Comparison of Finite Volume and High-Order Discontinuous Galerkin Based Aerodynamic Shape Optimization

Alexander W. Coppeans*, Krzysztof J. Fidkowski†, and Joaquim R.R.A. Martins‡
Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI, 48109

Aerodynamic shape optimization (ASO) requires a robust, accurate, and efficient flow solver. The second-order finite volume method (FVM) has been shown to satisfy these constraints when using a sufficiently fine mesh. However, during aerodynamic shape optimization, there are large geometry and flow solution changes that can lead to a decrease in solution accuracy over the course of the optimization. If the solution loses accuracy, the optimizer can find a spurious optimum. The discontinuous Galerkin (DG) method yields high-order accurate solutions that have less discretization error. However, the higher accuracy comes at a higher computational cost, so tha advantage of DG is not always clear. DG is suitable for both local order and mesh refinement which can allow it to be more accurate per degree of freedom than the FVM. In this work we show the benefits of using DG over FVM for optimization. We also develop a strategy for p-adaptation during optimization that reduces computational cost and obtains the same optimum as a fine-space solution.

I. Introduction

Aerodynamic shape optimization (ASO) in recent years has seen many advancements and is now quite mature. There are many key components required to perform ASO, such as geometric parameterization and the computational fluid dynamics (CFD) solver. The choice of the CFD solver is important as this solver has the highest percentage of computational cost during optimization. Therefore, the solver must be efficient as there could be hundreds of flow solutions required. Additionally, the CFD solver must be robust and able to provide a solution for shapes that a human designer would not normally ask for. If the CFD solver cannot provide a solution to the optimizer, then the whole optimization process could fail even if that design is nowhere close to the optimum. Finally, the solver must be accurate, as numerical errors affect the design space and can lead to a spurious optimum. Second-order finite volume method (FVM) CFD solvers have been shown to meet these criteria when using a sufficiently fine mesh. Additionally, high-order methods, such as the discontinuous Galerkin (DG), have been shown to offer highly accurate solutions that can beat low-order approximations in terms of both degrees of freedom, the number of unknowns for each state, and cost [1]. However, often to realize the benefits of high-order solutions, some form of adaptation is required to not waste degrees of freedom in regions that do not affect the solution and to concentrate them in regions that do [2].

Controlling discretization error plays an important role in optimization. Using a very fine discretization can be costly but necessary to find the "true" optimum. However, if using a fixed-fidelity approach computational, cost is wasted during initial iterations during optimization. Therefore, a multi-fidelity approach is ideal to prevent over optimizing a coarse mesh that leads to a spurious optimum and over refining designs that are not close to the optimum. Multiple approaches have been taken on this topic. Wu et al. [3] used a sequential multi-fidelity approach where the optimizer switched between multiple meshes and sets of governing equations during the optimization. Hicken and Alonso [4] computed errors in gradient norms and used these as an adaptive indicator to control first-order optimality error during the optimization. Chen and Fidkowski [5, 6] computed error in the objective function while accounting for how discretization error in constraints affects the objective and use it to drive the optimization based on how much the objective is changing. Brown and Nadarajah [7, 8] used adaptive tolerances for solving the residual and adjoint equations during optimization based on first-order optimality convergence.

In this work we compare the performance and results of using finite volume and DG in the same optimization framework. We also present an adaptation strategy that computes the Lagrangian error to account for constraint error in

*Ph.D Candidate, Department of Aerospace Engineering, AIAA Student Member.
†Professor, Department of Aerospace Engineering, Associate Fellow AIAA.
‡Professor, Department of Aerospace Engineering, Fellow AIAA.

the objective and use this as our adaptive indicator. We use the optimality convergence to specify when to adapt to meet a target error level at the end of the optimization and show that adaptation greatly reduces the computational cost without sacrificing accuracy of the optimization.

II. Computation Framework

In this work we use the MACH-Aero framework for optimization. MACH-Aero is an open-source framework for gradient-based ASO. It consists of various modules for geometric parameterization, mesh warping, CFD solver, and optimizers.

A. Discontinuous Galerkin CFD Solver

CFD requires discretizing a set of governing equations that model the fluid. In this work we use the Reynolds-averaged Navier Stokes (RANS) equations closed with the Spalart-Allmaras (SA) turbulence model [9]. The governing equations are written as,

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \vec{\mathbf{F}}(\mathbf{u}, \nabla\mathbf{u}) + \mathbf{S}(\mathbf{u}, \nabla\mathbf{u}) = \mathbf{0}, \tag{1}$$

where $\mathbf{u} \in \mathbb{R}^s$ is the s-component state vector, $\vec{\mathbf{F}} \in \mathbb{R}^{\text{dim}\times s}$ is the flux vector, dim is the number of spatial dimensions, and $\mathbf{S} \in \mathbb{R}^s$ is the source term arising from the turbulence model. The CFD solver we use is xflow which discretizes the governing equation by using the discontinuous Galerkin (DG) method with Roe [10] convective flux and the second form of Bassi and Rebay (BR2) [11] for viscous treatment. The state is approximated with polynomials of order p on an unstructured mesh of non-overlapping elements. Following a finite-element weak formulation and choice of polynomial basis functions, the semi-discrete form of the governing equations is,

$$\mathbf{M}\frac{d\mathbf{U}}{dt} + \mathbf{R}(\mathbf{U}) = \mathbf{0}. \tag{2}$$

Here $\mathbf{U} \in \mathbb{R}^N$ is the discrete state vector, N is the total number of unknowns, $\mathbf{R}(\cdot) \in \mathbb{R}^N$ is the nonlinear spatial residual, and $\mathbf{M} \in \mathbb{R}^{N\times N}$ is the block-element sparse mass matrix. For steady simulations used in this work the time derivative term drops out, although pseudo-time continuation [12] remains in the solver to converge the steady residual. The non-linear solver uses the Newton-Raphson method with the generalized minimum residual (GMRES) [13] linear solver, preconditioned by an element-line Jacobi smoother with coarse-level ($p = 1$) correction [14, 15].

B. Finite Volume CFD Solver

The second CFD solver used is ADflow [16], a second-order cell-centered finite volume flow solver for multiblock and overset meshes. ADflow includes a discrete adjoint implementation [17] and a robust approximate Newton-Krylov start up strategy [18].

C. Geometric Parameterization

To parameterize the geometry, we use the free-form deformation (FFD) approach [19] implemented using pyGeo [20]. The FFD approach uses a moveable control point box that embeds the baseline geometry. Moving the control points results in movement of nodes on the geometry surface. This approach is efficient because it doesn't directly parameterize the shape of the geometry but instead parametrizes deformations. This allows the movement of all surface nodes while also having far fewer design variables than surface nodes. pyGeo also includes modules for thickness and area constraints.

D. Mesh Warping

Once surface nodes are deformed using pyGeo, the rest of the volume nodes need to be updated. To do this, we use IDwarp [21], which uses an inverse-distance weighting method proposed in [22] to propagate deformations from the surface to the rest of the volume. IDwarp works for both structured and unstructured meshes because volume nodes are represented by a point cloud with no connectivity. However, connectivity is required on the surface and IDwarp does not support high-order meshes. Thus, high-order meshes are converted to linear meshes by linearly connecting all high-order surface nodes.

E. Optimizer

We use pyOptSparse [23], an optimization framework designed for large-scale gradient based optimizations. pyOptSparse provides wrappers for several optimization packages and in this work we use SNOPT [24].

III. Constrained Optimization

Aerodynamic shape optimization can be formulated as a constrained optimization [25] with design parameters, $\mathbf{x} \in \mathbb{R}^n_x$, that minimize an objective function $f(\mathbf{x}) \in \mathbb{R}^1$ subject to n_h equality constraints $\mathbf{h}(\mathbf{x}) = 0$, where $\mathbf{h} \in \mathbb{R}^{n_h}$, and n_g inequality constraints $\mathbf{g}(\mathbf{x}) \leq 0$, where $\mathbf{g} \in \mathbb{R}^{n_g}$, given as,

$$\begin{aligned} \text{minimize } & f(\mathbf{x}) \\ \text{by varying } & x_i \\ \text{subject to } & g_j(\mathbf{x}) \leq 0 \\ & h_l(\mathbf{x}) = 0. \end{aligned} \tag{3}$$

To solve this problem, the objective function is combined with all the constraints into a Lagrangian function given as,

$$\mathcal{L} = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{x}) + \boldsymbol{\sigma}^T (\mathbf{g}(\mathbf{x}) + \mathbf{s} \odot \mathbf{s}), \tag{4}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{n_h}$ and $\boldsymbol{\sigma} \in \mathbb{R}^{n_g}$ are Lagrange multipliers for the equality constraints and inequality constraints respectively, and $\mathbf{s} \in \mathbb{R}^{n_g}$ is a slack variable to turn inequality constraints into equality constraints [26], and $\odot$ represents element wise multiplication.

Taking the derivative of the Lagrangian with respect the design variables, $\mathbf{x}$, and Lagrange multipliers, $\boldsymbol{\lambda}$ and $\boldsymbol{\sigma}$, and setting them equal to 0 gives the following set of equations.

$$\nabla_x \mathcal{L} = \nabla_x f + \mathbf{J}_h^T \boldsymbol{\lambda} + \mathbf{J}_g^T \boldsymbol{\sigma} = \mathbf{0}, \tag{5}$$

$$\nabla_\lambda \mathcal{L} = \mathbf{h} = \mathbf{0}, \tag{6}$$

$$\nabla_\sigma \mathcal{L} = \mathbf{g} + \mathbf{s} \odot \mathbf{s} = \mathbf{0}, \tag{7}$$

$$\nabla_s \mathcal{L} = \sigma_i s_i = 0, \tag{8}$$

$$\boldsymbol{\sigma} \geq \mathbf{0}. \tag{9}$$

These equations are known as the Karush-Kuhn-Tucker (KKT) conditions that need to be solved at the optimal point which is a stationary point of the Lagrangian. Solving this system of equations requires values of both the objective and constraints as well as gradients of the objective and constraints with respect to the design variables.

