AIAA AVIATION Forum
June 15-19, 2020, VIRTUAL EVENT
AIAA AVIATION 2020 FORUM

10.2514/6.2020-3136

Advances in the Pseudo-Time Accurate Formulation of the Adjoint and Tangent Systems for Sensitivity Computation and Design

Emmett Padway [*]
Dimitri Mavriplis [†]
Department of Mechanical Engineering, University of Wyoming, Laramie, WY 82071

This paper presents progress in the pseudo-time accurate formulation of the adjoint and tangent systems for sensitivity computation and design to better compute sensitivities for non-converging simulations. Previous efforts presented the formulation and verification for two explicit schemes and a quasi-Newton method. The previous quasi-Newton method formulation had assumptions that limited the generality of its potential applications as it was only exact for Newton-Chord Method nonlinear solvers with fixed linear smoothers. This paper presents a new quasi-Newton method formulation using a more general assumption and shows that in the limit of exact solution of the linear system the exact sensitivities are recovered; additionally, this method is guaranteed to converge under certain conditions. Furthermore, this work shows that the approximation of the sensitivities also becomes more accurate as the nonlinear problem is solved. Finally, this method is applied to three design optimization cases with non-converging analysis problems.

I. Introduction

Multidisciplinary Design Optimization (MDO) along with Adaptive Mesh Refinement (AMR) have become larger parts of the expanding field of Computational Fluid Dynamics (CFD) as computers and algorithms have developed. Much MDO is done using gradient driven optimization as this allows for far fewer function evaluations when compared to global methods, such as genetic 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 and this has allowed researchers to provide sensitivities through a selection of commonly used methods: finite-difference (real or complex-step), tangent, or adjoint methods. The last two methods[4] are more accurate than the traditional finite-difference method (providing they are properly implemented) and are developed through conditions on convergence to generate mathematical equations to solve for the sensitivities. The main benefit of the complex-step finite-difference method is that for small complex steps it provides the exact sensitivities of the computational process; it provides these sensitivities because it is the complex-step diffennentiation of the analysis simulation process. We argue that these are the appropriate sensitivities to use in unconverged problems. The downsides of this method are the slowdown of the process due to the inclusion of complex arithmetic, as well as the fact that this method scales with the number of design variables. In unconverged flows, the finite-difference sensitivities computed either with real or complex-step perturbations are the sensitivities of the solution process; with the complex-step computed sensitivities lacking the round-off error of the traditional finite-difference method. The tangent formulation for the sensitivities of the steady-state problem scales similarly to the finite-difference method, in that it generates a linear system which scales with the number of design variables; however, the formulation is based on the requirement that the analysis problem be fully converged to generate its linear system. The adjoint problem for the steady-state analysis uses that same residual requirement, but is different in that it transposes the system and scales independently

*Graduate Student, Member AIAA; email: epadway@uwyo.edu
†Professor, AIAA Associate Fellow; email: mavripl@uwyo.edu

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. Additionally the adjoint solution is well suited for AMR as a tool to refine a computational mesh to increase accuracy in an output of interest.[5,6]

As stated above, both the adjoint and tangent systems require that the analysis problem be fully converged – or in the case of the unsteady adjoint, each implicit time step – indicating that the governing equations have been satisfied to machine precision. However, as CFD has developed and matured and the field has attempted more difficult problems (higher-order formulations, blunt geometries, or time-accurate simulations) this constraint has become difficult or impractical to obey, and numerous design or mesh refinement cycles have been done using adjoint systems linearized about partially converged analysis 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 analysis problem about which they are linearized when the analysis simulation is terminated.[7,8] For AMR, this can lead to obtaining different AMR patterns which can lead to refining the mesh in areas that may not contribute to the output of interest, and not refining in areas where the output of interest is affected. This can impact the accuracy of the analysis problem as the mesh adaptation proceeds. In the realm of design optimization, which is the focus of this paper, inaccurate adjoints 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]

As mentioned above, Krakos and Darmofal[7] illustrate that for a nonconvergent case, the state about which the adjoint is linearized can notably affect the sensitivity calculations. The authors then 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 lest CFD practitioners 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 analysis flows with time-averaged outputs of interest.[9] 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.[10] have demonstrated growing sensitivity error during the adjoint reverse time-integration due to partial convergence of the analysis problem at each implicit time step, which is a common practice in applied CFD problems.

To summarize, the works of Krakos and Darmofal show that the appropriate way to handle these systems is by running the time-accurate analysis problem and then using statistical techniques for less expensive unsteady adjoints. However, Mishra et al. show that the time-accurate problem is too expensive to solve and the error will grow through the backwards-in-time integration. In order to tackle these sorts of cases, Padway and Mavriplis[8] run the steady state problem and use the pseudo-time accurate formulation for a pseudo-time averaged objective functional to compute the adjoint and tangent sensitivities, which will correspond exactly to the sensitivities obtained through the complex step differentiation of the solution process. Their 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.

Luers et al.[11] illustrate the importance of accurate gradients in well converging cases that have not reached deep convergence. Their paper shows steady-state optimizations for a CRESCENDO turbine cascade, and contrasts the optimization for finite-difference computed gradients to that driven by adjoint computed gradients in a case with a 5 order drop in the analysis residual. It is shown that that by using the finite-difference provided gradients the optimized efficiency increases by a greater percentage than by using the adjoint provided gradients, while still obeying the constraint of the analysis problem. In an effort to address this expense of deep convergence, Brown and Nadarajah[12] investigate an upper bound for the error in the adjoint computed sensitivities arising from partial convergence of the analysis problem. 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 and accuracy in the sensitivity vector is at a premium. The work of Brown and Nadarajah is a natural solution to the issues demonstrated by Luers et al. in their volumetric optimization cases. Shimizu et al., rather than computing the error directly, have worked to lessen it by

applying least squares shadowing and windowing techniques towards better behaved adjoint systems and more accurate adjoint computed sensitivities with an application to chaotic adjoint problems.[13,14]
This work comes from the work of Padway and Mavriplis[8] on the effects of incomplete analysis problem convergence in steady state problems. In this paper we develop a new approach to the quasi-Newton nonlinear solver that in the limit of exact solution of the linear system will correspond exactly to the sensitivities obtained by the complex-step differentiation of the simulation at every step in the iterative history. Additionally, this method will also show good behavior at larger linear system tolerances. This paper shows verification of the methods through a comparison of the computed sensitivities with complex-step finite-difference sensitivities, and proceeds to examine the accuracy of sensitivities obtained in this manner as a function of the linear system tolerance and its application to design.

II. Background and In-House Solver

II.A. Governing Equations

We developed an in-house flow solver to solve the steady-state Euler equations on unstructured meshes. The steady-state compressible Euler equations (which may also be referred to as the analysis problem) can be written as follows.

$$\nabla \cdot F(u(D)) = 0 \tag{1}$$

Which can also be written as:

$$R(u(D), D) = 0 \tag{2}$$

where u is the conservative variable vector, D is the design variable vector, and $F(u)$ is the conservative variable flux.

II.B. Spatial Discretization

The residual about the closed control volume is given as:

$$R = \int_{dB} [F(u(D))] \cdot n(x(D)), \mathrm{dB} = \sum_{i=1}^{n_{edge}} F^{\perp}_{e_i}(u(D), n_{e_i}(x(D)))B_{e_i}(x(D)) \tag{3}$$

This equation gives the operator in the aforementioned requirement for the adjoint and tangent systems. In the discretized form of the residual operator, x is the mesh points, F is the numerical flux across the element boundary, B is the element boundary, and n is the edge normal on the element boundary. The solver used in this work is a steady-state finite-volume cell-centered Euler solver with second-order spatial accuracy implemented for triangular elements. Second-order accuracy is implemented through weighted least squares gradient reconstruction.[15] The solver has three different flux calculations implemented and linearized, these are the Lax-Friedrichs,[16] Roe,[17] and van Leer[18] schemes. In this work, only the Lax-Friedrichs and Van-Leer schemes are used. In order to slope limit the solution reconstruction near shock discontinuties, a modified Venkatakrishnan limiter is used; Venkatakrishnan's limiter[19] is modified in a few important ways.

1. All max and min functions use smooth max and mins to allow differentiability
2. All switches in the code dependent on the flow state are done in a smooth and blended manner using a sin shut-off function to aid differentiability
3. The limiter is augmented with a stagnation point fix[20] 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[21] that, if violated (pressure, energy, or density are reconstrcuted to below 5% of the cell center value), the cell gradient is set to 0. Then the cell (as a whole) becomes first order, this is used rather than employing face-based limiting.

This limiter in algorithm (1) provides better convergence properties than the standard VK 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 (4),[22] every step in this limiter is

Algorithm 1 Augmented Venkatakrishnan Limiter

```
 1: procedure VK LIMITER
 2:     $M_1 = .80, M_2 = .85, \epsilon_{lim} = 1e-5, \epsilon_{TFO_s} = -.95, \epsilon_{TFO_e} = -.9$
 3:     $\epsilon = \sqrt{\kappa(r_1)}^3$, where $r_1$ is the radius of the circumscribed circle of the triangular cell
 4:     $M_{max} = \bar{M}$
 5:     for $i = 1, ..., fields$ do
 6:         $\Phi_i = 1$
 7:         $\delta u_i^{min} = \infty$, $\delta u_i^{max} = 0$
 8:         for $j = 1, ..., neighbors$ do
 9:             $\delta u_i^{min} = \min(\delta u_i^{min}, \bar{u}_i - \bar{u_i^j})$
10:             $\delta u_i^{max} = \max(\delta u_i^{max}, \bar{u}_i - \bar{u_i^j})$
11:             $M_{max} = \max(M_{max}, M_i)$
12:         $\sigma_{Mach} = 1 - SSO(M_{max}, M_1, M_2)$
13:         for $j = 1, ..., neighbors$ do
14:             $\Delta_- = \nabla\bar{u}_i \cdot \vec{r_j}$
15:             $s = SSO(\Delta_-, 0, \epsilon_{lim})$
16:             $\Delta_+ = (1-s)(u_i^{min} - \bar{u}_i) + s(u_i^{max} - \bar{u}_i)$
17:             $\phi = \frac{\Delta_+^2 + \epsilon^2 + 2\Delta_+\Delta_-}{\Delta_+^2 + \epsilon^2 + 2\Delta_+\Delta_- + 2\Delta_-^2}$
18:             Stagnation point fix
19:             $\phi = \sigma_{Mach} + (1 - \sigma_{Mach})\phi$
20:             $\Phi_i = \min(\Phi_i, \phi)$
21:     Begin realizability check
22:     for $j = 1, ..., neighbors$ do
23:         $u_{rc_j} = u_i + \Phi\nabla\bar{u}_i \cdot \vec{r_j}$
24:         $\delta_\rho = \rho_{rc_j} - \bar{\rho}$
25:         $\delta_P = P_{rc_j} - \bar{P}$
26:         $\delta_E = E_{rc_j} - \bar{E}$
27:         $s_e = SSO(\delta_e, \epsilon_{TFO_s}, \epsilon_{TFO_e})$
28:         $s_p = SSO(\delta_p, \epsilon_{TFO_s}, \epsilon_{TFO_e})$
29:         $s_\rho = SSO(\delta_\rho, \epsilon_{TFO_s}, \epsilon_{TFO_e})$
30:         $s = \min(s_\rho, s_e, s_p)$
31:         $\Phi = s\Phi$
```

differentiable. This makes the linearization of the limiters possible and much easier to code, as we avoid multiple branching statements.

$$SSO(x, x_s, x_e) = \begin{cases} 0 & if x < x_s \\ \frac{1}{2}\left(sin\left(\frac{\pi}{2}\frac{2x-(x_e+x_e)}{x_e-x_s}\right)+1\right) & if xs < x < x_e \\ 1 & if x > x_e \end{cases} \tag{4}$$

II.C. Steady-State Solver

The solver technology for this code uses either explicit time stepping through pseudo time using a forward Euler time discretization, or a low storage five stage Runge-Kutta scheme, or a quasi-Newton method. The quasi-Newton method is implemented using pseudo-transient continuation (PTC) with a BDF1 pseudo temporal discretization scheme. For Newton's method the time-stepping procedure is written as:

$$u^k = u^{k-1} + \Delta u \tag{5}$$

where we compute Δu by solving the following system of linear equations.

$$[P]\,\Delta u = -R(u) \tag{6}$$

We can substitute our expression for Δu into the time-stepping equation (5) to obtain our final form of this equation.

$$u^k = u^{k-1} - [P_{k-1}]^{-1}\,R \tag{7}$$

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 (8).

$$[P_{k-1}] = \left[\frac{\partial R}{\partial u^{k-1}}\right]_1 + \frac{vol}{\Delta t CFL} \tag{8}$$

Please note the subscript on the jacobian above denotes that it is a 1st order jacobian; in this work the subscripts of 1 and 2 will denote first and second order spatially accurate jacobians respectively. The equation for the local explicit time step limit Δt is given as:

$$\Delta t_i = \frac{r_i}{\sqrt{(u^2+v^2)}+c} \tag{9}$$

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.
Furthermore, the CFL is scaled either with a simple ramping coefficient (β) and cap criterion:

$$CFL^{k+1} = min(\beta \cdot CFL, CFL_{max}) \tag{10}$$

or with a linesearch and CFL controller,[23] which seeks to minimize the L2 norm of the pseudo-temporal residual, defined as:

$$R_t(u+\alpha\Delta u) = \frac{vol}{\Delta t}\alpha\Delta u^n + R(u+\alpha\Delta u^n) \tag{11}$$

when the pseudo-temporal residual decreases, we consider this to be a satisfactory value for α and we change the CFL accordingly. The pseudocode explains the actual process for the linesearch and CFL controller. The parameters $iter_{max}$, c, α_{l_1}, α_{l_2}, β_1, and β_2 are all user defined input values, defaulted to 30, .9, .1, .75, .1, and 1.5 respectively. One benefit of this combined line-search and CFL controller is that we do not have to differentiate it, and can simply store the values of CFL and α and use these fixed values in the forward and reverse linearizations and still obtain machine-level correspondence when comparing those sensitivity values to those of the complex-step differentiated solution process. This is possible because the combined CFL controller and line-search is a piece-wise constant function with a zero derivative.

