AIAA JOURNAL
Vol. 63, No. 3, March 2025

Monolithic *p*-Adaptive High-Order Aerodynamic Shape Optimization

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

https://doi.org/10.2514/1.J063892

Aerodynamic shape optimization requires a robust, accurate, and efficient flow solver. However, during aerodynamic shape optimization, large geometry and flow solution changes may decrease solution accuracy and efficiency on fixed meshes. The optimizer may converge to a spurious optimum if the solution loses accuracy. We use the discontinuous Galerkin (DG) method to tackle this problem because it yields high-order-accurate solutions that often have less error per degree of freedom compared to second-order finite-volume methods. Because DG degrees of freedom often incur a higher computational cost, we take advantage of local adaptation to maximize accuracy at a given cost. However, during optimization, it is not clear when to adapt to avoid overoptimizing initial designs and to avoid errors polluting the optimal solution. We develop an adaptation strategy that reaches a target error at the end of a single optimization loop. Finally, we present results for two airfoil optimization test cases. Our results show that this adaptation procedure outperforms optimization using fixed-fidelity DG and second-order finite volume on a per-degree-of-freedom basis.

I. Introduction

IN RECENT years, aerodynamic shape optimization (ASO) has seen many advancements and is now mature. The key components required to perform ASO are a computational fluid dynamics (CFD) solver with adjoint derivative computation, a gradient-based optimizer, a differentiated geometric parameterization, and automated mesh movement with adjoint derivatives [1]. The choice of the CFD solver is consequential because the CFD solution is typically the most computationally costly step in an optimization iteration. Therefore, the solver must be efficient because the optimization typically requires hundreds of flow solutions. Additionally, the CFD solver must be robust and provide a solution for shapes that a human designer would not usually evaluate [1]. If the CFD solver cannot provide a solution to the optimizer, the whole optimization process could fail even if that design is nowhere near the optimum. Finally, the solver must be accurate because numerical errors affect the design space and may lead to a spurious optimum. Second-order finite volume method (FVM) CFD solvers meet these criteria when using a sufficiently fine mesh.

Additionally, high-order methods, such as discontinuous Galerkin (DG), offer highly accurate solutions better than low-order approximations in terms of degrees of freedom, number of unknowns for each state, and sometimes computational cost [2]. However, to realize the benefits of high-order DG solutions on practical problems, some form of adaptation is required to reduce the degrees of freedom in regions that do not require high resolution for accurate output prediction and concentrate them in regions that do [3].

Controlling discretization errors is vital in optimization. A fine discretization is costly but necessary to find the "true" optimum. However, using a fixed-fidelity approach wastes computational effort during initial optimization iterations. Therefore, a multifidelity approach is ideal for preventing overoptimizing a coarse mesh that leads to spurious optima and over-refining designs that are from the optimum.

Numerous multifidelity approaches for optimization have been previously proposed in the literature. Many approaches have taken a sequential optimization approach where the optimization problem is solved multiple times at varying cost or error targets [4–8]. These approaches require the user to set the number of sequential optimization levels and the error and convergence tolerances for each level, which are not always obvious choices. Additionally, many researchers have adapted the mesh at each design iteration to reach the prescribed error for that optimization [4,5,7–9]. Requiring multiple mesh adaptation iterations at each design significantly increases computational cost because each design requires multiple flow and adjoint solutions. Lu [10] controlled both discretization error, through *p*-adaptation, and iteration error, by changing the number of solution iterations. They used the quadratic penalty method for constraints and adapted based on the error of the penalized objective. This approach improved the final objective value and achieved a lower error when reanalyzed with a finer mesh. However, this approach does not account for how constraint errors affect the objective, especially when the airfoil has to be retrimmed. Nemec and Aftosmis [4] used a sequential optimization approach where the multiple optimizations were performed at user-specified mesh resolutions. They used a progressive strategy where each optimization they increased the number of adaptation iterations at each design point. Results showed that overoptimizing on too coarse of a mesh resulted in the optimizer taking more iterations on the finer meshes to backtrack from a spurious optimum to get to the true optimum. Hicken and Alonso [6] computed errors in gradient norms and used these as an adaptive indicator to control first-order optimality error during the optimization. To estimate the gradient norm, additional adjoint equations need to be solved that require second-order partial derivatives that are not easily computed. This approach did not require error estimation or adaptation at each design iteration, which reduces computational expense, as the error estimation requires two additional adjoint solutions. Their results showed a lower objective value and tighter optimality convergence compared to a fixed grid approach. However, this approach was limited to optimization problems with only partial differential equation (PDE) constraints and did not handle other constraints that are present in practical engineering problems. Chen and Fidkowski [7,8] computed error in the objective function while accounting for how discretization error in general constraints affects the objective and performed error and cost-based optimization with specified error tolerances and costs. This approach directly controls the discretization error of the objective function and

Presented as Paper 2023-1845 at the 2023 AIAA SciTech Forum, National Harbor, MD, January 23–27, 2023; received 7 March 2024; accepted for publication 16 December 2024; published online 28 January 2025.

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

uses a sequential approach relying on user to specify the number of optimization levels and the cost or error for each optimization. Additionally, the mesh was adapted at each optimization update, requiring multiple flow and adjoint solutions for each update. However, this approach was successful in significantly reducing the computational cost to achieve the same optimum as a fixed high-resolution optimization. Brown and Nadarajah [11,12] adaptively changed tolerances for solving the primal and adjoint equations during optimization based on first-order optimality convergence for PDE-constrained problems. This approach used relaxed tolerances for the primal and adjoint equations early in the optimization and progressively increased as the optimizer got closer to the optimum. Results showed a reduction in computational expense by adapting these solver tolerances. Wu et al. [13] used a sequential multifidelity approach where the optimizer switched between multiple meshes and sets of governing equations during the optimization. This approach requires quantifying the errors between the different fidelities on the initial design and does not update the error estimates as the design changes. Their results showed that, compared to a high fixed-fidelity case, their mixed-fidelity optimization required lower computational cost but more optimization steps due to the added noise of switching fidelities in the problem.

Much of the previous work has either adapted the solution at every iteration, performed sequential optimizations, or not accounted for general output constraint error. In this paper, we develop a monolithic approach to controlling discretization error during optimization using order (p) adaptation. Instead of performing sequential optimizations at various cost or error targets, we control the error during one optimization and require only one user input—the final target error. As the optimization progresses, the adaptation reduces the effect of discretization error on the augmented Lagrangian. Additionally, we do not adapt the approximation space at each design iteration. The solution is only adapted when the discretization error is too high for the current optimality. We present two test cases for our adaptation strategy: a subsonic airfoil and a transonic airfoil, where the initial design causes a strong shock. We compare our optimizations using adaptation to optimizations with uniform p refinement and optimizations using a second-order finite volume solver.

II. Computational Framework

We use MACH-Aero, an open-source framework for gradient-based ASO.[§] The framework consists of various modules for parameterizing the geometry, mesh warping, computing CFD solutions, and performing optimization. The MACH-Aero framework allows users to easily exchange modules, such as the CFD solver. In this work, we added a MACH-Aero interface for xflow, a high-order DG CFD solver. Note that xflow has a discrete adjoint, and that has been used previously for output-based error estimation and adaptation with both p and h adaptation.

