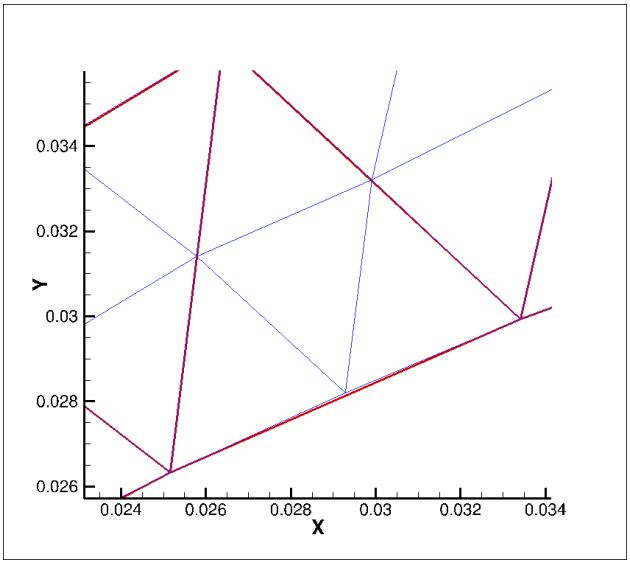


Figure 7.3: Curvature along a NACA0012 intermediate curvature: contrast between initial mesh (red) and refined mesh with curvature correction(blue)

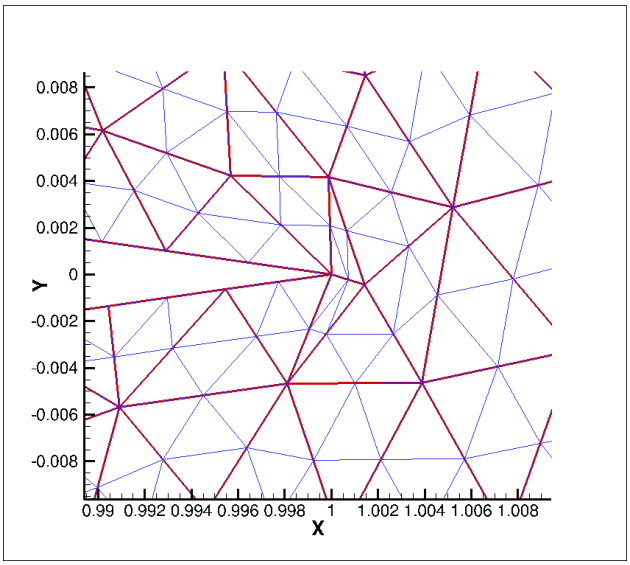


Figure 7.4: Curvature along a NACA0012 trailing edge: contrast between initial mesh (red) and refined mesh with curvature correction(blue)

7.3 Development of the Pseudo-Time Accurate Dual-Weighted Constraint

As stated before, the pseudo-time accurate adjoint method is drawn from the derivation of the unsteady adjoint; the algorithm works backwards through pseudo-time to get the pseudo-time accurate adjoint solution. This formulation seeks to find an estimate of how much of the error in the quantity of interest is due to the spatial discretization for these unconverged and oscillatory flows. In this derivation the output of interest is a pseudo-time averaged functional, averaged over the last m steps for a simulation that runs through n pseudo-time steps. This gives an output of interest L as:

$$L = L(u_h^n, u_h^{n-1}, ..., u_h^{n-m}) \tag{7.17}$$

where u_h^n is the conservative variable vector at the final time step n on the fine mesh. A Taylor series expansion about the fine mesh objective is then performed, where $\tilde{u}_h^k$ is the coarse (working) mesh (H) solution at iteration k interpolated onto the embedded mesh:

$$\begin{aligned} L = L(u_h^n, u_h^{n-1}, ..., u_h^{n-m}) \approx L(\tilde{u}_h^n, \tilde{u}_h^{n-1}, ..., \tilde{u}_h^{n-m}) + \frac{\partial L}{\partial \tilde{u}_h^n}(u_h^n - \tilde{u}_h^n) \\ + \frac{\partial L}{\partial \tilde{u}_h^{n-1}}(u_h^{n-1} - \tilde{u}_h^{n-1}) \\ + ... \\ + \frac{\partial L}{\partial \tilde{u}_h^{n-m}}(u_h^{n-m} - \tilde{u}_h^{n-m}) \end{aligned} \tag{7.18}$$

Unlike the typical adjoint and dual weighted residual method where the constraint that the residual is

zero is used, here this is impossible as this is not true at each pseudo-time step. Instead, a constraint is selected based on the pseudo-time evolution of the solution, for which, the k^{th} constraint will be referred to as G^k. Because of the algorithm implementation it is clear that the constraint is dependent only on the old time-step and the new time-step, expressed as follows.

$$G^k = G^k(u^k, u^{k-1}) = 0 \tag{7.19}$$

Linearizing about the states gives:

$$G^k(u_h^k, u_h^{k-1}) = 0 \approx G^k(\tilde{u}_h^k, \tilde{u}_h^{k-1}) + \frac{\partial G^k}{\partial \tilde{u}_h^k}(u_h^k - \tilde{u}_h^k) + \frac{\partial G^{k-1}}{\partial \tilde{u}_h^{k-1}}(u_h^{k-1} - \tilde{u}_h^{k-1}) \tag{7.20}$$

by isolating for the error in the conservative variable reconstruction the below equation is obtained.

$$(u_h^{k-1} - \tilde{u}_h^{k-1}) = -\left[\frac{\partial G^{k-1}}{\partial \tilde{u}_h^{k-1}}\right]^{-1}\left[G^k(\tilde{u}_h^k, \tilde{u}_h^{k-1}) + \frac{\partial G^k}{\partial \tilde{u}_h^k}(u_h^k - \tilde{u}_h^k)\right] \tag{7.21}$$

Substituting in the error in the conservative variables returns the below expression.

$$\begin{aligned} L(u_h^n, u_h^{n-1}, ..., u_h^{n-m}) &= L(\tilde{u}_h^n, \tilde{u}_h^{n-1}, ..., \tilde{u}_h^{n-m}) \\ &- \left(\frac{\partial L}{\partial u}\right)_{\tilde{u}_h^n}\left[\frac{\partial G^n}{\partial u_h^n}\right]^{-1}_{u_h^n}\left[G(\tilde{u}_h^n, \tilde{u}_h^{n-1}) + \left(\frac{\partial G^n}{\partial u_h^{n-1}}\right)_{u_h^{n-1}}(u_h^{n-1} - \tilde{u}_h^{n-1})\right] \\ &+ \left(\frac{\partial L}{\partial u}\right)_{\tilde{u}_h^{n-1}}(u_h^{n-1} - \tilde{u}_h^{n-1}) \\ &+ ... \\ &+ \left(\frac{\partial L}{\partial u}\right)_{\tilde{u}_h^{n-m}}(u_h^{n-m} - \tilde{u}_h^{n-m}) \end{aligned} \tag{7.22}$$

Defining the equation for the final state adjoint, as in equation 3.53, returns the equation below:

$$\Lambda^{nT} = -\left(\frac{\partial L}{\partial u}\right)_{\tilde{u}_h^n}\left[\frac{\partial G^n}{\partial u_h^n}\right]^{-1}_{u_h^n} \tag{7.23}$$

it is then necessary to substitute and separate terms to get the following expression.

$$\begin{aligned} L(u_h^n, u_h^{n-1}, ..., u_h^{n-m}) &= L(\tilde{u}_h^n, \tilde{u}_h^{n-1}, ..., \tilde{u}_h^{n-m}) \\ &+ \Lambda^{nT}\left[G(\tilde{u}_h^n, \tilde{u}_h^{n-1})\right] \\ &+ \left[\left(\frac{\partial L}{\partial u}\right)_{\tilde{u}_h^{n-1}} + \Lambda^{nT}\left(\frac{\partial G^n}{\partial u_h^{n-1}}\right)_{u_h^{n-1}}\right](u_h^{n-1} - \tilde{u}_h^{n-1}) \\ &+ ... \\ &+ \left(\frac{\partial L}{\partial u}\right)_{\tilde{u}_h^{n-m}}(u_h^{n-m} - \tilde{u}_h^{n-m}) \end{aligned} \tag{7.24}$$

Looking at the above equation, it is necessary to define the following adjoint equation, similar to the recurrence relation in equation 3.44.

$$\Lambda^{n-1T} = -\left[\left(\frac{\partial L}{\partial u}\right)_{\tilde{u}_h^{n-1}} + \Lambda^{nT}\left(\frac{\partial G^n}{\partial u_h^{n-1}}\right)_{u_h^{n-1}}\right]\left[\frac{\partial G^{n-1}}{\partial u_h^{n-1}}\right]^{-1}_{u_h^{n-1}} \tag{7.25}$$

Substituting in 7.25 to 7.24 returns the expression below.

$$\begin{aligned} L(u_h^n, u_h^{n-1}, ..., u_h^{n-m}) &= L(\tilde{u}_h^n, \tilde{u}_h^{n-1}, ..., \tilde{u}_h^{n-m}) \\ &+ \Lambda^{nT}\left[G(\tilde{u}_h^n, \tilde{u}_h^{n-1})\right] \\ &+ \Lambda^{n-1T}\left[G(\tilde{u}_h^{n-1}, \tilde{u}_h^{n-2})\right] \\ &+ \left[\left(\frac{\partial L}{\partial u}\right)_{\tilde{u}_h^{n-2}} + \Lambda^{n-1T}\left(\frac{\partial G^{n-1}}{\partial u_h^{n-2}}\right)_{u_h^{n-2}}\right](u_h^{n-2} - \tilde{u}_h^{n-2}) \\ &+ ... \\ &+ \left(\frac{\partial L}{\partial u}\right)_{\tilde{u}_h^{n-m}}(u_h^{n-m} - \tilde{u}_h^{n-m}) \end{aligned} \tag{7.26}$$

Working this recurrence relation m times back in pseudo-time out through the averaging window returns:

$$\begin{aligned} L(u_h^n, u_h^{n-1}, ..., u_h^{n-m}) &= L(\tilde{u}_h^n, \tilde{u}_h^{n-1}, ..., \tilde{u}_h^{n-m}) \\ &+ \Lambda^{nT}\left[G(\tilde{u}_h^n, \tilde{u}_h^{n-1})\right] \\ &+ \Lambda^{n-1T}\left[G(\tilde{u}_h^{n-1}, \tilde{u}_h^{n-2})\right] \\ &+ \Lambda^{n-2T}\left[G(\tilde{u}_h^{n-2}, \tilde{u}_h^{n-3})\right] \\ &+ ... \\ &+ \Lambda^{n-mT}\left[G(\tilde{u}_h^{n-m}, \tilde{u}_h^{n-(m+1)})\right] \\ &+ \Lambda^{n-mT}\left(\frac{\partial G^{n-m}}{\partial u_h^{n-(m+1)}}\right)(u_h^{n-(m+1)} - \tilde{u}_h^{n-(m+1)}) \end{aligned} \tag{7.27}$$

When the backwards-in-pseudo-time integration exits the averaging window the recurrence relation loses the source term from the linearization of the output of interest and simplifies to the below equation.

$$\Lambda^{k-1T} = -\left[\Lambda^{kT}\left(\frac{\partial G^k}{\partial u_h^{k-1}}\right)_{u_h^{k-1}}\right]\left[\frac{\partial G^{k-1}}{\partial u_h^{k-1}}\right]^{-1}_{u_h^{k-1}} \tag{7.28}$$

This returns a final expression for the averaged output of interest on the fine mesh as shown below.

$$\begin{aligned} L(u_h^n, u_h^{n-1}, ..., u_h^{n-m}) &= L(\tilde{u}_h^n, \tilde{u}_h^{n-1}, ..., \tilde{u}_h^{n-m}) \\ &+ \Lambda^{nT}\left[G(\tilde{u}_h^n, \tilde{u}_h^{n-1})\right] \\ &+ \Lambda^{n-1T}\left[G(\tilde{u}_h^{n-1}, \tilde{u}_h^{n-2})\right] \\ &+ ... \\ &+ \Lambda^{1T}\left[G(\tilde{u}_h^1, \tilde{u}_h^0)\right] \\ &+ \Lambda^{1T}\left[\frac{\partial G^1}{\partial u_h^0}\right](u_h^0 - \tilde{u}_h^0) \end{aligned} \tag{7.29}$$

Grouping the output of interest terms returns:

$$\begin{aligned}\Delta L(\tilde{u}_h^n, \tilde{u}_h^{n-1}, ..., \tilde{u}_h^{n-m}) \approx & \Lambda^{nT}\left[G(\tilde{u}_h^n, \tilde{u}_h^{n-1})\right] \\ & + \Lambda^{n-1T}\left[G(\tilde{u}_h^{n-1}, \tilde{u}_h^{n-2})\right] \\ & + ... \\ & + \Lambda^{1T}\left[G(\tilde{u}_h^1, \tilde{u}_h^0)\right] \\ & + \Lambda^{1T}\left[\frac{\partial G^1}{\partial u_h^0}\right]\left(u_h^0 - \tilde{u}_h^0\right)\end{aligned} \tag{7.30}$$

This could be viewed as the error in the output of interest due to the spatial discretization for an unconverged fixed point iteration with a correction included to compute the impact of the initial condition on the fixed point. For a uniform flow there is no correction introduced from the initial condition, but for a restart file the error in the solution reconstruction would lead to a nonzero value due to the nonuniform initial flow. In an AMR case, this could theoretically allow for calculating the error introduced by a mesh adaptation cycle, and then going all the way back through the adaptation chain back to the first mesh and freestream flow. While this may be theoretically possible, this is untenable in practice. This does however open up the door to only integrating partially back in iteration-space as was done in chapter 6 for sensitivities in design cases. This error could be modeled through the subtraction of the bilinear reconstruction from the biquadratic reconstruction and this could perhaps be sufficient as an correction to the error estimate.

7.3.1 Error Estimation for Newton's Method

For this section the analogues of the error estimation methods shown in the steady-state section are shown for Newton's method using the approximate adjoint linearization with the inverse identity. To begin, it is necessary to refer once again to equation (3.10):

$$u^k = u^{k-1} - \left[P_{k-1}\right]^{-1} R \tag{7.31}$$

Taking the derivative of the shifted fixed-point iteration at each pseudo-time-step with respect to both states returns the equation below.

$$\begin{aligned}\frac{\partial G^k}{\partial u_h^k} &= I \\ \frac{\partial G^k}{\partial u_h^{k-1}} &= -I + \left[P_{k-1}\right]^{-1}\left[\frac{\partial R(u_h^{k-1})}{\partial u_h^{k-1}}\right]_2 + \frac{\partial \left[P_{k-1}\right]^{-1}}{\partial u_h^{k-1}} R(u_h^{k-1})\end{aligned} \tag{7.32}$$

This work once again uses the differentiation of a matrix inverse first shown in equation 3.16 and reproduced here.

$$\frac{d\left[K\right]^{-1}}{dx} = -\left[K\right]^{-1}\left[\frac{dK}{dx}\right]\left[K\right]^{-1} \tag{7.33}$$

The derivative of the constraint term substituting in equation (7.33) is shown below.

$$\begin{aligned}\frac{\partial G^k}{\partial u_h^k} &= I \\ \frac{\partial G^k}{\partial u_h^{k-1}} &= -I + \left[P_{k-1}\right]^{-1} \left[\frac{\partial R(u_h^{k-1})}{\partial u_h^{k-1}}\right]_2 - \left[P_{k-1}\right]^{-1} \frac{\partial \left[P_{k-1}\right]}{\partial u_h^{k-1}} \left[P_{k-1}\right]^{-1} R(u_h^{k-1})\end{aligned} \tag{7.34}$$

Using the definition of the nonlinear solver increment, the above equation can be simplified as follows:

$$\begin{aligned}\frac{\partial G^k}{\partial u_h^k} &= I \\ \frac{\partial G^k}{\partial u_h^{k-1}} &= -I + \left[P_{k-1}\right]^{-1} \left[\left[\frac{\partial R(u_h^{k-1})}{\partial u_h^{k-1}}\right]_2 - \frac{\partial \left[P_{k-1}\right]}{\partial u_h^{k-1}} \Delta u_h\right]\end{aligned} \tag{7.35}$$

As before these Hessian vector products can be computed using complex Frechét derivatives, rather than hand differentiating the residual operator twice to obtain the Hessian operator. Using the equation for the adjoint at the final pseudo-time step with the constraint derivatives returns the identity below.

$$\left[I\right] \Lambda_h^n = -\left[\frac{\partial L}{\partial u_h^n}\right]^T \tag{7.36}$$

Substituting in the constraint derivatives from equation (7.35) into equation (5.1) returns:

$$\left[I\right] \Lambda^{k-1} = -\left[-I + \left[P_{k-1}\right]^{-1} \left[\left[\frac{\partial R(u_h^{k-1})}{\partial u_h^{k-1}}\right]_2 - \frac{\partial \left[P_{k-1}\right]}{\partial u_h^{k-1}} \Delta u_h\right]\right]^T \Lambda_h^k \tag{7.37}$$

This recurrence relation can be rewritten in delta form as below.