Algorithm 2 CFL controller

```
1: procedure LINE SEARCH AND CFL CONTROLLER
2:     $r_{t0} = \|R_t(u + \Delta u)\|_2$
3:     $r_{s0} = \|R(u)\|_2$
4:     $r_{s1} = \|R(u + \Delta u)\|_2$
5:     if $r_{s0} < r_{s1}$ then
6:         for $k = 1, ..., iter_{max}$ do
7:             $\alpha = c\alpha$
8:             $r_{t1} = \|R_t(u + \alpha\Delta u)\|_2$
9:             if $r_{t1} < r_{t0}$) then exit
10:    if $\alpha < \alpha_{l_1}$ then
11:        $\alpha = 0$
12:        $CFL = \beta_1 CFL$
13:    else if $\alpha_{l_1} < \alpha < \alpha_{l_2}$ then
14:        $CFL = CFL$
15:    else if $\alpha < \alpha_{l_2}$ then
16:        $CFL = min(\beta_2 CFL, CFL_{max})$
```

In order to solve the linear system we use a point-implicit Jacobi or point-implicit Gauss-Seidel solver. This is done by lagging the off-diagonal components, with the right hand side being the linear residual of the original system. In this work the residual is the second-order accurate spatial residual operator, and $[P_{k-1}]$ is based off the first-order accurate residual operator outlined in equation (8). Equation (6) is then solved iteratively as:

$$[D]\,\Delta(\Delta u)^l = -R(u) - [P_{k-1}]\,\Delta u^l \tag{12}$$

where the matrix $[D]$ is the element block diagonal entry in the Jacobian matrix.

$$\Delta u^{l+1} = \Delta u^l + \omega(\Delta(\Delta u))^l \tag{13}$$

These linear solvers can also be applied as smoothers either to a BiCGStab or a GMRES linear solver. The flexible GMRES linear solver is the more commonly used one in this Newton-Krylov nonlinear solver, and the algorithm below shows its implementation, where the operator M^{-1} is the preconditioning matrix using the block Jacobi or block Gauss-Seidel relaxation schemes outlined above. The FGMRES solver outlined in algorithm (3)[24] is implemented to solve the stiff steady state tangent and adjoint systems as well.

Algorithm 3 Flexible Restarted GMRES

```
1: procedure FLEXIBLE GMRES
2:     for $k = 1, ..., ncycles$ do
3:         $r_0 = b - Ax_0, \beta = \|r_0\|, v_1 = r_0/\beta$
4:         for $j = 1, ..., m$ do
5:             $z_j = M^{-1}v_j$
6:             $v_{j+1} = Az_j$
7:             for $i = 1, ..., j$ do
8:                 $h_{i,j} = (v_{j+1}, v_i)$
9:                 $v_{j+1} = v_{j+1} - h_{i,j}v_i$
10:            $h_{j+1,j} = \|v_{j+1}\|, v_{j+1} = v_{j+1}/h_{j+1,j}$
11:            Define $Z_m = [z_1, ..., z_m], \bar{H}_m = [h_{i,j}]_{1<i<j+1,1<j<m}$
12:        Solve least squares problem for $y_m$
13:        $x_0 = x_0 + Z_m y_m$
```

II.D. A Review of Tangent and Adjoint Systems

II.D.1. Tangent Formulation

For an aerodynamic optimization problem, we consider an objective functional $L(u(D), x(D))$, for example lift or drag, where u is the conservative variable vector, and x is the vector of point coordinates. In order to obtain an expression for the sensitivities we take the derivative of the objective functional:[25]

$$\frac{dL}{dD} = \frac{\partial L}{\partial x}\frac{\partial x}{\partial D} + \frac{\partial L}{\partial u}\frac{\partial u}{\partial D} \tag{14}$$

For the above expression $\frac{\partial L}{\partial x}$ and $\frac{\partial L}{\partial u}$ can be directly obtained by differentiating the corresponding subroutines in the code. $\frac{\partial x}{\partial D}$ is calculated by solving the spring analogy mesh deformation equation:

$$[K]\frac{\partial x_v}{\partial D_i} = \frac{\partial x_s}{\partial D_i} \tag{15}$$

and we calculate $\frac{\partial x_s}{\partial D_j}$ through differentiating the shape design variables. For the global inverse distance weighted method,[26] the mesh sensitivities are computed directly as a function of the surface coordinate sensitivities:

$$\frac{\partial x_{v_i}}{\partial D_j} = \frac{\sum w_{ik}(\vec{r}_{ik})\frac{\partial x_{s_k}}{\partial D_j}}{\sum w_{ik}(\vec{r}_{ik})} \tag{16}$$

It is not possible to obtain $\frac{\partial u}{\partial D}$ through linearization of the subroutines in the code without linearizing the entire analysis solution process, as will be covered in later sections. In order to solve for this term we use the constraint that for a fully converged flow $R(u(D), x(D)) = 0$. By taking the derivative of the residual operator we obtain the equation below.

$$\left[\frac{\partial R}{\partial x}\right]\frac{\partial x}{\partial D} + \left[\frac{\partial R}{\partial u}\right]_2\frac{\partial u}{\partial D} = 0 \tag{17}$$

We can isolate the sensitivity of the residual to the design variables to obtain the tangent system.

$$\left[\frac{\partial R}{\partial u}\right]_2\frac{\partial u}{\partial D} = -\left[\frac{\partial R}{\partial x}\right]\frac{\partial x}{\partial D} \tag{18}$$

We solve this linear system, using hand differentiated subroutines to provide the left hand matrix $\left[\frac{\partial R}{\partial u}\right]_2$, the right hand side $\left[\frac{\partial R}{\partial x}\right]\frac{\partial x}{\partial D}$ (which scales with the design variables), and obtain $\frac{\partial u}{\partial D}$. We then substitute $\frac{\partial u}{\partial D}$ into equation (14) to obtain the final sensitivities.

II.D.2. Discrete Adjoint Formulation

The adjoint formulation begins with the same sensitivity equation:

$$\frac{dL}{dD} = \frac{\partial L}{\partial x}\frac{\partial x}{\partial D} + \frac{\partial L}{\partial u}\frac{\partial u}{\partial D} \tag{19}$$

Using the condition $R(u(D), D) = 0$, we return to equation (18) and pre-multiply both sides of the equation by the inverse Jacobian matrix to obtain:

$$\frac{\partial u}{\partial D} = -\left[\frac{\partial R}{\partial u}\right]_2^{-1}\left[\frac{\partial R}{\partial x}\right]\frac{\partial x}{\partial D} \tag{20}$$

Substituting the above expression into the sensitivity equation yields:

$$\frac{dL}{dD} = \frac{\partial L}{\partial x}\frac{\partial x}{\partial D} - \frac{\partial L}{\partial u}\left[\frac{\partial R}{\partial u}\right]_2^{-1}\left[\frac{\partial R}{\partial x}\right]\frac{\partial x}{\partial D} \tag{21}$$

We then define an adjoint variable $\mathbf{\Lambda}$ such that:

$$\mathbf{\Lambda}^T = -\frac{\partial L}{\partial u}\left[\frac{\partial R}{\partial u}\right]_2^{-1} \tag{22}$$

which gives an equation for the adjoint variable:

$$\left[\frac{\partial R}{\partial u}\right]_2^T \mathbf{\Lambda} = -\left[\frac{\partial L}{\partial u}\right]^T \tag{23}$$

We can solve this linear system and obtain the sensitivities for the objective function as follows:

$$\frac{dL}{dD} = \left[\frac{\partial L}{\partial x} + \mathbf{\Lambda}^T \frac{\partial R}{\partial x}\right] \frac{\partial x}{\partial D} \tag{24}$$

We can then define a mesh adjoint variable $\mathbf{\Lambda_x}$:

$$[K]^T \mathbf{\Lambda_x} = \left[\frac{\partial L}{\partial x}\right]^T + \left[\frac{\partial R}{\partial x}\right]^T \mathbf{\Lambda} \tag{25}$$

and the final expression for sensitivity is given as:

$$\frac{dL}{dD} = \mathbf{\Lambda_x}^T \frac{\partial x_s}{\partial D} \tag{26}$$

The adjoint system is of interest, because as mentioned in the introduction, it results in an equation for the sensitivity that does not scale with the number of design variables.

III. Development of the Pseudo-Time Accurate Tangent

In this section we show the previous tangent formulation[8] and the new one developed in this paper for the quasi-Newton nonlinear solver. We also discuss the impacts on implementation and accuracy of some of the assumptions made in the previous formulation and the one presented here. As stated above, the assumptions made are exact when the linear system at each iteration is solved to machine precision. In such cases the tangent sensitivities would exactly correspond to the sensitivities of solution process itself, and will correspond exactly to the sensitivities obtained by the complex-step finite difference method at each step of the solution process. The idea is that a sensitivity vector obtained from the differentiation of the solution process is preferable to one obtained by linearizing the adjoint system about an unconverged state, because, as stated above, adjoint systems of unconverged analyses are very sensitive to the specific state about which they are linearized and we cannot say with certainty what they correspond to mathematically. This formulation is the equivalent of using the derivative of the solution process to drive the optimization. We note that in this section and the derivation of the adjoint section, when taking the derivative of an operator with respect to the design variables, $\frac{\partial}{\partial D}$ is an abbreviation of $\frac{\partial}{\partial x}\frac{\partial x}{\partial D}$.

III.A. Tangent System for Newton-Chord Method

For this section, 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 uses an approximation to the Jacobian matrix which is only first-order spatially accurate, and the resulting linear system is only approximately solved. For Newton's method the time-stepping procedure is written as in equation (7).

$$u^k = u^{k-1} - [P_{k-1}]^{-1} R \tag{27}$$

where $[P_{k-1}]$ is a first-order accurate Jacobian augmented with a diagonal term to ensure that it is diagonally dominant as in equation (8).

$$[P_{k-1}] = \left[\frac{\partial R}{\partial u^{k-1}}\right]_1 + \frac{vol}{\Delta t CFL} \tag{28}$$

If we take the derivative of each side of equation (27) we obtain:

$$\frac{du}{dD}^k = \frac{du}{dD}^{k-1} - [P_{k-1}]^{-1}\left[\frac{\partial R}{\partial D} + \left[\frac{\partial R}{\partial u^{k-1}}\right]_2 \frac{du}{dD}^k\right] - R(u^{k-1})\left[\frac{\partial [P_{k-1}]^{-1}}{\partial D} + \frac{\partial [P_{k-1}]^{-1}}{\partial u}\frac{du}{dD}^{k-1}\right] \tag{29}$$

If we choose to neglect the change in the preconditioner we can simplify the above equation a great deal, as is shown below, by avoiding taking derivatives of inverse Jacobians. This can be done either by using a frozen Jacobian, which is also known as a Newton-Chord method, or by arguing by egodicity that for partially converged cases, oscillations about some mean are independent of the initial state. It should be noted that although the derivative of the the inverse preconditioner is nonzero, the argument used is based on the assumption that the residual at this point is very small. This small residual vector will be multiplying the derivative of the preconditioner inverse in the last term on the right hand side of equation (29), and will have a negligible contribution. Alternatively, this is the exact linearization if we perform a smoothing operation independent of the simulation characteristics (a fixed number of smoothing passes, for example), thereby ensuring the derivative of the partial linear system solve is in fact equal to zero. This method will not work for any linear solvers that have a dependence on the right hand side of the linear system, and so is limited to relaxation methods. This of course excludes the use of GMRES (or any Krylov solver) because while they are linear solvers, they are path dependent. Proceeding with the simplified equation:

$$\frac{du}{dD}^{k} = \frac{du}{dD}^{k-1} - [P_{k-1}]^{-1}\left[\frac{\partial R}{\partial D} + \left[\frac{\partial R}{\partial u}\right]_2 \frac{du}{dD}^{k-1}\right] \tag{30}$$

This can then be rewritten as:

$$\frac{du}{dD}^{k} = \frac{du}{dD}^{k-1} + \Delta\left(\frac{du}{dD}\right) \tag{31}$$

and we solve the following linear system:

$$[P_{k-1}]\,\Delta\left(\frac{du}{dD}\right) = -\left[\frac{\partial R}{\partial D} + \left[\frac{\partial R}{\partial u}\right]_2 \frac{du}{dD}^{k-1}\right] \tag{32}$$

It is important to note that we are not taking the exact inverse of the preconditioner matrix in the iterative equation above, rather we are performing the exact number of steps to invert it as we did in the analysis problem, as we are drawing directly of the analysis solution process. If this iterative process is run at the converged state then it corresponds to the dual direct differentiation;[27,28] its adjoint presented later is the exact dual of that process.

III.B. Tangent System for quasi-Newton Method

For this section, we begin from equation (7)

$$u^k = u^{k-1} - [P_{k-1}]^{-1} R \tag{33}$$

where $[P_{k-1}]$ is again a first-order accurate Jacobian augmented with a diagonal term to ensure that it is diagonally dominant, as described in equation (8).

$$[P_{k-1}] = \left[\frac{\partial R}{\partial u^{k-1}}\right]_1 + \frac{vol}{\Delta t CFL} \tag{34}$$

If we take the derivative of each side of equation (34) we obtain:

$$\frac{du}{dD}^{k} = \frac{du}{dD}^{k-1} - [P_{k-1}]^{-1}\left[\frac{\partial R}{\partial D} + \left[\frac{\partial R}{\partial u^{k-1}}\right]_2 \frac{du}{dD}^{k}\right] - \left[\frac{d\,[P_{k-1}]^{-1}}{dD}\right] R(u^{k-1}) \tag{35}$$

Here, rather than neglecting the change in the preconditioner we use a definition of the derivative of the matrix inverse defined by:

$$\frac{d\,[K]^{-1}}{dx} = -\,[K]^{-1}\left[\frac{dK}{dx}\right][K]^{-1} \tag{36}$$

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 us to have a clearly defined source of error in our computation and evaluate how much the linear tolerance of the system affects the sensitivity computation. Furthermore, we can argue that the error from this sensitivity will go to 0 as we approach full convergence, and we can investigate the behavior in near-ergodic limit cycle oscillations. By computing the total derivative of the nonlinear solver–

as in equation (35)– and substituting in the expression from (36) into equation (35), the below expression is generated.

$$\frac{du^k}{dD} = \frac{du^{k-1}}{dD} - [P_{k-1}]^{-1}\left[\left[\frac{\partial R}{\partial u^{k-1}}\right]_2 \frac{du^{k-1}}{dD} + \left[\frac{\partial R}{\partial x}\right]\frac{dx}{dD}\right] + [P_{k-1}]^{-1}\frac{d\,[P_{k-1}]}{dD}[P_{k-1}]^{-1}R(u^{k-1},D) \tag{37}$$

Expanding the total derivative shows that this is in fact the sum of two matrix vector products:

$$\begin{aligned}\frac{du^k}{dD} &= \frac{du^{k-1}}{dD} - [P_{k-1}]^{-1}\left[\left[\frac{\partial R}{\partial u^{k-1}}\right]_2 \frac{du^{k-1}}{dD} + \left[\frac{\partial R}{\partial x}\right]\frac{dx}{dD}\right]\\ &+ [P_{k-1}]^{-1}\left[\frac{\partial P_{k-1}}{\partial u^{k-1}}\frac{du^{k-1}}{dD} + \frac{\partial P_{k-1}}{\partial x}\frac{dx}{dD}\right][P_{k-1}]^{-1}R(u^{k-1},D)\end{aligned} \tag{38}$$

While the full hessian is an undesireable item to compute, this formulation requires two hessian vector products that can be obtained through complex perturbations to the conservative variables and nodal coordinate vectors, and subsequent evaluation of the jacobian. This avoids the need for the full hessian computation. Furthermore, one of the matrix inverses can be removed by reusing the computation of Δu and rewriting the equation as:

$$\begin{aligned}\frac{du^k}{dD} &= \frac{du^{k-1}}{dD} - [P_{k-1}]^{-1}\left[\left[\frac{\partial R}{\partial u^{k-1}}\right]_2 \frac{du^{k-1}}{dD} + \left[\frac{\partial R}{\partial x}\right]\frac{dx}{dD}\right]\\ &+ [P_{k-1}]^{-1}\left[\frac{\partial P_{k-1}}{\partial u^{k-1}}\frac{du^{k-1}}{dD} + \frac{\partial P_{k-1}}{\partial x}\frac{dx}{dD}\right]\Delta u\end{aligned} \tag{39}$$

This can then be rewritten as:

$$\frac{du}{dD}^{k} = \frac{du}{dD}^{k-1} + \Delta\left(\frac{du}{dD}\right) \tag{40}$$

and we solve the following linear system:

$$[P_{k-1}]\,\Delta\left(\frac{du}{dD}\right) = -\left[\left[\frac{\partial R}{\partial u^{k-1}}\right]_2 \frac{du^{k-1}}{dD} + \left[\frac{\partial R}{\partial x}\right]\frac{dx}{dD}\right] + \left[\frac{\partial P_{k-1}}{\partial u^{k-1}}\frac{du^{k-1}}{dD} + \frac{\partial P_{k-1}}{\partial x}\frac{dx}{dD}\right]\Delta u \tag{41}$$

It is important to note that this expression is only exact for machine-zero solution of the linear system at every iteration, but for those cases we will have exact correspondence between this and the complex-step computed sensitivities. This saves us from needing to differentiate the entire linear solution process, which is intractable for many cases for the forward mode, and even more difficult for the reverse 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 solvers for the forward and the reverse linearizations. We can also note that, where in the initial formulation– in equation (38)– we see 3 approximate linear solves; when we group terms and use the already stored information from the analysis solve we are left with only one approximate linear solve, the same as in the analysis problem's nonlinear solver. Although we are not limited to using the same linear solver for the analysis and tangent problems and its dual for the adjoint problem, by doing so we gain in that we are guaranteed to have a converging tangent and adjoint solver. We would then be solving using the same algorithm that ran without divergence through the analysis portion, when we apply this algorithm to the tangent and adjoint problem we have the same eigenvalues, same condition numbers, and same convergence properties. Therefore, if our analysis problem did not diverge, neither will the tangent or the adjoint.