A. Discontinuous Galerkin CFD Solver

CFD requires discretizing a set of governing equations that model the fluid. We solve the Reynolds-averaged Navier–Stokes (RANS) equations closed with the Spalart–Allmaras (SA) turbulence model [14]. The governing equations can be written as

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

where $\boldsymbol{u} \in \mathbb{R}^s$ is the s-component state vector, $\vec{\boldsymbol{F}} \in \mathbb{R}^{d \times s}$ is the flux vector, d is the number of spatial dimensions, and $\boldsymbol{S} \in \mathbb{R}^s$ is the source term arising from the turbulence model. We use the xflow CFD solver [15,16], which discretizes the governing equations via the DG method with the Roe [17] convective flux and the second form of Bassi and Rebay (BR2) [18] 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

$$\boldsymbol{M}\frac{\mathrm{d}\boldsymbol{U}}{\mathrm{d}t} + \boldsymbol{R}(\boldsymbol{U}) = \boldsymbol{0} \tag{2}$$

where $\boldsymbol{U} \in \mathbb{R}^N$ is the discrete state vector, N is the total number of unknowns, $\boldsymbol{R}(\cdot) \in \mathbb{R}^N$ is the nonlinear spatial residual, and $\boldsymbol{M} \in \mathbb{R}^{N \times N}$ is the block-element sparse mass matrix. We neglect the time derivative term for the steady simulations, but pseudo-time continuation [19] remains in the solver to converge the steady residual. The nonlinear solver uses the Newton–Raphson method with the generalized minimum residual (GMRES) linear solver [20], preconditioned by an element-line Jacobi smoother with coarse-level ($p = 1$) correction [21,22].

B. Finite Volume CFD Solver

The second CFD solver used is ADflow, a second-order cell-centered finite volume flow solver for multiblock and overset meshes [23]. ADflow includes a discrete adjoint implementation [24] and a robust approximate Newton–Krylov startup strategy [25].

C. Geometric Parameterization

To parameterize the geometry, we use the free-form deformation (FFD) approach [26] through the pyGeo implementation [27]. The FFD approach uses a movable control point box that embeds the baseline geometry. Moving the control points results in the movement of nodes on the geometry surface. This approach is efficient because it does not directly parameterize the shape of the geometry but instead parameterizes deformations. This allows the movement of all surface nodes with far fewer design variables than surface nodes. Note that pyGeo also includes modules for thickness and area constraints.

D. Mesh Warping

Once surface nodes are deformed using pyGeo, the remaining volume nodes must be updated. To do this, we use IDwarp [28], which uses an inverse-distance weighting method proposed by Luke et al. [29] 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 finer linear meshes by linearly connecting all high-order surface nodes.

E. Optimizer

We use pyOptSparse [30], an optimization framework for large-scale gradient-based optimizations; pyOptSparse provides wrappers for several optimization packages. This work uses the SNOPT optimizer, a general-purpose optimizer that uses sequential quadratic programming (SQP) to minimize nonlinear functions subject to linear or nonlinear constraints [31].

III. Constrained Optimization

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

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

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

[§]Data available online at https://mdolab-mach-aero.readthedocs-hosted.com/en/latest/.

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

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; $\boldsymbol{s} \in \mathbb{R}^{n_g}$ is a slack variable to turn inequality constraints into equality constraints; and $\odot$ represents elementwise multiplication ([32], Sec. 5.5). Differentiating the Lagrangian with respect to the design variables, $\boldsymbol{x}$, slack variables, $\boldsymbol{s}$, and Lagrange multipliers, $\boldsymbol{\lambda}$ and $\boldsymbol{\sigma}$, and setting them equal to 0, yield the following equations:

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

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

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

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

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

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

A. Gradient Computation

To perform gradient-based optimization, we must compute the gradients of all objectives and constraints with respect to all design variables. There are several approaches for computing these gradients ([32], Chap. 6), such as finite differences, complex step [33], algorithmic differentiation [34], and the adjoint method [35].

This work uses the adjoint method, which performs well for large optimization problems because it scales with the number of functions of interest and not the number of design variables ([32], Sec. 6.7). We compute the gradient of some function J that is a function of both some state vector $\boldsymbol{U}$ and the design variables $\boldsymbol{x}$:

$$J = J(\boldsymbol{U}, \boldsymbol{x}) \tag{10}$$

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

$$\boldsymbol{R}(\boldsymbol{U}, \boldsymbol{x}) = \boldsymbol{0} \tag{11}$$

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

$$\frac{\mathrm{d}J}{\mathrm{d}\boldsymbol{x}} = \frac{\partial J}{\partial \boldsymbol{x}} + \frac{\partial J}{\partial \boldsymbol{U}} \frac{\mathrm{d}\boldsymbol{U}}{\mathrm{d}\boldsymbol{x}} \tag{12}$$

The term $\mathrm{d}\boldsymbol{U}/\mathrm{d}\boldsymbol{x}$ is solved by applying the chain rule to the residual $\boldsymbol{R}$ and computing the total derivative of the residual with respect to the design variables. This gradient is $\boldsymbol{0}$ if the residual is driven to $\boldsymbol{0}$ at each design point:

$$\frac{\mathrm{d}\boldsymbol{R}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}} \frac{\mathrm{d}\boldsymbol{U}}{\mathrm{d}\boldsymbol{x}} = \boldsymbol{0} \tag{13}$$

$$\frac{\mathrm{d}\boldsymbol{U}}{\mathrm{d}\boldsymbol{x}} = -\left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}}\right)^{-1} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} \tag{14}$$

The expression for $\mathrm{d}\boldsymbol{U}/\mathrm{d}\boldsymbol{x}$ is then substituted into Eq. (12) to yield

$$\frac{\mathrm{d}J}{\mathrm{d}\boldsymbol{x}} = \frac{\partial J}{\partial \boldsymbol{x}} - \frac{\partial J}{\partial \boldsymbol{U}} \left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}}\right)^{-1} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} \tag{15}$$

This equation could be solved naively by solving the linear system $(\partial \boldsymbol{R}/\partial \boldsymbol{U})^{-1} \partial \boldsymbol{R}/\partial \boldsymbol{x}$. This is known as the direct method and requires solving a linear system for each design variable.

Instead of using the direct method, we define the adjoint vector as

$$\boldsymbol{\Psi} = -\left(\frac{\partial \boldsymbol{R}^T}{\partial \boldsymbol{U}}\right)^{-1} \frac{\partial J^T}{\partial \boldsymbol{U}} \tag{16}$$

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 write the gradient of the function as

$$\frac{\mathrm{d}J}{\mathrm{d}\boldsymbol{x}} = \frac{\partial J}{\partial \boldsymbol{x}} + \boldsymbol{\Psi}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} \tag{17}$$

The remaining partial derivatives on the right-hand side are computationally inexpensive compared to the flow and adjoint solutions because 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 that affects the optimization performance. In ADflow, these partial terms are calculated through algorithmic differentiation. This yields exact gradients that have been verified with the complex-step method [24]. 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 compute the partial derivatives using a forward difference approximation. This leads to a higher cost to compute the gradient because we must warp the mesh for each shape design variable. It also leads to less accurate gradients, which causes the optimizer to perform more major iterations for the same optimality and feasibility tolerances.