At the optimum point the first KKT condition is given as,

$$\frac{\partial f}{\partial \mathbf{x}} = -\boldsymbol{\sigma}^T \frac{\partial g_a}{\partial \mathbf{x}} - \boldsymbol{\lambda}^T \frac{\partial h}{\partial \mathbf{x}}, \tag{10}$$

where g_a is the set of active inequality constraints. If we write the differential of the objective $df = (\frac{\partial f}{\partial \mathbf{x}} d\mathbf{x})$ and combine the active inequality constraints with equality constraints we can rewrite this equation as,

$$df = -\boldsymbol{\lambda}^T \frac{\partial h}{\partial \mathbf{x}} d\mathbf{x}. \tag{11}$$

This leads to the definition of the Lagrange multiplier λ_i for constraint i to be

$$\lambda_i = -\frac{df}{dh_i}. \tag{12}$$

The Lagrange multipliers therefore represent a linearization of the objective with respect to the constraints. They show how much the objective would change if the constraints were to be relaxed.

A. Gradient Calculation

To perform gradient-based optimization the gradient of all objectives and constraints must be computed with respect to all design variables. There are many ways to perform this calculation[25] such as: finite difference, complex step [27], algorithmic differentiation [28], and the adjoint method [29]

In this work we use the adjoint method which performs well for large optimization problems as it scales with the number of functions of interest and not the number of design variables. We compute the gradient of some function J that is a function of both some state vector $\mathbf{U}$ and the design variables $\mathbf{x}$,

$$J = J(\mathbf{U}, \mathbf{x}). \tag{13}$$

The states are solved for by driving a discrete residual vector, $\mathbf{R}$, to 0 for a given set of design variables,

$$\mathbf{R}(\mathbf{U}, \mathbf{x}) = \mathbf{0}. \tag{14}$$

The gradient of the function J is computed first by applying the chain rule

$$\frac{dJ}{d\mathbf{x}} = \frac{\partial J}{\partial \mathbf{x}} + \frac{\partial J}{\partial \mathbf{U}}\frac{d\mathbf{U}}{d\mathbf{x}}. \tag{15}$$

The term $\frac{d\mathbf{U}}{d\mathbf{x}}$ can be solved for by applying the chain rule to the residual $\mathbf{R}$ and computing the total derivative of the residual with respect to the design variables. This gradient is always $\mathbf{0}$ as the residual is driven to $\mathbf{0}$ at each design point,

$$\frac{d\mathbf{R}}{d\mathbf{x}} = \frac{\partial \mathbf{R}}{\partial \mathbf{x}} + \frac{\partial \mathbf{R}}{\partial \mathbf{U}}\frac{d\mathbf{U}}{d\mathbf{x}} = 0, \tag{16}$$

$$\frac{d\mathbf{U}}{d\mathbf{x}} = -\left(\frac{\partial \mathbf{R}}{\partial \mathbf{U}}\right)^{-1}\frac{\partial \mathbf{R}}{\partial \mathbf{x}}. \tag{17}$$

The expression for $\frac{d\mathbf{U}}{d\mathbf{x}}$ can then be plugged into Equation (15),

$$\frac{dJ}{d\mathbf{x}} = \frac{\partial J}{\partial \mathbf{x}} - \frac{\partial J}{\partial \mathbf{U}}\left(\frac{\partial \mathbf{R}}{\partial \mathbf{U}}\right)^{-1}\frac{\partial \mathbf{R}}{\partial \mathbf{x}}. \tag{18}$$

This equation can be solved naively by solving the linear system $\left(\frac{\partial \mathbf{R}}{\partial \mathbf{U}}\right)^{-1}\frac{\partial \mathbf{R}}{\partial \mathbf{x}}$. This is known as the direct method and requires solving a linear system for each design variable. Instead, we can define the adjoint vector as,

$$\mathbf{\Psi} = -\left(\frac{\partial \mathbf{R}}{\partial \mathbf{U}}^T\right)^{-1}\frac{\partial J}{\partial \mathbf{U}}^T. \tag{19}$$

This linear system does not scale with the number of design variables but instead scales with the number of functions of interest. From here we can write the gradient of the function as,

$$\frac{dJ}{d\mathbf{x}} = \frac{\partial J}{\partial \mathbf{x}} + \mathbf{\Psi}^T\frac{\partial \mathbf{R}}{\partial \mathbf{x}}. \tag{20}$$

The remaining partial derivatives on the right-hand side can be computed easily as they do not require solving the residual equations again. In both ADflow and xflow the adjoint is computed using analytic derivatives. However there is a difference in how the remaining partial derivatives are computed which affects performance of the optimization. In ADflow these partial terms are calculated through algorithmic differentiation. This leads to exact gradients that have been verified with complex step [17]. In xflow we compute these partial derivatives using finite differences. We perturb each design variable and compute the residual vector and output at the perturbed state and use a forward difference approximation. This leads to a larger cost to compute the gradient as we need to warp the mesh for each shape design variable. It also leads to less accurate gradients which cause the optimizer to take more major iterations for the same optimality and feasibility tolerances.

IV. Error Estimation and Adaptive Indicators

When solving solving a discretized PDE on a coarse-space H, discretization error that affects the outputs of interest. In practice, it is not possible to compute the amount of discretization error relative to the exact solution of the PDE, but it is possible to compute the error relative to a finer space h which can be used in place of the exact PDE,

$$\text{output error: } \delta J = J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h). \tag{21}$$

Here J represents some output of interest which can be the objective value or a constraint. In this work to obtain the fine-space, h, we increment each elements' approximation order, p_e, to $p_e + 1$. The output on the fine-space is not solved for directly instead we use the adjoint-weighted residual method [2, 30, 31] to compute the output error estimate. The fine-space adjoint solution $\mathbf{\Psi}_h$, linearized about $\mathbf{U}_h^H$, which is the coarse-space solution injected into the fine-space. The output error comes from perturbations in the state which lead to perturbations in the residual, as follows:

$$\begin{aligned}\delta J &= J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h),\\ &= J(\mathbf{U}_h^H) - J_h(\mathbf{U}_h) = \frac{\partial J_h}{\partial \mathbf{U}_h}\delta \mathbf{U},\\ &= -\mathbf{\Psi}_h^T \delta \mathbf{R}_h = -\mathbf{\Psi}_h^T(\mathbf{R}_h(\mathbf{U}_h^H) - \mathbf{R}_h(\mathbf{U}_h)),\\ &= -\mathbf{\Psi}_h^T \mathbf{R}_h(\mathbf{U}_h^H).\end{aligned} \tag{22}$$

Here $\mathbf{U}_h$ is the state associated with the solution to the fine-space problem which leads to $\mathbf{R}_h(\mathbf{U}_h) = 0$. This derivation relies on the assumption that the injection into the fine-space does not change the output of interest, $J_h(\mathbf{U}_h^H) = J_H(\mathbf{U}_H)$.

The error estimate can be thought of as the sum of the error on each element written as,

$$\begin{aligned}\delta J &= = J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h),\\ &= -\mathbf{\Psi}_h^T \mathbf{R}_h(\mathbf{U}_h^H),\\ &= -\sum_{e=1}^{n_e} \mathbf{\Psi}_{h,e}^T \mathbf{R}_{h,e}(\mathbf{U}_h^H).\end{aligned} \tag{23}$$

A common approach to determine which elements should be adapted is to compute the discretization error caused by each element and take the absolute value,

$$\epsilon_e \equiv \left| \mathbf{\Psi}_{h,e}^T \mathbf{R}_{h,e}(\mathbf{U}_h^H) \right| \tag{24}$$

A. Output estimation during optimization

The derivation above shows how the output error for a single function of interest is computed. However, in optimization problems, discretization errors in the constraints also affect the objective value. For example, when trimming an airfoil to a given lift coefficient, c_ℓ, under predicting lift in the coarse space would under-predict drag in the fine-space when the airfoil is trimmed. To account for this, we estimate the error of the Lagrangian, which relates the effects of errors or changes in constraints to the objective function,

$$\begin{aligned}\delta \mathcal{L} &= \mathcal{L}_H(\mathbf{U}_H) - \mathcal{L}_h(\mathbf{U}_h)\\ &= \delta f + \boldsymbol{\lambda}^T \delta \mathbf{h} + \boldsymbol{\sigma}^T \delta \mathbf{g}.\end{aligned} \tag{25}$$

Here the discretization error is computed for the objective and each constraint. For most aerodynamic optimization problems the number of outputs and constraints from the CFD solver is low so only a few fine-space adjoint solutions are needed. Other constraints not computed from the CFD solver are not affected by the discretization level and therefore, the associated discretization errors are $\mathbf{0}$.

When adapting on the error of the Lagrangian, we take the element wise contribution of error but also include the Lagrange multipliers in the absolute value. This leads to a more conservative approach and adapts elements even when errors in constraints could further reduce the output.

$$\epsilon_e = \left|\delta f_e\right| + \left|\boldsymbol{\lambda}^T \delta \mathbf{h}\right| + \left|\boldsymbol{\sigma}^T \delta \mathbf{g}\right|. \tag{26}$$

V. Adaptation Strategy

During the optimization process, it is important for the discretization error at the optimum to be low to avoid spurious optima while not wasting degrees of freedom early in the optimization process, when the design is changing rapidly. To do this, we use an an adaptation strategy based on the optimality value from SNOPT which represents a measure of the first KKT condition given in Equation (5).

Brown and Nadararajah derived a relationship between between residual from solving PDE constraints and their adjoint equations to the first order optimality condition [7, 8]. They showed that adapting how tightly the PDE is solved during the optimization reduces the total optimization time. Their algorithm looks at the relative convergence of the optimality and adapts the relative convergence of the PDE to be a multiplicative factor of the convergence of the optimality.