$$\Delta\Lambda_h = -\left[\left[P_{k-1}\right]^{-1} \left[\left[\frac{\partial R(u_h^{k-1})}{\partial u_h^{k-1}}\right]_2 - \frac{\partial \left[P_{k-1}\right]}{\partial u_h^{k-1}} \Delta u_h\right]\right]^T \Lambda_h^k \tag{7.38}$$

Distributing the transpose returns:

$$\Delta\Lambda_h = -\left[\left[\frac{\partial R(u_h^{k-1})}{\partial u_h^{k-1}}\right]_2 - \frac{\partial \left[P_{k-1}\right]}{\partial u_h^{k-1}} \Delta u_h\right]^T \left[P_{k-1}\right]^{-T} \Lambda_h^k \tag{7.39}$$

which motivates the definition of a secondary adjoint variable for each recurrence relation.

$$\left[P_{k-1}\right]^T \psi_h^k = \Lambda_h^k \tag{7.40}$$

It is possible to rewrite again into the delta form of the adjoint recurrence relation as follows.

$$\Delta\Lambda = -\left[\left[\frac{\partial R(u_h^{k-1})}{\partial u_h^{k-1}}\right]_2 - \frac{\partial \left[P_{k-1}\right]}{\partial u_h^{k-1}} \Delta u_h\right]^T \psi_h^k \tag{7.41}$$

It is important to note that, as noted previously, exact dual correspondence here between the adjoint solver and the forward one is not required. Additionally, all these operations are on the embedded mesh, which makes this adjoint tremendously expensive, as it requires nearly the expense of a nonlinear solve on the

much finer embedded mesh. To make this more tractable, in this work the adjoint is calculated on the coarse mesh, and then uses the same interpolants as in the steady state adjoint error estimate to get an estimate of the adjoint on the embedded mesh. This means that convergence of the adjoint problem is guaranteed if the dual solver of the analysis linear solver is used at each iteration. While this does make the process cheaper, it is still necessary to interpolate the coarse mesh solution onto the fine mesh and compute the constraints on the fine mesh, which is a nontrivial expense. Hence the interest in using the flow reconstruction correction mentioned previously. The error equation can then be rendered as:

$$L = L(u_h^n, u_h^{n-1}, ..., u_h^{n-m}) - L(\tilde{u}_h^n, \tilde{u}_h^{n-1}, ..., \tilde{u}_h^{n-m}) \approx -\epsilon_c - \epsilon_a \tag{7.42}$$

where ϵ_c (or the functional correction) is calculated similarly to the method in the steady state adjoint and is shown below.

$$\begin{aligned}
\epsilon_c = & {\Lambda_{BQ}^n}^T G^n(\tilde{u}_h^n, \tilde{u}_h^{n-1}) \\
& + {\Lambda_{BQ}^{n-1}}^T G^{n-1}(\tilde{u}_h^{n-1}, \tilde{u}_h^{n-2}) \\
& + ... \\
& + {\Lambda_{BQ}^1}^T G^1(\tilde{u}_h^1, \tilde{u}_h^0) \\
& + {\Lambda_{BQ}^1}^T \left[\frac{\partial G^1}{\partial \tilde{u}_h^0}\right](u_{BQ}^0 - u_{BL}^0)
\end{aligned} \tag{7.43}$$

ϵ_a (or the remaining error) is calculated again using the adjoint error correction similar to in the steady state adjoint:

$$\begin{aligned}
\epsilon_a = & (\Lambda_{BQ}^n - \Lambda_{BL}^n)^T G^n(\tilde{u}_h^n, \tilde{u}_h^{n-1}) \\
& + (\Lambda_{BQ}^{n-1} - \Lambda_{BL}^{n-1})^T G^{n-1}(\tilde{u}_h^{n-1}, \tilde{u}_h^{n-2}) \\
& + ... \\
& + (\Lambda_{BQ}^1 - \Lambda_{BL}^1)^T G^1(\tilde{u}_h^1, \tilde{u}_h^0) \\
& + (\Lambda_{BQ}^1 - \Lambda_{BL}^1)^T \left[\frac{\partial G^1}{\partial \tilde{u}_h^0}\right](u_{BQ}^0 - u_{BL}^0)
\end{aligned} \tag{7.44}$$

The flow reconstruction correction at iteration k, given by $(\Lambda_{BQ}^k - \Lambda_{BL}^k)^T \left[\frac{\partial G^k}{\partial \tilde{u}_h^{k-1}}\right](u_{BQ}^{k-1} - u_{BL}^{k-1})$ is implemented by differentiating the fixed point iteration on the fine mesh.

$$\frac{\partial G_h^k}{\partial u_h} = \left[-I + [P_{k-1}]_h^{-1} \frac{\partial P_{k-1}}{\partial u_h^{k-1}} [P_{k-1}]_h^{-1} R_h(\tilde{u}_h^{k-1}) - [P_{k-1}]_h^{-1} \frac{\partial R_h}{\partial u^{k-1}}\right] \tag{7.45}$$

7.3.1.1 Error Estimation on Virtual Mesh

For the virtual mesh method, the fine mesh fixed point iteration is done through using the residual evaluations on the virtual fine mesh as shown in 7.1.1, with the coarse mesh preconditioner matrix. This is expressed mathematically as:

$$G^k(u^k, u^{k-1})_H^h = u_H^k - u_H^{k-1} - [P_{k-1}]_H R(u_h^{k-1})_H^h \tag{7.46}$$

where $R(\tilde{u}_h^{(k-1)})_H^h$ is the fine mesh residual calculated on the virtual mesh from the interpolated flow solution $\tilde{u}_h^{k-1}$ and lifted back to the coarse mesh. A parallel Δu_H^h is defined that corresponds to the fine mesh Δu lifted back to the coarse mesh, and the error estimation derivation proceeds.

$$\Delta u_H^h = -\left[P_{k-1}\right]_H R(\tilde{u}_h^{k-1})_H^h \tag{7.47}$$

Alternatively, the preconditioner matrix used could also be the fine mesh preconditioner matrix lifted back to the coarse mesh, in which case the fine mesh fixed point iteration becomes:

$$G^k(u^k, u^{k-1})_h = u_H^k - u_H^{k-1} - \left[P_{k-1}\right]_H^h R(u_h^{k-1})_H^h \tag{7.48}$$

where $\left[P_{k-1}\right]_H^h$ uses the gradient reconstructed values to the face on the virtual fine mesh to form the preconditioner on the coarse mesh. In such a case the lifted Δu becomes:

$$\Delta u_H^h = -\left[P_{k-1}\right]_H^h R(\tilde{u}_h^{k-1})_H^h \tag{7.49}$$

While this method showed promise it did not meaningfully affect the results and so the cheaper method that uses the virtual mesh for the fine mesh residual evaluation only was used. The computation of the adjoint is unaffected, but the computation of the error becomes:

$$d\eta_H^k = |\Lambda_H^{kT} G^k(u^k, u^{k-1})_H^h| \tag{7.50}$$

where:

$$\eta_H = \sum_{k=1}^{n} d\eta_H^k \tag{7.51}$$

The analogue of the flow reconstruction correction shown in the embedded mesh methods on the virtual mesh uses the derivative of the fixed point on the virtual fine mesh and could be expressed as:

$$\frac{\partial G_h^k}{\partial u^{k-1}} = \left[-I + \left[P_{k-1}\right]_H^{-1} \frac{\partial P_{k-1}}{\partial u^{k-1}}_H \left[P_{k-1}\right]_H R(\tilde{u}_h^{k-1})_H^h - \left[P_{k-1}\right]_H^{-1} \frac{\partial R}{\partial u^{k-1}}\right] \tag{7.52}$$

where the flow reconstruction error is computed on the virtual mesh and lifted back to the coarse mesh following the same procedure for the virtual mesh lifted residual. This is done to avoid linearization of the interpolation processes, so that the flow Jacobian can be computed off the coarse mesh values, however this correction returned poor results, and so the correction is neglected for the virtual mesh method, and the error estimate is kept to the expressions shown in equations 7.50 and 7.51.

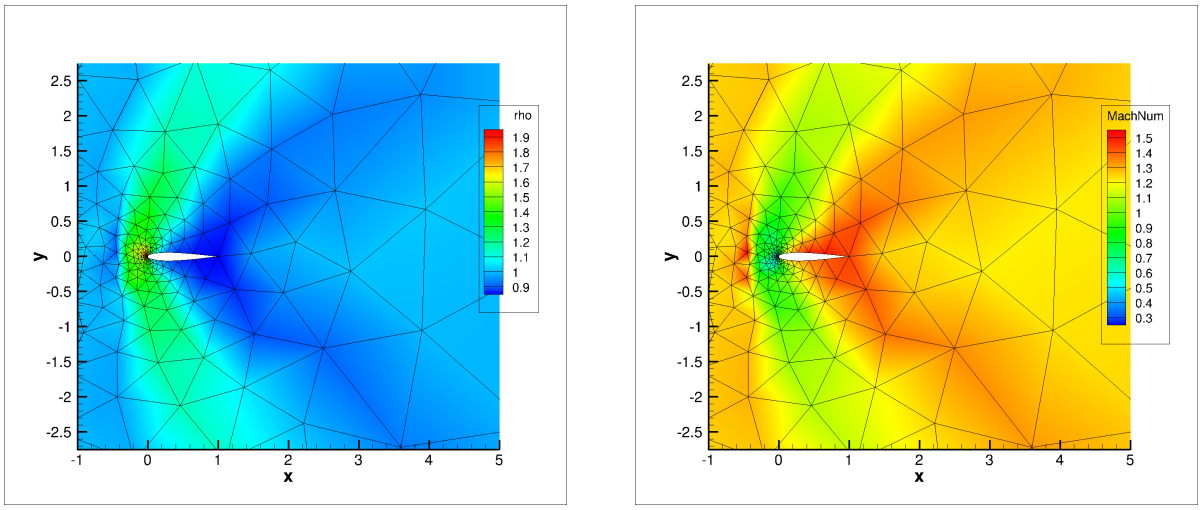
7.4 Mesh Refinement Results

The test cases in this section show the results of a supersonic case and a transonic case. They will be examined on the following criteria: consistency of the error estimate, convergence of the error estimate, and the mesh quality itself. The consistency of the error estimate means that as the error estimate is calculated

on successively finer meshes, not only does the magnitude of the error estimate shrink, but the cells with the highest error estimates are refined; i.e. that the highest magnitude error estimates are targeted first. This is the most important factor in our analysis, as we expect to see oscillations or lack of convergence in the objective due to the averaging windows used in this analysis. This method computes the error due to the spatial discretization, not due to the averaging window and oscillatory behavior of the function. There are two caveats to keep mind for the results shown here as the adjoint correction formulation assumes smoothness of the fixed-point iteration and use of the fine mesh adjoint vector. First, the functional correction is computed using a biquadratic interpolation of the coarse mesh adjoint, this was done to decrease expense, but it also decreases accuracy of the functional correction, which becomes small rapidly with refinement. Second, these algorithms make heavy use of gradient reconstruction, which becomes highly inaccurate on stretched meshes [67].

7.4.1 Detached Bow Shock Error Estimation

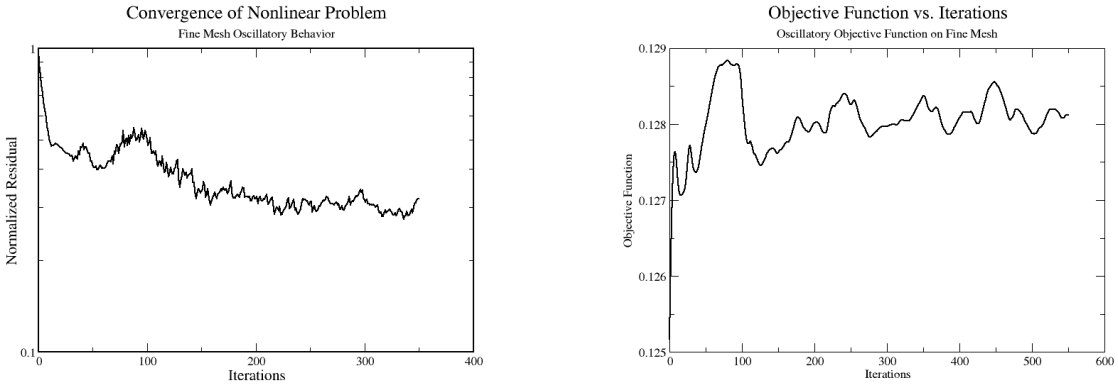
The supersonic error estimation case is similar to the supersonic design case, it is a NACA0012 airfoil in $M = 1.25$ flow, but with no angle of incidence. This case shows small scale unsteadiness due to lack of convergence due to the limiter behavior. The initial mesh is very coarse as shown in Figure 7.5, and does not resolve the flow features well. On the coarse mesh the detached bow shock is not well resolved and the trailing edge fishtail shock that will appear as the mesh refines is not present except as a general smeared high Mach region near the trailing edge. The mesh was then refined by the three different algorithms for pseudo-temporal error estimation presented previously: virtual mesh error estimate, embedded mesh error estimate, and the embedded mesh error estimate with functional correction, with the results of these refinements presented in the following sections. The output of interest for these cases is a sum of lift, drag, and entropy, $J = c_L + c_D + s$. The choice of entropy is used to illustrate the fineness of the refinement of the shock as it passes to the edges of the domain as the mesh is refined. For this case, the objective function was calculated at the final converged state for the first five meshes, at which point the mesh was fine enough that simulations would no longer converge and the objective became the pseudo-time averaged objective function calculated over the final 75 iterations. The mesh refinement was held isotropic for the first eleven meshes, at which point the mesh refinement used the Mach gradient to determine anisotropy, and the flow reconstruction correction with partial backwards in time integration to reduce cost. The behavior on the fine mesh in Figure 7.6 shows a very ill-converged flow with an oscillatory objective function that looks to have oscillations with an amplitude of approximately 2% of the average value.



(a) Coarse Mesh Density

(b) Coarse Mesh Mach Number

Figure 7.5: Coarse mesh for detached bow shock error estimation case



(a) Fine Mesh Convergence

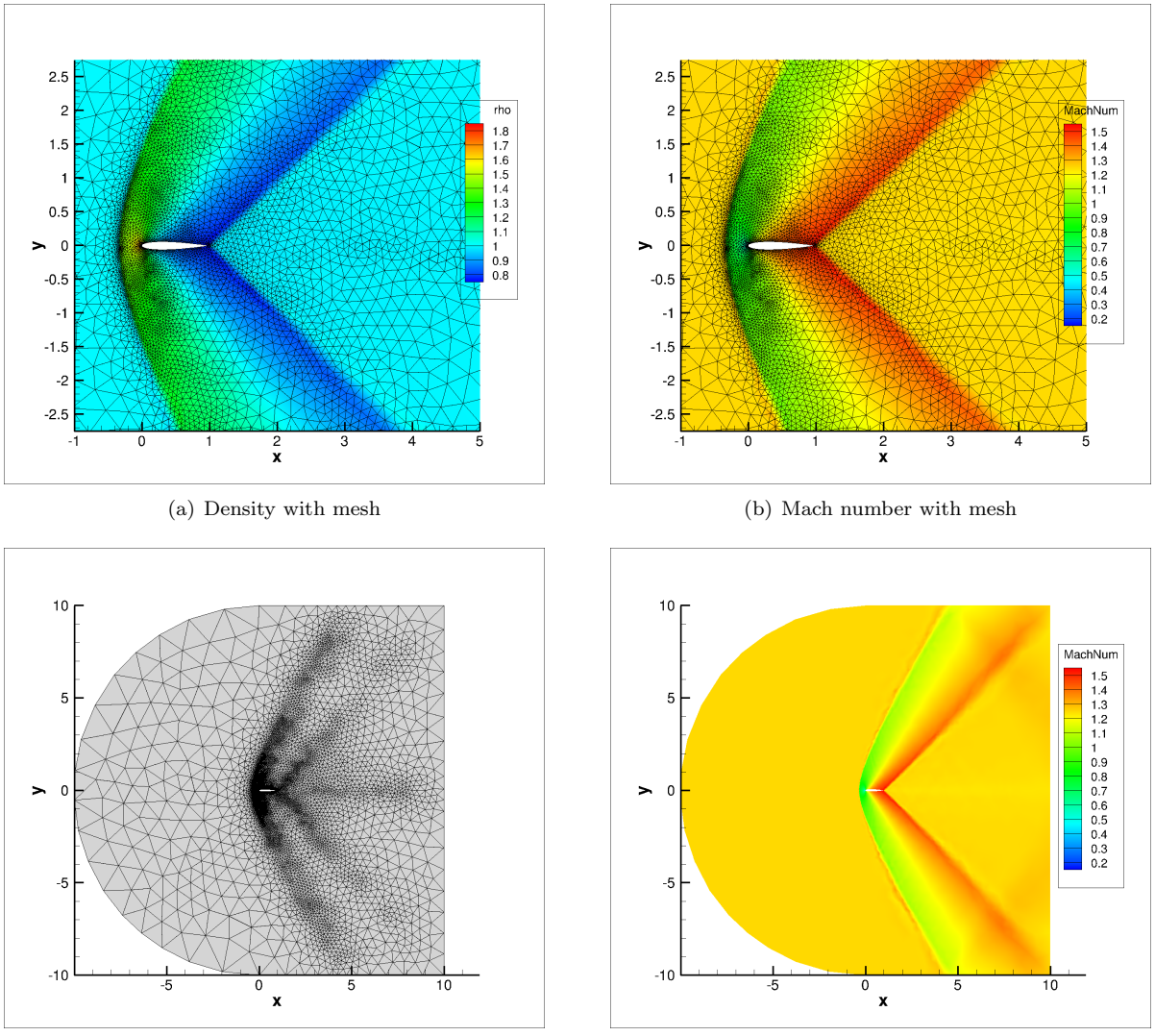
(b) Fine Mesh Objective

Figure 7.6: Objective and convergence behavior on a fine mesh

7.4.1.1 Virtual Mesh Error Estimation

The mesh from the tenth adaptation cycle shows good behavior in the refinement pattern as shown in Figure 7.7. The detached bow shock is resolved well and is carried all the way to the outer boundary, where it reflects off the boundary and crosses through the refined fishtail shock coming off the trailing edge. The reflection happens because the characteristic boundary condition used at the outer boundary is not a non-reflecting boundary condition. There also looks to be the beginning of a refinement of a line of increased entropy coming off the trailing edge of the airfoil. This is an effective and efficient refinement pattern, as shown by the heavy coarsening in front of the bow shock.