IV. Error Estimation and Adaptive Indicators

When solving a discretized PDE on a coarse approximation space, H, discretization error 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 estimate the error relative to a finer space h that can be used in place of the exact PDE:

$$\text{output error} \approx \delta J = J_H(\boldsymbol{U}_H) - J_h(\boldsymbol{U}_h) \tag{18}$$

Here, J represents some output of interest that is the objective value or a constraint. To obtain the fine space h, we increment each element's approximation order p_e to $p_e + 1$. The output on the fine-space is not solved for directly. Instead, we compute the output-error estimate using the adjoint-weighted residual method [3,36,37]. The fine-space adjoint solution $\boldsymbol{\Psi}_h$ is computed about $\boldsymbol{U}_h^H$, the coarse-space solution injected into the fine-space. The output error comes from perturbations in the state, leading to perturbations in the residual, as follows:

$$\begin{aligned} \delta J &\approx J_H(\boldsymbol{U}_H) - J_h(\boldsymbol{U}_h), \\ &= J_h(\boldsymbol{U}_h^H) - J_h(\boldsymbol{U}_h) \approx \frac{\partial J_h}{\partial \boldsymbol{U}_h} \delta \boldsymbol{U}, \\ &= -\boldsymbol{\Psi}_h^T \delta \boldsymbol{R}_h = -\boldsymbol{\Psi}_h^T (\boldsymbol{R}_h(\boldsymbol{U}_h^H) - \boldsymbol{R}_h(\boldsymbol{U}_h)) \\ &= -\boldsymbol{\Psi}_h^T \boldsymbol{R}_h(\boldsymbol{U}_h^H) \end{aligned} \tag{19}$$

Here, $\boldsymbol{U}_h$ is the state associated with the solution to the fine-space problem, leading to $\boldsymbol{R}_h(\boldsymbol{U}_h) = \boldsymbol{0}$. This derivation assumes that the

injection into the fine space does not change the output of interest, $J_h(\boldsymbol{U}_h^H) = J_H(\boldsymbol{U}_H)$.

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

$$\begin{aligned}\delta J &= J_H(\boldsymbol{U}_H) - J_h(\boldsymbol{U}_h) \\ &= -\boldsymbol{\Psi}_h^T \boldsymbol{R}_h(\boldsymbol{U}_h^H) \\ &= -\sum_{e=1}^{n_e} \boldsymbol{\Psi}_{h,e}^T \boldsymbol{R}_{h,e}(\boldsymbol{U}_h^H)\end{aligned} \tag{20}$$

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

$$\epsilon_e \equiv |\boldsymbol{\Psi}_{h,e}^T \boldsymbol{R}_{h,e}(\boldsymbol{U}_h^H)| \tag{21}$$

A. Error 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, underpredicting lift in the coarse space would underpredict 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:

$$\begin{aligned}\delta\mathcal{L} &= \mathcal{L}_H(\boldsymbol{U}_H) - \mathcal{L}_h(\boldsymbol{U}_h) \\ &= \delta f + \boldsymbol{\lambda}^T \delta \boldsymbol{h} + \boldsymbol{\sigma}^T \delta \boldsymbol{g}\end{aligned} \tag{22}$$

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 unaffected by the CFD discretization level; therefore, the associated discretization errors are zero.

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

$$\epsilon_e = |\delta f_e| + |\boldsymbol{\lambda}^T||\delta \boldsymbol{h}_e| + |\boldsymbol{\sigma}^T||\delta \boldsymbol{g}_e| \tag{23}$$

V. Adaptation Strategy

During the optimization process, the discretization error must be low enough at the optimum to avoid spurious optima. At the same time, it is inefficient to have many degrees of freedom early in the optimization process when the design and the distribution of discretization error in the mesh change rapidly. To achieve this desired behavior, we use an adaptation strategy based on the convergence of the SNOPT optimality value, which is the residual of the first KKT condition given in Eq. (5), to determine when to adapt. This is different from the approach taken by Chen and Fidkowski [7], who ran sequential optimizations at different error tolerances and cost levels.

Brown and Nadarajah [12] derive a relationship between the primal and adjoint residual convergence to how tightly the first-order optimality conditions can be solved. They use this relationship to adapt how tightly the primal and adjoint are solved to avoid wasting computational resources early in the optimization process. Their algorithm looks at the relative convergence of the optimality and adapts the relative convergence of the primal and adjoint based on the optimality convergence. This approach only considers how the PDE constraint error propagates to the gradient of the objective. However, when there are constraints other than the governing PDE, the discretization error of the constraints directly affects the optimization. Additionally, the derivation for this approach assumes a steepest descent optimization algorithm with no constraints except for the PDE. In practical engineering problems, there are nearly always design constraints beyond the PDE. Also, the steepest descent algorithm converges slowly compared to alternatives such as quasi-Newton algorithms ([32], Example 4.18).

Chen and Fidkowski [8] present an approach that controls the discretization error of the objective, including the effects of the constraint error. Their approach runs sequential optimizations where each optimization is performed at a specified cost or error tolerance. During one of these optimizations, the mesh is adapted to meet the target cost or error tolerance at each design iteration. This approach requires the user or a heuristic algorithm to specify each optimization's cost or error tolerance.

We combine these approaches, but we change output error tolerances instead of changing residual tolerances based on optimality. Consider the case of an airfoil in supersonic flow. Large residuals downstream of the airfoil do not affect outputs calculated on the surface of the airfoil. Therefore, it is not always necessary to drive all PDE residual tolerances to zero, even close to the optimum. Instead, we look at which residuals directly affect the output through the adjoint-weighted residual error estimate in Eq. (19). From here, we can directly follow the approach of Brown and Nadarajah [12] and drive constraint errors down to a specified tolerance as we optimize. Instead, we take one further step and look at the error of the Lagrangian and drive this down to a specified tolerance by the end of the optimization process, using the optimality condition to tell us when to adapt.

Specifically, we look at the relative convergence of the output error, $R_{\delta\mathcal{L}}$, defined as

$$R_{\delta\mathcal{L}} \equiv \frac{\delta\mathcal{L}}{\epsilon_{\delta\mathcal{L}}^{\text{tol}}} \tag{24}$$

where $\epsilon_{\delta\mathcal{L}}^{\text{tol}}$ is the user-specified error tolerance of the Lagrangian. We specify that the relative convergence of the Lagrangian error is at most one order of magnitude lower than the relative convergence of the optimality:

$$R_{\delta\mathcal{L}} \leq \max\left(\frac{\tau_{\text{opt}}}{10\tau_{\text{opt}}^{\text{tol}}}, 1\right) \tag{25}$$

where τ_{opt} is the current optimality, and $\tau_{\text{opt}}^{\text{tol}}$ is the optimality tolerance. The factor of 10 ensures that the error tolerance is met before the optimality criteria, preventing overoptimization on a coarse mesh. When Eq. (25) is not satisfied, we adapt our solution.

Computing a fine-space adjoint for each CFD output at each major iteration would add significant computational expense to the optimization. To avoid this, we do not compute the error in the Lagrangian at every iteration; instead, we compute it in a lagged fashion. This is a similar approach taken by Hicken and Alonso [6], where they computed the error of the gradient only at the start of each sequential optimization. After the initial stages of the optimization, the design does not change rapidly between iterations, and on a fixed mesh, the discretization error also does not change significantly. Thus, 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 at which fine-space adjoint computations are required. Additionally, no adaptation takes place during the line search of the optimization to avoid introducing additional noise that could slow down the optimization. The updated optimization process is detailed in Algorithm 1.

We modify the MACH-Aero framework so that Lagrange multipliers are passed from the optimizer to the CFD solver, and the discretization error is passed to the optimizer. The extended design matrix (XDSM) diagram [38] is shown in Fig. 1.