We take a similar approach but, instead of changing residual tolerances based on optimality, we adapt by changing output error tolerances. Consider the case of an airfoil in supersonic flow. Large residuals downstream of the airfoil have no effect on outputs calculated on the surface of the airfoil. Therefore, it is not always necessary to drive all PDE residuals tolerances to zero. Instead we look at which residuals directly affect the output through the adjoint-weighted residual error estimate in Equation (22). From here, we can directly follow their approach and drive constraint errors down to a specified tolerance as we optimize. Instead, we take one further step and look at the error in the Lagrangian and drive this down to a specified tolerance by the end of the optimization process, looking at the optimality condition to tell us when to adapt. We also do not compute the error on the Lagrangian every iteration; instead we compute it in a lagged fashion. After the initial stages of the optimization, the design does not change rapidly between iterations and on a fixed mesh the discretization error is not changing significantly. So instead we only compute the error estimate after adaptation and then every n_{adapt} iterations which is a user defined parameter. This saves computational cost by reducing the frequency that the fine space adjoint is required. Additionally, no adaptation takes place during the line search of the optimization to avoid introducing extra noise that could slow down the optimization. The updated optimization process is shown in Algorithm 1.

We modify the MACH-Aero framework so Lagrange multipliers are passed from the optimizer to the CFD solver and so the discretization error is passed to the optimizer. The extended design matrix (XDSM) diagram [32] is shown in Figure 1.

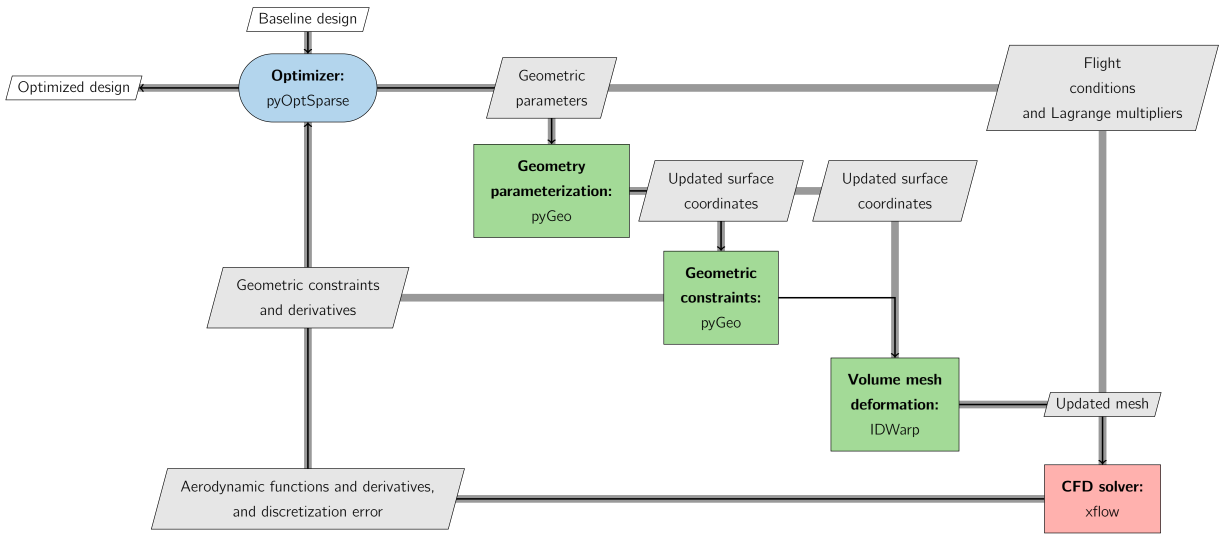
Baseline design
Optimized design
Optimizer: pyOptSparse
Geometric parameters
Flight conditions and Lagrange multipliers
Geometry parameterization: pyGeo
Updated surface coordinates
Updated surface coordinates
Geometric constraints and derivatives
Geometric constraints: pyGeo
Volume mesh deformation: IDWarp
Updated mesh
Aerodynamic functions and derivatives, and discretization error
CFD solver: xflow

Fig. 1 Modified MACH-Aero XDSM diagram showing how discretization error and Lagrange multipliers are passed between the CFD solver and the optimizer.

Algorithm 1 Optimization with error estimation and adaptation

```
$\tau_{\text{opt}}^{\text{tol}}, \tau_{\text{feas}}^{\text{tol}}$    ▷ Optimality and Feasibility Tolerance
$x_i = x_0$    ▷ Set initial design
$\epsilon_{\mathcal{L}}^{\text{tol}}$    ▷ Desired final error tolerance
adaptFlag $\leftarrow$ False    ▷ Initialize flag for CFD solver to adapt solution to false
errorFlag $\leftarrow$ False    ▷ Initialize flag for CFD solver to compute discretization error to false
$n_{\text{adapt}}$    ▷ Initialize $n_{\text{adapt}}$
$n = 0$    ▷ Initialize number of major iterations since last adaptation
while $\tau_{\text{opt}} > \tau_{\text{opt}}^{\text{tol}}$ AND $\tau_{\text{feas}} > \tau_{\text{feas}}^{\text{tol}}$ do    ▷ While optimality and feasibily criteria not met
    if CFD SOLVER then    ▷ CFD Solver portion of optimization iteration
        Compute $f$, $\mathbf{g}$, $\mathbf{h}$, $\nabla f$, $\nabla \mathbf{g}$, $\nabla \mathbf{h}$ and    ▷ Compute objective, constraint values and their gradients
        if adaptFlag then
            Adapt solution order of elements with highest discretization error
            Re-compute Lagrangian error estimate, $\delta\mathcal{L}$
        end if
        if errorFlag then
            Re-compute Lagrangian error estimate
        end if
    end if
    if Optimizer then
        Compute $\lambda, \sigma$    ▷ Compute Lagrange Multipliers
        Update $x_{i+1}$    ▷ Update Design
        Update $\tau_{\text{opt}}$ and $\tau_{\text{feas}}$
        if End of Major Iteration then
            $n+ = 1$
            if $n > n_{\text{adapt}}$ then    ▷ If its been too many major iterations next time compute error estimate
                errorFlag $\leftarrow$ True
                $n = 0$
            end if
            if $\delta\mathcal{L} > \max(\frac{\tau_{\text{opt}}}{\tau_{\text{opt}}^{\text{tol}} \times 10}, 1) \times \epsilon_{\mathcal{L}}^{\text{tol}}$ then
                adaptFlag$\leftarrow$ True    ▷ If error is too high based on optimality adapt next iteration
            end if
        end if
    end if
end while
```

VI. Drag Minimization of Subsonic Turbulent NACA 0012 Airfoil

For our first test problem, we minimize drag for a NACA 0012 airfoil at a Mach number of $M = 0.5$ and Reynolds number of 1×10^6. We impose a lift constraints of $c_\ell = 1.0$, an area constraint $A \geq A_0$ where the area of the airfoil cannot be less than the baseline NACA 0012, and impose thickness constraint, $t/c \geq 0.3t_0/c$, where the thickness cannot be less than 30% of the original thickness at any location along the chord. We also impose FFD constraints at the leading and trailing edge to ensure the airfoil does not rotate and only the angle of attack variable, α, changes the angle of attack. The optimization problem is summarized in Table 1.

Table 1 Optimization Problem for Subsonic NACA 0012 drag minimization

Category	Name	Quantity	Lower	Upper
Objective	c_d	1	—	—
Variable	y	20	-0.05	0.05
	α	1	0	10
Constraints	c_ℓ	1	1.0	1.0
	A	1	A_{initial}	—
	$y_{\text{LE,lower}} = -y_{\text{LE,upper}}$	1	0.0	0.0
	$y_{\text{TE,lower}} = -y_{\text{TE,upper}}$	1	0.0	0.0
	t/c	400	0.3	—

The optimality and feasibility tolerances in SNOPT were both set to 1×10^{-6} for all optimizations in this section. We run the optimization on the family of meshes with ADflow. In xflow we use the $L2$ mesh, detailed in Table 2, and run at a fixed resolution with different solution approximation orders of $p = 3$, $p = 2$, $p = 1$. At $p = 3$ using tensor product quad-Lagrange basis functions there are $72,960$ degrees of freedom which is equal to the number of degrees of freedom when running ADflow on the $L0$ mesh.

After each optimization, we reanalyze the design and re-trim the c_ℓ to a tolerance of 1×10^{-8} in both xflow and ADflow. In xflow we use the $L1$ mesh modified with curved $q = 3$ elements with a solution order of $p = 3$ with a total of $291,840$ degrees of freedom. In ADflow we use the $L00$ mesh which has $291,840$ degrees of freedom. The FFD points used are shown with the $L0$ mesh in Figure 2

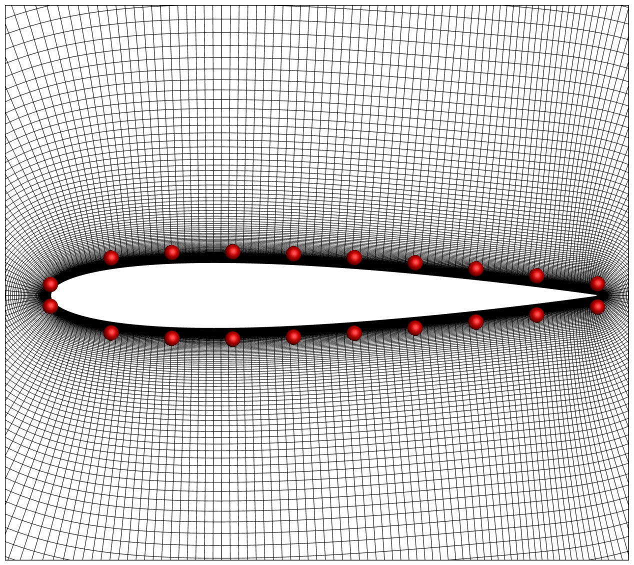

Fig. 2 $L0$ mesh for NACA 0012 Airfoil with 20 FFD points used for optimization.

A. NACA 0012 Meshes

We generated a family of O topology meshes around the NACA 0012 airfoil to study the different flow solvers on the same mesh. We created 4 meshes summarized in Table 2.

Table 2 NACA 0012 O-Grid Mesh Family Summary

Name	Cells On Wall	Total Cells
$L00$	480	291840
$L0$	240	72960
$L1$	120	18240
$L2$	60	4560

High-order methods require meshes that are curved around the geometry. Once we generated the family of meshes we took the $L2$ mesh and added geometric nodes on the wall to make each cell cubic, $q = 3$, and used linear-elasticity to curve the mesh.