Figure 7.8 shows the final adapted anisotropic mesh. The final mesh shows a great deal of refinement



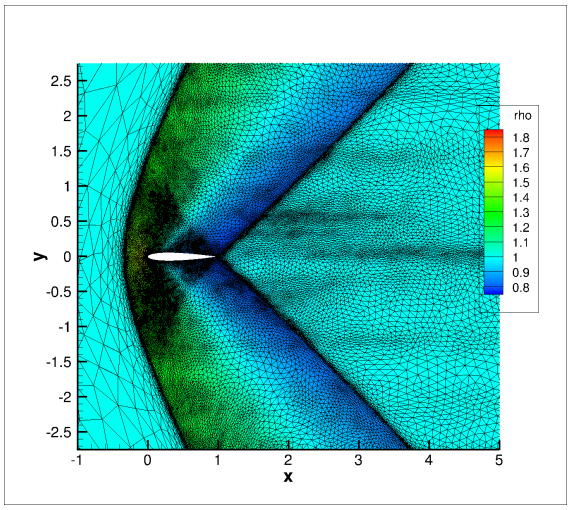
(a) Density with mesh

(b) Mach number with mesh

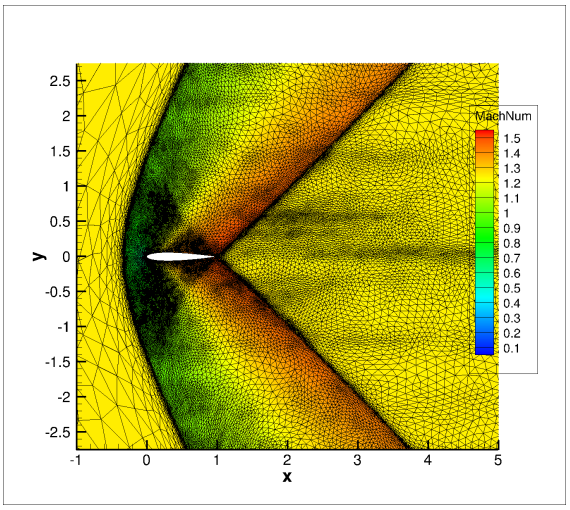
(c) Mesh in full domain

(d) Mach number in full domain

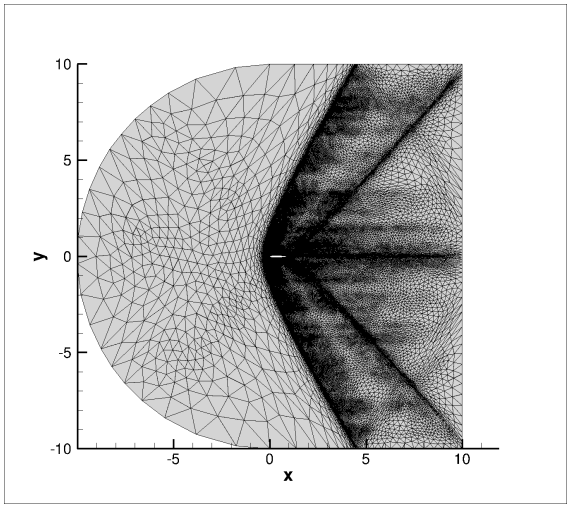
Figure 7.7: Tenth adaptation cycle for detached bow shock with error estimation (final isotropic mesh)



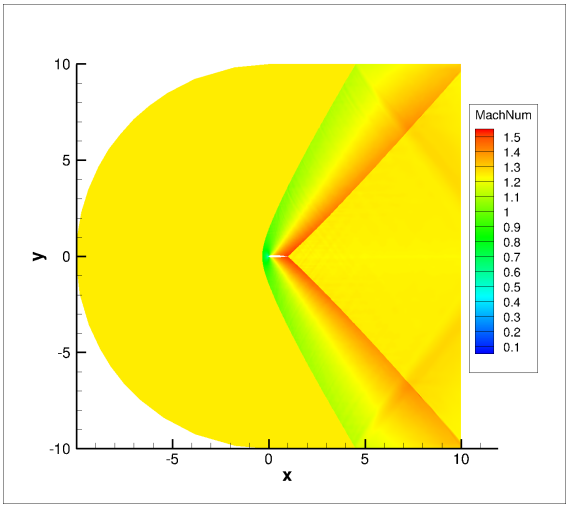
(a) Density with mesh



(b) Mach number with mesh



(c) Mesh in full domain

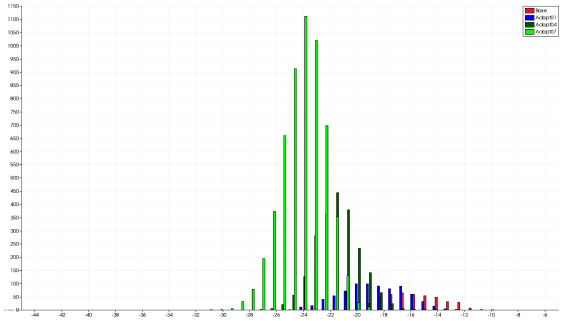


(d) Mach number in full domain

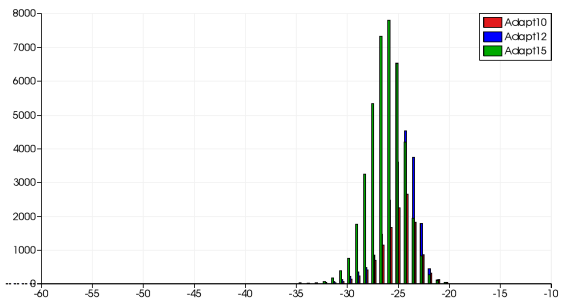
Figure 7.8: 18th and final adaptation cycle for detached bow shock with error estimation

along the bow shock and the reflection off the boundary, the fishtail shock, and the lines of increased entropy coming off the shocks. It also shows a high degree of refinement coming off the trailing edge forward to an intersection with the bow shock, enveloping the airfoil in an area of high refinement. The line of increased entropy has seen increased refinement as well. The Mach number plot shows the high degree of detail for the physical flow features, the sharpness of the shocks is specifically notable and occurs because by including entropy in the objective function the error estimation criteria drives the AMR to capture the entropy behind the shocks.

Having investigated the behavior of the mesh refinement, the next concern is the consistency of the error estimate. For the purposes of this analysis, this is the most important thing, as we should see consistency from mesh to mesh to have an accurate error estimate. It is expected that there will be oscillations in the

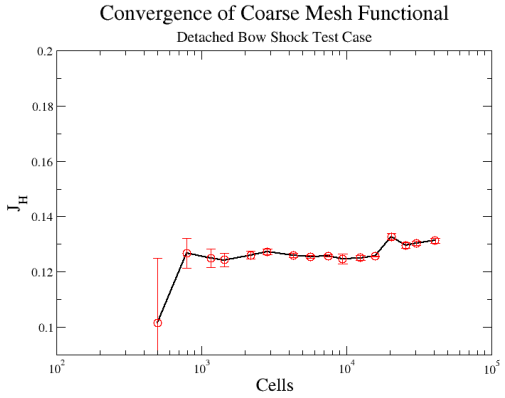


(a) Baseline to seventh adaptation cycle

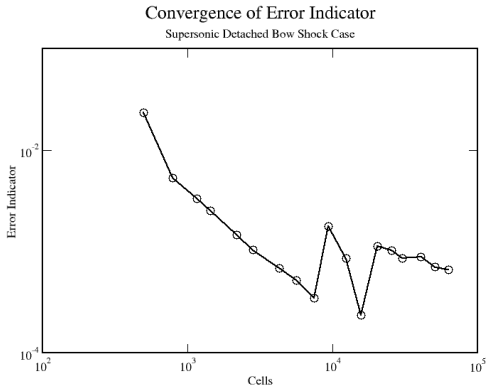


(b) Tenth to 15th adaptation cycle

Figure 7.9: Error histograms for detached bow shock case



(a) Coarse mesh functional with error bars

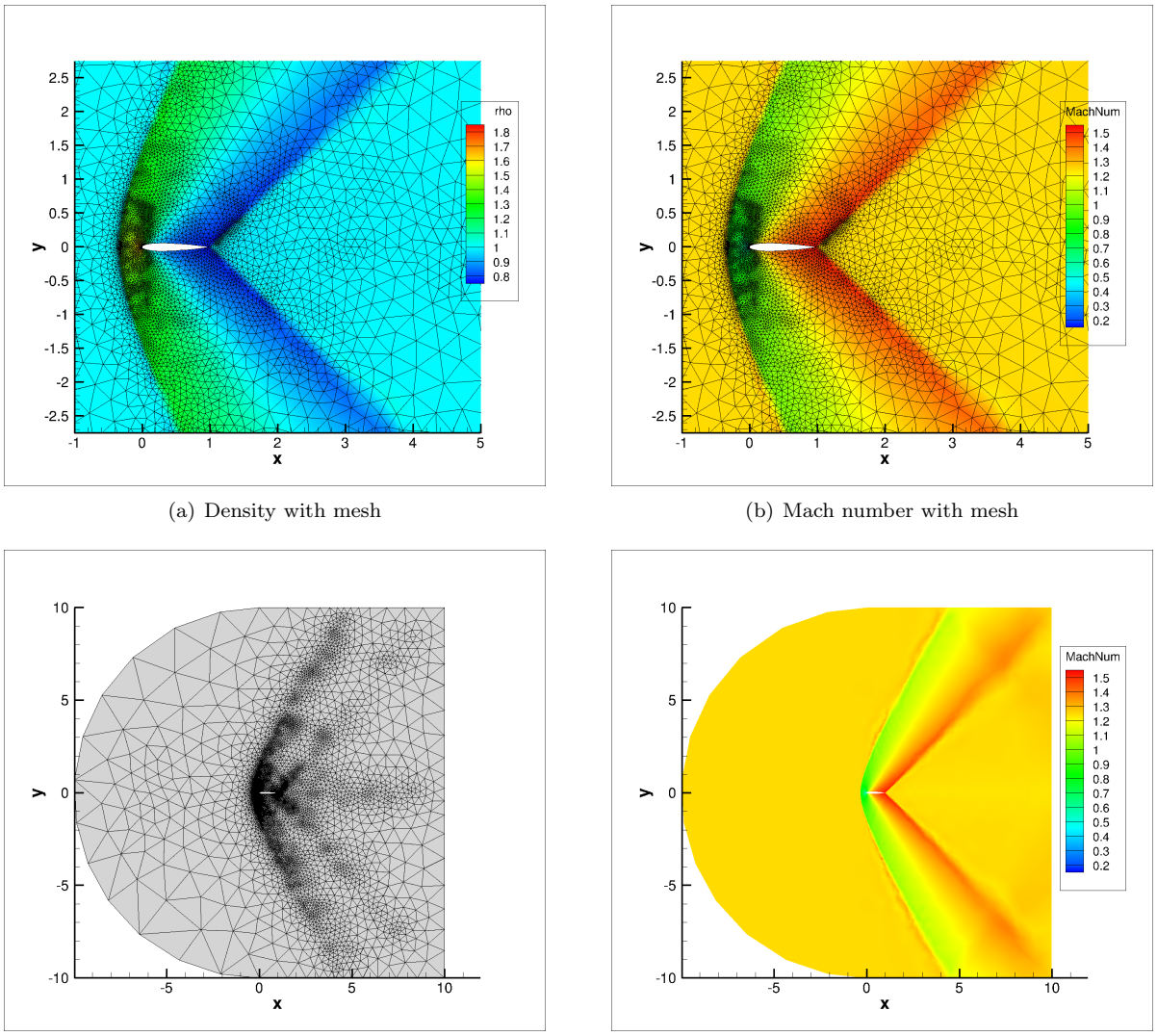


(b) Error convergence

Figure 7.10: Functional and error estimate convergence for supersonic detached bow shock

functional output from mesh to mesh as the output is oscillatory on any given mesh, as such the consistency of the error estimate is our primary concern. The histograms of the error estimates shown in Figure 7.9 show the cells sorted into bins according to the $log_2(\eta_H)$, where the y-axis is the number of elements in a given bin. The expected behavior is that the highest error elements are consistently refined and that the mean steadily moves to the left – signifying decreasing error. The histogram showing the early adaptation behavior shows good consistency, whereas the histogram with the finer mesh error values shows less consistency, indicating a possible issue with this approach.

Figure 7.10 shows the convergence of the functional with red error bars in the left plot and the logarithmic behavior of the error indicator on the right. The functional convergence looks to converge within a bound acceptable for an oscillatory function like the ones examined in this work. The error convergence is well behaved until the finer meshes, where as was shown in the histograms, consistency and uniform decrease of error is lost.



(a) Density with mesh

(b) Mach number with mesh

(c) Mesh in full domain

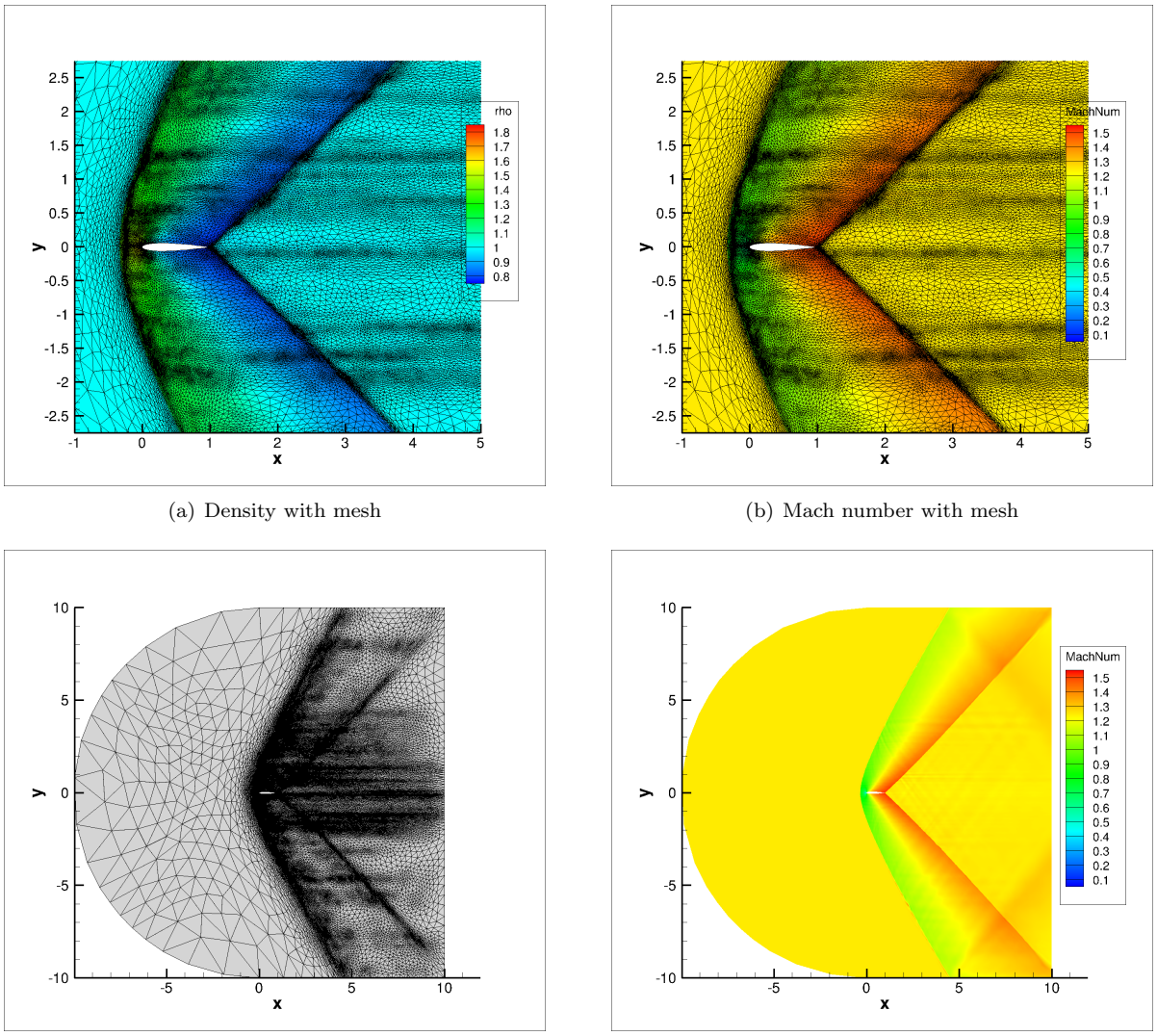
(d) Mach number in full domain

Figure 7.11: Tenth adaptation cycle for detached bow shock with error estimation (final isotropic mesh)

7.4.1.2 Embedded Mesh Error Estimation

Figure 7.11 shows the tenth adapted mesh, the last isotropic mesh, for the embedded mesh error estimation technique. This shows, as in the virtual mesh error estimation technique, refinement along the detached bow shock, the fish tail shock, and the area of increased entropy behind the trailing edge of the airfoil. The mesh is refined well enough along both shocks to carry the bow shock and the fishtail shock all the way to the boundary and captures the reflecting shock interaction –even though it is fainter than on the final mesh. This case shows more refinement near the airfoil leading edge than that shown in Figure 7.11 for the virtual mesh refinement.