IV. Development of the Pesudo-Time Accurate Adjoint

As stated above, the pseudo-time accurate adjoint method is drawn from the derivation of the unsteady adjoint. In this method we look at each pseudo-time step and work backwards through pseudo-time to get the pseudo-time accurate adjoint solution. In this derivation the objective function is a pseudo-time averaged functional, averaged over the last m steps for a program that runs through n pseudo-time steps. From this we define our objective function L as:

$$L = L(u^n, u^{n-1}, ..., u^{n-m}, D) \tag{42}$$

where u^n is the conservative variable vector at the final time step n and D is the design variable vector. For the constraint we cannot select $R(u, D) = 0$, as this is not true at each pseudo-time step; instead, we select a constraint based on the pseudo-time evolution of the solution, for which, the kth constraint will be referred to as G^k. We know because we are using first order time-stepping that the constraint is dependent only on the old time-step, the new time-step, and the design variables, expressed as follows.

$$G^k = G^k(u^k, u^{k-1}, D) = 0 \tag{43}$$

We define an augmented objective function with n constraints and n Lagrange multipliers:

$$\begin{aligned} J(D, u^n, u^{n-1}, u^{n-2}, ..., \Lambda^n, \Lambda^{n-1}, ...) = {} & L(u^n, u^{n-1}, ..., u^{n-m}, D) \\ & + \Lambda^{nT} G^n(u^n(D), u^{n-1}(D), D) \\ & + \Lambda^{n-1T} G^{n-1}(u^{n-1}(D), u^{n-2}(D), D) \\ & + ... \\ & + \Lambda^{1T} G^1(u^1(D), u^0(D), D) \end{aligned} \tag{44}$$

In order to get an expression for the adjoint we take the derivative of the augmented objective function with respect to the conservative variables at different pseudo-time steps, and choose our Lagrange multiplier such that these partial derivatives are equal to 0.

$$\begin{aligned} \frac{\partial J}{\partial u^n} &= \frac{\partial L}{\partial u^n} + \Lambda^{nT} \frac{\partial G^n}{\partial u^n} = 0 \\ \frac{\partial J}{\partial u^{n-1}} &= \frac{\partial L}{\partial u^{n-1}} + \Lambda^{nT} \frac{\partial G^n}{\partial u^{n-1}} + \Lambda^{n-1T} \frac{\partial G^{n-1}}{\partial u^{n-1}} = 0 \\ \frac{\partial J}{\partial u^{n-2}} &= \frac{\partial L}{\partial u^{n-2}} + \Lambda^{n-1T} \frac{\partial G^{n-1}}{\partial u^{n-2}} + \Lambda^{n-2T} \frac{\partial G^{n-2}}{\partial u^{n-2}} = 0 \\ & ... \\ \frac{\partial J}{\partial u^1} &= \frac{\partial L}{\partial u^1} + \Lambda^{2T} \frac{\partial G^2}{\partial u^1} + \Lambda^{1T} \frac{\partial G^1}{\partial u^1} = 0 \end{aligned} \tag{45}$$

Using $L = L(u^n, u^{n-1}, ..., u^{n-m}, D)$, we get the following equation.

$$\begin{aligned} \frac{\partial J}{\partial u^n} &= \frac{\partial L}{\partial u^n} + \Lambda^{nT} \frac{\partial G^n}{\partial u^n} = 0 \\ \frac{\partial J}{\partial u^{n-1}} &= \frac{\partial L}{\partial u^{n-1}} + \Lambda^{nT} \frac{\partial G^n}{\partial u^{n-1}} + \Lambda^{n-1T} \frac{\partial G^{n-1}}{\partial u^{n-1}} = 0 \\ & ... \\ \frac{\partial J}{\partial u^{n-m}} &= \frac{\partial L}{\partial u^{n-m}} + \Lambda^{n-(m-1)T} \frac{\partial G^{n-(m-1)}}{\partial u^{n-m}} + \Lambda^{n-mT} \frac{\partial G^{n-m}}{\partial u^{n-m}} = 0 \\ \frac{\partial J}{\partial u^{n-(m+1)}} &= \Lambda^{n-mT} \frac{\partial G^{n-m}}{\partial u^{n-(m+1)}} + \Lambda^{n-(m+1)T} \frac{\partial G^{n-(m+1)}}{\partial u^{n-(m+1)}} = 0 \\ & ... \\ \frac{\partial J}{\partial u^1} &= \Lambda^{2T} \frac{\partial G^2}{\partial u^1} + \Lambda^{1T} \frac{\partial G^1}{\partial u^1} = 0 \end{aligned} \tag{46}$$

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

$$\left[\frac{\partial G^n}{\partial u^n}\right]^T \Lambda^n = -\left[\frac{\partial L}{\partial u^n}\right]^T \tag{47}$$

using the other adjoint equations we get an adjoint recurrence relation for k = 2, 3, ...,m:

$$\frac{\partial G^{k-1}}{\partial u^{k-1}}^T \Lambda^{k-1} = -\frac{\partial G^k}{\partial u^{k-1}}^T \Lambda^k \tag{48}$$

with the adjoint recurrence relation for k = m+1, ..., n-1 as follows.

$$\frac{\partial G^{k-1}}{\partial u^{k-1}}^T \Lambda^{k-1} = -\frac{\partial G^k}{\partial u^{k-1}}^T \Lambda^k - \left[\frac{\partial L}{\partial u^{k-1}}\right]^T \tag{49}$$

If the objective function is only dependent on the final time-step we only have one recurrence relation for all k = 2, 3, ..., n-1:

$$\frac{\partial G^{k-1}}{\partial u^{k-1}}^T \Lambda^{k-1} = -\frac{\partial G^k}{\partial u^{k-1}}^T \Lambda^k \tag{50}$$

Lastly, we can take the derivative of equation (44) with respect to the design variables to get the sensitivity equation.

$$\frac{dJ}{dD} = \frac{\partial L}{\partial D} + \Lambda^{nT}\frac{\partial G^n}{\partial D} + \Lambda^{n-1T}\frac{\partial G^{n-1}}{\partial D} + \Lambda^{n-2T}\frac{\partial G^{n-2}}{\partial D} + ... \tag{51}$$

IV.A. Adjoint Computed Sensitivites for Newton-Chord Method

We refer once again to equation (7)

$$u^k = u^{k-1} - [P_{k-1}]^{-1} R \tag{52}$$

We move all terms to one side and obtain the following equation as the constraint.

$$G^k(u^k(D), u^{k-1}(D), D) = u^k - u^{k-1} + [P_{k-1}]^{-1} R(u^{k-1}) = 0 \tag{53}$$

We have the preconditioner matrix defined as seen previously in (8):

$$[P_k] = \left[\frac{\partial R(u^k)}{\partial u^k}\right]_1 + \frac{vol}{\Delta t^k CFL^k} \tag{54}$$

We then take the derivatives of our constraint equations. These are written as follows.

$$\begin{aligned}\frac{\partial G^k}{\partial u^k} &= I \\ \frac{\partial G^k}{\partial u^{k-1}} &= -I + [P_{k-1}]^{-1}\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 + \frac{\partial\,[P_{k-1}]^{-1}}{\partial u^{k-1}} R(u^{k-1}) \\ \frac{\partial G^k}{\partial D} &= [P_{k-1}]^{-1}\frac{\partial R(u^{k-1})}{\partial D} + \frac{\partial\,[P_{k-1}]^{-1}}{\partial D} R(u^{k-1})\end{aligned} \tag{55}$$

By using the same logic as in the pseudo-time accurate Newton-Chord tangent method, we choose to neglect terms multiplied by the residual and simplify the constraint derivatives to those below. As in the tangent system we are now making only an approximation of the exact sensitivity and error equations, with the goal 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.

$$\begin{aligned}\frac{\partial G^k}{\partial u^k} &= I \\ \frac{\partial G^k}{\partial u^{k-1}} &= -I + [P_{k-1}]^{-1}\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 \\ \frac{\partial G^k}{\partial D} &= [P_{k-1}]^{-1}\frac{\partial R(u^{k-1})}{\partial D}\end{aligned} \tag{56}$$

Using the equation for the adjoint at the final pseudo-time step with our constraint derivatives we get the same initial source term.

$$[I]\,\Lambda^n = -\left[\frac{\partial L}{\partial u^n}\right]^T \tag{57}$$

Substituting in the constraint derivatives into equation (50) returns:

$$[I]\,\Lambda^{k-1} = -\left[-I + [P_{k-1}]^{-1}\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2\right]^T \Lambda^k \tag{58}$$

we can also write this recurrence relation in delta form so that it more closely follows the analysis problem. To this end, we define a $\Delta\Lambda$ such that $\Lambda^{k-1} = \Lambda^k + \Delta\Lambda$. Which gives the recurrence relation as:

$$\Delta\Lambda = -\left[[P_{k-1}]^{-1}\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2\right]^T \Lambda^k \tag{59}$$

distributing the transpose allows us to rewrite the equation.

$$\Delta\Lambda = -\left[\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2\right]^T [P_{k-1}]^{-T}\,\Lambda^k \tag{60}$$

This motivates us to define a secondary adjoint variable for each recurrence relation:

$$[P_{k-1}]^T\,\psi^k = \Lambda^k \tag{61}$$

we then rewrite the delta form of the adjoint recurrence relation as follows.

$$\Delta\Lambda = \left[\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2\right]^T \psi^k \tag{62}$$

It is important to note that, as in the tangent mode, to obtain exact correspondance, these linear systems must be solved as the exact dual of the analysis solve, as we are attempting to transpose exactly the analysis solve. This secondary adjoint variable will then be used in the sensitivity equation (51) and we obtain:

$$\frac{dJ}{dD} = \frac{\partial L}{\partial D} + \psi^{nT}\frac{\partial R(u^{n-1})}{\partial D} + \psi^{n-1T}\frac{\partial R(u^{n-2})}{\partial D} + ... + \psi^{1T}\frac{\partial R(u^0)}{\partial D} \tag{63}$$

IV.B. Adjoint Computed Sensitivites for quasi-Newton Method

For this section we begin with the derivatives of our constraint equations as shown in equation (55), before the simplification shown for the Newton-Chord method. These are written as follows.

$$\begin{aligned}
\frac{\partial G^k}{\partial u^k} &= I \\
\frac{\partial G^k}{\partial u^{k-1}} &= -I + [P_{k-1}]^{-1}\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 + \frac{\partial\,[P_{k-1}]^{-1}}{\partial u^{k-1}} R(u^{k-1}) \\
\frac{\partial G^k}{\partial D} &= [P_{k-1}]^{-1}\frac{\partial R(u^{k-1})}{\partial D} + \frac{\partial\,[P_{k-1}]^{-1}}{\partial D} R(u^{k-1})
\end{aligned} \tag{64}$$

By using the differentiation of a matrix inverse shown in equation (36) we can obtain the derivative of the constraint term shown below:

$$\begin{aligned}
\frac{\partial G^k}{\partial u^k} &= I \\
\frac{\partial G^k}{\partial u^{k-1}} &= -I + [P_{k-1}]^{-1}\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 - [P_{k-1}]^{-1}\frac{\partial\,[P_{k-1}]}{\partial u^{k-1}}[P_{k-1}]^{-1} R(u^{k-1}) \\
\frac{\partial G^k}{\partial D} &= [P_{k-1}]^{-1}\frac{\partial R(u^{k-1})}{\partial D} - [P_{k-1}]^{-1}\frac{\partial\,[P_{k-1}]}{\partial D}[P_{k-1}]^{-1} R(u^{k-1})
\end{aligned} \tag{65}$$

Using the definition of the nonlinear solver increment we can simplify the above equation with:

$$\begin{aligned}
\frac{\partial G^k}{\partial u^k} &= I \\
\frac{\partial G^k}{\partial u^{k-1}} &= -I + [P_{k-1}]^{-1}\left[\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 - \frac{\partial\,[P_{k-1}]}{\partial u^{k-1}}\Delta u\right] \\
\frac{\partial G^k}{\partial D} &= [P_{k-1}]^{-1}\left[\frac{\partial R(u^{k-1})}{\partial D} - \frac{\partial\,[P_{k-1}]}{\partial x}\frac{dx}{dD}\Delta u\right]
\end{aligned} \tag{66}$$

Please note that we can compute these hessian vector products using complex frechet derivatives, rather than hand differentiating the residual operator twice to obtain the Hessian operator; even though we are in the adjoint mode, the hessian vector products are not transpose matrix vector products and can therefore be computed using frechet derivatives. Using the equation for the adjoint at the final pseudo-time step with our constraint derivatives we get the same initial source term as in the Newton-Chord formulation.

$$[I]\,\Lambda^n = -\left[\frac{\partial L}{\partial u^n}\right]^T \tag{67}$$

Substituting in the constraint derivatives from equation (66) into equation (50) returns:

$$[I]\,\Lambda^{k-1} = -\left[-I + [P_{k-1}]^{-1}\left[\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 - \frac{\partial\,[P_{k-1}]}{\partial u^{k-1}}\Delta u\right]\right]^T \Lambda^k \tag{68}$$

we can write this recurrence relation in delta form as in the Newton-Chord formulation.

$$\Delta\Lambda = -\left[[P_{k-1}]^{-1}\left[\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 - \frac{\partial\,[P_{k-1}]}{\partial u^{k-1}}\Delta u\right]\right]^T \Lambda^k \tag{69}$$

Distributing the transpose returns:

$$\Delta\Lambda = -\left[\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 - \frac{\partial\,[P_{k-1}]}{\partial u^{k-1}}\Delta u\right]^T [P_{k-1}]^{-T}\,\Lambda^k \tag{70}$$

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

$$[P_{k-1}]^T\,\psi^k = \Lambda^k \tag{71}$$

we then rewrite the delta form of the adjoint recurrence relation as follows.

$$\Delta\Lambda = -\left[\left[\frac{\partial R(u^{k-1})}{\partial u^{k-1}}\right]_2 - \frac{\partial\,[P_{k-1}]}{\partial u^{k-1}}\Delta u\right]^T \psi^k \tag{72}$$

It is important to note that, as noted before we do not need exact dual correspondence here between the adjoint solver and the tangent one. Regardless of duality this formulation will recover machine precision accuracy in the sensitivities in the limit of the linear system being solved to machine precision. We substitute the constraint derivatives from equation (66) into the sensitivity equation (51) and we obtain:

$$\frac{dJ}{dD} = \frac{\partial L}{\partial D} + \Lambda^{nT}\,[P_{n-1}]^{-1}\left[\frac{\partial R(u^{n-1})}{\partial D} - \frac{\partial\,[P_{n-1}]}{\partial x}\frac{dx}{dD}\Delta u\right] + \ldots + \Lambda^{1T}\,[P_0]^{-1}\left[\frac{\partial R(u^0)}{\partial D} - \frac{\partial\,[P_0]}{\partial x}\frac{dx}{dD}\Delta u\right] \tag{73}$$

We can refer back to our definition of the secondary adjoint variable to simplify this equation and remove an additional linear solve and get the equation below:

$$\frac{dJ}{dD} = \frac{\partial L}{\partial D} + \psi^{nT}\left[\frac{\partial R(u^{n-1})}{\partial D} - \frac{\partial\,[P_{n-1}]}{\partial x}\frac{dx}{dD}\Delta u\right] + \ldots + \psi^{1T}\left[\frac{\partial R(u^0)}{\partial D} - \frac{\partial\,[P_0]}{\partial x}\frac{dx}{dD}\Delta u\right] \tag{74}$$

We can note that the adjoint sensitivity formulation uses only one approximate linear solver per nonlinear step, like the forward and tangent solvers. Should we want to guarantee convergence of the adjoint problem, we can use the dual solver of the primal linear solver used at each nonlinear iteration.