B. Finite Volume NACA 0012 Optimization

We ran the optimization in ADflow 4 times on the $L2$, $L1$, $L0$, and $L00$ meshes. Table 3 summarizes optimized final drag and optimization time.

Table 3 NACA 0012 optimization with ADflow summary

Mesh	DoFs	Time (hr)	Optimized $c_d \times 10^4$	xflow True $c_d \times 10^4$	ADflow True $c_d \times 10^4$
$L00$	291,840	14.4	137.59	136.91	137.59
$L0$	72,960	0.712	138.48	137.04	137.66
$L1$	18,240	0.078	140.89	137.38	137.98
$L2$	4,560	0.025	153.28	138.06	138.81

The “true" final drag of each of the optimizations are all within 1 drag count ($c_d = 10^4$) when reanalyzing with both xflow and ADflow even though the geometries are all different. The final geometries and c_p distributions generated from the “true” xflow cross analysis are shown in Figure 3. The upper surface of these airfoils is very similar but there are noticeable differences around the leading edge on the lower surface. As the mesh is refined, the airfoils’ nose gets thinner and more area is concentrated towards the back. The coefficient of pressure plots show as the mesh is refined the optimized design tries to increase pressure on the lower surface around the leading edge where the geometry gets thinner.

C. Fixed Resolution Discontinuous Galerkin NACA 0012 Optimization

For our DG results at a fixed resolution we used the $L2$ mesh curved with $q = 3$ cubic quad elements and solution order $p = 1$, $p = 2$, and $p = 3$. The results are summarized in Table 4.

Table 4 NACA 0012 optimization with xflow summary

Solution Order	DoFs	Time (hr)	Optimized $c_d \times 10^4$	xflow True $c_d \times 10^4$	ADflow True $c_d \times 10^4$
3	72,960	6.6	137.21	136.90	137.62
2	41,040	2.0	137.65	136.90	137.60
1	18,240	0.2	142.10	137.10	137.84

The optimized airfoil geometries and “true” c_p distributions from xflow are shown in Figure 4. The geometries obtained using DG are all very similar and the $p = 2$ geometry almost directly matches the $p = 3$ geometry. The airfoils’

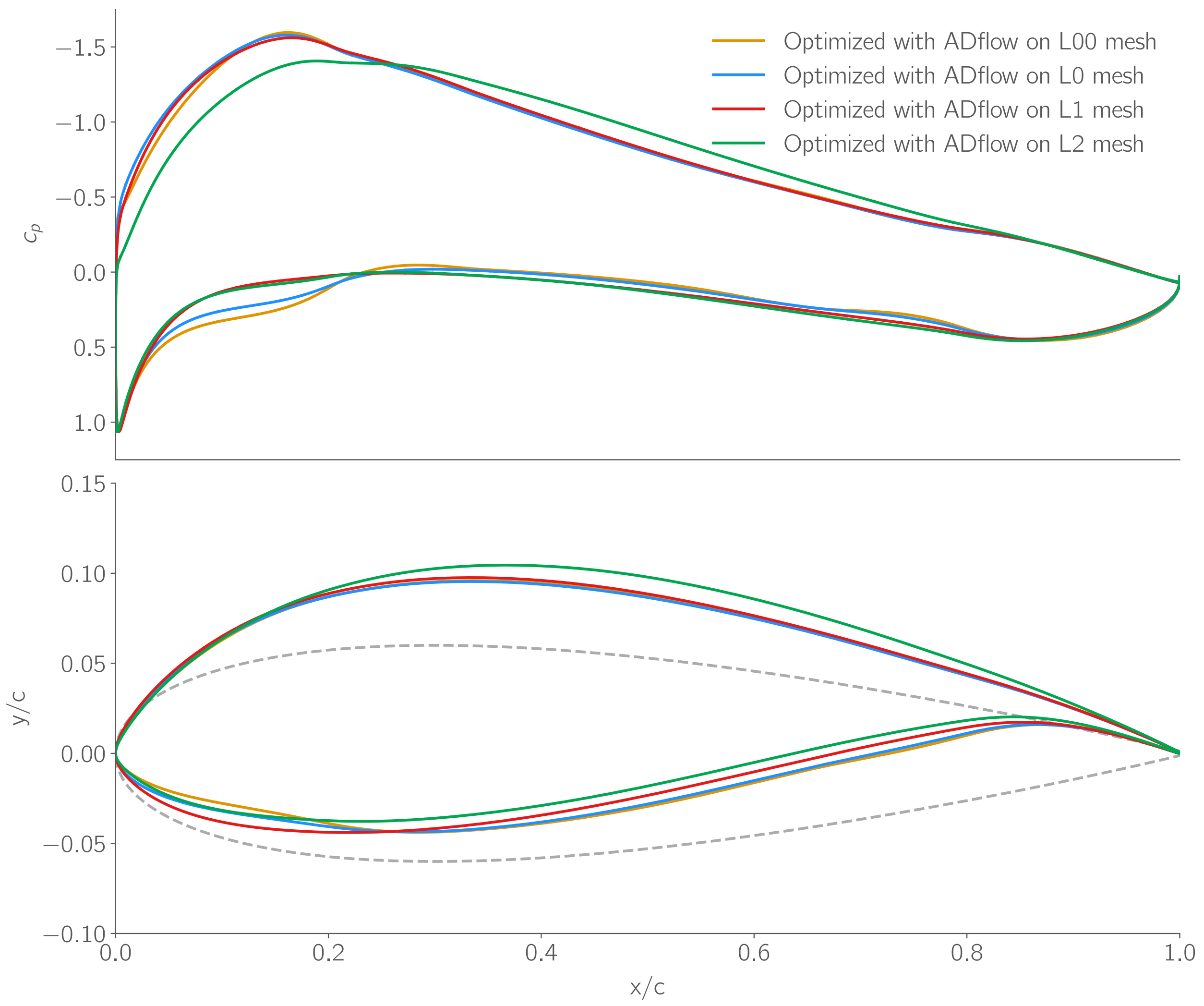

Fig. 3 Optimized airfoils using ADflow using different mesh resolutions show as the mesh is refined the leading edge gets more pinched. c_p plots generated from "true" xflow cross analysis.

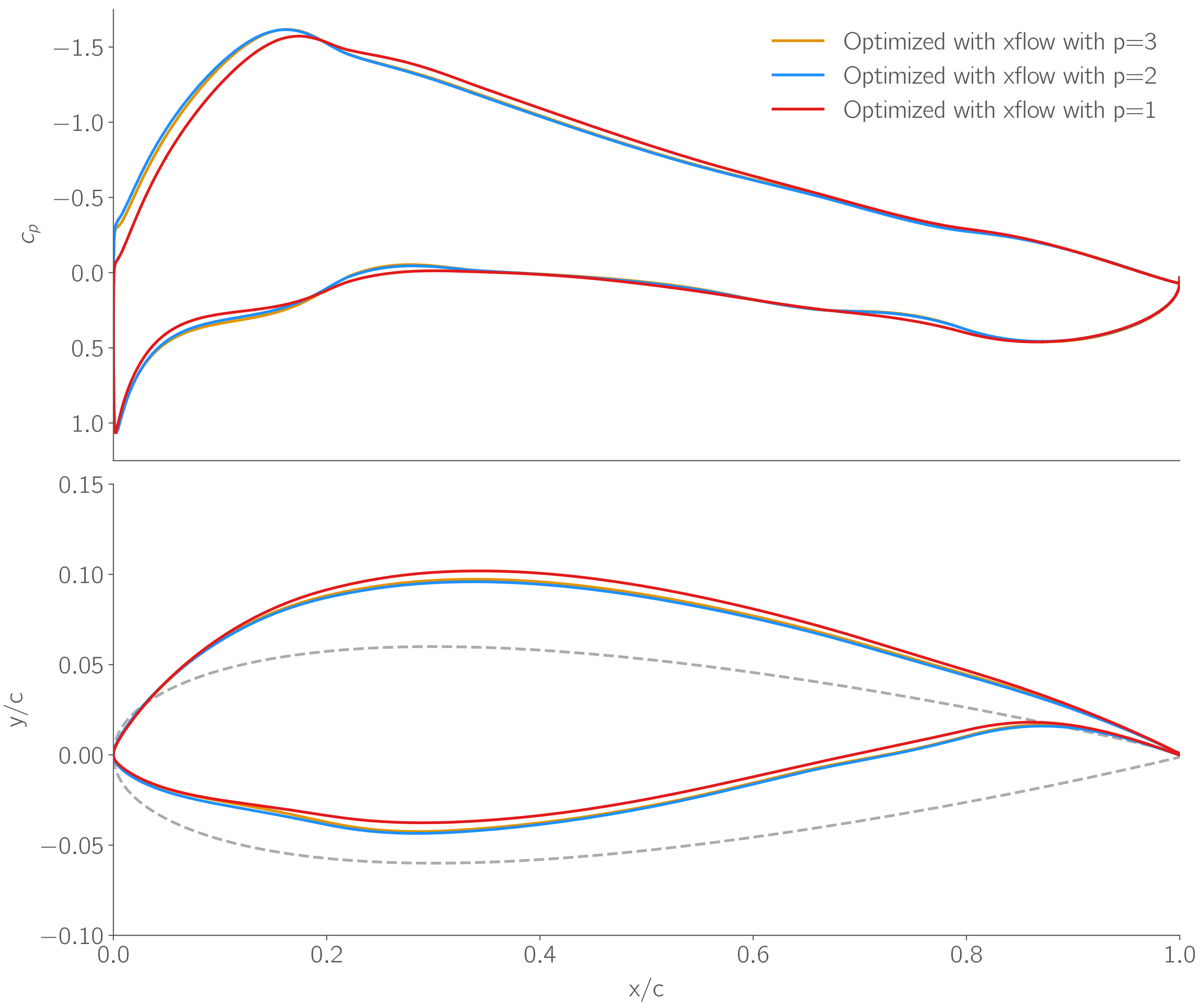

Fig. 4 Optimized airfoils using xflow using different solution orders all achieve pinched leading edge. c_p plots generated from "true" xflow cross analysis.

leading edge gets thinner at all solution orders and is almost identical unlike when using finite-volume where as the resolution was increased the leading edge got noticeably thinner. The airfoil optimized using $p = 1$ has an upper surface and lower surface slightly moved up from the airfoils optimized using $p = 2$ and $p = 3$ and the bump at around 20% chord is not as pronounced which is shown by the lower pressure in that region in the c_p plot. The $p = 2$ results match almost identically with the $p = 3$ results which is also shown when looking at the “True” c_d values. The minimum c_p on the airfoil optimized with $p = 1$ is not get as low on the upper surface or as high around the bump at around 20% chord. However, it is slightly lower on the upper surface from 20% to 60% chord moving the lift distribution from the front to the back.