Figure 7.12 shows the final adapted anisotropic mesh; both shocks are more heavily refined and the



(a) Density with mesh

(b) Mach number with mesh

(c) Mesh in full domain

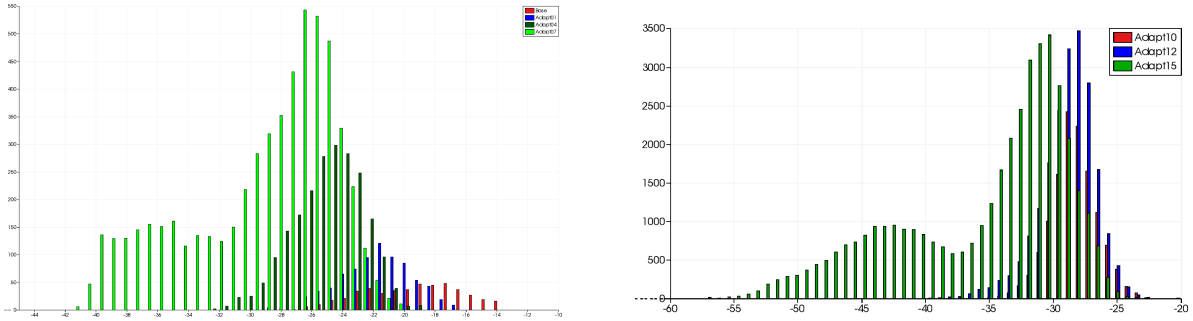
(d) Mach number in full domain

Figure 7.12: 18th and final adaptation cycle for detached bow shock with error estimation

shock reflection and interaction is well captured. The mesh is heavily coarsened in front of the bow shock, and tightly refined all along areas of increased entropy due to the shock interactions and reflections. The lines of increased refinement post fishtail shock correspond well to the areas of slightly higher Mach number due to the shock reflections and entropy creation due to the flux function. The high level of refinement along these lines of increased entropy points to the importance of the behavior of the flux functions on the embedded mesh when it comes to entropy creation, as these lines were nearly absent for the virtual mesh refinement.

Figure 7.13 shows the histograms for the same refinement cycles as in Figure 7.9, but using the embedded mesh error estimate. In this case, the error estimates show the expected consistency even on the fine meshes.

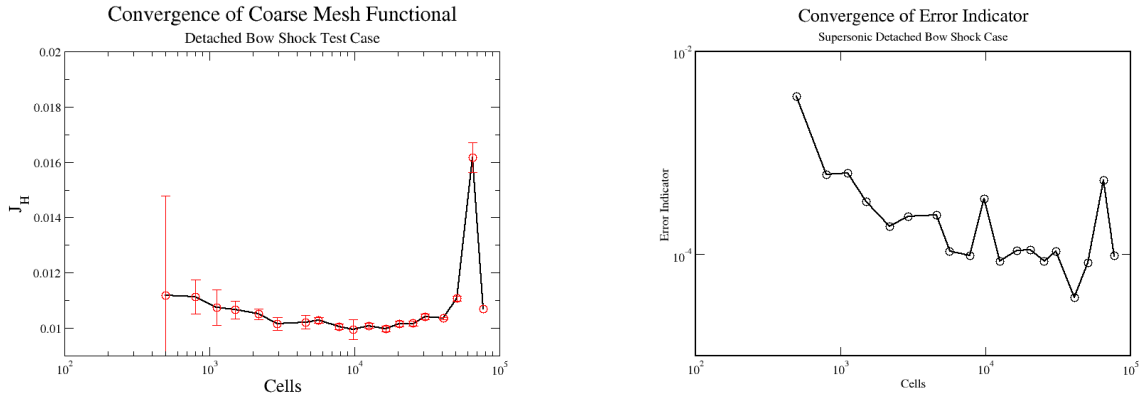
Figure 7.14 shows the expected functional convergence with the exception of the penultimate mesh.



(a) Baseline to seventh adaptation cycle

(b) Tenth to 15th adaptation cycle

Figure 7.13: Error histograms for detached bow shock case



(a) Coarse mesh functional with error bars

(b) Error convergence

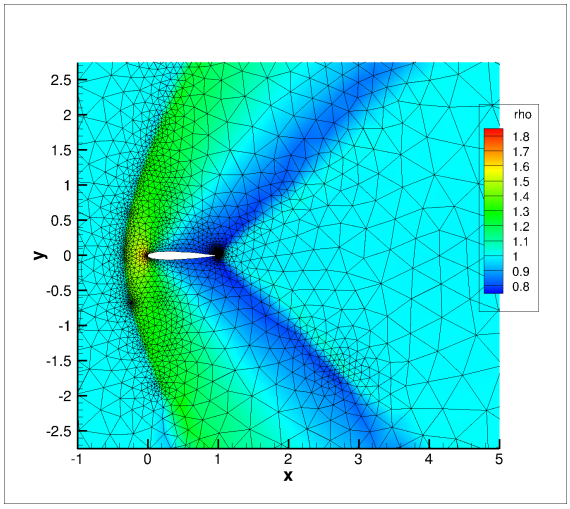
Figure 7.14: Functional and error estimate convergence for supersonic detached bow shock

Similarly, the error convergence decreases and then stagnates, with the exception being a spike at that same mesh. Examining that mesh shows that the issue is due to the stagnation of the nonlinear problem at a non-physical flow state markedly different from that on every other mesh, and the error estimate increases significantly. If the fine mesh pseudo-temporal adjoint were utilized then the adjoint based error estimate would be more likely to pick up such errors, but the biquadratic reconstruction of the adjoint is only a good approximation for smooth flows; additionally the adjoint-based functional correction assumes smoothness of the fixed-point itself

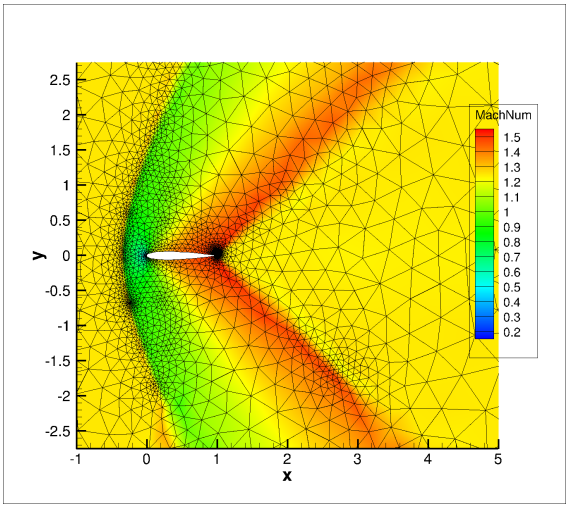
7.4.1.3 Embedded Mesh Error Estimation and Functional Correction

This case has similar behavior as that exhibited by the previous two methods, carrying the shocks to the domain boundaries and capturing the shock reflection, as captured in Figure 7.15. This method showed less refinement along the leading edge and the high entropy line coming off the trailing edge. This can be attributed to the use of the functional correction.

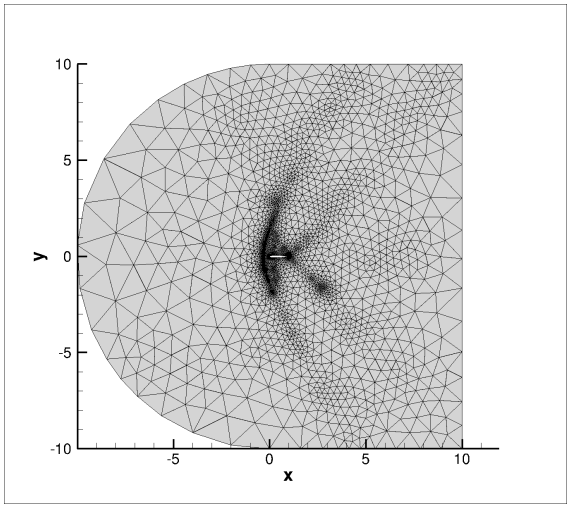
The final adapted mesh in Figure 7.16 shows high refinement along the shocks and the shock reflection



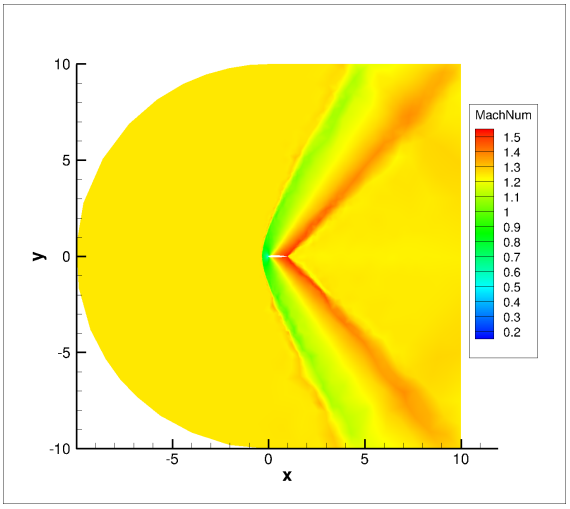
(a) Density with mesh



(b) Mach number with mesh

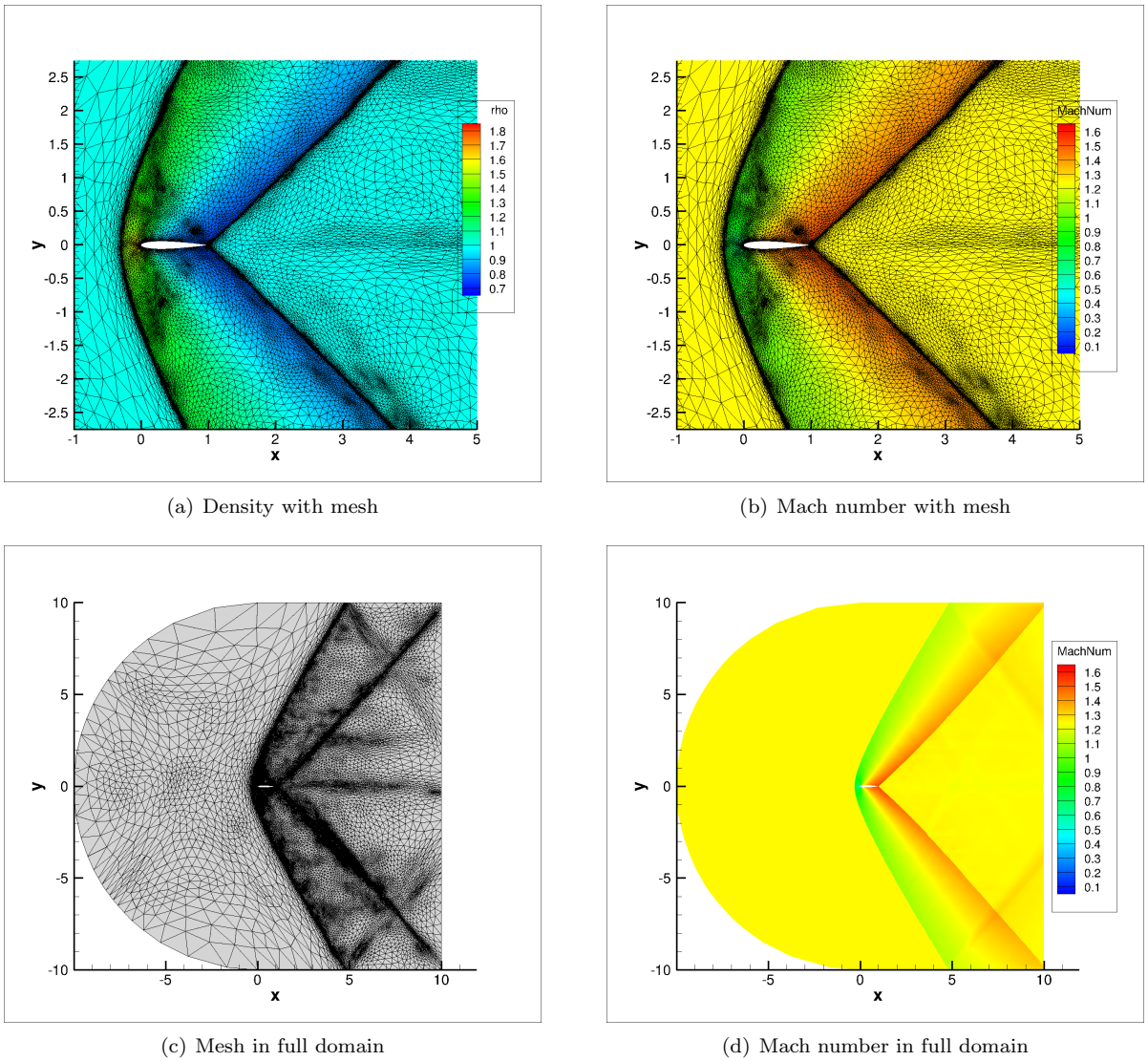


(c) Mesh in full domain



(d) Mach number in full domain

Figure 7.15: Tenth adaptation cycle for detached bow shock with error estimation and functional correction



(a) Density with mesh

(b) Mach number with mesh

(c) Mesh in full domain

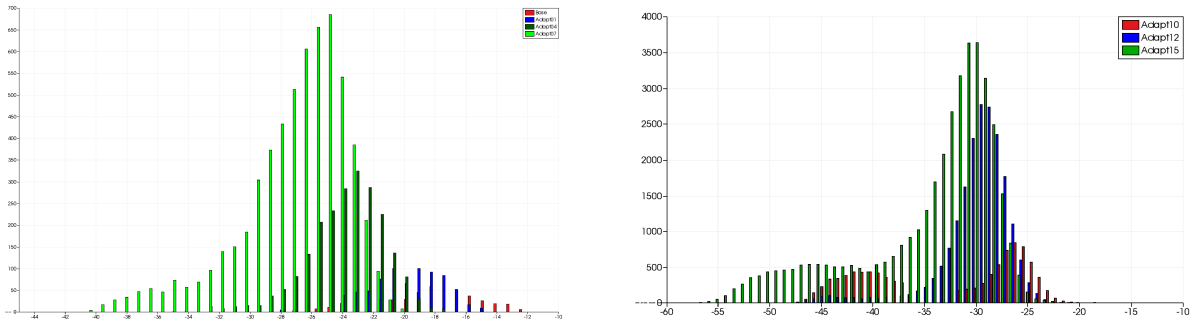
(d) Mach number in full domain

Figure 7.16: 18th and final adaptation cycle for detached bow shock with error estimation and functional correction

as well as some refinement from the trailing edge forward to the detached bow shock. There is also refinement along the high entropy line coming off the trailing edge, but there is not a high degree of refinement in other places due to these areas of higher entropy. This again can be attributed to the functional correction.

Figure 7.17 shows consistency in the error estimate as desired with distinct peaks for each mesh, moving towards the left with decreasing error.

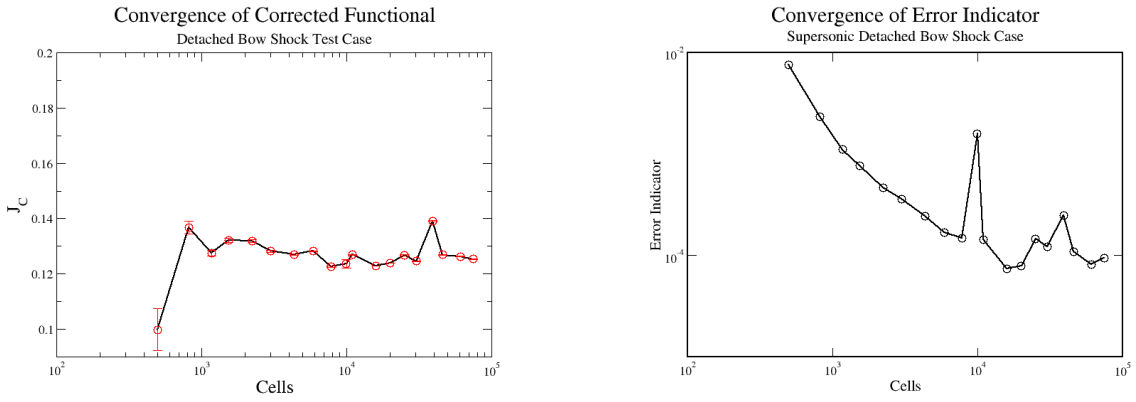
Figure 7.18 shows the expected functional and error convergence, with stagnation of the error at approximately $1e-4$. The one outlier is the behavior at the 15th adapted mesh; this can again be attributed to pathological behavior in the flow field, reflected by an outlier in the objective function and a spike in the error.