V. Verification

In this section, we run on an unstructured triangular mesh consisting of 4212 elements shown in Figure (1). All verification cases were run in $M = .6$ flow with $\alpha = 1^o$. We use the tangent and adjoint computed sensitivities of the lift objective functional as proxies for the behavior of the tangent and adjoint systems, and as a means to examine their behavior, convergence, and verify their correctness. The design variables are equidistant Hicks-Henne bump functions and the mesh sensitivities were calculated with the spring analogy outlined previously.

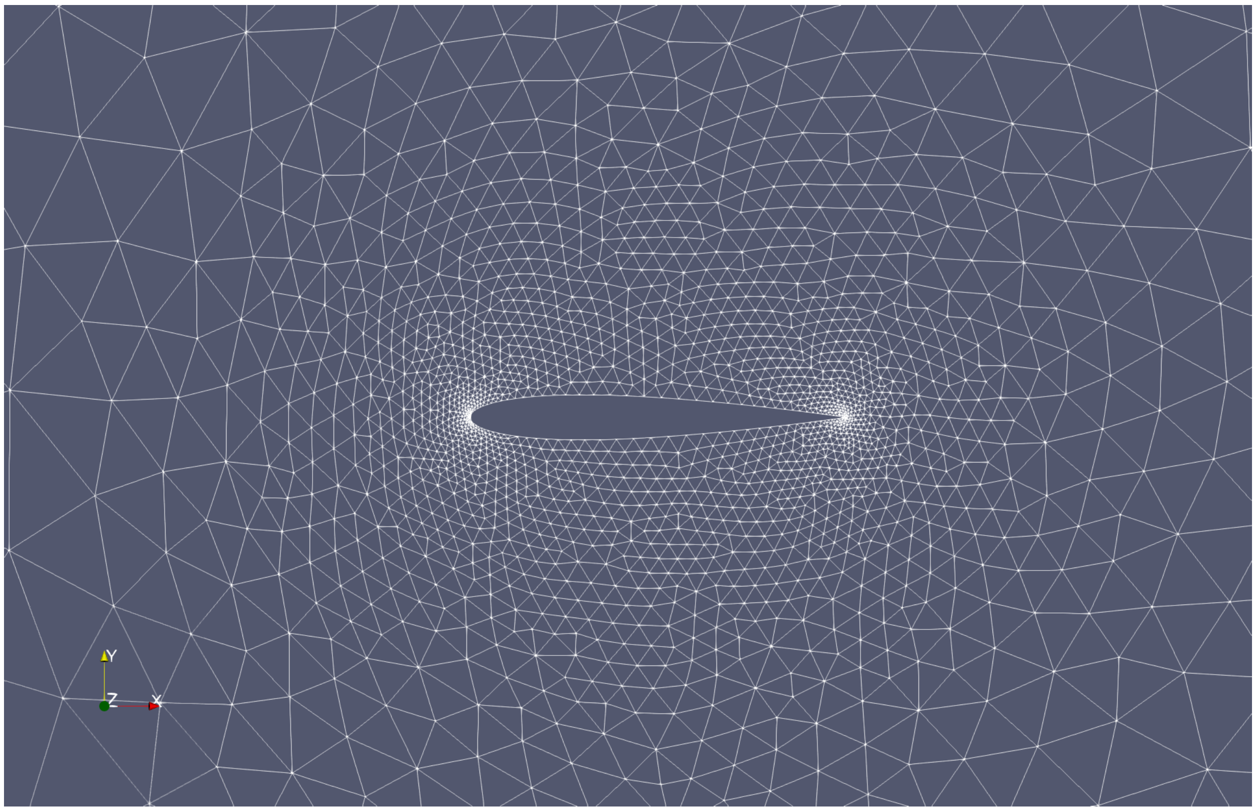


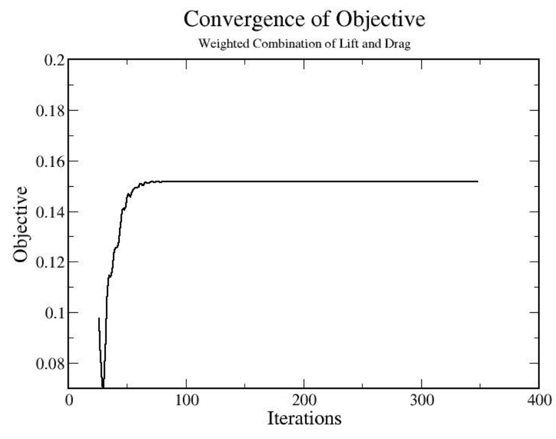
Figure 1. Computational mesh for NACA0012 airfoil

V.A. Tangent Verification

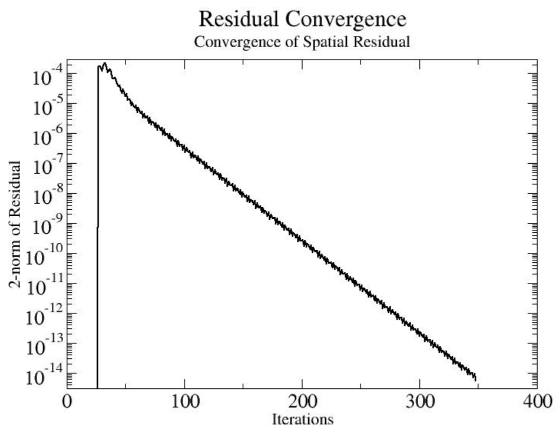
For the pseudo-time accurate tangent system verification we compare the complex step computed sensitivities to those provided by the tangent system. Since the pseudo-time accurate tangent formulation is designed to compute the exact derivative we expect to see exact correspondence between the two methods, i.e. differences of the order of machine zero.

V.A.1. Newton-Chord Method

The verification in this section is done by implementing a Newton-Chord method restarted at the 26th iteration. This verification is accomplished by starting the analysis problem at initial conditions and running a quasi-Newton algorithm for 25 steps, at which point we freeze the Jacobian and continue solving the analysis problem with the frozen Jacobian. The complex-step finite-difference sensitivities were calculated by restarting at the 26th iteration with the frozen Jacobian and introducing a complex perturbation to the design variables, therefore ensuring the complex perturbation does not affect the frozen Jacobian. 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 (32) 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. As we have taken the exact derivative of the process outlined in the analysis solution process, we expect exact correspondence between the complex step derivative and the pseudo-time accurate tangent sensitivity. The functional and residual behavior are depicted in Figure (2). Figure (3) plots the difference between the complex and pseudo-time accurate adjoint method sensitivities at each pseudo time step, and confirms that we obtain sensitivities which agree to machine precision.

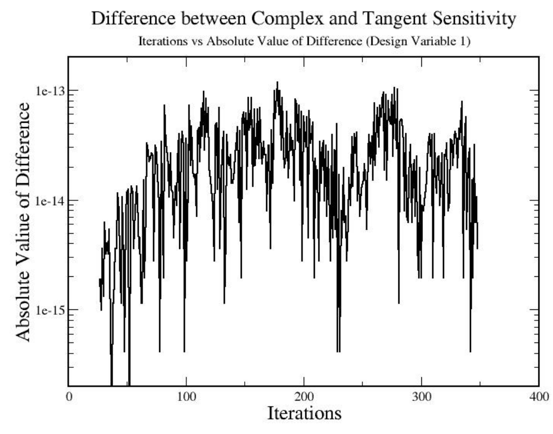


(a) Functional

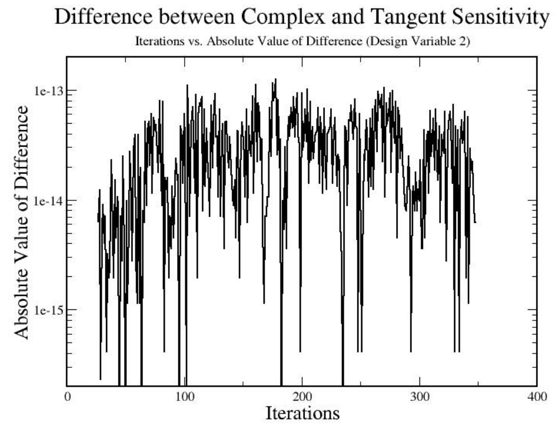


(b) Residual

Figure 2. Convergence of objective function and residual for Newton-Chord Method



(a) Design Variable 1



(b) Design Variable 2

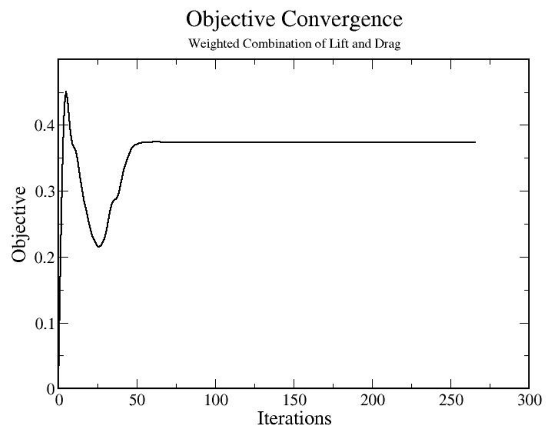
Figure 3. Difference between pseudo-time accurate tangent computed sensitivities and complex sensitivities for Newton-Chord method at each pseudo-time step

V.A.2. quasi-Newton Method

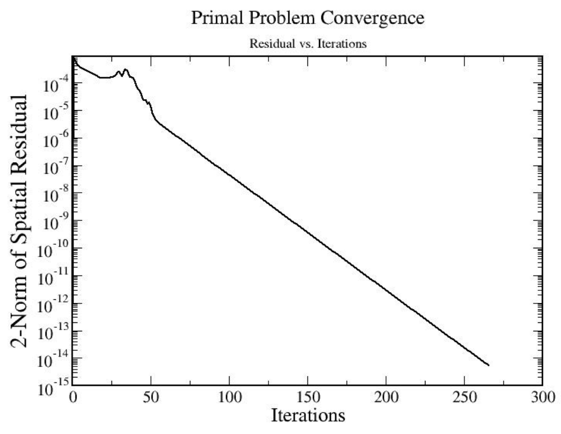
The verification in this section is done by using a very tight tolerance on the linear system of $1e-13$. The hessian vector products are computed using complex-step perturbation to the conservative variables and nodal coordinates as stated previously. We plot the convergence of the nonlinear residual, and the objective of the analysis system in Figure (4). We also plot the difference between the complex and pseudo-time accurate tangent method sensitivities at each pseudo-time step in Figure (5). At each step we obtain machine level correspondence between the complex-step and tangent computed sensitivities provided that the linear system is solved to machine precision; this is sufficient verification of the implementation.

V.B. Adjoint Verification

Because the adjoint computed 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, we have provided Table (1) showing the difference between the complex and adjoint provided sensitivities on truncated runs for verification purposes. The Newton-Chord (NC) and quasi-Newton inverse identity (QNII) adjoints and complex sensitivities are shown to verify the implementation. Please note that for the Newton-Chord formulation, we use a dual block jacobi smoother as the linear solver (for which the dual is trivial to

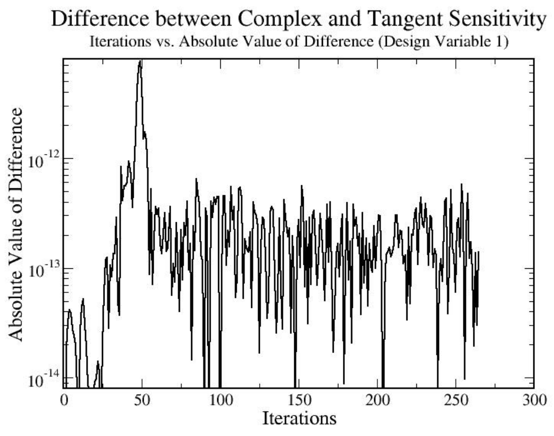


(a) Functional

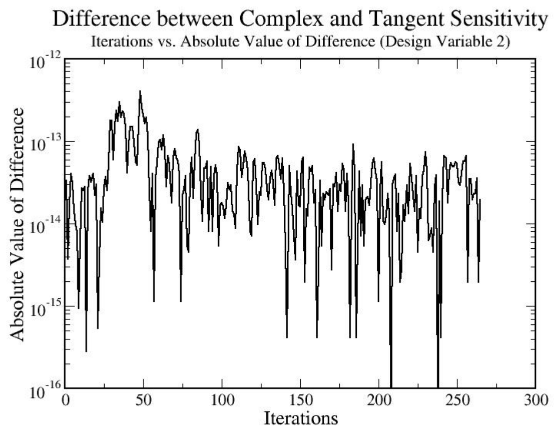


(b) Residual

Figure 4. Convergence of objective function and residual for quasi-Newton Method



(a) Design Variable 1



(b) Design Variable 2

Figure 5. Difference between pseudo-time accurate tangent computed sensitivities and complex sensitivities for quasi-Newton method at each pseudo-time step

implement), but for the quasi-Newton method we are not restricted to dual solvers, or even the same type of solver. Some tests were performed using BiCGStab for the analysis and tangent, and Gauss-Seidel for the adjoint to verify the theory, whose result are not shown here. In the table, we can see that regardless of the convergence of the residual, which we marked by reporting the L_2-norm of the spatial residual in the final column of the table, we obtain exact correspondence between the sensitivities provided by the adjoint and complex-step methods.