Algorithm 1: Optimization with error estimation and adaptation

$\tau_{\text{opt}}^{\text{tol}}, \tau_{\text{feas}}^{\text{tol}}$ ▷ Set optimality and feasibility tolerances
$i = 0$ ▷ Major iteration counter
$x_i = x_0$ ▷ Set initial design
$\epsilon_{\mathcal{L}}^{\text{tol}}$ ▷ Set desired final error tolerance
n_{adapt} ▷ Initialize n_{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 feasibility criteria not met
  Compute $f, \boldsymbol{g}, \boldsymbol{h}, \nabla f, \nabla \boldsymbol{g}, \nabla \boldsymbol{h}$
  Perform line search updating $x_{i+1}, \lambda_{i+1}, \sigma_{i+1}, \tau_{\text{opt}}$, and τ_{feas}
  if $i = 0$ OR $n \pmod{n^{\text{adapt}}} = 0$ **then**
    Compute $\delta\mathcal{L}$ and $R_{\delta\mathcal{L}}$ ▷ Compute error estimate of Lagrangian and relative convergence of error
    $n = 0$ ▷ Reset n after computing error estimate
  end if
  if $R_{\delta\mathcal{L}} \geq \max\left(\frac{\tau_{\text{opt}}}{10\tau_{\text{opt}}^{\text{tol}}}, 1\right)$ **then**
    Adapt the discretization
  end if
  $n = n + 1$
  $i = i + 1$
end while

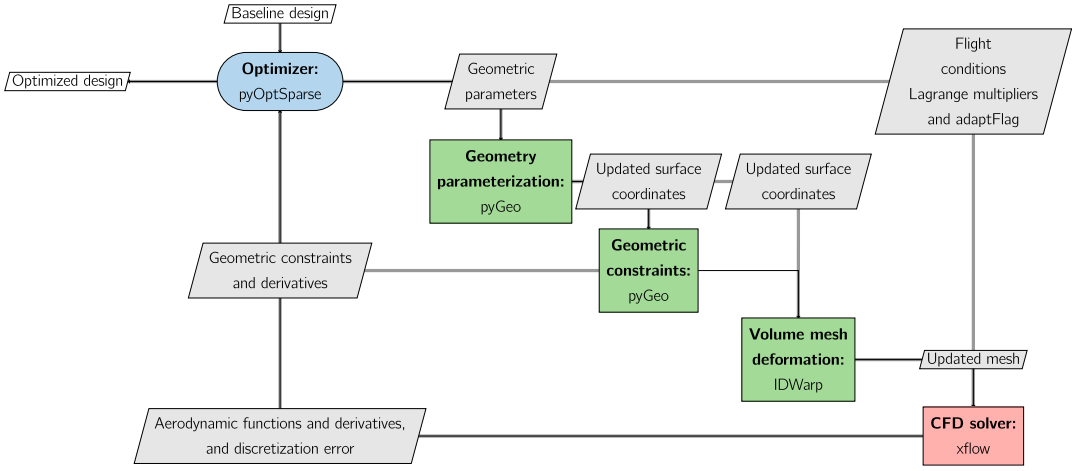


Fig. 1 Modified MACH-aero XDSM diagram.

VI. Drag Minimization of a Subsonic Turbulent Airfoil

For our first test problem, we minimize drag starting with a NACA 0012 airfoil at a Mach number of $M = 0.5$ and Reynolds number of $Re = 10^6$. We impose a lift constraint of $c_\ell = 1.0$ and constrain the area of the airfoil to be greater than or equal to that of the baseline NACA 0012. We prevent the thickness from being less than 30% of the original thickness at any location along the chord. We also impose FFD constraints at the leading and trailing edges to ensure that 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.

The optimality and feasibility tolerances in SNOPT are both set to 10^{-6} for all optimizations in this section. We run the optimization on the family of O topology meshes with ADflow summarized in Table 2. Once we generated the family of meshes, we took the $L1$ and $L2$ meshes and added geometric nodes on the wall to make each cell cubic, $q = 3$, and used linear-elasticity analogy to curve the mesh. For optimizations run with xflow, we use the $L2$ mesh and run at a fixed resolution with different solution approximation orders of $p = 3$, $p = 2$, and $p = 1$. 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 retrim the c_ℓ to a tolerance of 10^{-8} in both xflow and ADflow on finer solution spaces to obtain the "true" values for the drag. In ADflow, we

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	— —

Table 2 NACA 0012 O-mesh family

Name	Elements on wall	Total elements
$L00$	480	29,1840
$L0$	240	72,960
$L1$	120	18,240
$L2$	60	4,560

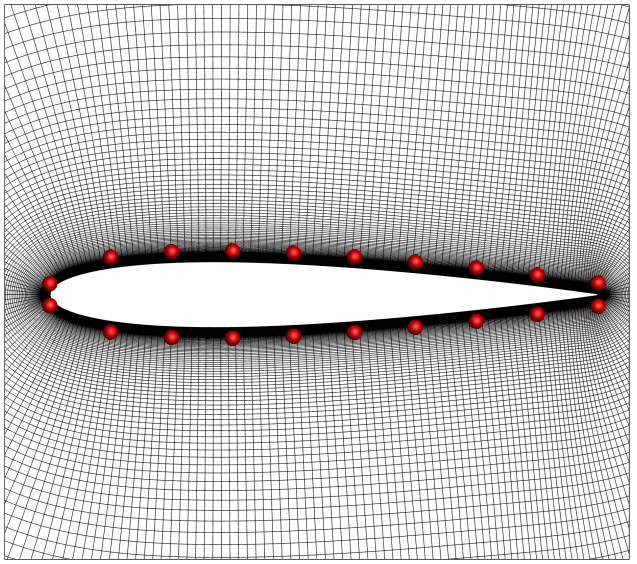


Fig. 2 $L0$ mesh for the NACA 0012 airfoil with 20 FFD points used for optimization.

use the $L00$ mesh, which has 291,840 degrees of freedom. 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.

The FFD points used are shown with the $L0$ mesh in Fig. 2.

Timings for this case were done on an AMD Ryzen Threadripper 3960x using 18 cores.

A. Discontinuous Galerkin NACA 0012 Optimization

For the DG results at a fixed resolution, we use the $L2$ mesh curved with $q = 3$ cubic quadrilateral elements and solution orders $p = 1$, $p = 2$, and $p = 3$. For the p-adaptive case, we set a tolerance of the Lagrangian error to be 0.5 drag counts ($\delta\mathcal{L} \leq 5 \times 10^{-5}$). In xflow we set the absolute convergence criterion of the primal residual vector to $\|\boldsymbol{R}\|_1 < 1 \times 10^{-8}$ and the relative convergence of the adjoint residual, $\boldsymbol{R}^{\Psi}$, to $\|\boldsymbol{R}^{\Psi}/\boldsymbol{R}_0^{\Psi}\|_2 < 1 \times 10^{-12}$. We start on the $L2$ 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 of the elements with the lowest 2.5% of error. We limit the order of the elements to be between $p = 1$ and $p = 3$. Additionally, we compute the error on the Lagrangian every five major iterations. The optimization reached a final error estimate of 0.42 drag counts, $\delta\mathcal{L} = 4.2 \times 10^{-5}$, which is below the set tolerance. The results are summarized in Table 3. The column for optimized c_d is the final objective function reported by SNOPT using each given solution order, while the columns for "true" c_d are from reanalyzing the optimized designs using a $p = 3$ solution order on the $L1$ curved with $q = 3$ elements mesh in xflow.