(a) Baseline mesh to seventh adaptation cycle

(b) 10th to 15th adaptation cycle

Figure 7.17: Error histograms for detached bow shock case with functional correction



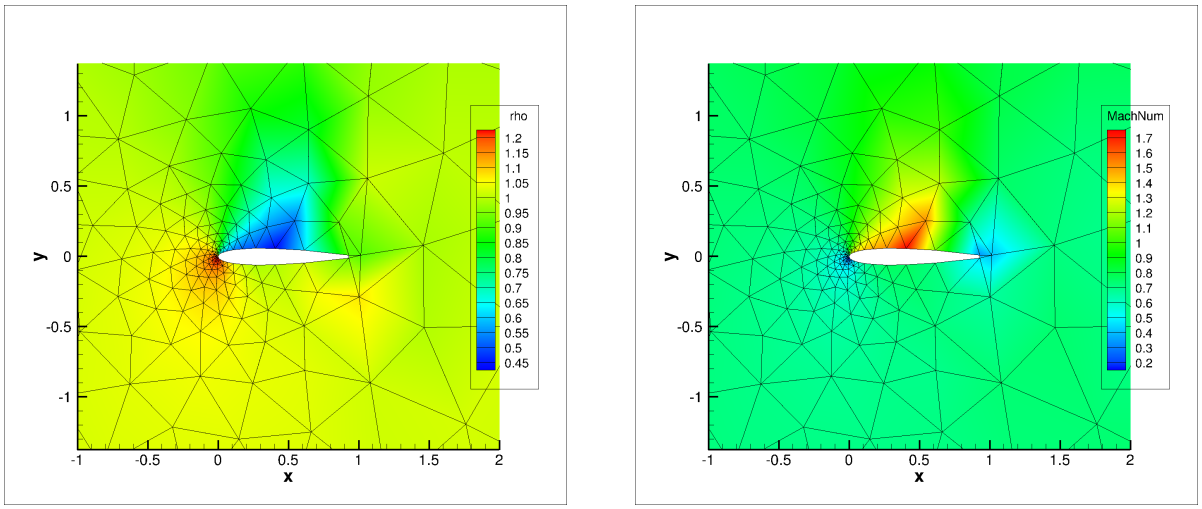
(a) Corrected functional with error bars

(b) Error convergence

Figure 7.18: Corrected functional and error estimate convergence for detached bow shock case

7.4.2 Transonic Airfoil With Blunt Trailing Edge Error Estimation

Having shown good behavior on a case with very small trailing edge unsteadiness, it is necessary to look at a case with stronger unsteadiness. This transonic error estimation case is similar to the transonic design case; it is a NACA0012 airfoil truncated at 93% of the chord in $M = .75, \alpha = 5^o$ flow. This case shows small scale unsteadiness due to lack of convergence at the trailing edge, where the flow enters noticeable limit cycle oscillations. The initial mesh is very coarse as shown in Figure 7.19, and does not resolve the flow features well, including allowing the flow to converge. On the coarse mesh the upper surface shock is not well resolved and the trailing edge unsteadiness is not present and the flow converges. It was then refined by the same three algorithms as in the detached bow shock simulation. The output of interest for these cases is a sum of lift and drag, $J = c_L + c_D$. The first 6 meshes are refined isotropically, at which point anisotropic refinement is invoked.

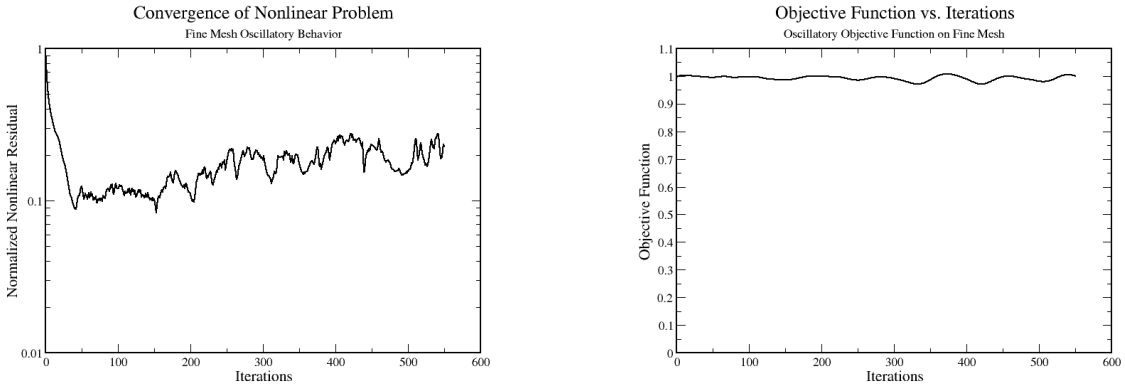


(a) Coarse Mesh Density

(b) Coarse Mesh Mach Number

Figure 7.19: Coarse mesh for transonic blunt trailing edge error estimation case

The behavior on the fine mesh shows a very ill-converged flow with an oscillatory objective function that looks to have oscillations with an amplitude of approximately 5% of the average value. This shows much more oscillatory behavior than the detached bow shock case, which can be attributed to the blunt trailing edge. The functional behavior with the refined meshes reflect this oscillatory behavior.



(a) Fine Mesh Convergence

(b) Fine Mesh Objective

Figure 7.20: Objective and convergence behavior on a fine mesh

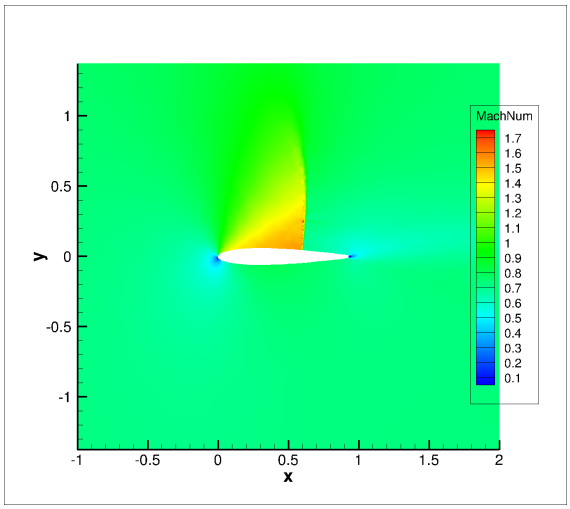
7.4.2.1 Virtual Mesh Error Estimation

The sixth adapted mesh is the final isotropic mesh and shows noticeable refinement along the shock and the streamline, as is expected. The flow variables show noticeable unsteadiness at the trailing edge as can be seen in Figure 7.21.

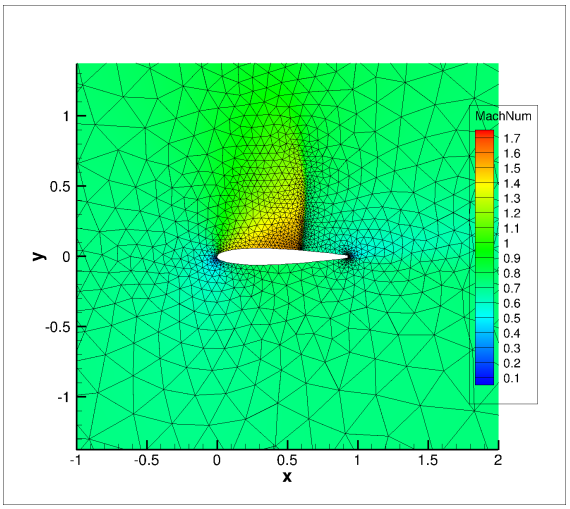
The final mesh and flow shown in Figure 7.22 shows poor behavior. The mesh shows minimal refinement along the shock which will poorly capture the drag and lift. The streamline Mach number plot shows very strong unsteadiness which is expected in this case.

Figure 7.23 shows poor error estimate consistency on the coarse meshes, whereas before the virtual mesh error estimate was consistent on the coarse meshes but lost consistency as the meshes got finer. Here, the finer meshes show almost no consistency in the virtual mesh error estimate, and consistency of the error estimate is lost early on.

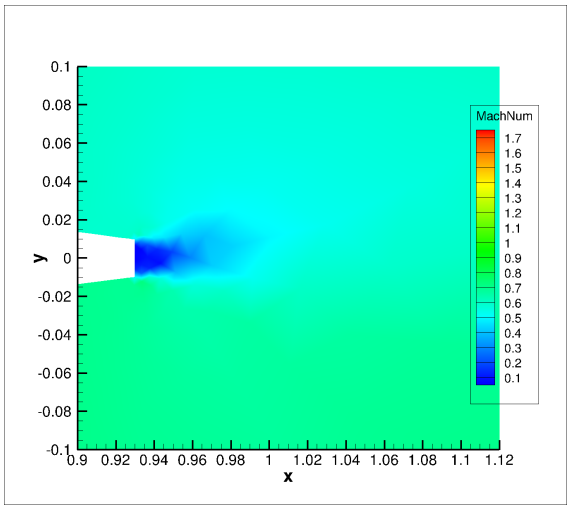
Figure 7.24 explains the loss of consistency; the error estimate spikes on later meshes. The reason for this has to do with calculating the residual on the virtual fine mesh. The virtual fine mesh appears to have highly inaccurate residual computation for the flow states when the flow expands around the trailing edge. The lack of limiters on the virtual fine mesh combined with the anisotropy of the mesh leads to unphysical or nearly unphysical states with very large residuals, which destroys the ability of this method to provide reliable or consistent error estimates. Investment in better interpolation and gradient reconstruction techniques could solve this problem. It is also hypothesized that a move to FEM codes, which do not require interpolation onto the embedded mesh and have better mechanics for dealing with high p-order residual evaluations, might not suffer from this problem.



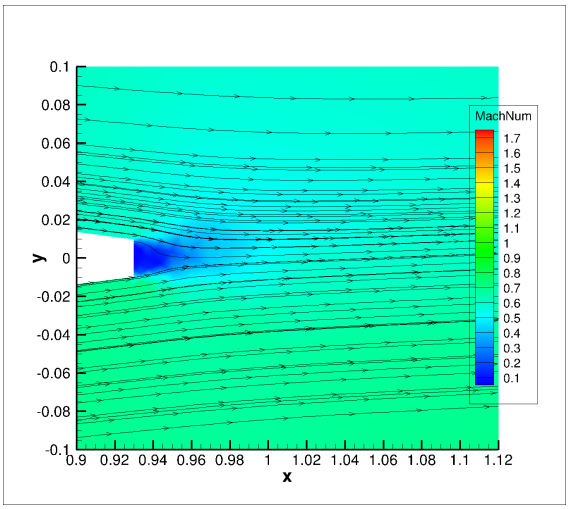
(a) Mach number without mesh



(b) Mach number with mesh

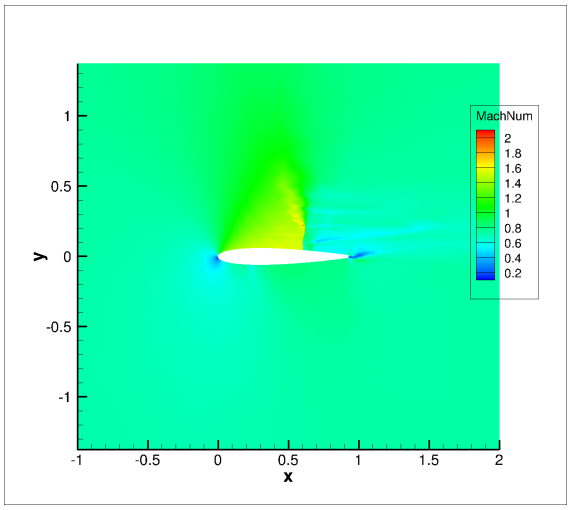


(c) Mach number at trailing edge

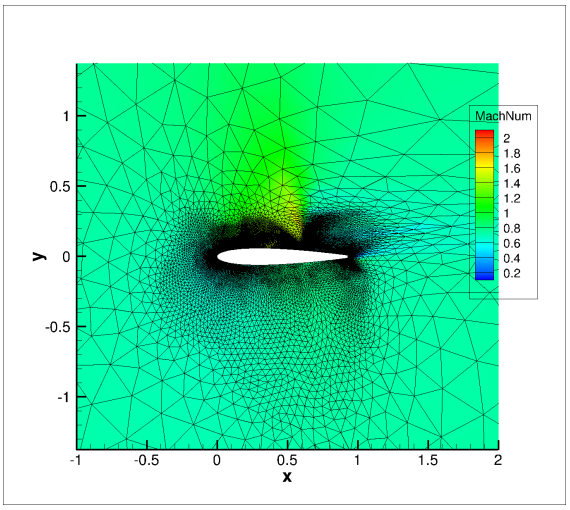


(d) Mach number with streamlines

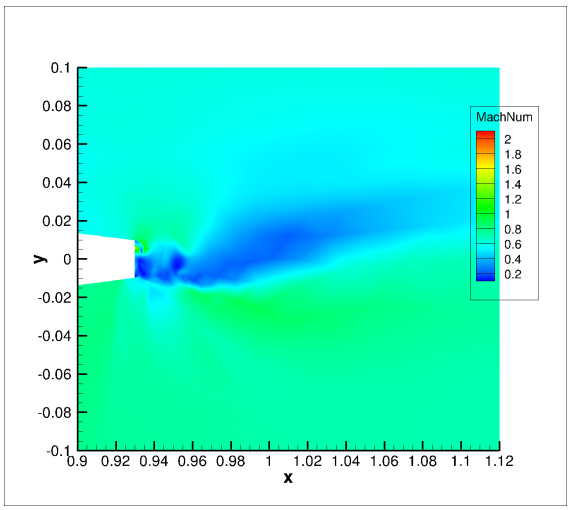
Figure 7.21: Sixth adaptation cycle for transonic blunt trailing edge with error estimation (final isotropic mesh)



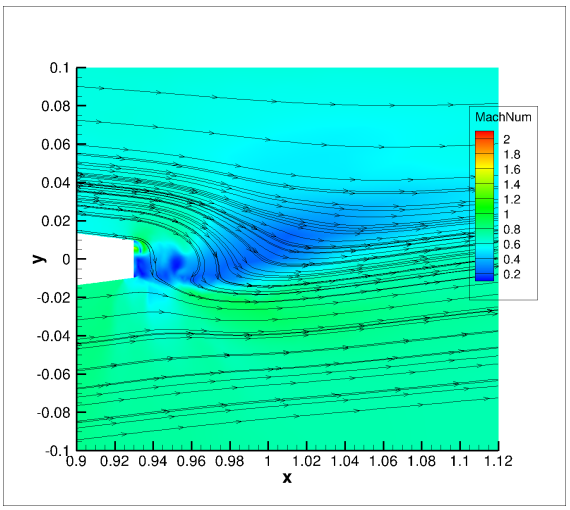
(a) Mach number without mesh



(b) Mach number with mesh

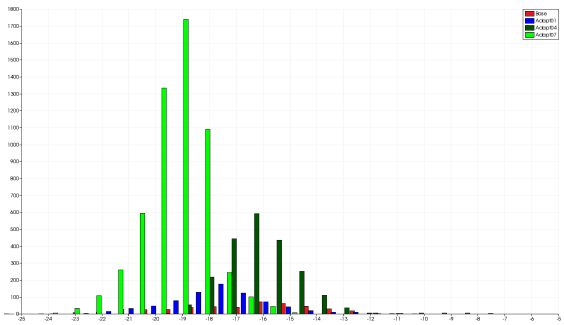


(c) Mach number at trailing edge

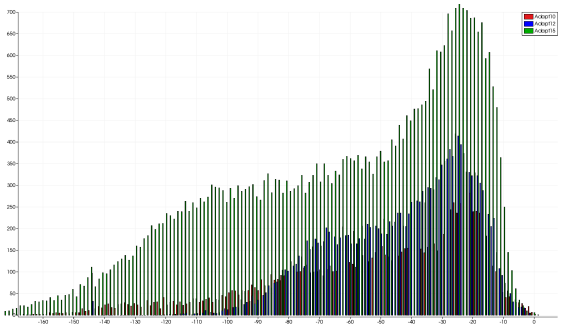


(d) Mach number with streamlines

Figure 7.22: 16th and final adaptation cycle for transonic blunt trailing edge with error estimation

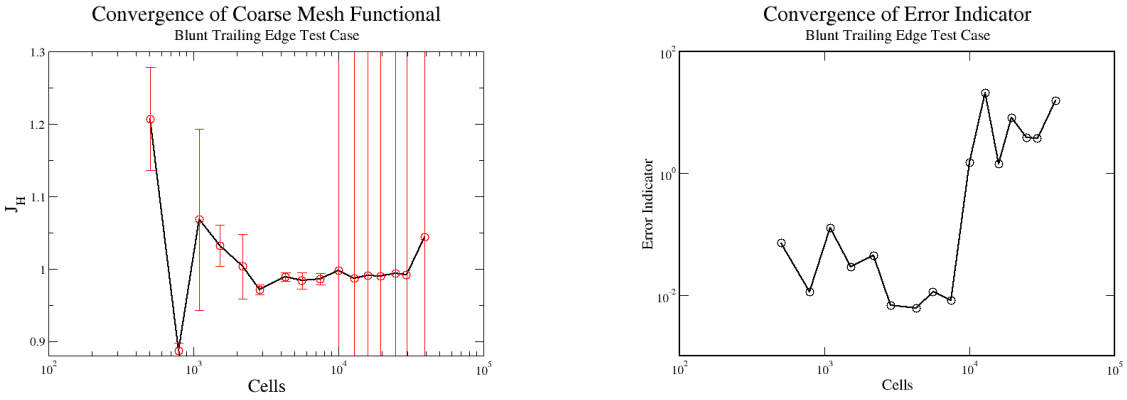


(a) Baseline to 7th adaptation cycle



(b) Tenth to 15th adaptation cycle

Figure 7.23: Error histograms for transonic blunt trailing edge case



(a) Coarse mesh functional with error bars

(b) Error convergence

Figure 7.24: Functional and error estimate convergence for transonic blunt trailing edge

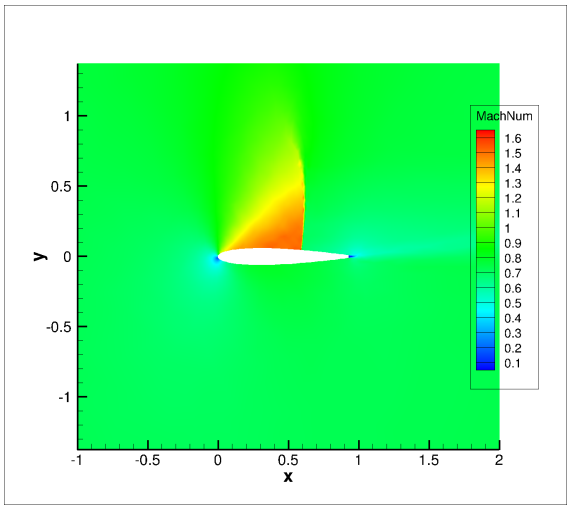
7.4.2.2 Embedded Mesh Error Estimation

As in the virtual mesh adaptation the sixth mesh, the final isotropic mesh, shows good refinement patterns which accord with expected results as shown in Figure 7.25. The mesh is adapted along the trailing edge unsteadiness, the leading edge and along the shock, and the flow also shows noticeable unsteadiness as expected.