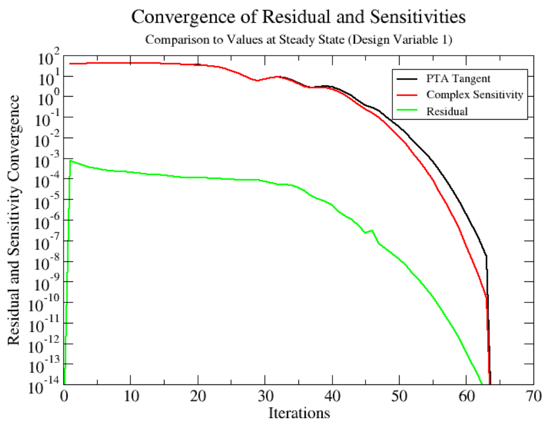
Scheme	Steps	Complex Sensitivity	Adjoint Sensitivity	$\|\|Residual\|\|_2$
NC	10	-2.3443155620267**46**	-2.3443155620267**35**	0.3205858355993253E-03
NC	20	-5.11707061016280**4**	-5.51170706101628**07**	0.2522158472255535E-03
QNII	10	-3.4821964390757**67**	-3.4821964390757**86**	0.2982633094835574E-03
QNII	20	-2.2644939350717**75**	-2.2644939350717**03**	0.1691766297621063E-03

Table 1. Comparison of adjoint and complex-step computed sensitivities

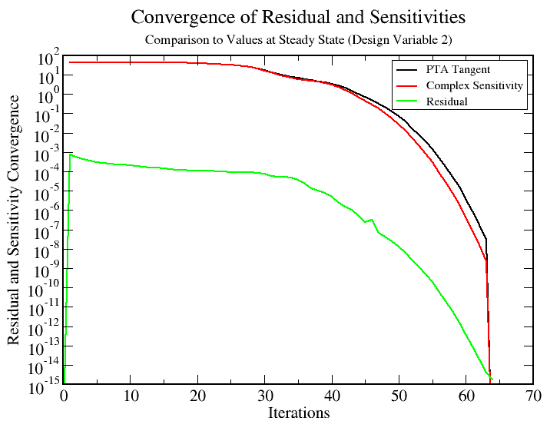
VI. Investigation of Sensitivity Computation

VI.A. Impact of Appromixate Linear Solves on Sensitivity Accuracy

This section shows the effect of an approximate linear solution on the accuracy of the identity of the differentiation of the inverse matrix in the sensitivity computation shown in equation (36). We can see in Figure (6) 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. We can then see in Figure (7), which depicts the maximum difference between the complex and tangent sensitivities over the iteration history of the analysis solution process, 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. We can see that 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 would also correspond to the steady-state tangent and adjoint computed sensitivities linearized about the converged analysis state.



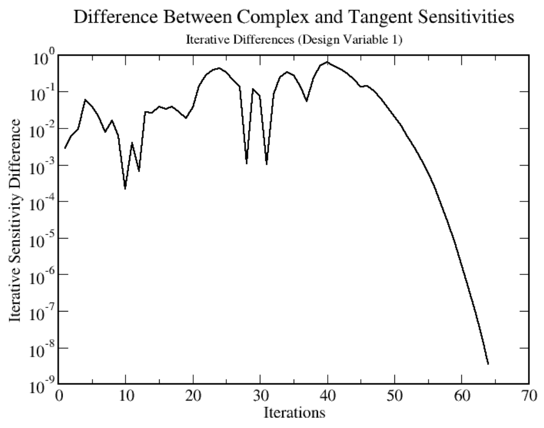
(a) Design Variable 1



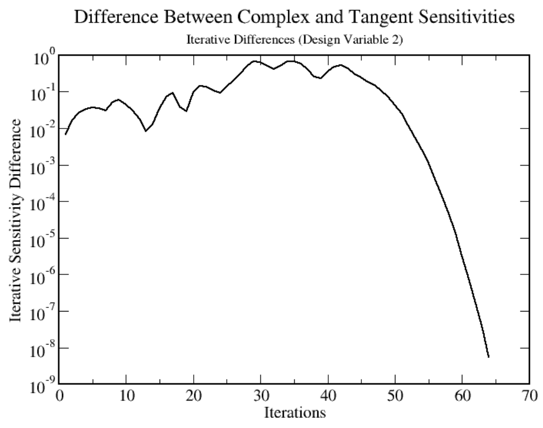
(b) Design Variable 2

Figure 6. Sensitivity convergence for linear tolerance, $1e-1$: difference between current and final sensitivity values

Figure (8) contains the same information as Figure (6), and Figure (7) is the sister plot of Figure (9) but the two later plots show resuls with a tighter linear system tolerance of $1e-4$. We can see that as before the maximum iterative difference is again on the order of the linear system tolerance, and that 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, we simulate the analysis problem with linear tolerances at every order from $1e-1$ to $1e-13$, and plot the maximum iterative difference in Figure (10). 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, but it is a function of the linear tolerance and the convergence of the non-linear problem.

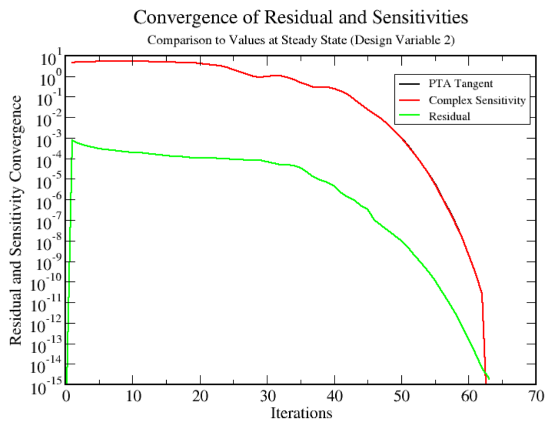


(a) Design Variable 1

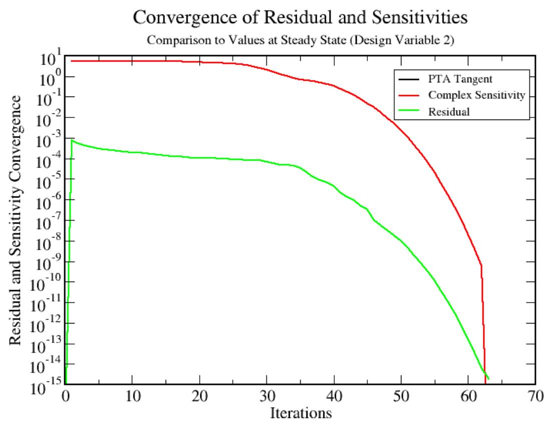


(b) Design Variable 2

Figure 7. Iterative sensitivity difference for linear tolerance, $1e-1$

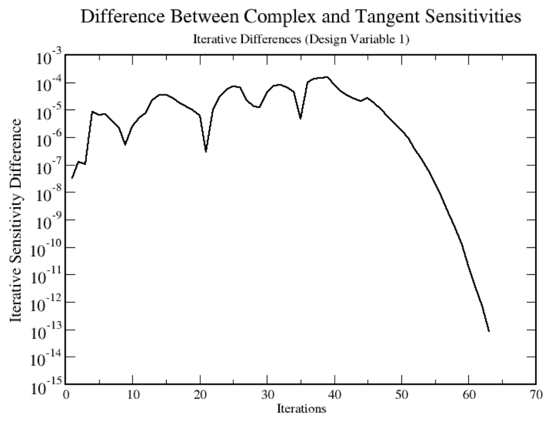


(a) Design Variable 1

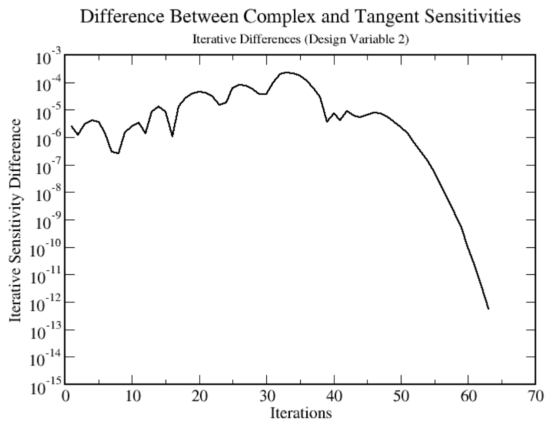


(b) Design Variable 2

Figure 8. Sensitivity convergence for linear tolerance, $1e-4$: difference between current and final sensitivity values

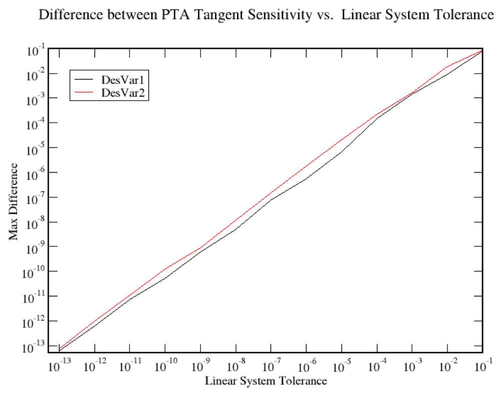


(a) Design Variable 1

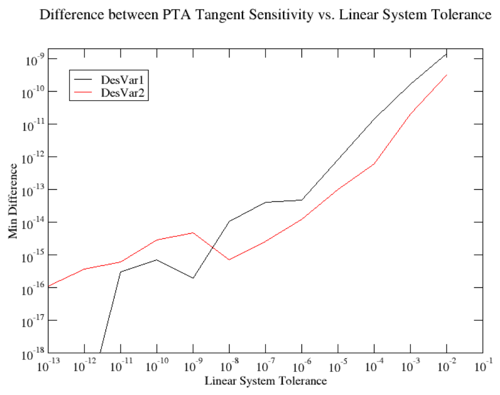


(b) Design Variable 2

Figure 9. Iterative sensitivity difference for linear tolerance, $1e-4$



(a) Maximum Iterative Difference

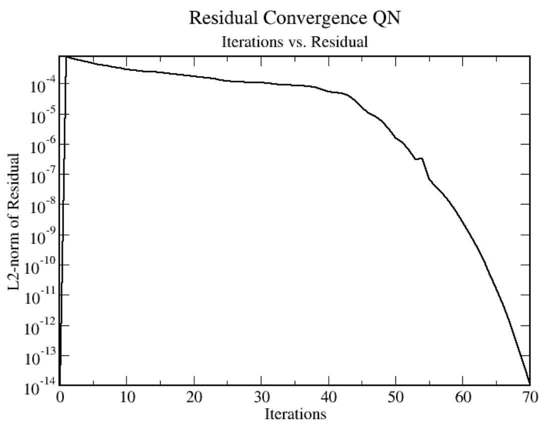


(b) Minimum Iterative Difference

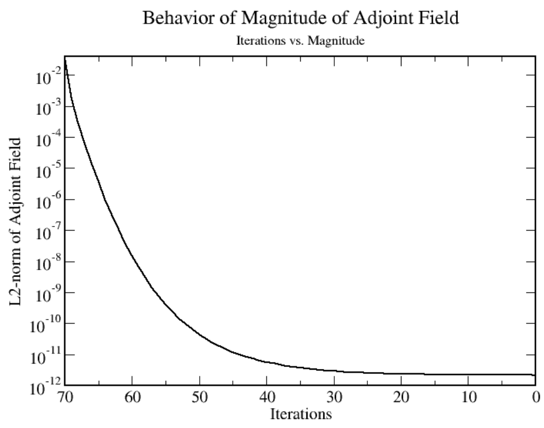
Figure 10. Iterative difference vs. linear tolerance

VI.B. Iterative Dependence of Sensitivity Computation

Given that the adjoint variable at a given iteration is the dependence of the objective function at the function evaluation iteration to a perturbation at the adjoint computation iteration, we can learn a great deal from looking at the behavior of the adjoint as measured by the magnitude of the L_2-norm of the adjoint field variables. We show in Figure (11) that the magnitude of the norm of the adjoint field behaves like the inverse of the analysis problem convergence. We can see that the adjoint magnitude, plotted on a reversed x-axis, is greatest at convergence of the analysis problem and near machine zero at initialization. Furthermore, we can see that the adjoint magnitude is negligible during the region dominated by the pseudo-transient behavior early on in the analysis problem simulation and that it is only significant during the region dominated by the quadratic convergence behavior. Figure (12) 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 differentation of the analysis solve is not necessary, rather the differentiation algorithm could be run only through the quadratic convergence region of the simulation.

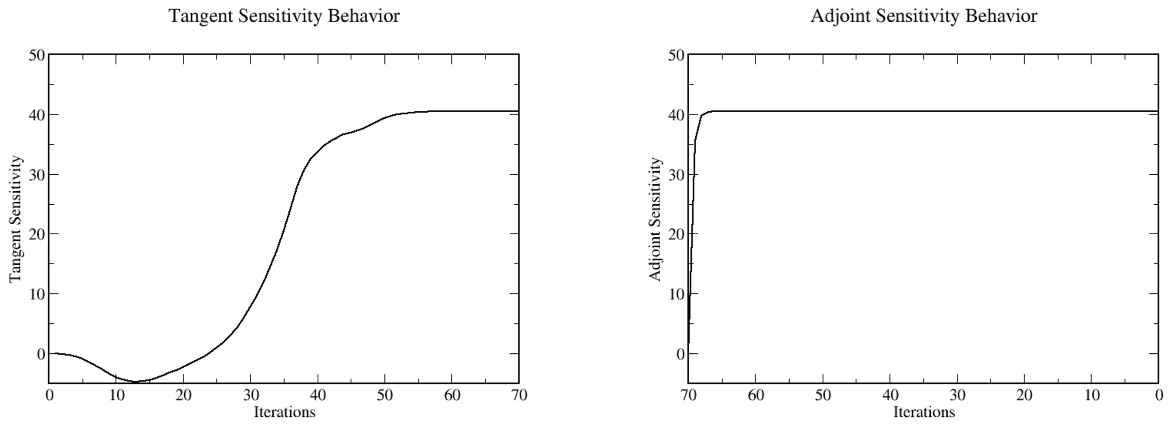


(a) Residual Convergence



(b) Adjoint Magnitude Behavior

Figure 11. Residual convergence and adjoint magnitude behavior for Quasi-Newton scheme



(a) Tangent Sensitivity Convergence

(b) Adjoint Sensitivity Convergence

Figure 12. Sensitivity convergence for Quasi-Newton scheme

VII. Optimization Results

VII.A. Optimization of Symmetric Airfoil with Detached Bow Shock

The case presented in this section is a NACA0012 airfoil shown in Figure (1) with symmetric design variables in $M = 1.25$ flow with $\alpha = 3^o$. The objective function here is a iteration averaged composite objective function of lift, drag, and entropy, where $m = 75$.

$$L = \sum_{i=n-m}^{n} \omega_L(c_L - C_{L_T})_i^2 + \omega_D(c_D - C_{D_T})_i^2 + \omega_s(s - s_T)_i^2 \tag{75}$$

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 basline 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 try to keep the lift constant, minimize the drag, and decrease the shock strength. The limiter used here to prevent divergence is the modified VK limiter presented in the beginning of the paper. The nonlinear convergence plot in Figure (13) shows that the convergence has stalled; for this case, the linear system in the Newton-Krylov solver is converged 5 orders of magnitude at each nonlinear iteration. If we run the adjoint formulation and look at the accumulation of the sensitivities as we integrate backwards in pseudo-time, we see these sensitivities are very well behaved, shown in Figure (14). This indicates that we could only partially integrate backwards in time, and optimize based off those sensitivities. We do so in this work, and average the objective function over the last 75 iterations and integrate backwards only through that averaging window to calculate the sensitivities.