When using a fixed resolution DG vs finite volume we see that per degree of freedom DG is able to outperform finite volume. The $p = 2$ design when analyzed with both xflow and ADflow for the “True” solution is identical to the $p = 3$ solution and outperforms all airfoils optimized using ADflow except the airfoil optimized using the $L00$ mesh. The airfoil optimized in ADflow with the $L00$ mesh has almost identical drag values but in terms of both time and number of degrees of freedom the optimization performs worse. Per degree of freedom finite volume is faster but the optimized airfoil is worse than the optimized airfoil using DG.

D. p-Adaptive Discontinuous Galerkin NACA 0012 Optimization

To improve the DG performance we use p-adaptation and set a tolerance of the Lagrangian error to be 0.5 drag counts (5×10^{-5}) at an optimality of 1×10^{-5}. This ensures the last order of convergence is on the most accurate solution space to avoid getting stuck at a spurious optimum. We start on the $L1$ mesh with $p = 1$ elements and set a maximum solution order of $p = 3$ and a minimum order of $p = 1$. At each adaptation iteration we increase the order of the elements with the top 7.5% of the error and reduce the order fo the elements with the lowest 2.5% of error. We also clip the order so the maximum order any element can have is $p = 3$ and the lowest order an element can have is $p = 1$. Additionally, we compute the error on the Lagrangian every 5 major iterations. The results are summarized in Table 5.

Table 5 NACA 0012 optimization with adaptive DG summary

Solution Order	final DoFs	Time (hr)	Optimized $c_d \times 10^4$	xflow True $c_d \times 10^4$	ADflow True $c_d \times 10^4$
1/2/3	30,758	1.8	137.50	136.90	137.61

Figure 5 shows the number of degrees of freedom, optimality, and c_d at each major iteration. Each time the optimality drops below the adaptation tolerance the number of degrees of freedom increases and the optimality increases because the new discretization creates a different design space. The first two adaptation iteration show a noticeable drop in c_d which is due to the finer space removing numerical diffusion.

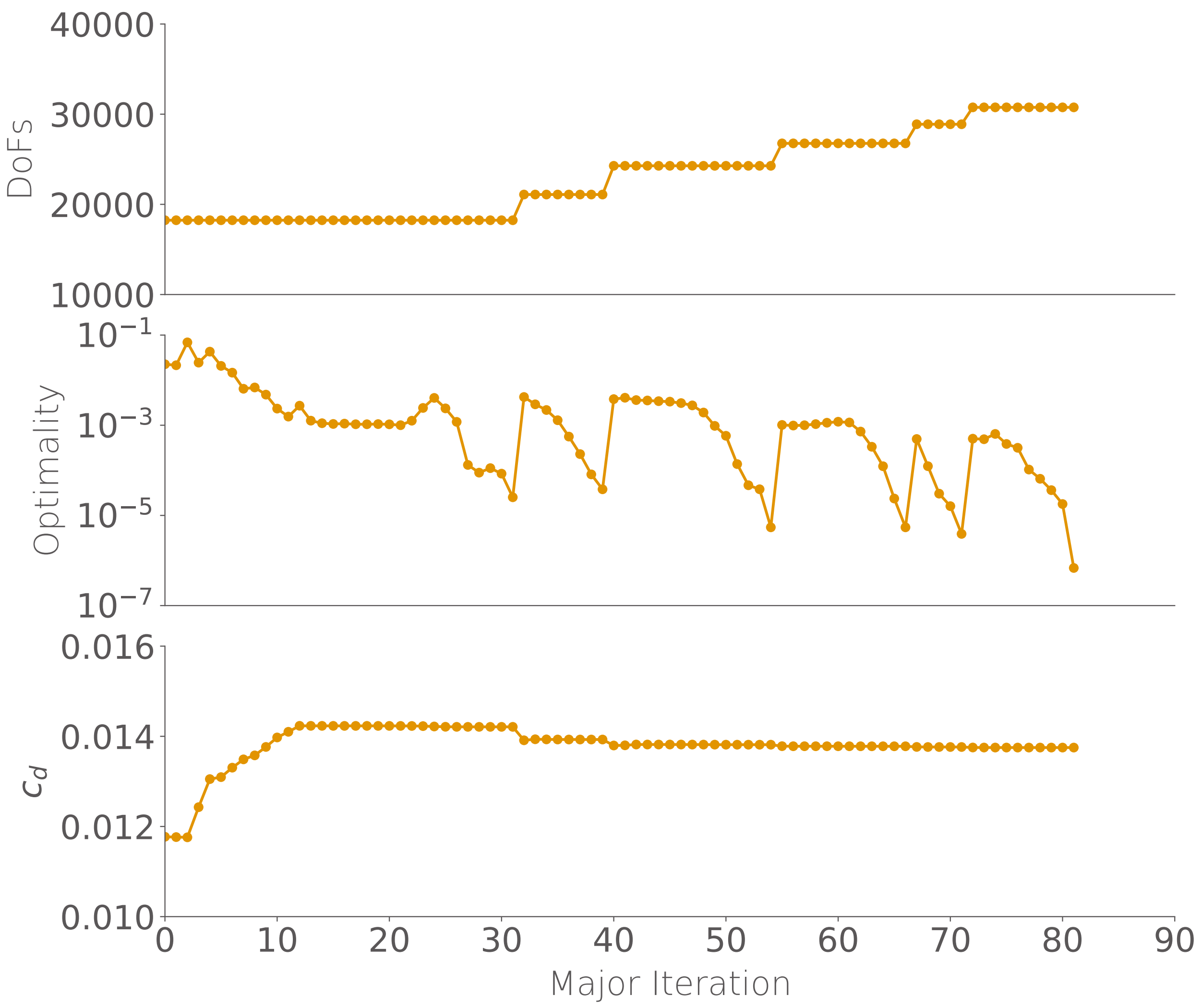

Fig. 5 Optimization history for p-adaptive NACA 0012 optimization

The geometry and c_p of the optimized airfoil with adaptation are compared to the finest fixed-fidelity optimizations in both xflow and ADflow in Figure 6.

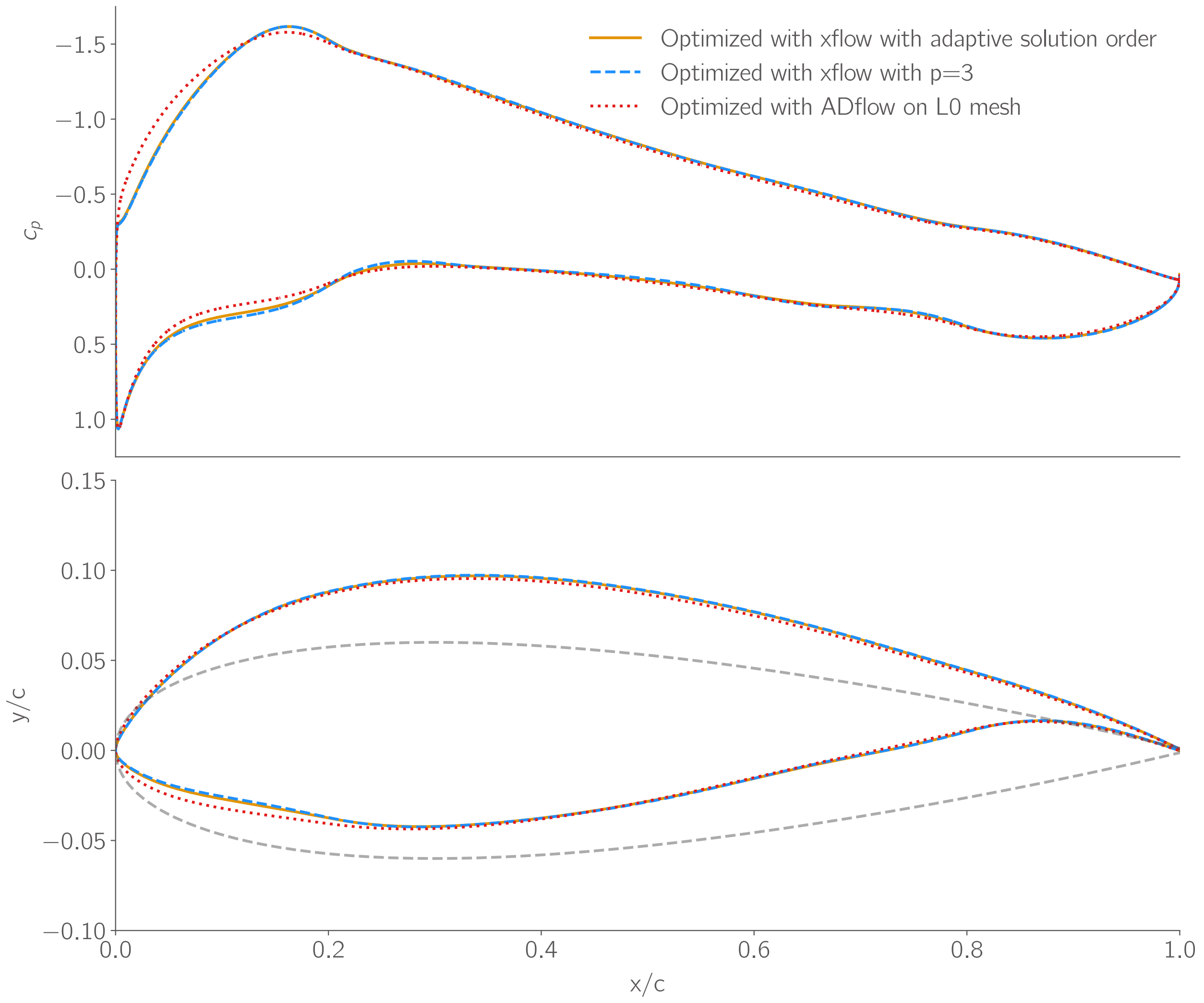

Fig. 6 Comparison of optimized airfoil with DG p-adaptation, DG fixed fidelity, and FVM shows the designs from DG are identical and the FVM airfoil is very similar. c_p distributions from "true" xflow results.