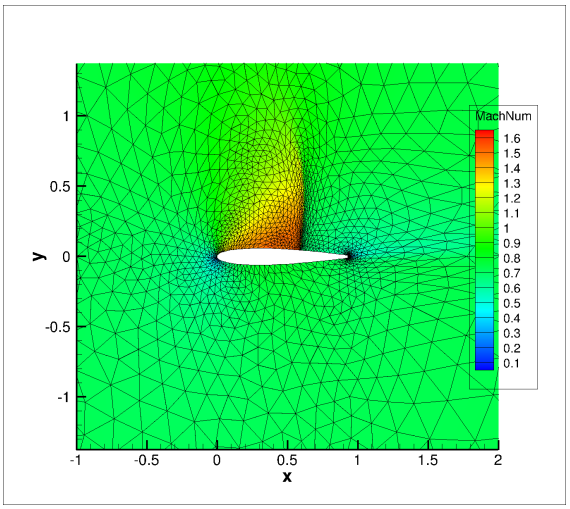
The final adapted mesh is refined well along the lower portion of the shock as well as showing good refinement along the forward streamline portion of the flow and the trailing edge as shown in Figure 7.26. The flow field at the trailing edge shows noticeable unsteadiness, which explains the oscillatory nature of the flow as was expected.

Figure 7.27 shows the histograms of the error, and it shows good consistency with the exception of the seventh adapted mesh; the seventh adapted mesh appears to have higher mean error than the baseline mesh. The error distribution on the finer meshes regains consistency as shown in the finer mesh histogram in Figure 7.27b.

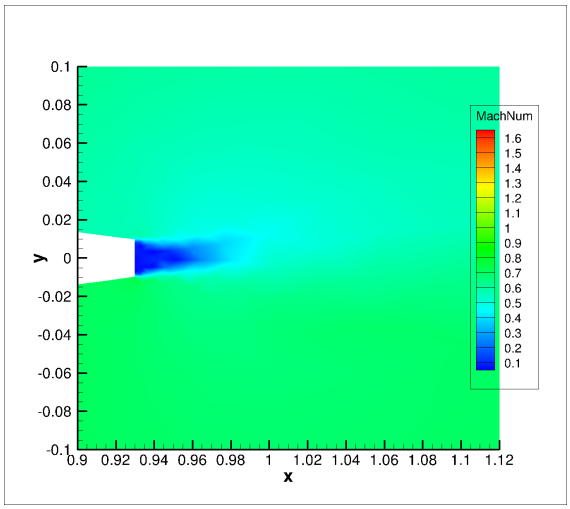
Figure 7.28 explains why the seventh adapted mesh loses the consistency. The seventh and ninth adapted meshes sees spikes in the error for the same reason observed in the virtual mesh error estimation. The functional convergence is reasonable for such an oscillatory case.



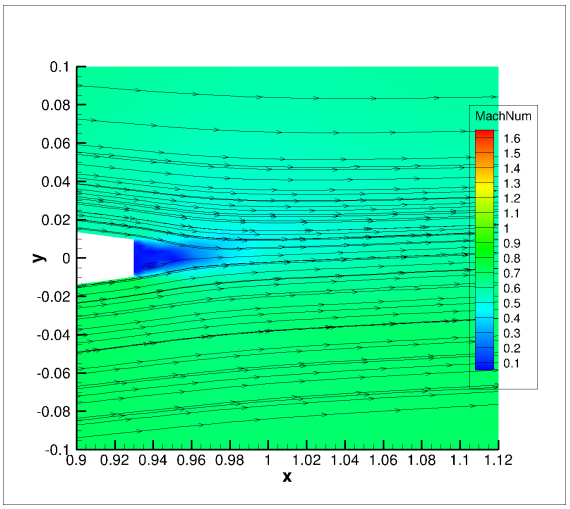
(a) Mach number without mesh



(b) Mach number with mesh

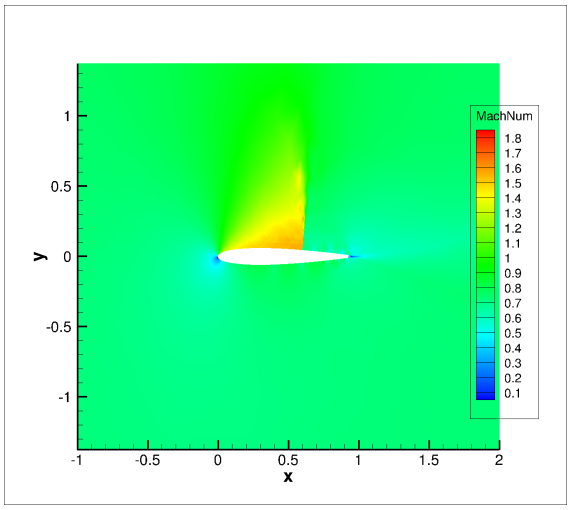


(c) Mach number at trailing edge

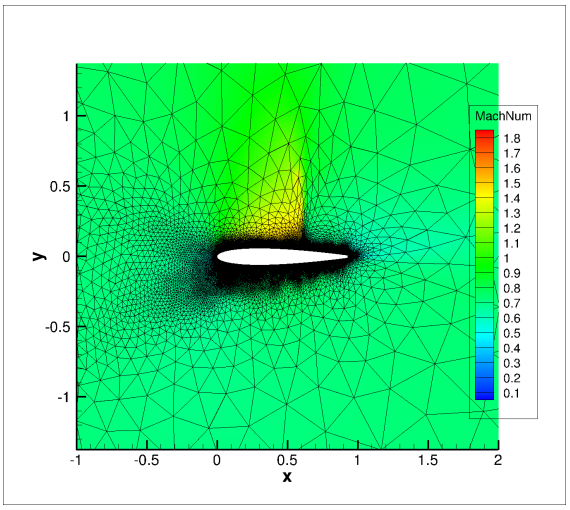


(d) Mach number with streamlines

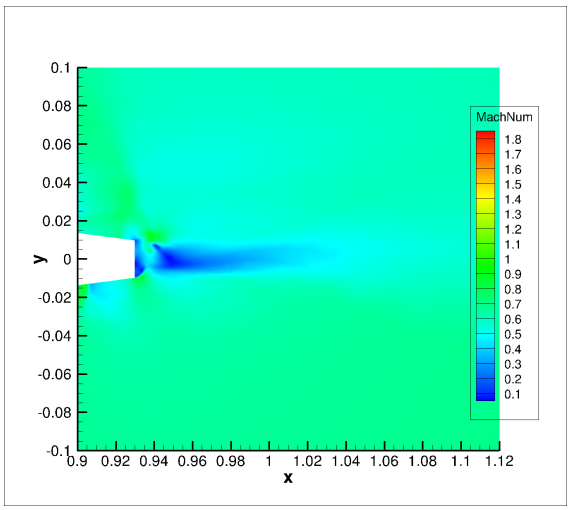
Figure 7.25: Sixth adaptation cycle for transonic blunt trailing edge with error estimation (final isotropic mesh)



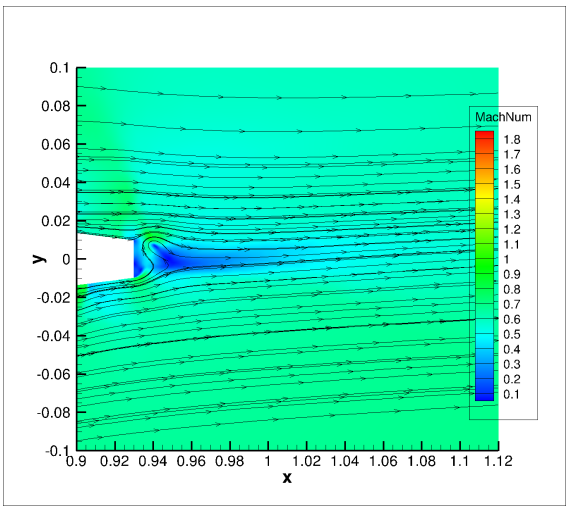
(a) Mach number without mesh



(b) Mach number with mesh

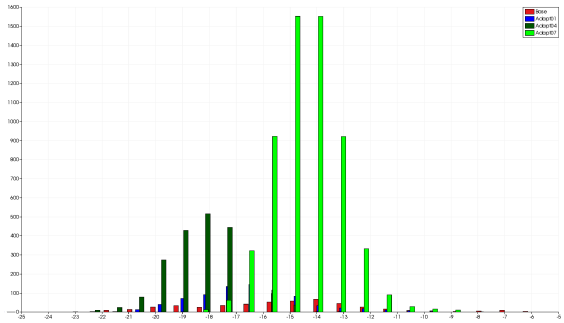


(c) Mach number at trailing edge

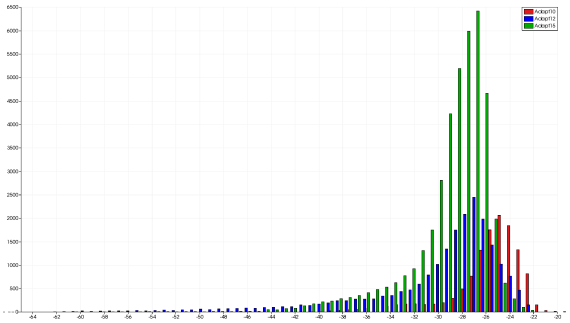


(d) Mach number with streamlines

Figure 7.26: 16th and final adaptation cycle for transonic blunt trailing edge with error estimation

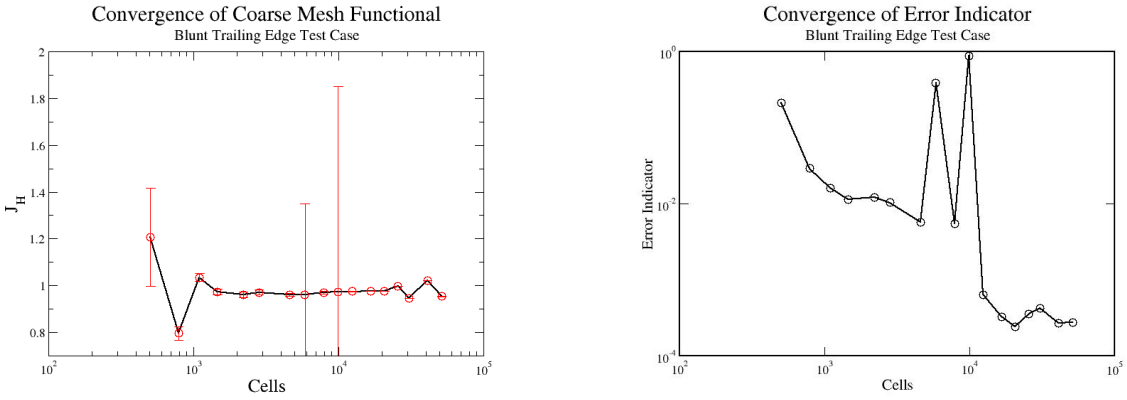


(a) Baseline to 7th adaptation cycle



(b) Tenth to 15th adaptation cycle

Figure 7.27: Error histograms for transonic blunt trailing edge case



(a) Coarse mesh functional with error bars

(b) Error convergence

Figure 7.28: Functional and error estimate convergence for transonic blunt trailing edge

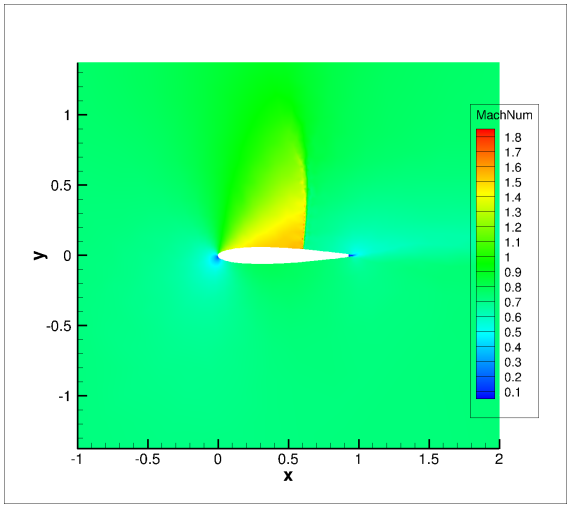
7.4.2.3 Embedded Mesh Error Estimation and Functional Correction

This is the final error estimation case and it shows excellent refinement behavior on the sixth adapted mesh shown in Figure 7.29. The shock is well refined, and the leading and trailing edges show refinement.

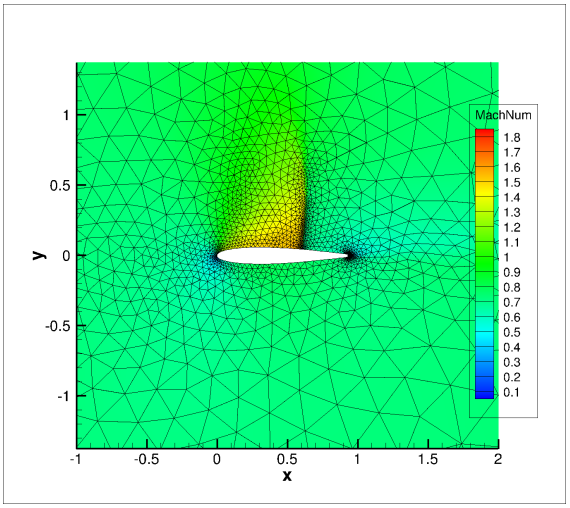
The final adapted mesh shown in Figure 7.30 shows good refinement along the shock as well as along the leading and trailing edge flows. This pattern is very similar to what is seen in steady state adjoint refinement for converged flows. This mesh does show greater refinement along the latter half of the airfoil than is typical due to the impact of the trailing edge unsteadiness along the rear section of the airfoil. In contrast with the previous error estimate, the method with the functional correction does not refine the forward half of the airfoil or the incoming flow region as heavily. This could be because the functional correction can calculate the error in that smooth flow region, and so the refinement gets focused more heavily on the shock and the unsteadiness at the trailing edge which are less smooth. The Mach number plots show the high degree of unsteadiness at the trailing edge, as is expected.

The error histograms shown in 7.31 show good consistency of the error estimate with the mean error steadily decreasing even on the finer meshes.