The summary of the design cycle is in Figure (15), 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.

When we look at the density and mach number fields in Figure (16) we can see that there is a lower flow speed-up on the back half of the suction side of the airfoil, this lowers the pressure differential and decreases the drag.

We then can compare this adjoint based optimization to one that uses complex sensitivities to drive the optimization. Since we want the sensitivity of the process and the complex sensitivities are these sensitivities we compare the results of the two optimizations to get a sense of how much we are harmed by using this approximation of the derivative of the linear system solve, for both wide and narrow design variable bounds.

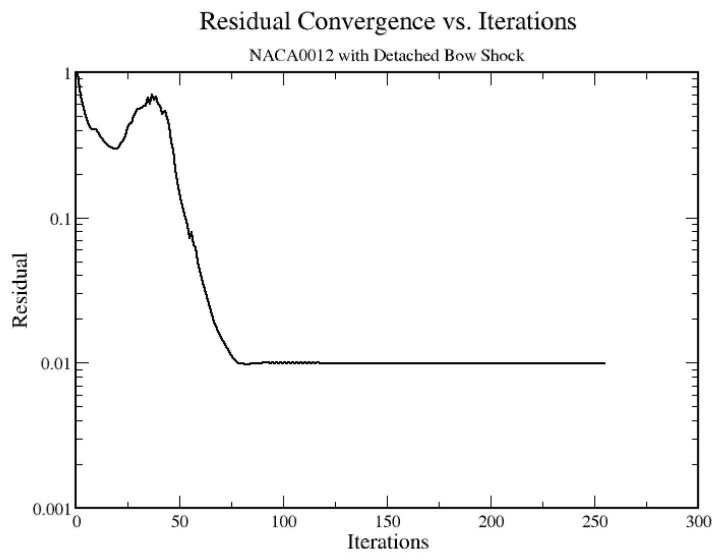


Figure 13. Analysis convergence plot

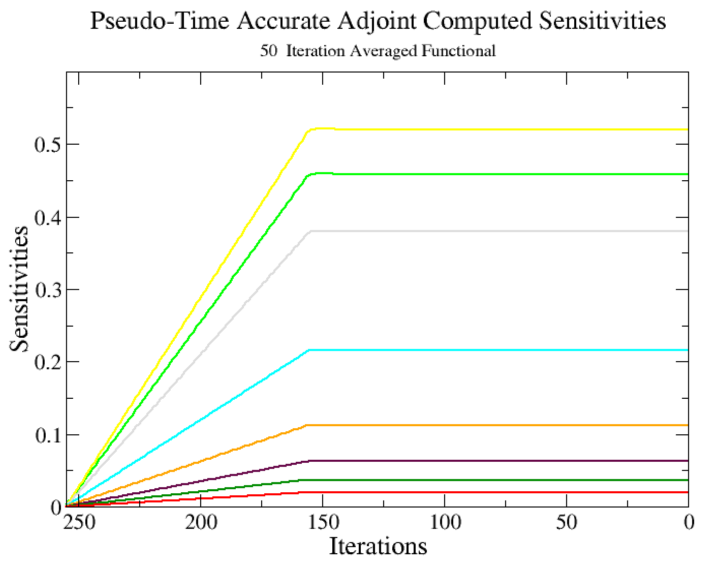
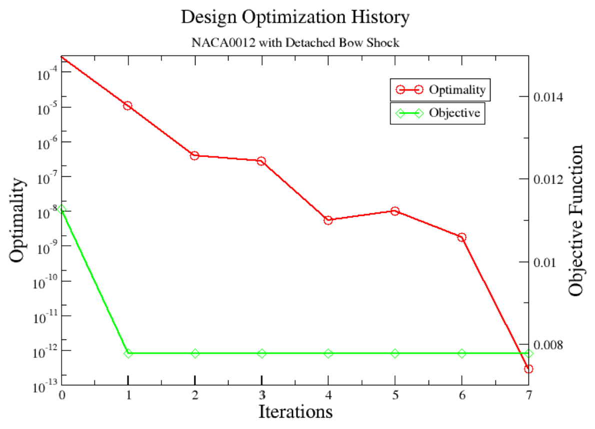


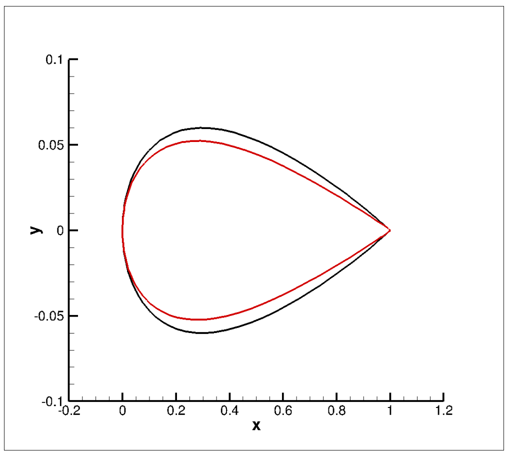
Figure 14. Backwards in iteration-space integretation of sensitivites

The bounds refer to the maximum and minimum 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.

Table (2) shows very close correspondence between the complex and adjoint optimizations; the 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 all at the bounds, the 20th design variable (the aftmost one) is not at the bound but has a machine zero gradient; so the optimality condition is satisfied. We also ran this case 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



(a) Design Cycle Convergence



(b) Airfoil Comparison

Figure 15. Design cycle summary

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 2. Comparison of adjoint and complex-step optimizations

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 far more restrictive tolerance than what is typically required, especially for inexact Newton solvers. 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 sensitivities.

VII.A.1. Investigation of Linear Tolerance on Design Optimization

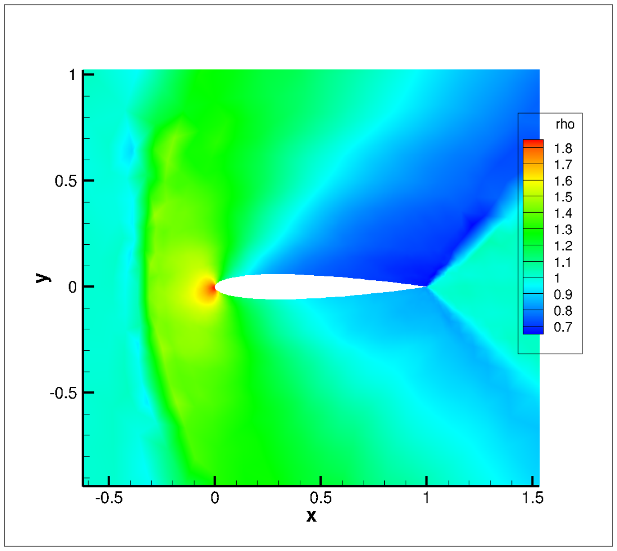
Tolerance	Bounds	Complex Optimality	Adjoint Optimality	Complex Objective	Adjoint Objective
1e-2	Narrow	1.7E-12	2.9E-13	7.76855**83E-03**	7.76855**33E-03**
1e-3	Narrow	2.9E-12	3.0E-13	7.7685583E-03	7.7685583E-03
1e-4	Narrow	2.9E-13	7.8E-13	7.7685583E-03	7.7685533E-03

Table 3. Comparison of adjoint and complex-step optimizations

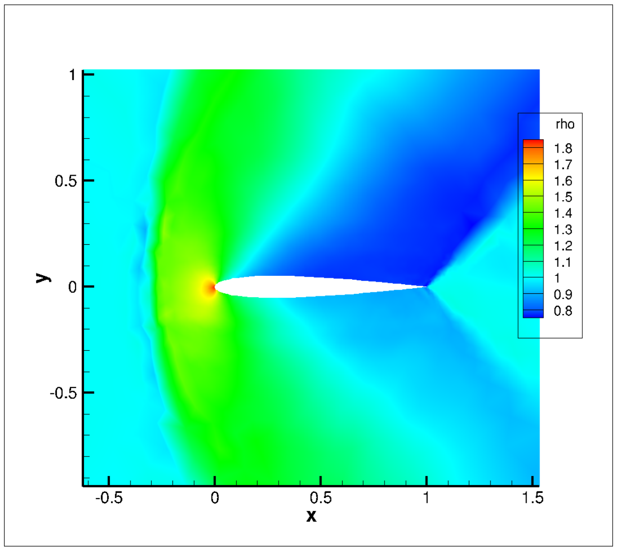
Table (3) shows that the linear tolerance has little effect on the adjoint sensitivities, as the adjoint sensitivity based optimizations were very similar to one another regardless of linear tolerance. This is an encouraging result, that allows further investigation into additional optimization cases with confidence. It is worth noting that the optimality is slightly better behaved for the adjoint rather than the complex sensitivities.The overall behavior indicates that using the pseudo-time adjoint calculated 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. We will do so in the following results sections.

VII.B. Optimization of Symmetric Airfoil with Trailing Edge Unsteadiness

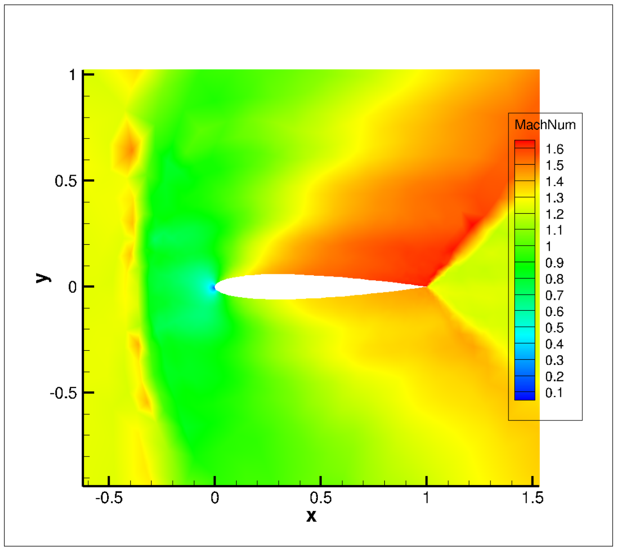
We then move onto a case with trailing edge unsteadiness in transonic flow. This case is unsteady because we simulate a NACA0012 with a truncated (at 97% of the chord) blunt trailing edge shown in Figure (17). This mesh went through 4 refinement cycles (using an in-house mesh refinement module) to get sufficient fineness near the trailing edge and shock location without exerting 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



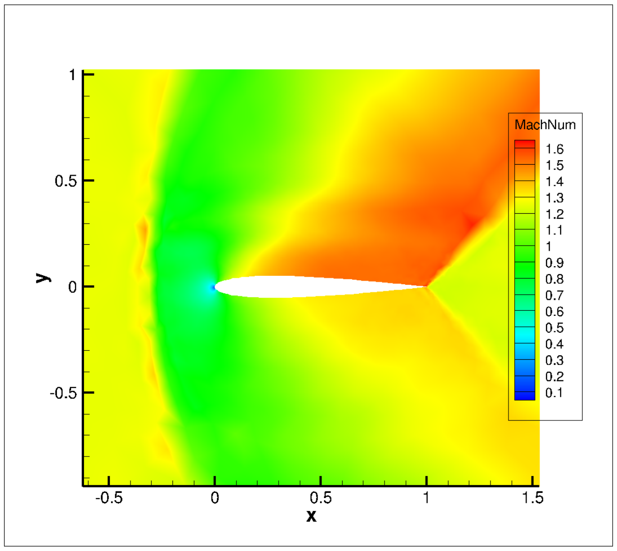
(a) Baseline Design Density



(b) Optimized Design Density



(c) Baseline Design Mach Number



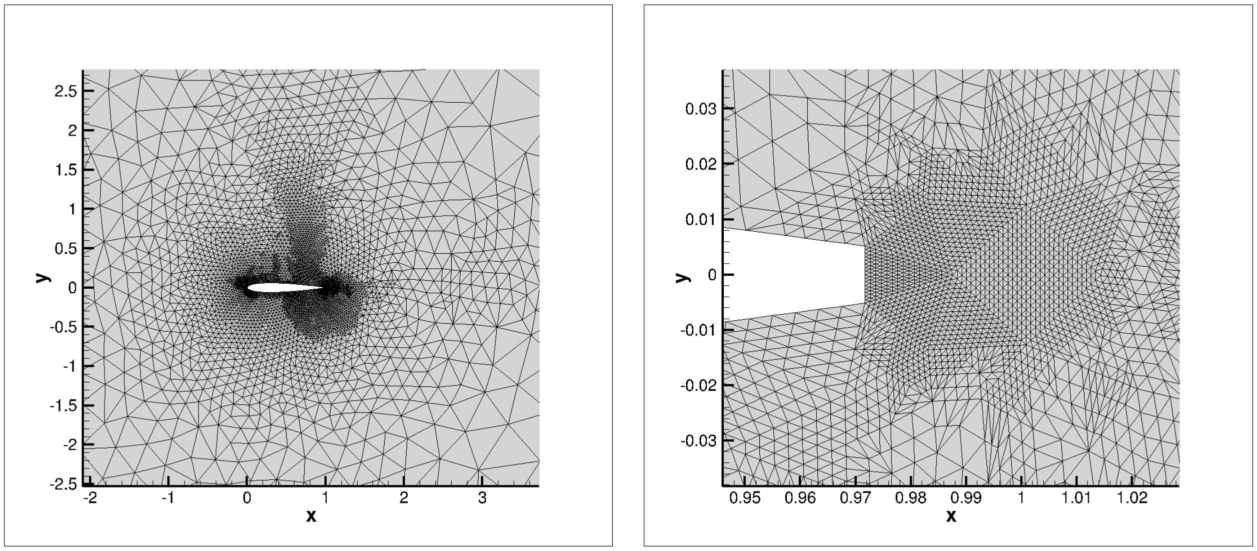
(d) Optimized Design Mach Number

Figure 16. Flow field comparison

$\alpha = 3^o$. The objective function here is a composite objective function of lift and drag averaged over the last 200 iterations:

$$L = \sum_{i=n-m}^{n} \omega_L (c_L - C_{L_T})_i^2 + \omega_D (c_D - C_{D_T})_i^2 \tag{76}$$

where we have targets for lift and drag, 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 limiter used here to prevent divergence is the modified VK limiter presented in algorithm (1). We can see that the nonlinear convergence is shown in Figure (18), which shows that the convergence has stalled and has small scale oscillations. The linear system from the Newton-Krylov solver is converged 4 orders of magnitude at each nonlinear iteration.
Looking at the design cycle summary we can see a large decrease in the functional even though the optimality condition is not satisfied to machine precision. We see in Figure (19) that the objective function is decreased to a factor of $\frac{1}{6}$ and the airfoil shows interesting behavior. The front 60% of the airfoil gets thinner, but