The optimized airfoil geometries and "true" c_p distributions from xflow are shown in Fig. 3. The $p = 2$ results are almost identical to the $p = 3$ ones; this is also evident when looking at the "true" c_d values. The minimum c_p on the airfoil optimized with $p = 1$ does not reach 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

Table 3 NACA 0012 optimization with xflow

Solution order	DoFs	Time, h	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
p-adaptive	30,758	1.8	137.50	136.90	137.61

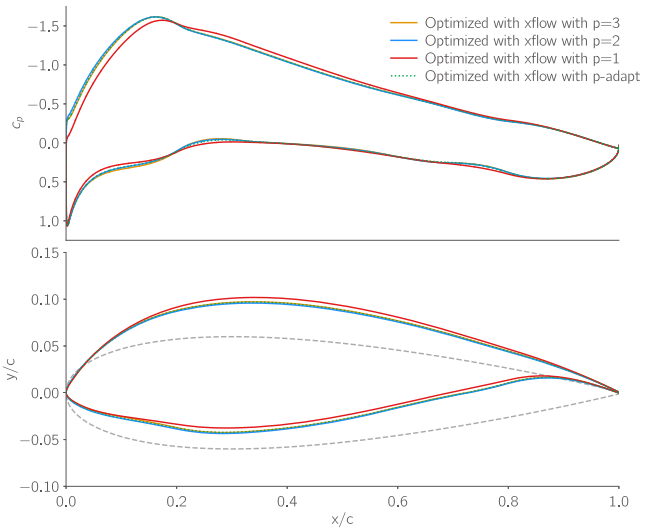


Fig. 3 NACA 0012 optimized airfoils using xflow; c_p plots generated from "true" xflow cross analysis.

from 20–60% chord, moving the lift distribution from the front to the back. The adaptive case final drag, c_p distribution, and shape are almost identical to the airfoils optimized at fixed $p = 2$ and $p = 3$. When looking at the total computational cost, however, we obtain this result with fewer degrees of freedom than the $p = 2$ case and less computational time. This shows that our adaptation strategy reduces computational cost without sacrificing the accuracy of the solution or optimization.

Figure 4 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 iterations show a noticeable drop in c_d due to the finer space removing numerical diffusion.

Additionally, the adaptive case converges in more than double the major iterations of the fixed-resolution cases. This is due to the changing design space the optimizer experiences when adapting. However, because each iteration is done with fewer degrees of freedom, the overall optimization is faster. When looking at the optimality and c_d converge between iterations 15 and 20, the p-adaptive and the fixed $p = 2$ cases have very similar optimality values and c_d values. This indicates that they follow a similar optimization trajectory before the adaptation occurs and changes the path.

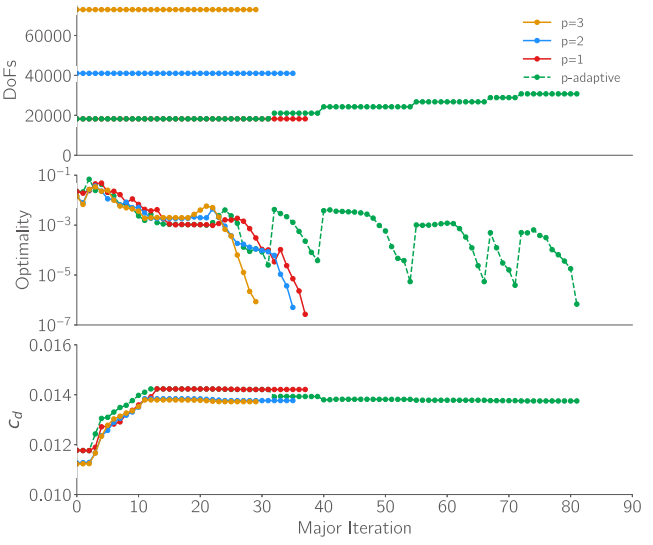


Fig. 4 Optimization history for NACA 0012 optimization.

B. Discontinuous Galerkin Compared to Finite Volume NACA 0012 Optimization

We compare the DG results to optimizations run in ADflow on the $L2$, $L1$, $L0$, and $L00$ meshes. In ADflow, we set the relative convergence criterion of both the primal and adjoint residual vector to 1×10^{-12}. Table 4 summarizes optimized final drag and optimization time.

The "true" final drag values of each optimization are all within one drag count ($c_d = 10^{-4}$) when reanalyzing with both xflow and ADflow, even though the geometries are different. This indicates a relatively flat design space where significant geometric changes have little impact on the objective function. The final geometries and c_p distributions generated from the "true" xflow cross-analysis are shown in Fig. 5. 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' leading edge becomes thinner, and the area moves toward the back. However, even the $L0$ mesh with 72,960 elements still does not converge to the shape optimized with the $L00$ mesh, indicating that more elements are needed to reach the true minimum.

When comparing these results to the DG results, we see that per degree of freedom, DG outperforms finite volume. A comparison of the DG $p = 3$ and adaptive airfoils and the finite volume $L0$ and $L00$ airfoils are shown in Fig. 6. The DG $p = 3$ case has the same number of degrees of freedom as the ADflow case on the $L0$ mesh but has a lower c_d. The $L00$ mesh performs similarly to the DG airfoils, but there are still minor shape changes around the leading edge and in the c_p distribution. The adapted case is still slower than the ADflow case on the $L0$ mesh, taking more than double the time, but it performs better when looking at the final drag values. Finite volume is faster per degree of freedom, but the optimized airfoil is worse than the optimized airfoil using DG.

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

Table 4 NACA 0012 optimization with ADflow summary

Mesh	DoFs, final	Time, h	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

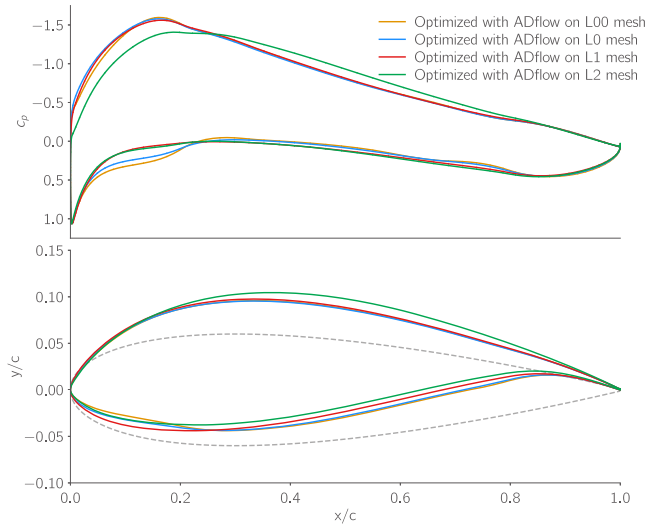


Fig. 5 NACA 0012 optimized airfoils using ADflow. The c_p plots are from "true" xflow cross analysis.

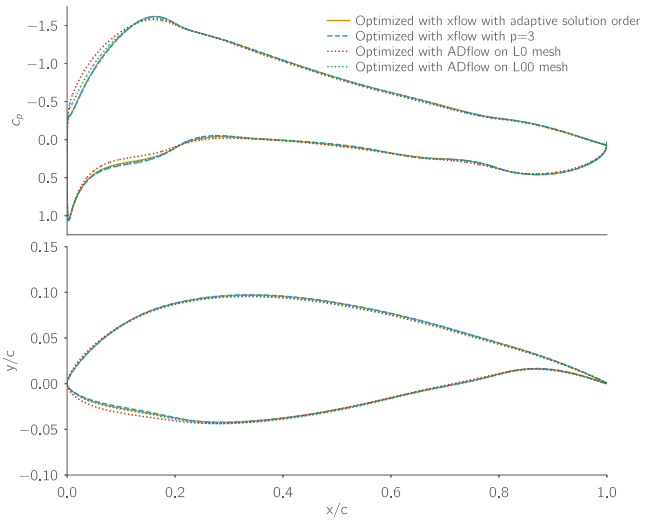


Fig. 6 Comparison of airfoils optimized with DG p-adaptation, DG fixed fidelity, and FVM. The c_p distributions are from "true" xflow.