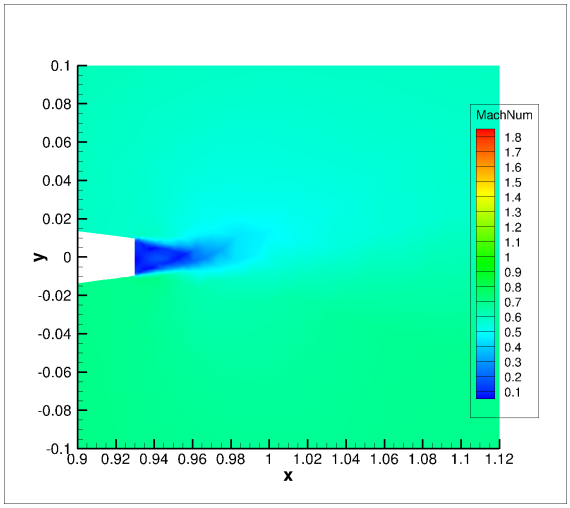
Figure 7.32 shows mostly the expected behavior for an oscillatory flow such as this, with mostly good behavior in the corrected functional and convergence in the error estimate until stagnation. The third adapted mesh has a very large error estimate and its corrected functional is a large outlier – this is due to the interpolation issues highlighted previously. While the interpolation becomes less important for the embedded mesh methods as they use limiters on the fine mesh, it can still lead to non physical states, especially for the highly non-smooth iterations that occur early on in the solution process in the highly transient domain.



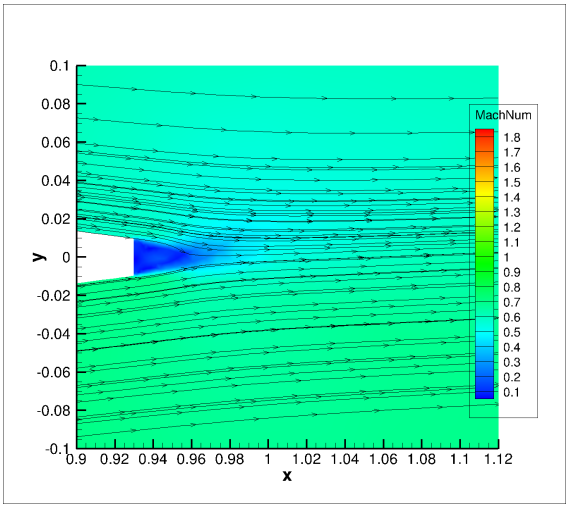
(a) Mach number without mesh



(b) Mach number with mesh

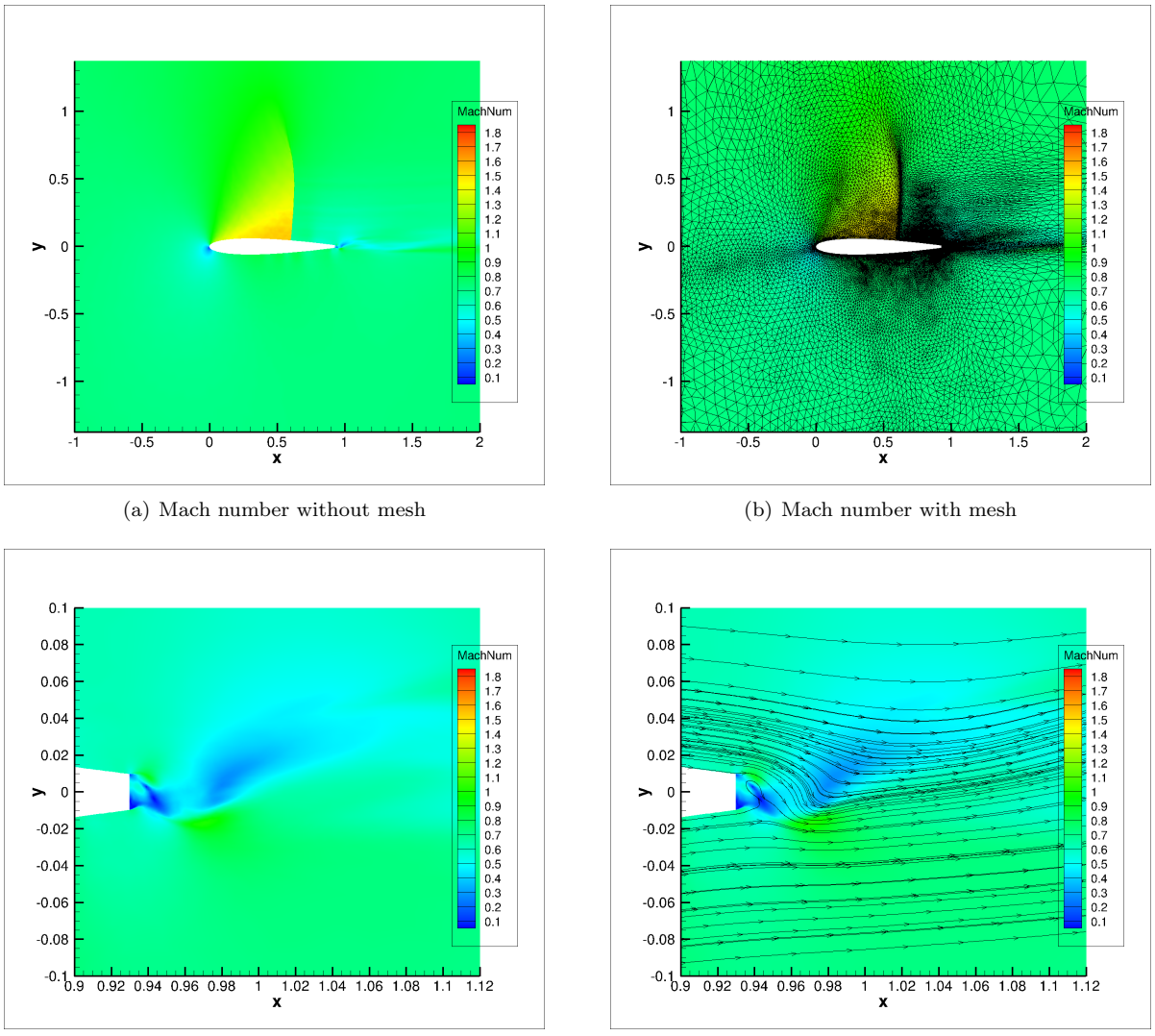


(c) Mach number at trailing edge



(d) Mach number with streamlines

Figure 7.29: Sixth adaptation cycle for transonic blunt trailing edge with error estimation and functional correction (final isotropic adaptation)



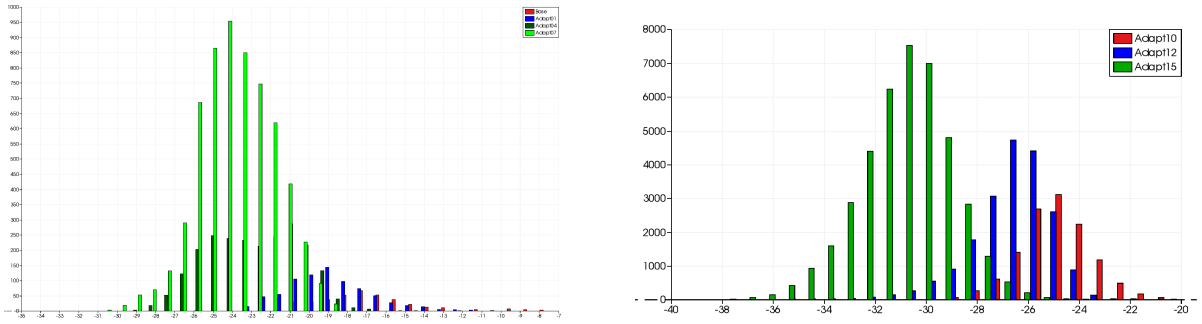
(a) Mach number without mesh

(b) Mach number with mesh

(c) Mach number at trailing edge

(d) Mach number with streamlines

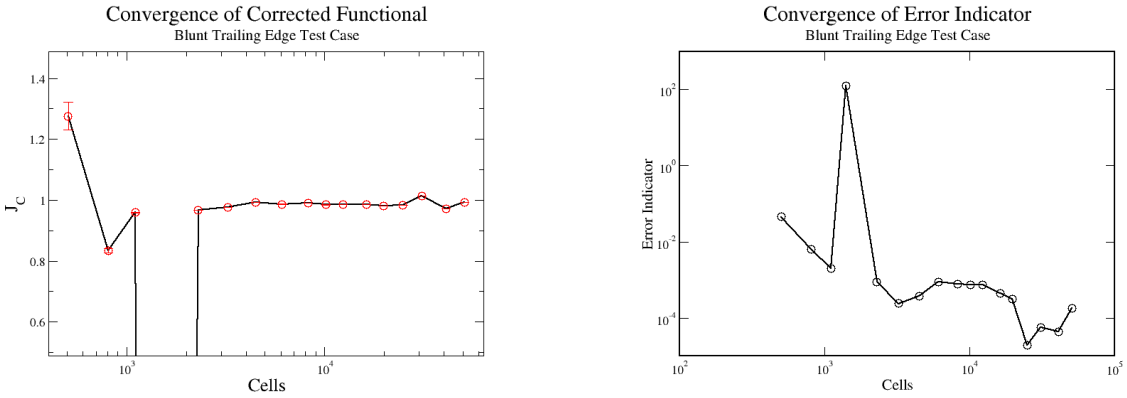
Figure 7.30: 16th and final adaptation cycle for transonic blunt trailing edge with error estimation and functional correction



(a) Baseline mesh to seventh adaptation cycle

(b) Tenth to 17th adaptation cycle

Figure 7.31: Error histograms for transonic blunt trailing edge case with functional correction



(a) Corrected functional with error bars

(b) Error Convergence

Figure 7.32: Corrected functional and error estimate convergence for transonic blunt trailing edge case

7.5 Summary

All three error estimation methods perform well on the detached bow shock with smaller scale unsteadiness. The virtual mesh error estimate loses consistency on the finer meshes when measured by the mean on the error histogram, but the embedded mesh methods maintain consistency even on the finer meshes. All methods show decreasing error and converging objective functions in the presence of small scale oscillations. The virtual mesh method and the functional correction method do best in terms of refinement patterns due to their lack of refinement on the lines of increased entropy. The virtual mesh method does not refine in these places because it cannot detect them, whereas the functional correction method corrects for the error in those smoother areas without having to refine in them.

In the transient blunt trailing edge case, the virtual mesh method, which had previously shown promise is shown to be unworkable. It loses consistency of the error estimate on the coarser meshes and on the fine meshes its high dependence on the interpolation functions leads to issues in the computation of the error estimate as it integrates back in pseudo-time on the anisotropic meshes. The embedded mesh error estimate performs much better –it still has some interpolation related issues with the error estimation, but these show up with less frequency. Additionally, it keeps consistency of the error estimate until stagnation of the convergence of the error estimate. The best results come from the functional correction method, it has only one non-convergent error estimate due to the interpolation and keeps consistency of the error estimate throughout the finer meshes. The final adapted mesh shows nice adaptation patterns of the shock, the incoming flow, and the trailing edge unsteadiness; with the functional correction preventing over-refinement in smooth regions of the flow.

This method is tempting due to the high quality mesh refinement patterns and the consistent error estimates even in nonconvergent cases – showing approximately two to three orders of magnitude reduction

in the error, with consistent error estimates for even the very low error cells with error values of approximately $1e-11$ even for these cases where the residual is not converging more than half an order. However it shows some undesirable pathologies due to the heavy reliance on many different assumptions to limit the expense and the high reliance on gradient reconstruction and interpolation, methods that can be highly inaccurate on anisotropic meshes. In some of the finer meshes used, the inaccuracy of the gradient reconstruction leads to near divergence of the residual operator in some of the less smooth time-steps in the transient dominated portion of these flows. As such, for these meshes the error estimates and functional corrections are very inaccurate and dominated by the pathological residual evaluations. In order for this method to work more consistently in flows with high-scale unsteadiness – such as the blunt trailing edge case – an investment in better interpolation mechanics and gradient reconstruction is necessary, high order limiting techniques may also be useful. Barycentric interpolation methods, which were designed for triangles and unstructured interpolation, could assist with interpolation for these non-smooth flows on highly anisotropic meshes. An interpolation related improvement could be handling of the adjoint reconstruction on the boundaries, for which currently the gradient in those cells is set to zero, and this impacts the nodal aggregation portion of the combined bilinear and biquadratic interpolation. It should be noted that the virtual mesh method will work only for smooth solutions and is limited in that regard and the embedded mesh methods should work better on the non-smooth solutions that are shown and investigated in this work. Alternatively, application of this method to an FEM method, where the error estimate can be performed with a higher-order discretization without having to deal with the effect of gradient reconstruction and other methods that struggle with the anisotropy that naturally arises in adaptive mesh refinement. A final improvement that could generally assist with the utility of this method would be the windowing regularization methods mentioned in the optimization chapter and investigated by Krakos et al. [35] and Schotthofer et al. [63]. These regularization methods have been shown to be effective for optimization methods and could assist with the functional convergence as the mesh is refined.

Chapter 8

Conclusions and Future Work

8.1 Summary

This work was targeted at addressing a long-standing area of concern in the field of CFD regarding the accuracy of tangent and adjoint sensitivities in partially converged flows and their utility in driving optimizations and adaptive mesh refinement. This work developed and demonstrated new adjoint and tangent formulations for use in partially converged flows for the purposes of optimization and output-based mesh refinement. These were developed through beginning from exact linearizations of the varying nonlinear solution algorithms and then introducing varying approximations. The end results are families of linearizations in both the tangent and adjoint modes that allow for varying degrees of fidelity in the sensitivity and error calculation that allow for design and error estimation in cases that were previously intractible.

This work began with the derivation of tangent and adjoint formulations for a variety of different nonlinear solution methods (both explicit and implicit), through introducing different approximations the linearization of the Newton type solver resulted in a series of tangent and adjoint linearizations of increasing fidelity and accuracy. The new formulations were proven to work first by computing sensitivities in arbitrary partially converged simulations and then by comparing the tangent and adjoint computed sensitivities to those provided by complex-step finite differentiation; this also functioned as proof of proper implementation.

This work then presented an analysis of the sensitivity behavior in both the tangent and adjoint modes, first showing that for problems that enter limit cycle oscillations averaging of the objective function is of utmost importance to obtain useful sensitivities, and that a partial backwards-in-pseudo-time integration is possible with only minor effects on the sensitivities to be used for optimization. When comparing the pseudo-time accurate adjoint to the steady-state adjoint it is clear that the sensitivity vectors have different magnitudes and different directions in the design space; this leads to an assumption of stagnation issues in optimizations driven by the steady state adjoint and this is borne out in the later section on optimization. Furthermore, the steady-state adjoint linearized about these unconverged states are expensive to solve as

shown in previous works.

This work then showed investigations of the various approximate linearizations and their behaviors; during this investigation two points of significant interest were the discovered. The first was that accuracy as compared to the complex-step method is improved by further convergence of the nonlinear problem for all approximate linearizations of the fixed-point iteration. The second was that for Newton-type solvers a practitioner can increase the accuracy of the sensitivities obtained by solving the linear system at each nonlinear step more accurately; the tradeoff being the increased cost of this linear system. Taken together, the accuracy in the sensitivities is determined by both the tolerance of the linear system and the convergence of the nonlinear problem, allowing practitioners a degree of control over the accuracy of their sensitivity calculations. The same is hypothesized to be true for error estimation due to the similarity of the respective derivations.

These partial backwards-in-time integrated sensitivities with approximate linearization of the pseudo-temporal discretization were then used to drive the optimization for analyses which do not converge and obtain better final designs as compared to the typical steady-state adjoint or a pseudo-time averaged objective function with a steady-state adjoint. This method has shown efficacy specifically for shocked flows, where the location of the shock is very important for the value of the objective function. For such transonic or supersonic flows the unsteadiness at the trailing edge not only explicitly affects the objective function, but it can also affect the shock location which will have a much larger impact on the objective function.

Finally this work showed the pseudo-time accurate adjoint applied to error estimation and mesh refinement. These error estimation and refinement techniques showed good behavior for unconverged cases with small scale limit cycle oscillations. These methods had issues with larger scale unsteadiness that can be traced to issues with interpolation and residual evaluation on the fine mesh. In order to make these methods cheaper, some techniques were developed to ameliorate the expense of computing the fixed-point iteration on the embedded mesh at each nonlinear iteration.

8.2 Contributions to the Field

This work showed the development and application of an adjoint and tangent formulation that can be guaranteed to converge even in conditions where the typical steady-state adjoint becomes very stiff and highly inaccurate. It also showed better results for optimizations that were previously intractable or sub-optimal due to the small scale oscillations in the residual and objective functions. This work also proved mathematically and demonstrated numerically that this method can be applied to compute the steady state adjoint sensitivities without exact linearization of the fixed point iteration, thus allowing for computation of the steady state adjoint only by solving less stiff systems with guaranteed convergence. Additionally, to be able to attempt the supersonic test cases and optimizations in this work, a new more robust limiter

technology was developed that enforces realizability as part of the limiter. Finally, three different adjoint based error estimation techniques were developed for the pseudo-time accurate adjoint formulations that are parallel to three existing steady-state output-based error estimation techniques. The embedded mesh techniques developed in this work showed good accuracy and mesh refinement qualities in unconverged cases although further improvements are still possible.