The airfoil optimized with p-adaptation performs identically to the airfoil optimized at $p = 2$ and $p = 3$. The geometry and c_p distribution is the same as the airfoil optimized with $p = 3$ while the $L00$ airfoil is quite different. The difference comes from the fact that the optimizer is able to reduce drag from numerical diffusion in each solver. Finite volume methods have more diffusion than high-order methods and that is why at even lower degrees of freedom, the airfoils optimized with DG all have similar geometries while the airfoils optimized with finite volume vary as the mesh is refined. As the mesh for finite volume is refined, the optimized airfoil nose gets thinner and approaches the design achieved by the DG optimization.

For this problem DG is able to take advantage of the smooth solution space and optimize the airfoil to a lower drag using fewer degrees of freedom when compared to using finite volume CFD. While DG takes longer than all but the optimization on the $L00$ mesh, it is more accurate and adaptation is able to reduce the time relative to the $p = 3$ optimization by 70% and 10% relative to the $p = 2$ optimization.

VII. Transonic Turbulent Airfoil

The second test case we ran was a viscous transonic airfoil optimization of the RAE 2822 airfoil defined by the second test case of the Aerodynamic Design Optimization Discussion Group (ADODG). This test case is a drag minimization of the RAE2822 airfoil subject to a lift, pitching moment, and area constraints at a Mach number of $M = 0.734$ and Reynolds number of $Re = 6.5 \times 10^6$. We parameterize the airfoil with 20 FFD points 10 points in the chord-wise direction along the upper and lower surface. Additionally we add thickness constraints such that the thickness at any chord-wise location is greater than 10% of the original thickness at that location. This ensures that the airfoil top and bottom surfaces never cross although for all optimizations no thickness constraint is ever at the bounds. Finally there are constraints on the FFD pairs at the leading and trailing edge to prevent geometric rotation and an angle of attack design variable α is added to trim the airfoil. The problem is summarized below in Table 6.

Table 6 Optimization Problem for Transonic RAE 2822 drag minimization

Category	Name	Quantity	Lower	Upper
Objective	c_d	1	—	—
Variable	y	20	-0.05	0.05
	α	1	0	10
Constraints	c_ℓ	1	1.0	1.0
	c_m	1	-0.092	—
	A	1	A_{initial}	—
	$y_{\text{LE,lower}} = -y_{\text{LE,upper}}$	1	0.0	0.0
	$y_{\text{TE,lower}} = -y_{\text{TE,upper}}$	1	0.0	0.0
	t/c	400	0.3	—

The $L0$ mesh used in ADflow along with the 20 FFD points used for all optimizations are shown in Figure 7.

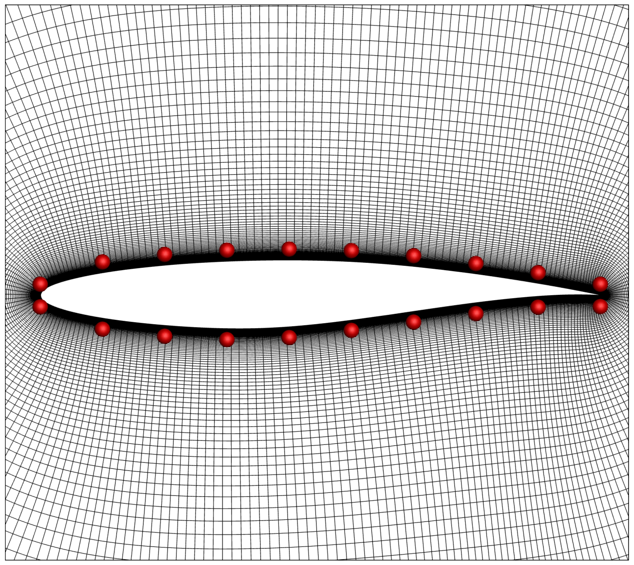

Fig. 7 $L0$ mesh for RAE 0012 Airfoil with 20 FFD points used for optimization.

A. RAE 2822 Meshes

Optimizations using xflow use a C-topology mesh to conform to the sharp trailing edge of the geometry. The mesh uses cubic elements, $q = 3$, and 150 elements on the airfoil and 20 elements in the wake with 24 elements in the normal wall normal direction. In ADflow however, we use a family of O-topology meshes to be consistent with previous studies for the RAE2822 airfoil [33]. We generated a family of O-topology meshes, summarized in Table 7, with the finest, $L00$, having $291,840$ cells and the coarsest mesh, $L2$, with $4,560$ cells.

Table 7 RAE 2822 O-Topology Mesh Family Summary

Name	Cells On Wall	Total Cells
$L00$	480	291,840
$L0$	240	72,960
$L1$	120	18,240
$L2$	60	4,560

B. ADflow Results

The optimization in ADflow was run on the $L0$, $L1$, and $L2$ meshes and the final drag results are summarized below in Table 8. The "True" values for the drag come from taking each design and reanalyzing it in xflow at $p = 3$ on C-topology mesh described above and again in ADflow on the $L00$ mesh.

Table 8 RAE 2822 optimization with ADflow summary

Mesh	DoFs	Time (hr)	Optimized $c_d \times 10^4$	xflow True $c_d \times 10^4$	ADflow True $c_d \times 10^4$
$L0$	72,960	5.4	105.86	104.93	105.24
$L1$	18,240	0.078	111.00	111.01	106.86
$L2$	4,560	0.025	134.47	111.79	112.21

The "true" drag coefficient, in both xflow and ADflow, for each design goes down as the mesh on which the optimization is done is refined. This is due to the optimizer taking advantage of discretization error to smooth out the shock rather than geometric features. The shocks come back when analyzed in both xflow and in ADflow on the $L00$. This is shown in Figure 8 where the $L2$ mesh design has the largest pressure drop and the $L0$ mesh design has a very weak shock on the upper surface.

These plots shows that each mesh level causes the optimized shape to be noticeably different. As the mesh gets more refined the lower surface towards the front of the airfoil gets pulled down more and the upper surface along the whole chord moves to lower y/c values.

C. xflow Results

We ran the same problem again in xflow on the a fixed C-topology mesh described above in Section VII.A. We tested a fixed-fidelity optimization running with solution orders of $p = 3$, $p = 2$, and $p = 1$. Finally, we tested an adaptive fidelity with a final tolerance on the Lagrangian discretization error of 5×10^{-5} and started all elements $p = 1$. We used a fixed fraction p-adaptation approach where 12.5% of elements with the highest error were refined and and the 2.5% of elements with the lowest error were coarsened and limited the approximation order to be between 1 and 3. We compute the error estimate used to decide when to adapt every 10 major iterations. This is because for this problem we require 3 fine-space adjoint solutions that if calculated every major iteration will add significant computational cost. Results for the DG optimizations are shown in Table 9.

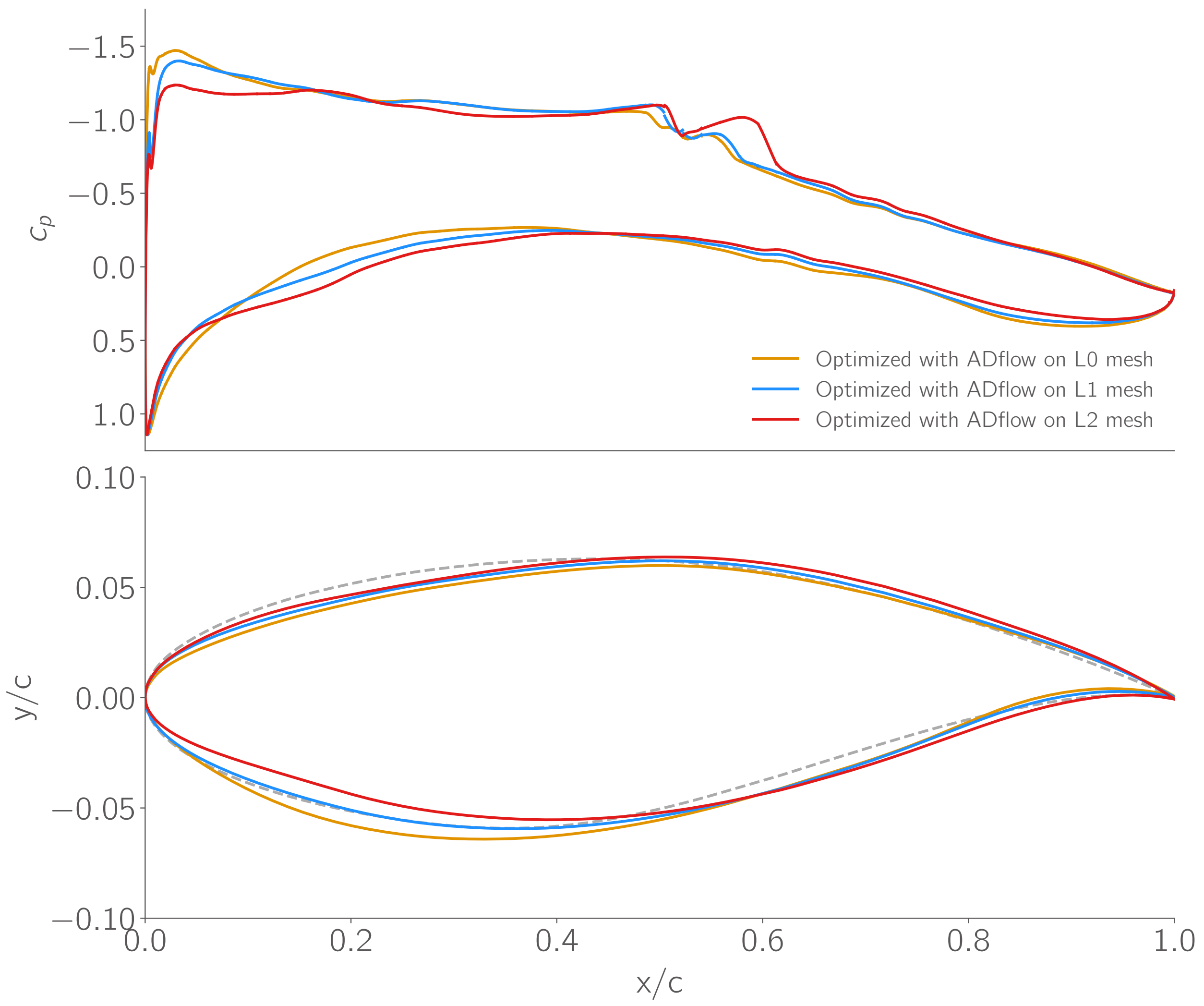

Fig. 8 Optimized airfoil geometries using ADflow with various mesh resolution shows the leading edge gets thicker as the mesh gets finer. c_p distributions generated from xflow "true" solution.