(a) Airfoil Mesh (b) Airfoil Trailing Edge

Figure 17. Truncated Airfoil Mesh

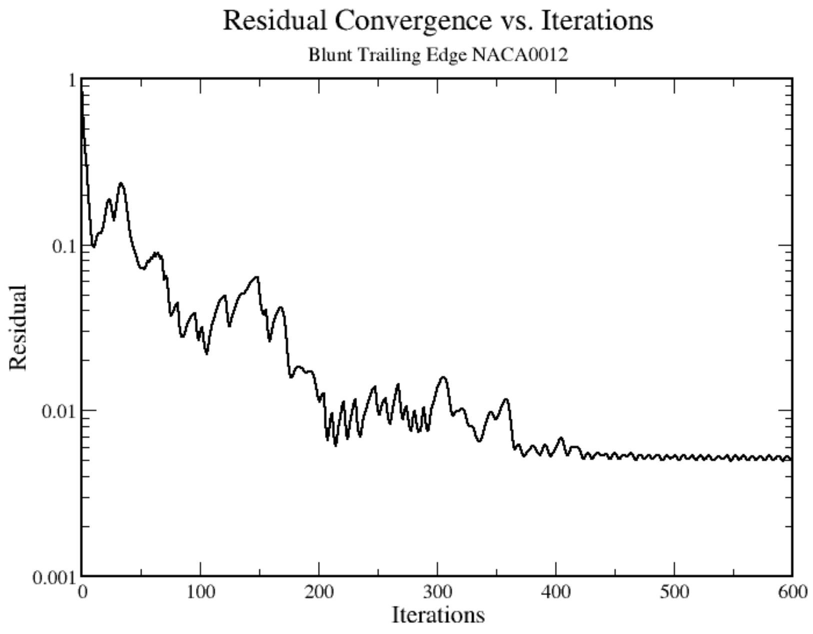
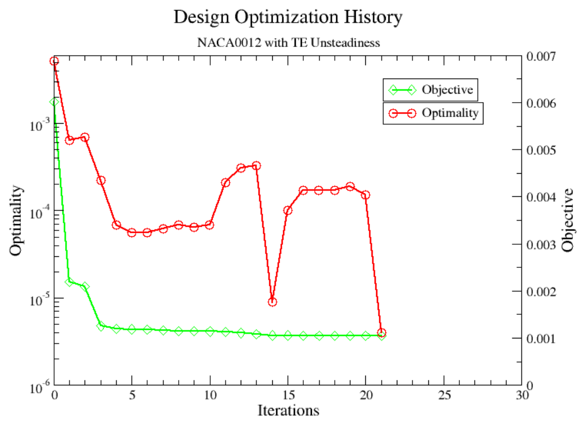


Figure 18. Analysis Convergence Plot

the rear 37% gets thicker. We hypothesize 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.[29]

When we look at the density field plots in Figure (20) at the final nonlinear iteration and compare the baseline to the optimized we can see the shock has gotten much weaker and moved further back along the airfoil.

The change in shock strength and location becomes clearer when we look at the mach field in Figure (21). We can also see that although the streamlines and shedding appears to be stronger in the optimized case than the baseline case, we are optimizing based off the last 200 nonlinear iterations, and that over that



(a) Design Cycle Convergence

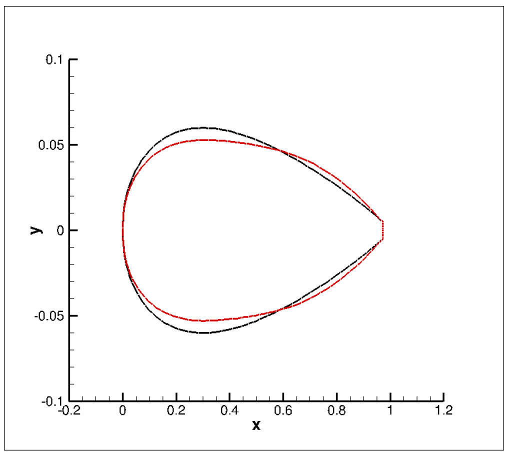
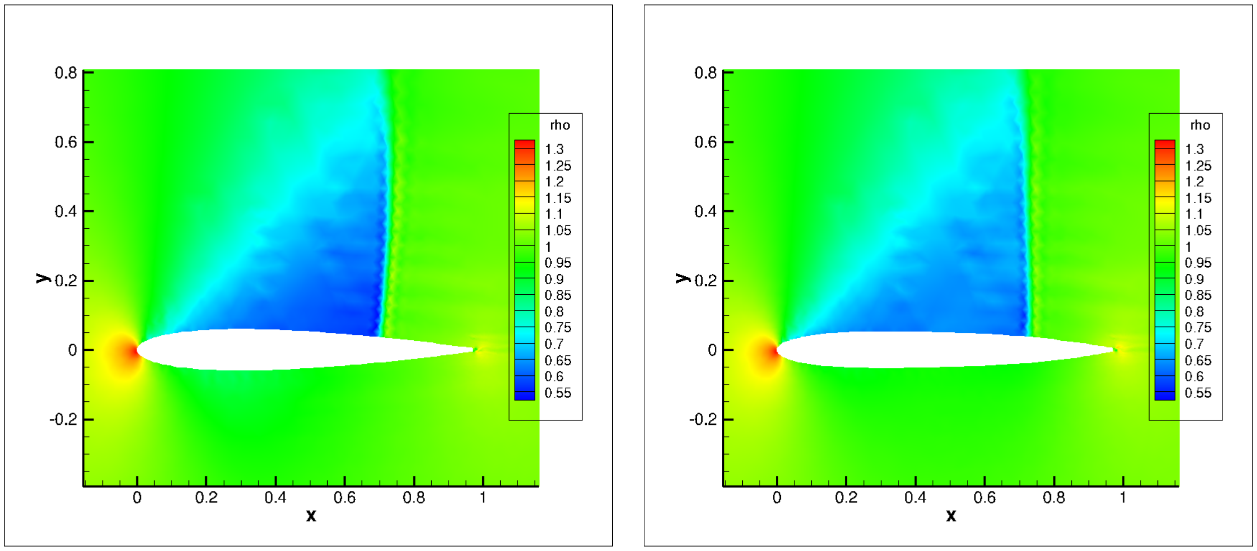

(b) Airfoil Comparison

Figure 19. Design cycle summary

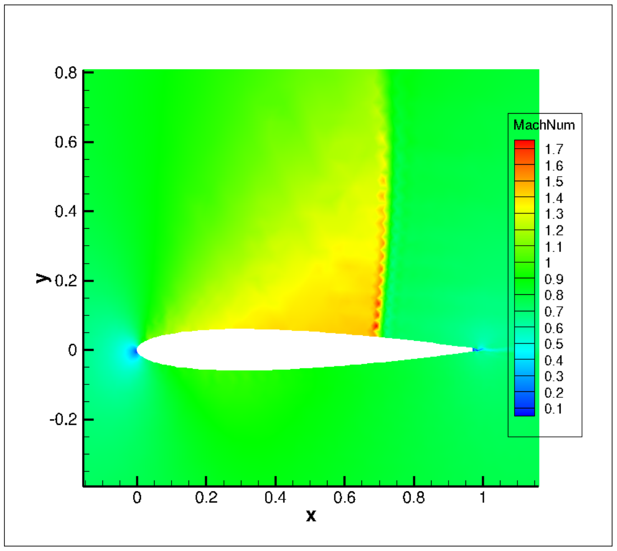


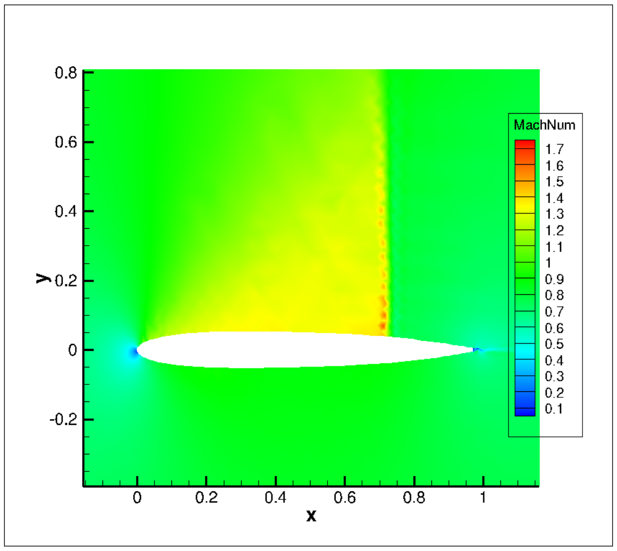
(a) Baseline Design

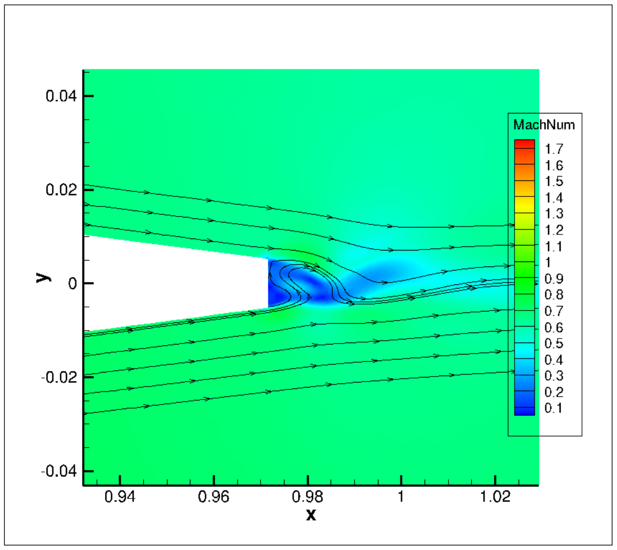
(b) Optimized Design

Figure 20. Density field comparison

window, the objective function average is far decreased, as we saw in Figure (19). When we ran using the steady adjoint as the source of the sensitivities as a comparison, in order for the optimizer to be successful we had to greatly increase the computational expense due to strengthening the linear solver and requiring more function evaluations. To obtain a successful optimization our preconditioned FGMRES linear solver used 25 krylov vectors with 10 restarts and 250 gauss-seidel smoothing iterations (per vector). The final design using the steady state adjoint had a higher optimality value and objective function (for a greater computational expense) than the optimization using the pseudo-time accurate approach.


(a) Baseline Design


(b) Optimized Design


(c) Baseline Trailing Edge with Streamlines

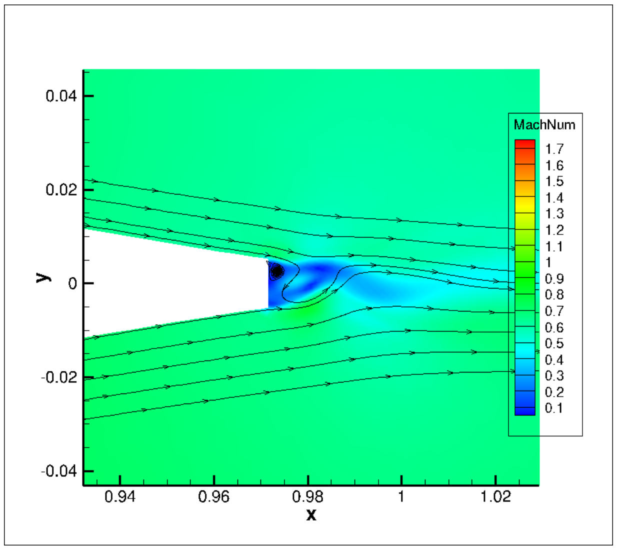

(d) Optimized Trailing Edge with Streamlines

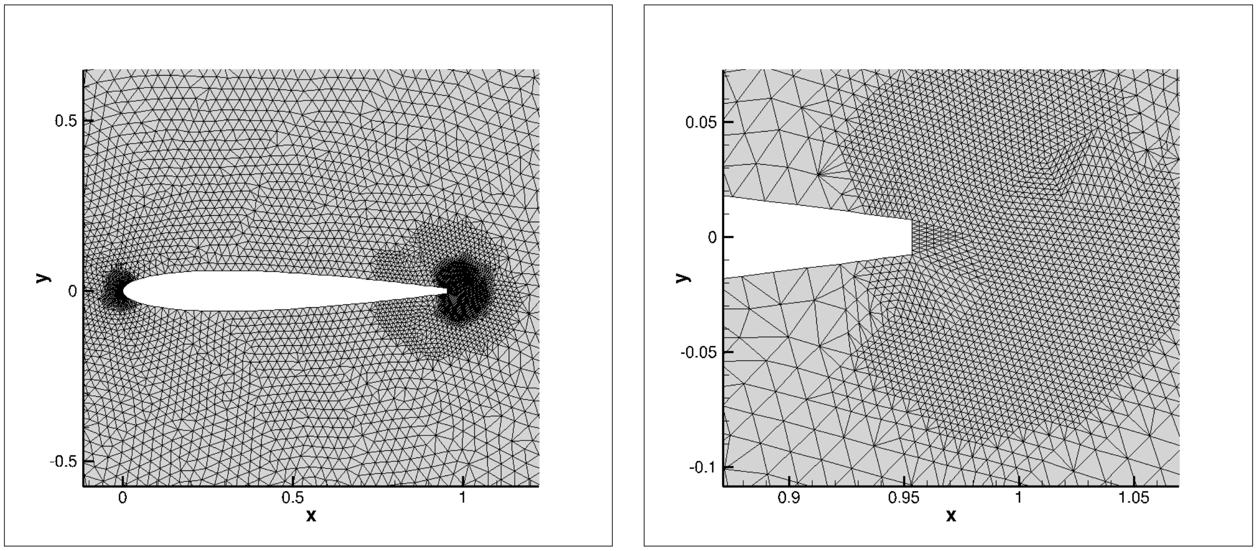
Figure 21. Mach field comparison

VII.C. Optimization of ADODG NACA0012 Airfoil with Trailing Edge Unsteadiness

The final case has a similar set up to the ADODG NACA0012 case[29] but 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) blunt trailing edge shown in Figure (22). This mesh went through 3 refinement cycles (using an in-house mesh refinement module) to get sufficient fineness near the trailing edge while minimizing overall computational expense. The case presented in this section is a NACA0012 airfoil with symmetric design variables in $M = 0.85$ flow with $\alpha = 0^o$. The objective function here is drag averaged over the last 200 iterations:

$$L = \sum_{i=n-m}^{n} c_{D_i} \tag{77}$$

This objective function will make the optimizer try to minimize the drag. On a symmetric structured mesh, lift would be equal to zero, in our unstructured case this is not the case. We chose to not optimize with a target lift of 0 because that would ask the optimizer to optimize based off the error. The limiter used here to prevent divergence is again the modified VK limiter presented in algorithm (1). Figure (23) shows



(a) Airfoil Mesh (b) Airfoil Trailing Edge

Figure 22. Truncated Airfoil Mesh

that convergence of the nonlinear problem has ceased and the analysis is stalled in limit cycle oscillations, the largest scale oscillations shown in this work. The linear system from the Newton-Krylov solver is, as in previous cases, converged 4 orders of magnitude at each nonlinear iteration.