VII. Drag Minimization for a Transonic Turbulent Airfoil

The second test case is a viscous transonic airfoil optimization of the RAE 2822 airfoil, defined by the second test case of the Aerodynamic Design Optimization Discussion Group (ADODG) [1]. This test case is a drag minimization of the RAE 2822 airfoil subject to 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 chordwise direction along the upper and lower surfaces. Additionally, we add thickness constraints such that the thickness at any chordwise location is greater than 10% of the original thickness at that location. This is to ensure that the airfoil top and bottom surfaces never cross, although no thickness constraint was active in any optimization. Finally, there are constraints on the FFD pairs at the leading and trailing edges to prevent geometric rotation, and an angle of attack design variable α is used to meet the lift constraint. The problem is detailed in Table 5.

The $L0$ mesh used in ADflow is shown in Fig. 7, along with the 20 FFD points used for all optimizations.

Optimizations using xflow use a C-topology mesh to conform to the sharp trailing edge of the geometry. The mesh uses cubic ($q = 3$) elements, with a distribution of 150 elements on the airfoil, 40 elements in the wake, and 24 elements in the wall-normal direction. However, in ADflow, we use a family of O-topology meshes to be consistent with previous studies of the RAE 2822 airfoil [39]. We

Table 5 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	0.824	0.824
	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	— —

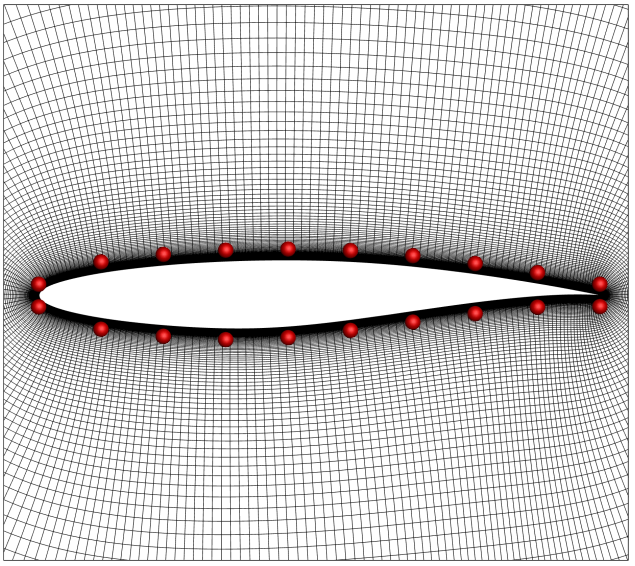


Fig. 7 $L0$ mesh for RAE 2822 airfoil with 20 FFD points used for optimization.

Table 6 RAE 2822 O-topology mesh family

Name	Elements on wall	Total elements
$L00$	480	291,840
$L0$	240	72,960
$L1$	120	18,240
$L2$	60	4,560

generated the family of O-topology meshes listed in Table 6. The finest mesh ($L00$) has 291,840 elements, and the coarsest ($L2$) has 4560 elements.

Timings for this case were done on an AMD Ryzen Threadripper 3960x using 18 cores.

A. RAE 2822 with xflow

We tested a fixed-fidelity optimization at solution orders of $p = 3$, $p = 2$, and $p = 1$, and an adaptive fidelity optimization with a final tolerance on the Lagrangian discretization error of 5×10^{-5}, starting with all elements $p = 1$. We used a fixed fraction p-adaptation approach, where 12.5% of elements with the highest error were refined, and 2.5% of elements with the lowest error were coarsened. The approximation order is limited to be between 1 and 3. In xflow, we set the absolute convergence criterion of the primal residual vector to $\|\boldsymbol{R}\|_1 < 1 \times 10^{-8}$ and the relative convergence of the adjoint residual, $\boldsymbol{R}^{\Psi}$, to $\|\boldsymbol{R}^{\Psi}/\boldsymbol{R}_0^{\Psi}\|_2 < 1 \times 10^{-12}$. Because this problem has three CFD outputs, three fine-space adjoint solutions are required for the Lagrangian error estimate. Therefore, we increase the number of iterations between error estimate computations from 5 to 10. The optimization reached a final error estimate of 0.48 drag counts, $\delta\mathcal{L} = 4.8 \times 10^{-5}$, which is below the set tolerance. To evaluate the "true" c_d in xflow, we uniformly refine the C-mesh used for the optimization to obtain a new mesh with 18,240 elements and run with a uniform $p = 3$ solution order totaling 291,840 degrees of freedom. In ADflow, we use the $L00$ mesh. Results for the DG optimizations are listed in Table 7.

The geometries and c_p plots, generated in xflow at $p = 3$, are shown in Fig. 8. The airfoil optimized at $p = 1$ differs significantly from all other optimized airfoils. This is due to significant discretization errors that the optimizer exploits to reduce drag. When this design is reanalyzed in xflow, the drag is much lower, and the flowfield is significantly different, as shown in Fig. 9. When analyzing this design at $p = 3$, a double shock appears on the upper surface, even though the airfoil is shock-free at $p = 1$. While weak, the

Table 7 RAE 2822 optimization with xflow

Solution order	DoFs	Time, h	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	104.10	105.53
2	41,040	7.56	106.69	105.26	106.35
1	18,240	1.04	147.48	111.62	115.12
p-adaptive	41,041	20.75	104.58	104.14	105.57

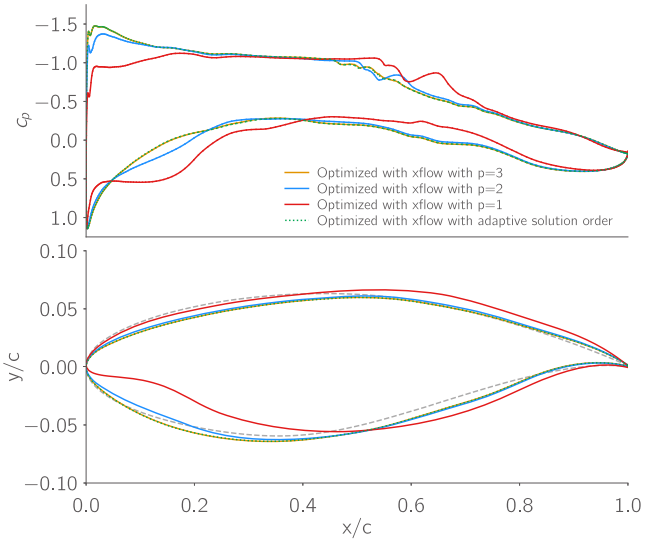


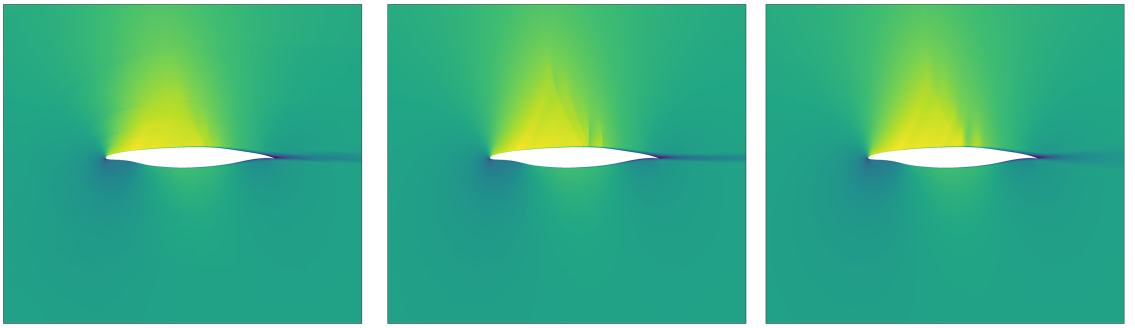
Fig. 8 RAE 2822 optimized airfoil geometries from fixed fidelity and p-adapted DG optimizations; c_p generated from the "true" xflow result.

double shock that reappears on the finer mesh $p = 3$ solution adds wave drag. However, the true drag is much lower than predicted because of the lower numerical dissipation drag.