Table 9 RAE 2822 optimization with xflow summary

Solution Order	DoFs	Time (hr)	Optimized $c_d \times 10^4$	xflow True $c_d \times 10^4$	ADflow True $c_d \times 10^4$
3	72,960	39.88	104.34	104.34	105.53
2	41,040	7.56	106.69	105.48	106.35
1	18,240	1.04	147.48	146.57	115.12
1/2/3	41,041	20.75	104.58	104.34	105.57

The geometries and c_p plots, generated in xflow at $p = 3$, are shown in Figure 9. The airfoil optimized at $p = 1$ is

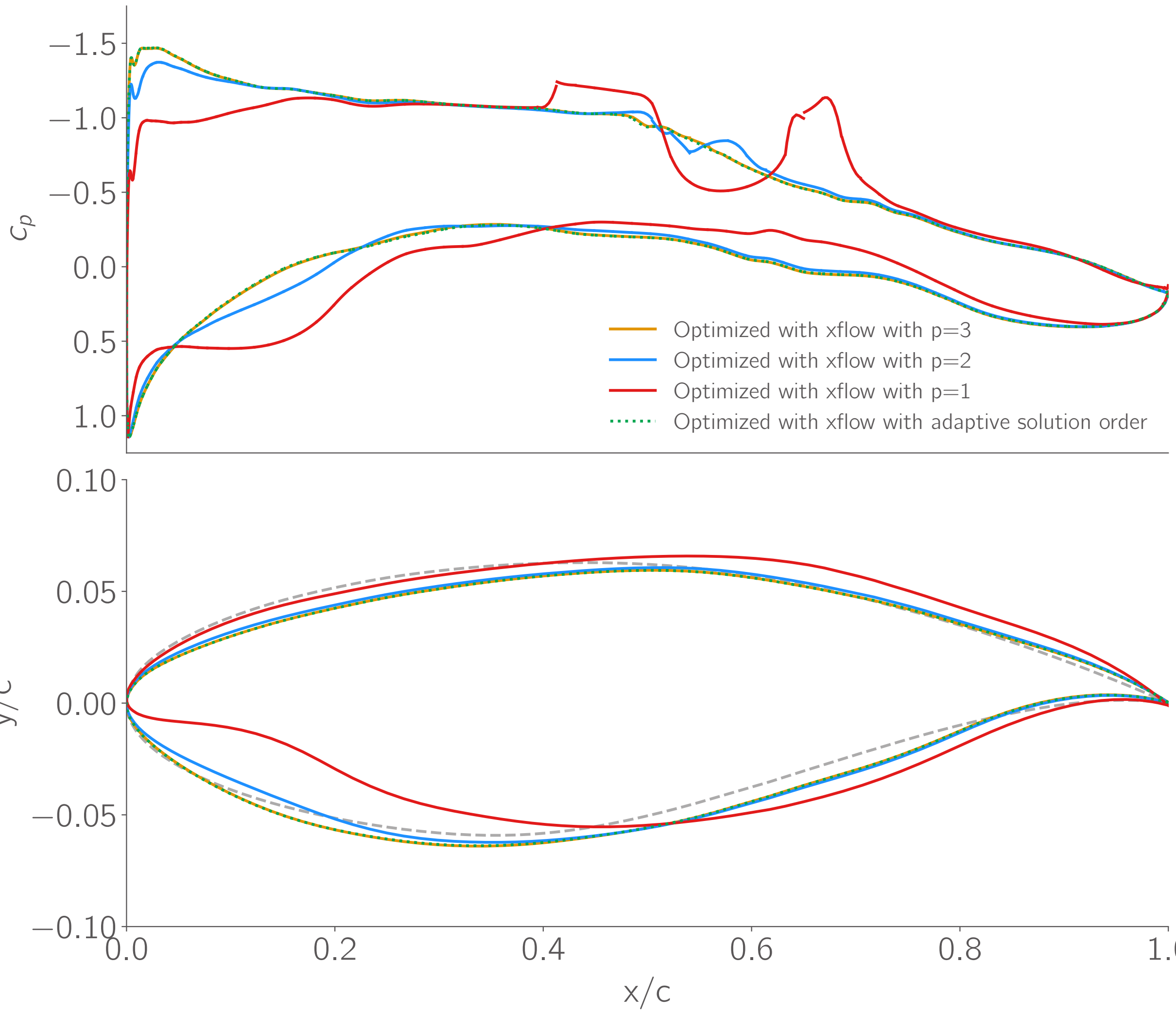

Fig. 9 Optimized airfoil geometries from fixed fidelity and p-adapted DG optimizations show p-adaption achieves the same result as $p = 3$. c_p generated from "true" xflow result.

very different than all other optimized airfoils. This is due to large discretization error that the optimizer uses to reduce drag. When this design is reanalyzed in xflow, the drag is very similar but the flow field is very different shown in Figure 10. A double shock shows up on the upper surface when analyzing this design at $p = 3$ but at $p = 1$ the airfoil is shock free. The double shock counteracts the numerical drag from the $p = 1$ discretization error so that the final drag counts are very similar. When looking at this design in ADflow two shocks are on the upper surface but are weaker. There is also less numerical drag so that the final drag coefficient is lower than in xflow at $p = 1$ and at $p = 3$.

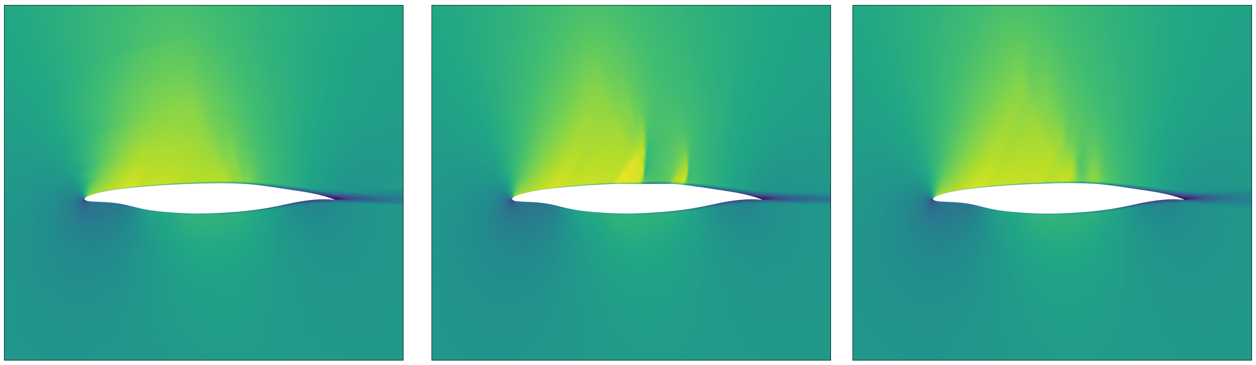
(a) xflow at $p = 1$. **(b) xflow at $p = 3$.** **(c) ADflow on $L00$ mesh.**

Fig. 10 Mach contours (0 to 1.3) on optimized airfoil from xflow with $p = 1$ in different solvers.

In the case of fixed-fidelity DG, $p = 3$ gets the lowest drag optimized design. The airfoil optimized at $p = 2$ is 1 drag count higher from the airfoil optimized at $p = 3$ due to the shock reappearing.

Using adaptation, the final number of degrees of freedom is nearly the same as the $p = 2$ solution but the final design matches the $p = 3$ design to .01 drag counts using fewer degrees of freedom and is almost 50% faster. This is shown when looking at the geometries and c_p plots the adapted case and $p = 3$ are both right on top of each other while the $p = 2$ is close but offset in some regions. However, even though the adaptation case uses the same number of degrees of freedom, the total optimization time takes much longer due to the extra fine-space adjoint solutions needed for both computing the error estimates and the adaptation indicators. Additionally, when adapting, the optimizer gets "kicked" off a spurious optimum: a shock appears that must be smoothed out.

When comparing to the designs from the optimizations using ADflow, even though the $L0$ mesh has the same number of degrees of freedom as $p = 3$, the drag is 0.6 counts higher. Additionally, once we adapt, xflow is able to perform even better using fewer degrees of freedom but when time is a constraint ADflow is much faster. This is due to ADflow being a code that takes advantage of structured meshes for more efficient memory access and the code being optimized for optimization problems. xflow on the other hand is and unstructured code that was developed for research purposes to be modular and easy to develop rather than being fast. Additionally, ADflow has fully analytic derivatives while in xflow there is an analytic adjoint but partial terms required for total derivatives are computed using finite difference which takes longer and is less accurate. The less accurate derivatives cause the optimizer to take more iterations to find the optimum.

VIII. Conclusion

When doing aerodynamic shape optimization the CFD solver plays a key role in the overall computational cost and accuracy of the final design. During ASO, there could be hundreds of primal and adjoint CFD solutions that are required. Additionally, during initial iterations the optimizer might request a solution on an obscure geometry on which the CFD solver needs to find a solution. Finally, discretization error in CFD can lead to the optimizer finding spurious optima by prioritizing lowering numerical drag. These factors leads to the necessity of having a efficient, robust, and accurate CFD solution.