8.3 Future Work

The investigations in this thesis are very encouraging and motivate further investigations and applications of the pseudo-time accurate linearizations to enhance CFD capabilities. Some possible regions of interest are outlined below:

- **Mean pseudo-temporal flow decomposition and linearization** Another method of interest that would be a nice addition to this field would be taking the mean flow to satisfy the average discretized shifted fixed-point iteration, i.e. $R(u)/= 0, \overline{u - G(u)} = 0$, averaging the flow states in pseudo-time and then linearizing the fixed point iteration about the averaged flow state. This could allow for simpler tangent and adjoint systems in partially converged flows. Investigations into the eigenvalues and convergence of these modified linear operators would be informative as to whether this would be preferable not only from a sensitivity accuracy standpoint but also from a linear system stiffness standpoint.

- **Windowing regularization** This method currently uses a simple averaging approach to computation of the objective function and the sensitivities. Windowing regularization techniques seem to be useful to allow for smaller averaging windows and therefore less expensive analysis problems and sensitivity evaluations, or for more stable objective function values as the mesh is refined.

- **Improvement of error estimation routines** The error estimates for this work, while showing promise, had some suboptimal choices of interpolation routines and gradient reconstruction techniques. It could be helpful to use more robust interpolation and gradient reconstruction techniques in the more challenging problems this methodology to which could be extended.

- **Application to viscous flows** Extending this to RANS problems is the next logical step and no major issues are expected in doing so.

- **Three-dimensional problems** Similarly, extension to three-dimensional problems is straightforward and while increased expense is anticipated, it should scale with the primal as was shown for these two-dimensional cases.

- **Unsteady flows** This work presents only steady cases. Time accurate cases like the ones shown by Mishra et al. [36] in the introduction are an important future area of inquiry. Another point of interest

could be applications to space-time solvers as this would be easier in terms of implementation. Rather than storing many nonlinear iterations at many time steps, the unsteady problem is solved as a steady problem and the extension from steady to unsteady would require the same effort as moving from two-dimensional flows to three-dimensional ones.

References

[1] Gill, P. E., Murray, W., and Saunders, M. A., "Users Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming," .

[2] Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM review*, Vol. 47, No. 1, 2005, pp. 99–131.

[3] Adams, B., Bauman, L., Bohnhoff, W., and et al., "Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.0 Users Manual," 2015.

[4] Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., "The Complex-Step Derivative Approximation," *ACM Trans. Math. Softw.*, Vol. 29, No. 3, Sept. 2003, pp. 245262.

[5] Nadarajah, S. K., *The Discrete Adjoint Approach to Aerodynamic Shape Optimization, Ph.D. Dissertation*, Department of Aeronautics and Astronautics, Stanford University, USA, 2003.

[6] Nemec, M. and Aftosmis, M. J., "Toward Automatic Verification of Goal-Oriented Flow Simulations," Tech. Rep. Tech. Rep. TM-2014-218386, NASA Ames Research Center, August 2014.

[7] Venditti, D. A. and Darmofal, D. L., "Anisotropic Grid Adaptation for Functional Outputs: Application to Two-Dimensional Viscous Flows," *J. Comput. Phys.*, Vol. 187, No. 1, May 2003, pp. 2246.

[8] Lighthill, M. J., "The hodograph transformation in trans-sonic flow. I. Symmetrical channels," *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, Vol. 191, No. 1026, 1947, pp. 323–341.

[9] McFadden, G. B., "An artificial viscosity method for the design of supercritical airfoils," Tech. Rep. Tech. Rep. NASA-CR-158840, New York Univ.; Courant Mathematics and Computing Lab.; New York, NY, United States, July 1979.

[10] Murman, E. M. and Cole, J. D., "Calculation of plane steady transonic flows," *AIAA Journal*, Vol. 9, No. 1, 1971, pp. 114–121.

[11] Jameson, A., Caughey, D. A., Newman, P. A., and Davis, R. M., "A brief description of the Jameson-Caughey NYU transonic swept-wing computer program: FLO 22," Tech. Rep. Tech. Rep. NASA-TM-X-73996, NASA Langley Research Center; Hampton, VA, United States, December 1976.

[12] Jameson, A., Schmidt, W., and Turkel, E., "Numerical solution of the Euler equations by finite volume methods using Runge Kutta time stepping schemes," 14th Fluid and Plasma Dynamics Conference, AIAA Paper 1981-1259, Palo Alto, CA. June 1981. https://doi.org/10.2514/6.1981-1259.

[13] Jameson, A. and Baker, T., "Solution of the Euler equations for complex configurations," 6th Computational Fluid Dynamics Conference, AIAA 1983-1929, Danvers, MA. July 1983, https://arc.aiaa.org/doi/abs/10.2514/6.1983-1929.

[14] Lions, J. L., *Optimal Control of Systems Governed by Partial Differential Equations*, Springer, 1971.

[15] Pironneau, O., *Optimal shape design for elliptic systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, 1982, pp. 42–66.

[16] Jameson, A., "Aerodynamic Design via Control Theory," Tech. Rep. Tech. Rep. NASA-CR-181749, ICASE, NASA Langley Research Center; Hampton, VA, United States, November 1988.

[17] Hicks, R. M. and Henne, P. A., "Wing Design by Numerical Optimization," *Journal of Aircraft*, Vol. 15, No. 7, 1978, pp. 407–412.

[18] Mani, K. and Mavriplis, D. J., "Unsteady Discrete Adjoint Formulation for Two-Dimensional Flow Problems with Deforming Meshes," *AIAA Journal*, Vol. 46, No. 6, 2008, pp. 1351–1364.

[19] Zhang, Z. J. and Zingg, D. W., "Efficient Monolithic Solution Algorithm for High-Fidelity Aerostructural Analysis and Optimization," *AIAA Journal*, Vol. 56, No. 3, 2018, pp. 1251–1265.

[20] Mavriplis, D. J., Fabiano, E., and Anderson, E., "Recent Advances in High-Fidelity Multidisciplinary Adjoint-Based Optimization with the NSU3D Flow Solver Framework," 55th AIAA Aerospace Sciences Meeting, AIAA Paper 2017-1669, Grapvine, TX. https://arc.aiaa.org/doi/abs/10.2514/6.2017-1669.

[21] Anderson, E. M., Bhuiyan, F. H., Mavriplis, D. J., and Fertig, R. S., "Adjoint-Based High-Fidelity Aeroelastic Optimization of Wind Turbine Blade for Load Stress Minimization," 2018 Wind Energy Symposium, AIAA Paper 2018-1241, Kissimmee, FL. https://arc.aiaa.org/doi/abs/10.2514/6.2018-1241.

[22] Kamali, S., Mavriplis, D. J., and Anderson, E. M., "Sensitivity Analysis for Aero-Thermo-Elastic Problems Using the Discrete Adjoint Approach," 59th AIAA Aerospace Sciences Meeting, AIAA Paper 2020-3138, Virtual Event, June 2020. https://doi.org/10.2514/6.2020-3138.

[23] Berger, M. J., *Adaptive mesh refinement for hyperbolic partial differential equations, Ph.D. Dissertation*, Department of Computer Science, Stanford University, USA, 1982.

[24] Berger, M. J. and Jameson, A., "Automatic adaptive grid refinement for the Euler equations," *AIAA Journal*, Vol. 23, No. 4, 1985, pp. 561–568.

[25] Berger, M. and Leveque, R., "An adaptive Cartesian mesh algorithm for the Euler equations in arbitrary geometries," 9th Computational Fluid Dynamics Conference, AIAA Paper 1989-1930, Buffalo, NY. https://arc.aiaa.org/doi/abs/10.2514/6.1989-1930.

[26] Lohner, R., "An adaptive finite element scheme for transient problems in CFD," *Computer Methods in Applied Mechanics and Engineering*, Vol. 61, No. 3, 1987, pp. 323 – 338.

[27] "Cart3D Web Page," https://www.nas.nasa.gov/publications/software/docs/cart3d/pages/cart3Dhome.html, Accessed: 2020-06-08.

[28] "Refine Repository," https://github.com/nasa/refine, Accessed: 2020-06-08.

[29] Kirby, A. C., Brazell, M. J., Yang, Z., Roy, R., Ahrabi, B. R., Stoellinger, M. K., Sitaraman, J., and Mavriplis, D. J., "Wind farm simulations using an overset hp-adaptive approach with blade-resolved turbine models," *The International Journal of High Performance Computing Applications*, Vol. 33, No. 5, 2019, pp. 897–923.

[30] Burgess, N. and Mavriplis, D., "An hp-Adaptive Discontinuous Galerkin Solver for Aerodynamic flows on Mixed-Element Meshes," AIAA Paper 2011-490, AIAA Aerospace Sciences Meeting, Orlando, FL, January 2011, https://doi.org/10.2514/6.2011-490.

[31] Becker, R. and Rannacher, R., "An optimal control approach to a posteriori error estimation in finite element methods," *Acta numerica*, Vol. 10, 2001, pp. 1–102.

[32] Mani, K. and Mavriplis, D., "Error Estimation and Adaptation for Functional Outputs in Time-Dependent Flow Problems," AIAA Paper 2009-1495, AIAA Aerospace Sciences Meeting, Orlando, FL, January 2009, https://doi.org/10.2514/6.2009-1495.

[33] Krakos, J. A. and Darmofal, D. L., "Effect of Small-Scale Output Unsteadiness on Adjoint-Based Sensitivity," *AIAA Journal*, Vol. 48, No. 11, 2010, pp. 2611–2623.

[34] Padway, E. and Mavriplis, D. J., "Toward a Pseudo-Time Accurate Formulation of the Adjoint and Tangent Systems," 57th AIAA Aerospace Sciences Meeting, AIAA Paper 2019-0699, San Diego CA, January 2019. https://doi.org/10.2514/6.2019-0699.

[35] Krakos, J. A., Wang, Q., Hall, S. R., and Darmofal, D. L., "Sensitivity analysis of limit cycle oscillations," *Journal of Computational Physics*, Vol. 231, No. 8, 2012, pp. 3228 – 3245.

[36] Mishra, A., Mavriplis, D. J., and Sitaraman, J., "Multipoint Time-Dependent Aero-elastic Adjoint-based Aerodynamic Shape Optimization of Helicopter Rotors," May 2015, pp. 828 – 844, AHS Forum 71,Virginia Beach VA, May 2015, pp 828 - 844.

[37] Luers, M., Sagebaum, M., Mann, S., Backhaus, J., Grossmann, D., and Gauger, N. R., "Adjoint-based Volumetric Shape Optimization of Turbine Blades," AIAA Paper 2018-3638, 2018 Multidisciplinary Analysis and Optimization Conference, Atlanta, GA, June 2018, https://doi.org/10.2514/6.2018-3638.

[38] Brown, D. A. and Nadarajah, S., "An Adaptive Constraint Tolerance Method for Optimization Algorithms Based on the Discrete Adjoint Method," 2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, AIAA SciTech Forum, AIAA Paper 2018-0414, Kissimmee, Florida, 01/2018, https://doi.org/10.2514/6.2018-0414.

[39] Padway, E. and Mavriplis, D. J., "Advances in the Pseudo-Time Accurate Formulation of the Adjoint and Tangent Systems for Sensitivity Computation and Design," 59th AIAA Aerospace Sciences Meeting, AIAA Paper 2020-3136, Virtual Event, June 2020. https://doi.org/10.2514/6.2020-3136.

[40] Burgess, N. K., *An Adaptive Discontinuous Galerkin Solver for Aerodynamic Flows, Ph.D. Dissertation*, Department of Mechanical Engineering, University of Wyoming, USA, 2011.

[41] LeVeque, R. J., *Numerical Methods for Conservation Laws*, Vol. 3, Springer, 1992.

[42] van Leer, B., "Flux-vector splitting for the Euler equations," *Eighth International Conference on Numerical Methods in Fluid Dynamics*, edited by E. Krause, Springer Berlin Heidelberg, Berlin, Heidelberg, 1982, pp. 507–512.

[43] Roe, P., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 135, No. 2, 1997, pp. 250 – 258.

[44] Hirsch, C., *Numerical Computation of Internal and External Flows, Volume 2: Computational Methods for Inviscid and Viscous Flows*, Elsevier Science, 2019.

[45] Mavriplis, D., "Revisiting the Least-Squares Procedure for Gradient Reconstruction on Unstructured Meshes," 16th AIAA Computational Fluid Dynamics Conference, Fluid Dynamics and Co-located Conferences, AIAA Paper 2003-3986, Orlando, Florida, 06/2003. https://doi.org/10.2514/6.2003-3986.

[46] Anderson, W. K., Newman, J. C., and Karman, S. L., "Stabilized finite elements in FUN3D," *Journal of Aircraft*, Vol. 55, No. 2, 2018, pp. 696–714.

[47] van der Vorst, H. A., "Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems," *SIAM Journal on Scientific and Statistical Computing*, Vol. 13, No. 2, 1992, pp. 631–644.

[48] Saad, Y., *Iterative methods for sparse linear systems*, Vol. 82, Society for Industrial and Applied Mathematics, 2003.

[49] Mavriplis, D. J., "Discrete Adjoint-Based Approach for Optimization Problems on Three-Dimensional Unstructured Meshes," *AIAA Journal*, Vol. 45, No. 4, 2007, pp. 741–750.

[50] Batina, J., "Unsteady Euler airfoil solutions using unstructured dynamic meshes," 27th Aerospace Sciences Meeting, AIAA Paper 1989-115, Reno, NV. https://arc.aiaa.org/doi/abs/10.2514/6.1989-115.

[51] Luke, E., Collins, E., and Blades, E., "A fast mesh deformation method using explicit interpolation," *J. Comput. Physics*, Vol. 231, 01 2012, pp. 586–601.

[52] Kedward, L., Allen, C. B., and Rendall, T. C. S., "Comparing Matrix-based and Matrix-free Discrete Adjoint Approaches to the Euler Equations," AIAA Scitech 2020 Forum, AIAA Paper 2020-1294, Orlando FL, January 2020, https://arc.aiaa.org/doi/abs/10.2514/6.2020-1294.

[53] Mavriplis, D. J., "VKI Lecture Series: 38th Advanced Computational Fluid Dynamics. Adjoint methods and their application in CFD, Time Dependent Adjoint Methods for Single and Multi-disciplinary Problems," Sep 2015.

[54] Anderson, W. K. and Bonhaus, D. L., "Airfoil Design on Unstructured Grids for Turbulent Flows," *AIAA Journal*, Vol. 37, No. 2, 1999, pp. 185–191.

[55] Giuliani, A. and Krivodonova, L., "Edge coloring in unstructured CFD codes," *ArXiv*, Vol. abs/1601.07613, 2016.

[56] Günther, S., Gauger, N. R., and Wang, Q., "Simultaneous single-step one-shot optimization with unsteady PDEs," *J. Comput. Appl. Math.*, Vol. 294, 2016, pp. 12–22.

[57] Nielsen, E., Lu, J., Park, M., and Darmofal, D., "An Exact Dual Adjoint Solution Method for Turbulent Flows on Unstructured Grids," 41st Aerospace Sciences Meeting and Exhibit, Aerospace Sciences Meetings, AIAA Paper 2003-272, Reno, Nevada, 01/2009. https://doi.org/10.2514/6.2003-272.

[58] Mavriplis, D. J., "Multigrid Solution of the Discrete Adjoint for Optimization Problems on Unstructured Meshes," *AIAA Journal*, Vol. 44, No. 1, 2006, pp. 42–50.

[59] Nambu, T., Mavriplis, D. J., and Mani, K., "Adjoint-based Shape Optimization of High-lift Airfoil using the NSU2D Unstructured Mesh Solver," 52nd Aerospace Sciences Meeting, AIAA Paper 2014-0554, National Harbor MD, January 2014. https://doi.org/10.2514/6.2014-0554.

[60] "UMESH2D," https://server.scientific-sims.com/cfdlab/scientific-sims/nsu2d.html, Accessed: 2020-11-18.

[61] Anderson, G. R., Nemec, M., and Aftosmis, M. J., "Aerodynamic Shape Optimization Benchmarks with Error Control and Automatic Parameterization," 53rd AIAA Aerospace Sciences Meeting, AIAA Paper 2015-1719, Kissimee, FL. January 2015. https://doi.org/10.2514/6.2015-1719.

[62] Bisson, F. and Nadarajah, S., "Adjoint-Based Aerodynamic Optimization of Benchmark Problems," AIAA Paper 2015-1948, 53rd AIAA Aerospace Sciences Meeting, Kissimmee, FL, January 2015, https://doi.org/10.2514/6.2015-1948.

[63] Schotthofer, S., Zhou, B. Y., Albring, T. A., and Gauger, N. R., "Windowing Regularization Techniques for Unsteady Aerodynamic Shape Optimization," AIAA Aviation 2020 Forum, AIAA Paper 2020-3130, Virtual Conference, June 2020, https://arc.aiaa.org/doi/abs/10.2514/6.2020-3130.

[64] "Adaptive Precision Floating-Point Arithmetic and Fast Robust Predicates for Computational Geometry," https://www.cs.cmu.edu/~quake/robust.html, Accessed: 2020-08-17.

[65] Venditti, D. A., *Grid Adaptation for Functional Outputs of Compressible Flow Simulations, Ph.D. Dissertation*, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, USA, 2002.

[66] Kulfan, B. M., "Universal parametric geometry representation method," *Journal of aircraft*, Vol. 45, No. 1, 2008, pp. 142–158.

[67] Diskin, B. and Thomas, J., "Accuracy of Gradient Reconstruction on Grids with High Aspect Ratio," 01 2009.