In the case of fixed-fidelity DG, $p = 3$ achieves the lowest drag-optimized design. Due to a weak shock reappearing, the airfoil optimized at $p = 2$ has a drag one count higher than the airfoil optimized at $p = 3$.

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 .04 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 nearly identical, while the $p = 2$ case has minor differences. These minor shape differences lead to the reappearance of the shock on the upper surface when analyzed with $p = 3$. Coincidentally, the adapted case ends with 1-degree-of-freedom difference from the $p = 2$ case. Even with the same number of degrees of freedom, the total optimization time is 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 jumps off a spurious optimum: a shock appears that must be smoothed out, taking more optimization iterations. This leads to noise in the optimization problem, which makes the optimization take more iterations, which is shown in Fig. 10.

Shocks, in particular, are sensitive to discretization errors and thus disappear and reappear on different meshes. This case shows that the importance of ensuring the discretization error is low during transonic optimization, where the optimizer uses discretization error to reduce wave drag instead of geometric features. While the adaptation case takes longer than the $p = 2$ optimization, there is no reappearance of the shock and, therefore, a lower true drag value using the same number of degrees of freedom. Much of this increased time is due to the optimizer taking more iterations rather than the flow solutions taking longer.



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

Fig. 9 Mach contours (0–1.3) on the $p = 1$ xflow-optimized airfoil, analyzed using different solvers.

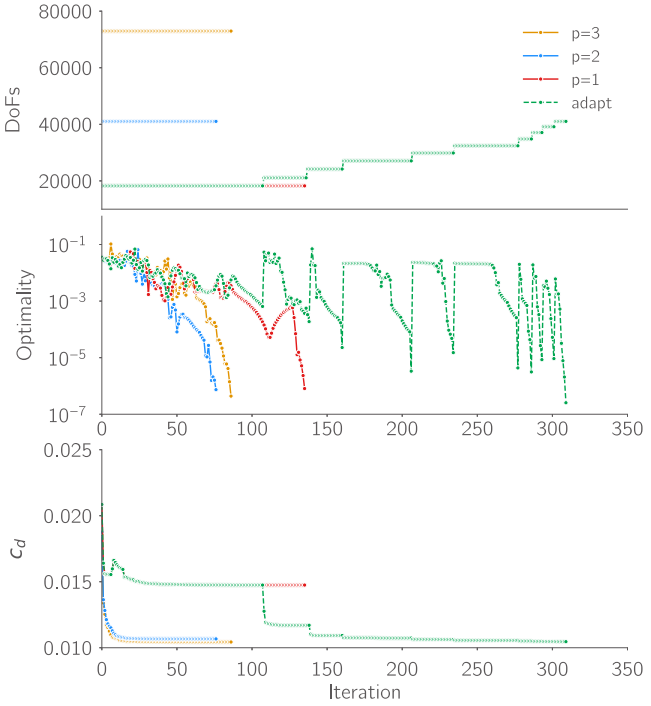


Fig. 10 Optimization history for RAE 2822 optimization.

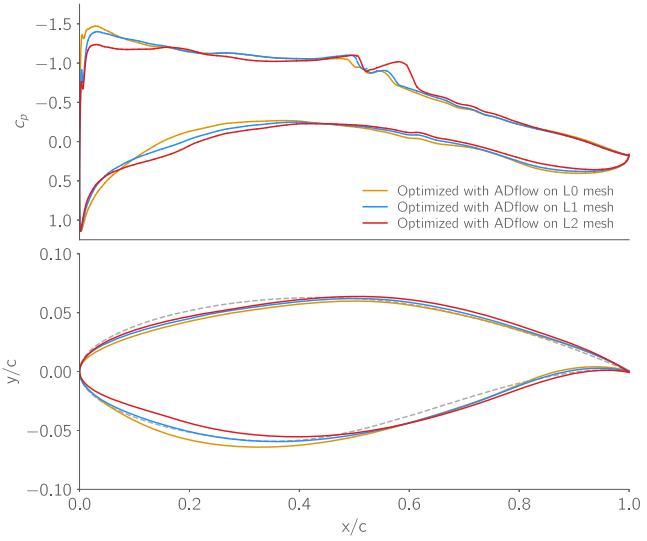


Fig. 11 RAE 2822 optimized airfoil geometries using ADflow. The c_p distributions are from the xflow "true" solution.

B. RAE 2822 with ADflow

The optimization in ADflow was run on the $L0$, $L1$, and $L2$ meshes, and the final drag results are summarized below in Table 8. In ADflow we set the relative convergence criterion of both the primal and adjoint residual vector to 1×10^{-12}. The "True" values for the drag come from taking each design and reanalyzing it in xflow at $p = 3$ on the C-topology mesh described above and again in ADflow on the $L00$ mesh.

The "true" drag coefficient for each design goes down in both xflow and ADflow as the meshes are refined. This is because the optimizer exploits discretization errors to smooth out the shock rather than using shape changes. The shocks come back when analyzed in both xflow and ADflow on the $L00$ mesh. This is shown in Fig. 11, where the $L2$ mesh design has the largest pressure decrease and the $L0$ mesh design has a very weak shock on the upper surface.

Table 8 RAE 2822 optimization with ADflow

Mesh	DoFs	Time, h	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	104.51	105.24
$L1$	18,240	0.078	111.00	106.00	106.86
$L2$	4,560	0.025	134.47	111.43	112.21

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

When comparing to the designs from the optimizations using xflow, 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 performs even better using fewer degrees of freedom, but when time is a constraint, ADflow is much faster. This is because ADflow takes advantage of structured meshes for more efficient memory access, and its code is developed with optimization in mind. On the other hand, xflow is an unstructured code developed for research purposes, prioritizing modularity and ease of development. Running xflow on unstructured triangular meshes optimized using a metric field generally leads to significant speedups compared to running on structured meshes, even those that are p-adapted [40]. Finally, ADflow has fully analytic derivatives, while xflow does not—some of its partial derivatives are computed using finite differences, which take longer to compute and are less accurate. The less accurate derivatives cause the optimizer to take more iterations to find the optimum. Thus, a direct time comparison favors ADflow. Although ADflow is faster, the adaptive DG in xflow finds a better-performing optimum than the finite volume cases with fewer degrees of freedom than using fixed-fidelity DG.

VIII. Conclusions

When performing ASO, the CFD solver significantly impacts the overall computational cost and accuracy of the optimization. ASO may

require hundreds of primal and adjoint CFD solutions. Additionally, discretization error in CFD can lead to the optimizer finding spurious optima, for example, by prioritizing lowering numerical drag. These factors motivate the need for an efficient and accurate CFD solution capability. Adaptation solves both of these problems by adding and removing degrees of freedom in a targeted manner that increases the accuracy of the solution.