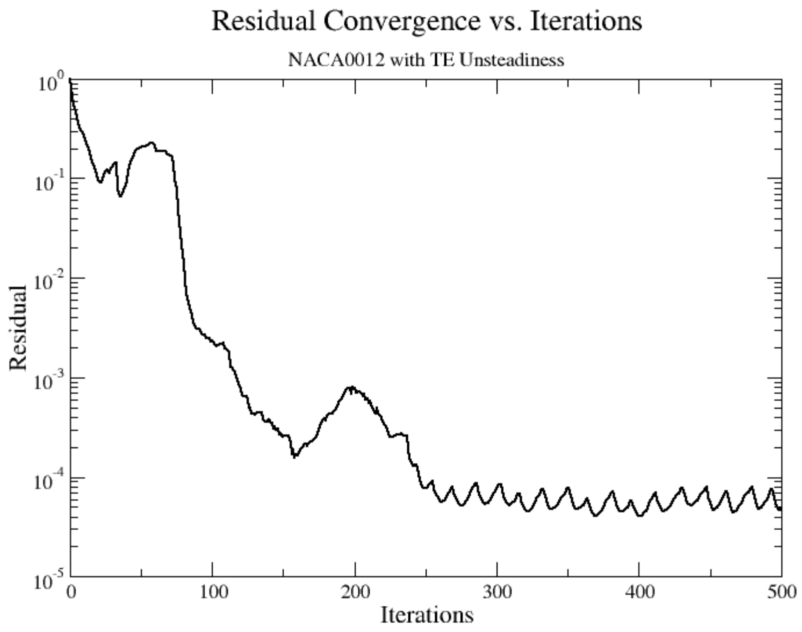
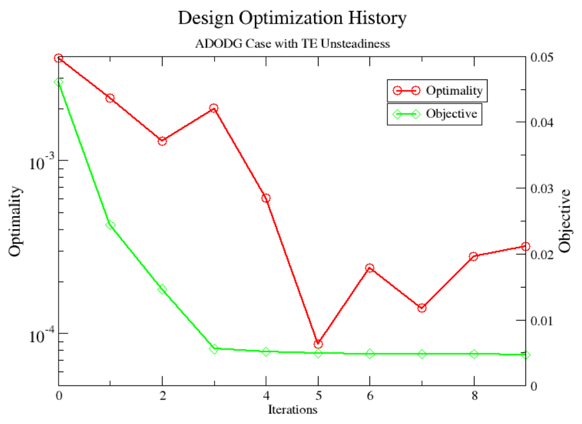
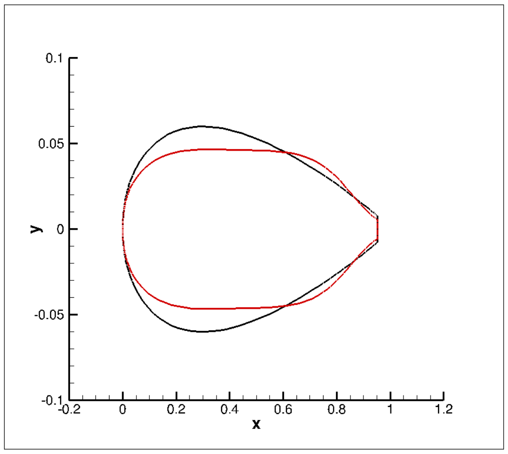


Figure 23. Analysis Convergence Plot

Looking at the design cycle summary in Figure (24) we can again see a large decrease in the functional even though the optimality condition is still not satisfied to machine precision. We see the objective function get decreased to a factor of $\frac{1}{10}$ and the airfoil shows interesting behavior. The front 60% of the airfoil gets thinner, but the rear 35% gets thicker, similarly to the previous trailing edge unsteadiness case. We hypothesize, as before, that this helps control the drag and shock behavior. Future cases could be run using the ADODG NACA0012 thickness constraints in the optimization to verify correspondance between that well-tested benchmark and this optimization methodology.



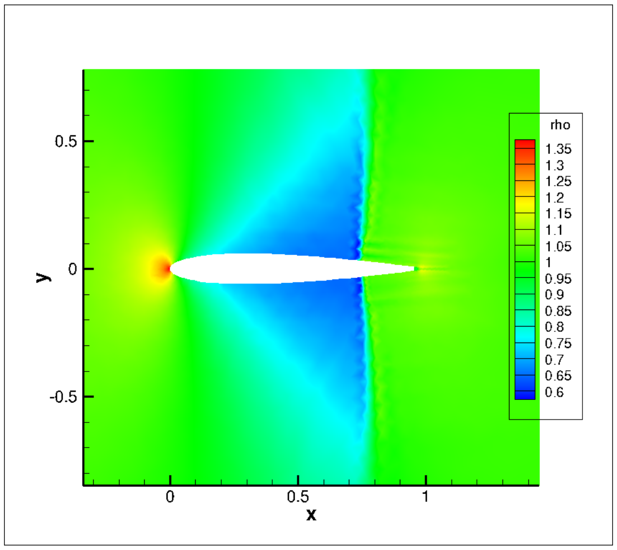
(a) Design Cycle Convergence



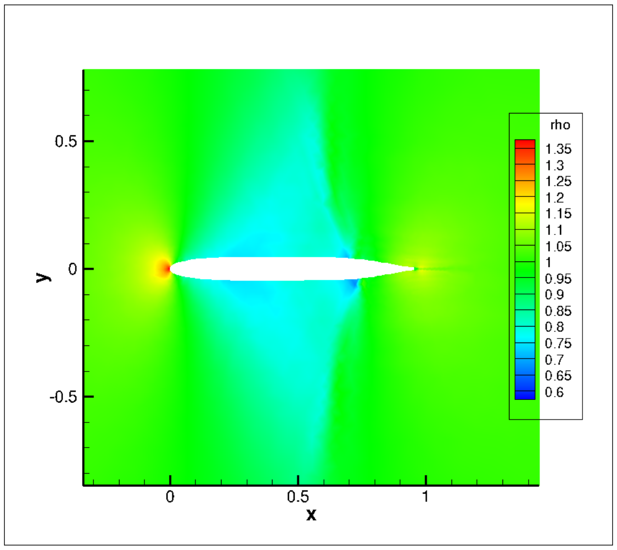
(b) Airfoil Comparison

Figure 24. Design cycle summary

When we look at the density field in Figure (25) at the final nonlinear iteration and compare the baseline to the optimized we can see that the shock has again gotten much weaker and moved further back along the airfoil as in the previous case.



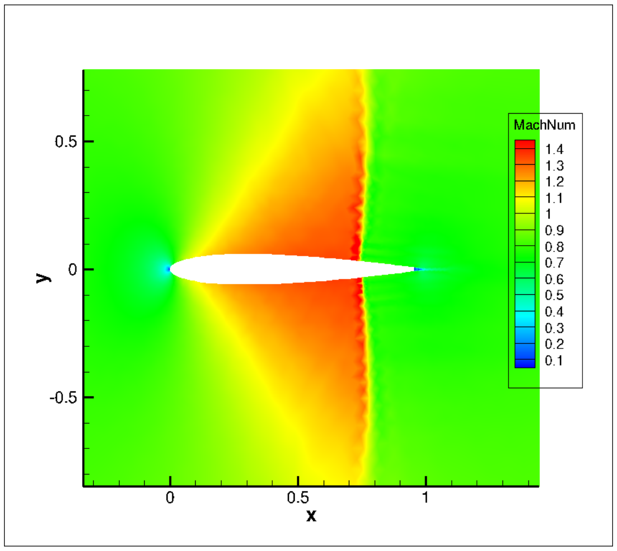
(a) Baseline Design



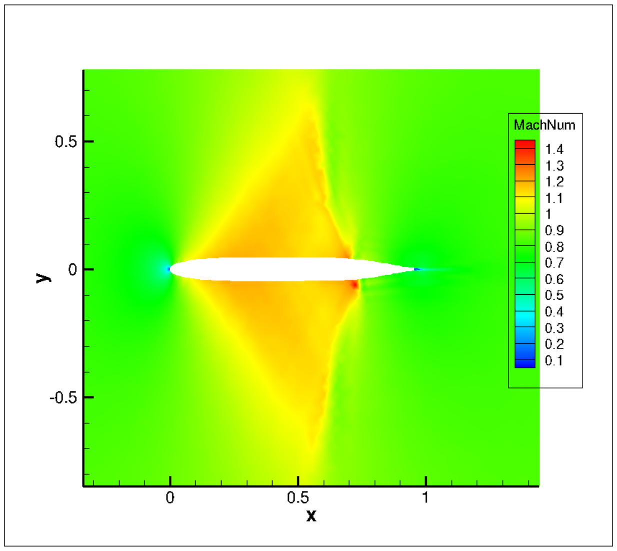
(b) Optimized Design

Figure 25. Density field comparison

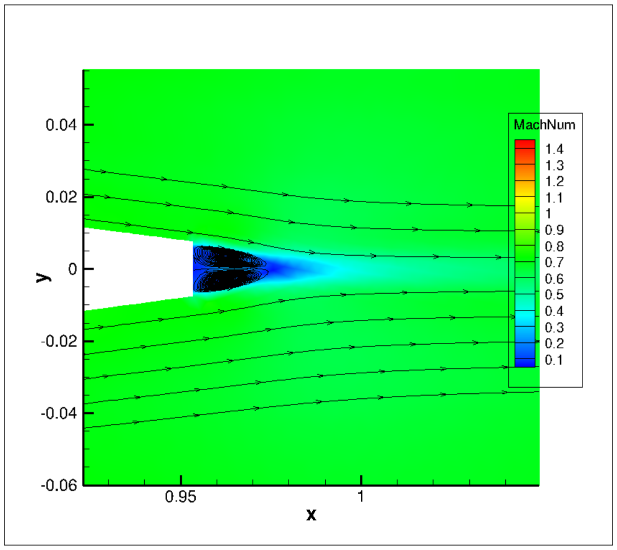
The change in shock strength and location becomes clearer when we look at the mach field in Figure (26). We can also there is little difference in the streamlines and shedding between the optimized case and the baseline case, and the shock – which is the primary driver of the drag – has nearly disappeared.



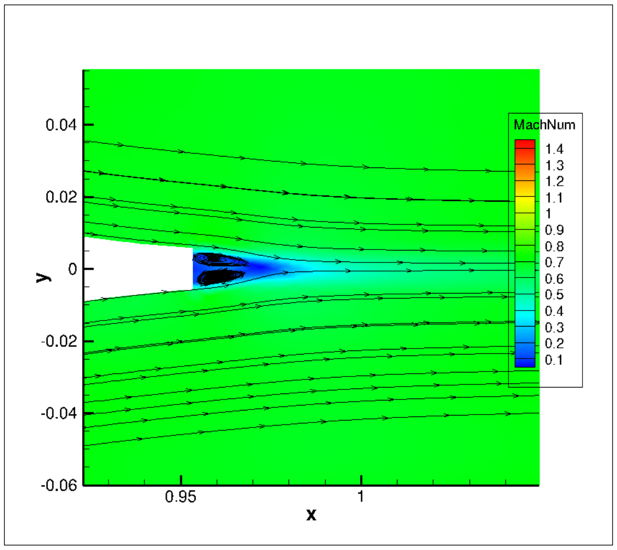
(a) Baseline Design



(b) Optimized Design



(c) Baseline Trailing Edge with Streamlines



(d) Optimized Trailing Edge with Streamlines

Figure 26. Mach field comparison

VIII. Conclusion

We developed pseudo-time accurate formulations of the tangent and adjoint systems for quasi-Newton solvers that return machine precision sensitivities in the limit of exact solution of the linear problem at each nonlinear iteration. These formulations are guaranteed to converge if the analysis problem converges and the adjoint uses the exact dual of the analysis linear solver. We first verified that these formulations are exact when the linear system is solved to near machine-precision. We then applied this formulation to varying levels of linear tolerance to see behavior of these formulations in non-converged and inexactly solved linear systems and showed that we can obtain an error estimate of these sensitivities as a function of both the linear tolerance and non-linear system convergence. We then showed that when these adjoint formulations are applied to a non-converging detached bow shock case, the sensitivities are negligibly affected by any backwards-in-pseudo-time integration outside of the objective function averaging window. We then ran an optimization on this case using varying linear tolerances and varying design bounds and found that the complex and adjoint sensitivities return very close designs to one another, which further strengthens the correspondence between these approaches. We then applied this adjoint formulation to two more non-converging cases with trailing edge unsteadiness and showed a large drop in the objective function value when driving the optimizer with these sensitivities for both cases. This method of sensitivity analysis in such flows was more robust and cheaper than using the steady-state adjoint and obtained sensitivities and designs that were comparable to those by complex-step finite differentiation.

IX. Acknowledgments

This work was supported in part by NASA Grant NNX16AT23H and the NASA Graduate Aeronautics Scholars Program. Computing time was provided by ARCC on the Teton supercomputer. The first author would like to thank Jan Kiviaho for his suggestion of certain resources and Hiro Nishikawa for his comments on limiters.

References

[1]Gill, P. E., Murray, W., and Saunders, M. A., "User's 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 Others, "Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.0 User's Manual," 2015.

[4]Nadarajah, S. K., *The Discrete Adjoint Approach to Aerodynamic Shape Optimization, Ph.D. Dissertation*, Department of Aeronautics and Astronautics, Stanford University, USA, 2003.

[5]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.

[6]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. 22–46.

[7]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.

[8]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.

[9]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.

[10]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.

[11]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.

[12]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.

[13]Shimizu, Y. S. and Fidkowski, K., "Output Error Estimation for Chaotic Flows," 46th AIAA Fluid Dynamics Conference, AIAA AVIATION Forum, AIAA Paper, 2016-3806, Washington D.C., 06/2016. https://doi.org/10.2514/6.2016-3806.

[14]Shimizu, Y. S. and Fidkowski, K., "Output-Based Error Estimation for Chaotic Flows Using Reduced-Order Modeling," 2018 AIAA Aerospace Sciences Meeting, AIAA SciTech Forum, (AIAA Paper 2018-0826), Kissimmee, Florida, 01/2018. https://doi.org/10.2514/6.2018-0826.

[15]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.

[16]LeVeque, R. J., *Numerical Methods for Conservation Laws*, Vol. 3, Springer, 1992.

[17]Roe, P., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 135, No. 2, 1997, pp. 250 – 258.

[18]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.

[19]Venkatakrishnan, V., "On the accuracy of limiters and convergence to steady state solutions," 31st Aerospace Sciences Meeting, AIAA Paper 1993-880, Reno NV, January 1993. https://doi.org/10.2514/6.1993-880.

[20]Michalak, C. and Ollivier-Gooch, C., "Accuracy preserving limiter for the high-order accurate solution of the Euler equations," *Journal of Computational Physics*, Vol. 228, No. 23, 2009, pp. 8693 – 8711.

[21]Nishikawa, H., "Robust numerical fluxes for unrealizable states," *Journal of Computational Physics*, Vol. 408, 2020, pp. 109244.

[22]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.

[23]Mavriplis, D., "A residual smoothing strategy for accelerating Newton method continuation," *arXiv preprint arXiv:1805.03756*, 2018.

[24]Saad, Y., *Iterative methods for sparse linear systems*, Vol. 82, Society for Industrial and Applied Mathematics, 2003.

[25]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.

[26]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.

[27]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.

[28]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.

[29]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.