In this work we presented a direct comparison of aerodynamic shape optimization using both finite volume and high-order discontinuous Galerkin CFD. We added our DG solver, xflow, into the existing MACH-Aero gradient based optimization framework. Additionally, we modified the framework to pass additional information between the optimizer and CFD solver to perform error estimation on the and p-adaptation during the optimization. The adaptation was automated based on a final desired error tolerance and scheduling based on the KKT conditions.

In both test cases studied here, we showed that using DG in optimization is able to find the same optimum as FVM using fewer degrees of freedom. In the finite volume cases, refining a mesh showed drastic design differences at the optimum indicating that the coarser meshes found a spurious optimum due to the numerical error. In DG when on a fixed coarse mesh, increasing the order everywhere showed less design changes, indicating that even on a coarse mesh high-order solutions are able to converge faster. Additionally, with our adaptation strategy we were able to find the same optimum as when using the finest CFD solution with fewer degrees of freedom and time. While in this work the time for finite volume was still much faster than using DG, a direct comparison between the two had multiple factors that favored the specific finite volume solver chosen. If a DG solver is developed with ASO in mind, the time could be drastically reduced. While this work only used p-adaptation to easily plug into the current MACH-Aero framework, using mesh refinement or optimization could further improve the number of degrees of freedom and time required to reach the optimum at a desired error tolerance.

Acknowledgments

The first author would like to acknowledge the support received through various stages of the research by the Department of Defense (DoD) through the National Defense Science and Engineering Graduate (NDSEG) Fellowship Program and the Rackham Graduate School at University of Michigan through the Rackham Merit Fellowship. The first author also received partial support from the University of Michigan Institute for Computational Discovery and Engineering Fellowship.

References

[1] Wang, Z., Fidkowski, K., Abgrall, R., Bassi, F., Caraeni, D., Cary, A., Deconinck, H., Hartmann, R., Hillewaert, K., Huynh, H., Kroll, N., May, G., Persson, P.-O., van Leer, B., and Visbal, M., "High-Order CFD Methods: Current Status and Perspective," *International Journal for Numerical Methods in Fluids*, 2013. doi:10.1002/fld.3767.

[2] Fidkowski, K. J., and Darmofal, D. L., "Review of output-based error estimation and mesh adaptation in computational fluid dynamics," *AIAA Journal*, Vol. 49, No. 4, 2011, pp. 673–694. doi:10.2514/1.J050073.

[3] Wu, N., Mader, C. A., and Martins, J. R. R. A., "A Gradient-based Sequential Multifidelity Approach to Multidisciplinary Design Optimization," *Structural and Multidisciplinary Optimization*, Vol. 65, 2022, pp. 131–151. doi:10.1007/s00158-022-03204-1.

[4] Hicken, J. E., and Alonso, J. J., "PDE-constrained optimization with error estimation and control," *Journal of Computational Physics*, Vol. 263, 2014, pp. 136–150. doi:10.1016/j.jcp.2013.12.050.

[5] Chen, G., and Fidkowski, K. J., "Discretization Error Control for Constrained Aerodynamic Shape Optimization," *Journal of Computational Physics*, Vol. 387, 2019, pp. 163–185. doi:10.1016/j.jcp.2019.02.038.

[6] Chen, G., and Fidkowski, K. J., "Variable-fidelity multipoint aerodynamic shape optimization with output-based adapted meshes," *Aerospace Science and Technology*, Vol. 105, 2020. doi:10.1016/j.ast.2020.106004.

[7] Brown, D. A., and Nadarajah, S., "Inexactly constrained discrete adjoint approach for steepest descent-based optimization algorithms," *Numerical Algorithms*, Vol. 78, No. 3, 2018, pp. 983–1000. doi:10.1007/s11075-017-0409-7.

[8] Brown, D. A., and Nadarajah, S., "Effect of inexact adjoint solutions on the discrete-adjoint approach to gradient-based optimization," *Optimization and Engineering*, Vol. 23, 2022, pp. 1643–1676. doi:10.1007/s11081-021-09681-5.

[9] Spalart, P., and Allmaras, S., "A One-Equation Turbulence Model for Aerodynamic Flows," *La Recherche Aerospatiale*, Vol. 1, 1994, pp. 5–21.

[10] Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 43, 1981, pp. 357–372. doi:10.1016/0021-9991(81)90128-5.

[11] Bassi, F., and Rebay, S., "Numerical evaluation of two discontinuous Galerkin methods for the compressible Navier-Stokes, equations," *International Journal for Numerical Methods in Fluids*, Vol. 40, 2002, pp. 197–207.

[12] Ceze, M., and Fidkowski, K., "Pseudo-transient Continuation, Solution Update Methods, and CFL Strategies for DG Discretizations of the RANS-SA Equations," *21st AIAA Computational Fluid Dynamics Conference*, American Institute of Aeronautics and Astronautics, 2013. doi:10.2514/6.2013-2686.

[13] Saad, Y., and Schultz, M. H., "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM Journal on Scientific and Statistical Computing*, Vol. 7, No. 3, 1986, pp. 856–869. doi:10.1137/0907058.

[14] Fidkowski, K. J., Oliver, T. A., Lu, J., and Darmofal, D. L., "p-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier–Stokes equations," *Journal of Computational Physics*, Vol. 207, No. 1, 2005, pp. 92–113. doi:10.1016/j.jcp.2005.01.005, URL `http://www.sciencedirect.com/science/article/pii/S0021999105000185`.

[15] Persson, P., and Peraire, J., "Newton-GMRES Preconditioning for Discontinuous Galerkin Discretizations of the Navier–Stokes Equations," *SIAM Journal on Scientific Computing*, Vol. 30, No. 6, 2008, pp. 2709–2733. doi:10.1137/070692108, URL `https://epubs.siam.org/doi/abs/10.1137/070692108`.

[16] Mader, C. A., Kenway, G. K. W., Yildirim, A., and Martins, J. R. R. A., "ADflow: An open-source computational fluid dynamics solver for aerodynamic and multidisciplinary optimization," *Journal of Aerospace Information Systems*, Vol. 17, No. 9, 2020, pp. 508–527. doi:10.2514/1.I010796.

[17] Kenway, G. K. W., Mader, C. A., He, P., and Martins, J. R. R. A., "Effective Adjoint Approaches for Computational Fluid Dynamics," *Progress in Aerospace Sciences*, Vol. 110, 2019, p. 100542. doi:10.1016/j.paerosci.2019.05.002.

[18] Yildirim, A., Kenway, G. K. W., Mader, C. A., and Martins, J. R. R. A., "A Jacobian-free approximate Newton–Krylov startup strategy for RANS simulations," *Journal of Computational Physics*, Vol. 397, 2019, p. 108741. doi:10.1016/j.jcp.2019.06.018.

[19] Sederberg, T. W., and Parry, S. R., "Free-form Deformation of Solid Geometric Models," *SIGGRAPH Comput. Graph.*, Vol. 20, No. 4, 1986, pp. 151–160. doi:10.1145/15886.15903.

[20] Kenway, G. K., Kennedy, G. J., and Martins, J. R. R. A., "A CAD-Free Approach to High-Fidelity Aerostructural Optimization," *Proceedings of the 13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, Fort Worth, TX, 2010. doi:10.2514/6.2010-9231.

[21] Secco, N., Kenway, G. K. W., He, P., Mader, C. A., and Martins, J. R. R. A., "Efficient Mesh Generation and Deformation for Aerodynamic Shape Optimization," *AIAA Journal*, Vol. 59, No. 4, 2021, pp. 1151–1168. doi:10.2514/1.J059491.

[22] Luke, E., Collins, E., and Blades, E., "A Fast Mesh Deformation Method Using Explicit Interpolation," *Journal of Computational Physics*, Vol. 231, No. 2, 2012, pp. 586–601. doi:10.1016/j.jcp.2011.09.021.

[23] Wu, N., Kenway, G., Mader, C. A., Jasa, J., and Martins, J. R. R. A., "pyOptSparse: A Python framework for large-scale constrained nonlinear optimization of sparse systems," *Journal of Open Source Software*, Vol. 5, No. 54, 2020, p. 2564. doi:10.21105/joss.02564.

[24] 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. doi:10.1137/S0036144504446096.

[25] Martins, J. R. R. A., and Ning, A., *Engineering Design Optimization*, Cambridge University Press, Cambridge, UK, 2021. doi:10.1017/9781108980647, URL `https://mdobook.github.io`.

[26] Boyd, S. P., and Vandenberghe, L., *Convex Optimization*, Cambridge University Press, Cambridge, UK, 2004.

[27] Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., "The Complex-Step Derivative Approximation," *ACM Transactions on Mathematical Software*, Vol. 29, No. 3, 2003, pp. 245–262. doi:10.1145/838250.838251.

[28] Griewank, A., *Evaluating Derivatives*, SIAM, Philadelphia, 2000.

[29] Jameson, A., "Aerodynamic Design via Control Theory," *Journal of Scientific Computing*, Vol. 3, No. 3, 1988, pp. 233–260. doi:10.1007/BF01061285.

[30] Becker, R., and Rannacher, R., "An optimal control approach to a posteriori error estimation in finite element methods," *Acta Numerica*, edited by A. Iserles, Cambridge University Press, 2001, pp. 1–102.

[31] Park, M. A., "Adjoint-Based, Three-Dimensional Error Prediction and Grid Adaptation," AIAA Paper 2002-3286, 2002.

[32] Lambe, A. B., and Martins, J. R. R. A., "Extensions to the Design Structure Matrix for the Description of Multidisciplinary Design, Analysis, and Optimization Processes," *Structural and Multidisciplinary Optimization*, Vol. 46, No. 2, 2012, pp. 273–284. doi:10.1007/s00158-012-0763-y.

[33] He, X., Li, J., Mader, C. A., Yildirim, A., and Martins, J. R. R. A., "Robust aerodynamic shape optimization—from a circle to an airfoil," *Aerospace Science and Technology*, Vol. 87, 2019, pp. 48–61. doi:10.1016/j.ast.2019.01.051.