This work presented a monolithic approach to controlling discretization error during optimization. Our approach uses a single optimization rather than sequential optimizations at fixed cost or error to reach the objective function's target error tolerance. This approach only relies on one user input, the error of the objective at the end of the optimization, instead of relying on multiple user inputs for the target error or cost for each sequential optimization. The algorithm ensures that the relative error convergence tolerance tracks the relative convergence of the optimality convergence condition. Taking this approach, we do not over-refine designs far from the optimum or overoptimize on coarse solution spaces, reducing computational cost and reaching the true optimum.

Two test cases show that high-order discretization converges to the true optimal design with fewer degrees of freedom than traditional second-order methods. Even fixed-fidelity ($p = 2$) shows the influence of discretization error on the final design. When combined with adaptation, high-order discretizations perform even better using far fewer degrees of freedom. The adaptation strategy reaches the same designs as the fixed $p = 3$ optimization, using the same number or even fewer degrees of freedom as the fixed $p = 2$ optimization cases.

While this work only focused on p-adaptation, to more easily work with the current MACH-Aero framework, using mesh adaptation and optimization (h-refinement) or combinations of mesh adaptation and order adaptation (hp-refinement) 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 acknowledges the support received at various stages of the research from the Department of Defense through the National Defense Science and Engineering Graduate Fellowship Program and the Rackham Graduate School at the University of Michigan through the Rackham Merit Fellowship. The first author received partial support from the University of Michigan Institute for Computational Discovery and Engineering Fellowship. The authors are also grateful for Ella Wu's insightful discussion throughout the development of the work.

References

[1] Martins, J. R. R. A., "Aerodynamic Design Optimization: Challenges and Perspectives," *Computers & Fluids*, Vol. 239, 2022, Paper 105391. https://doi.org/10.1016/j.compfluid.2022.105391

[2] Wang, Z., Fidkowski, K., Abgrall, R., Bassi, F., Caraeni, D., Cary, A., Deconinck, H., Hartmann, R., Hillewaert, K., et al., "High-Order CFD Methods: Current Status and Perspective," *International Journal for Numerical Methods in Fluids*, Vol. 72, No. 8, 2013, pp. 811–845. https://doi.org/10.1002/fld.3767

[3] 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. https://doi.org/10.2514/1.J050073

[4] Nemec, M., and Aftosmis, M., "Output Error Estimates and Mesh Refinement in Aerodynamic Shape Optimization," *51st AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, AIAA Paper 2013-0865, 2013.

[5] Anderson, G. R., Nemec, M., and Aftosmis, M. J., "Aerodynamic Shape Optimization Benchmarks with Error Control and Automatic Parameterization," *53rd AIAA Aerospace Sciences Meeting*, AIAA Paper 2015-1719, 2015.

[6] 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. https://doi.org/10.1016/j.jcp.2013.12.050

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

[8] Chen, G., and Fidkowski, K. J., "Variable-Fidelity Multipoint Aerodynamic Shape Optimization with Output-Based Adapted Meshes," *Aerospace Science and Technology*, Vol. 105, 2020, Paper 106004. https://doi.org/10.1016/j.ast.2020.106004

[9] Li, D., and Hartmann, R., "Adjoint-Based Airfoil Optimization with Discretization Error Control," *International Journal for Numerical Methods in Fluids*, Vol. 77, No. 1, 2015, pp. 1–17. https://doi.org/10.1002/fld.3971

[10] Lu, J., "An a Posteriori Error Control Framework for Adaptive Precision Optimization Using Discontinuous Galerkin Finite Element Method," Ph.D. Thesis, Massachusetts Inst. of Technology, Cambridge, MA, 2005.

[11] 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. https://doi.org/10.1007/s11075-017-0409-7

[12] 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, No. 3, 2022, pp. 1643–1676. https://doi.org/10.1007/s11081-021-09681-5

[13] 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, No. 4, 2022, pp. 131–151. https://doi.org/10.1007/s00158-022-03204-1

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

[15] Ceze, M. A., and Fidkowski, K. J., "An Anisotropic hp-Adaptation Framework for Functional Prediction," *AIAA Journal*, Vol. 51, No. 2, 2013, pp. 492–509. https://doi.org/10.2514/1.J051845

[16] Ceze, M. A., and Fidkowski, K. J., "High-Order Output-Based Adaptive Simulations of Turbulent Flow in Two Dimensions," *AIAA Journal*, Vol. 54, No. 9, 2016, pp. 2,611–2,625. https://doi.org/10.2514/1.J054517

[17] Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 43, No. 2, 1981, pp. 357–372. https://doi.org/10.1016/0021-9991(81)90128-5

[18] 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, Nos. 1–2, 2002, pp. 197–207. https://doi.org/10.1002/fld.338

[19] 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*, AIAA Paper 2013-2686, 2013. https://doi.org/10.2514/6.2013-2686

[20] 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. https://doi.org/10.1137/0907058

[21] 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. https://doi.org/10.1016/j.jcp.2005.01.005

[22] 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. https://doi.org/10.1137/070692108

[23] 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. https://doi.org/10.2514/1.I010796

[24] 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, Paper 100542. https://doi.org/10.1016/j.paerosci.2019.05.002

[25] 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,

Paper 108741.
https://doi.org/10.1016/j.jcp.2019.06.018
[26] Sederberg, T. W., and Parry, S. R., "Free-Form Deformation of Solid Geometric Models," *SIGGRAPH Computer Graphics*, Vol. 20, No. 4, 1986, pp. 151–160.
https://doi.org/10.1145/15886.15903
[27] Hajdik, H. M., Yildirim, A., Wu, N., Brelje, B. J., Seraj, S., Mangano, M., Anibal, J. L., Jonsson, E., Adler, E. J., Mader, C. A., et al., "PyGeo: A Geometry Package for Multidisciplinary Design Optimization," *Journal of Open Source Software*, Vol. 8, No. 87, 2023, Paper 5319.
https://doi.org/10.21105/joss.05319
[28] 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.
https://doi.org/10.2514/1.J059491
[29] 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.
https://doi.org/10.1016/j.jcp.2011.09.021
[30] 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, Paper 2564.
https://doi.org/10.21105/joss.02564
[31] 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.
https://doi.org/10.1137/S0036144504446096
[32] Martins, J. R. R. A., and Ning, A., *Engineering Design Optimization*, Cambridge Univ. Press, Cambridge, England, U.K., 2022, https://mdobook.github.io.
[33] 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.
https://doi.org/10.1145/838250.838251
[34] Griewank, A., *Evaluating Derivatives*, SIAM, Philadelphia, 2000.
[35] Jameson, A., "Aerodynamic Design via Control Theory," *Journal of Scientific Computing*, Vol. 3, No. 3, 1988, pp. 233–260.
https://doi.org/10.1007/BF01061285
[36] 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 Univ. Press, Cambridge, England, U.K., 2001, pp. 1–102.
[37] Park, M. A., "Adjoint-Based, Three-Dimensional Error Prediction and Grid Adaptation," *32nd AIAA Fluid Dynamics Conference and Exhibit*, AIAA Paper 2002-3286, 2002.
https://doi.org/10.2514/6.2002-3286
[38] 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.
https://doi.org/10.1007/s00158-012-0763-y
[39] 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.
https://doi.org/10.1016/j.ast.2019.01.051
[40] Fidkowski, K. J., "A Local Sampling Approach to Anisotropic Metric-Based Mesh Optimization," *AIAA SciTech Forum*, AIAA Paper 2016-0835, 2016.
https://doi.org/10.2514/6.2016-0835

V. Sankaran
Associate Editor