DAFoam: An Open-Source Adjoint Framework for Multidisciplinary Design Optimization with OpenFOAM

Ping He, Charles A. Mader, Joaquim R. R. A. Martins, and Kevin J. Maki
University of Michigan, Ann Arbor, MI, 48109

Abstract

The adjoint method is an efficient approach for computing derivatives that allow gradient-based optimization to handle systems parameterized with a large number of design variables. Despite this advantage, implementing the adjoint method for a partial-differential-equation-based primal solver is a time-consuming task. To lower the barrier for adjoint implementations, we propose DAFoam[1]: an object-oriented framework to rapidly implement the discrete adjoint method for any steady-state OpenFOAM primal solver by adding or modifying only a few hundred lines of source code. In this paper, we introduce the DAFoam framework and illustrate the proposed object-oriented adjoint development process. Using this strategy, we implement the adjoint method for eight primal solvers, five turbulence models, and one radiation model in OpenFOAM. We achieve excellent adjoint speed and scalability, with up to 10 million cells and 1536 CPU cores, and an average error in the adjoint derivatives of less than 0.1%. Finally, we integrate the implemented adjoint solvers and models into a gradient-based optimization framework and showcase four distinct design optimizations: multipoint aerodynamic optimization of a low-speed UAV wing, aerodynamic optimization of a transonic aircraft configuration, aerothermal optimization of a turbine internal cooling passage, and aerostructural optimization of a compressor rotor. DAFoam is available under an open-source license and is a powerful tool for the high-fidelity multidisciplinary design optimization of engineering systems such as aircraft, ground vehicles, marine vessels, and turbomachinery.

Keywords
Multidisciplinary Design Optimization; Open Source; CFD; Adjoint; OpenFOAM

[1]https://dafoam.rtfd.io

1 Introduction

Aerospace engineering designs often require using a large number of design variables to parameterize complex design surfaces, such as aircraft wings. Changing these variables by hand is time-consuming and is not likely to achieve the best possible design. Gradient-based optimization is a powerful approach to solve the above problem, because it automatically finds the set of design variables that maximizes the performance. To efficiently compute the derivatives, we can use the adjoint method, whose computational cost is independent of the number of design variables. The combination of adjoint method and gradient-based optimization thus enables the solution of complex design problems.

The adjoint method was first introduced in fluid mechanics by Pironneau [1] and then extended for aerodynamic shape optimization by Jameson [2]. Since then, the adjoint method has been widely used in gradient-based optimization for applications involving aerodynamics [3–10], hydrodynamics [11, 12], heat transfer [13, 14], and structures [15, 16].

Many engineering systems are composed of multiple disciplines, requiring multidisciplinary design optimization (MDO) techniques [17]. The adjoint method has been generalized to MDO problems [18, 19] and implemented in the OpenMDAO framework [20]. Coupled-adjoint implementations have been used to solve aerostructural [21–24], hydrostructural [25], aerothermal [26], aeropropulsive [27, 28], and aeroelastic [29, 30] MDO problems.

There are two different approaches to formulate the adjoint equations for a partial differential equation (PDE) based primal solver: continuous and discrete [31–33]. The continuous approach derives the adjoint formulation from the original governing equations and then discretizes them for numerical solution. This approach was used in the early work including Jameson [2], Anderson and Venkatakrishnan [34], as well as the initial adjoint implementations for OpenFOAM [35, 36] and SU2 [37]. The continuous adjoint is faster and requires less memory than the discrete adjoint. However, the accuracy of the continuous adjoint method suffers on coarse meshes [38] and it is challenging to implement for complex terms, such as those encountered in turbulence models [31–33].

On the other hand, the discrete approach starts directly from the discretized governing equations for the adjoint formulation. Therefore, the adjoint derivatives are consistent with the primal flow solutions, independent of the mesh density; a favorable feature that makes the optimization process more robust. Given this advantage, we opt to use the discrete adjoint approach in this paper.

There are two main tasks when implementing the discrete adjoint method for a PDE-based primal solver, which we elaborate on in Sec. 2.1: 1. Compute the partial derivatives or the matrix-vector products; 2. Solve the adjoint linear equation. In the past few decades, researchers have used various options for the above two tasks in discrete adjoint implementations [32, 33]. The partial derivatives have been computed using the analytic method, finite differences, the complex-step method [39], and algorithmic differentiation (AD) [40]. AD has also been used to efficiently compute the matrix-vector products in a matrix-free manner [33].

To solve the adjoint equation, both fixed-point iteration and Krylov methods have been proposed. The combination of these options has been used in a number of discrete adjoint solvers, such as ADflow [33, 41], Cart3D [42], FUN3D [3], HYDRA [43], Jetstream [44–46], NSU3D [5, 47], piggy- and reverseAcc-SimpleFoam [48], STAMPS [49], SU2 [37, 50], and TAU-Code [51].

Despite the progress cited above, developing a discrete adjoint solver remains a time-consuming task. This is primarily because of the strong connection between the primal and adjoint solvers. As a consequence, developers need to have detailed knowledge of low-level implementations for both primal and adjoint solvers. In addition, to ensure adjoint consistency, modification and extension in the primal solver require corresponding changes in the adjoint solver. The above two factors cause the adjoint solver to have the same amount of development and maintenance effort as the primal solver does; a discrete adjoint solver typically contains thousands of lines of source code, and its development may take years. Although the above-mentioned adjoint solvers provide a certain amount of flexibility for extension, such as adding new terms or boundary conditions, they do not offer rapid adjoint development options for a new set of PDEs. Having the capability to rapidly implement the adjoint method for a PDE-based primal solver from scratch is useful in practice because it allows us to handle a wide range of engineering design problems involving multiple disciplines (e.g., aerodynamics, structures, and heat transfer), configurations (e.g., wings, wing-body-tail, and airframe-propulsion integration), and flow conditions (e.g., incompressible, subsonic, and transonic).

One way to address the large adjoint development effort is to use reverse-mode AD to differentiate the entire primal solver (full-code AD) [48, 52, 53]. The full-code AD approach treats the primal solver as a black box and thus requires minimal development effort. However, the intermediate variables that are used in the primal nonlinear solution process need to be stored in memory, which is not feasible for large three-dimensional problems. Although advanced techniques (e.g., checkpointing and local pre-accumulation) can be used to trade speed for memory [48], the size of problems the full-code AD can handle is still limited [33].

The objective of this paper is to lower the barrier of adjoint development while maintaining the efficiency of adjoint computation. To this end, we propose DAFoam[2]: an open-source, object-oriented framework to rapidly develop discrete adjoint solvers with OpenFOAM [54]. OpenFOAM is an open-source, multiphysics package that contains more than 80 PDE-based primal solvers, involving a wide range of disciplines such as aerodynamics, hydrodynamics, structures, heat transfer, combustion, and multiphase flow. Borrowing the idea of object-oriented primal solver development in OpenFOAM, DAFoam provides a high-level interface that allows us to implement the discrete adjoint method for existing or new steady-state OpenFOAM primal solvers by adding or modifying only a few hundred lines of source code. The central recipe of the proposed framework is to use a generalized framework for partial derivative computation and adjoint equation solution, and then provide an interface that allows developers to define solver-specific implementations, such as the residual functions and connectivity

[2]https://github.com/mdolab/dafoam

information.

In this paper, we introduce the overall structure of DAFoam and illustrate the object-oriented adjoint development process for the Navier–Stokes (NS) equations. Using the proposed recipe, we implement the adjoint method for eight primal solvers, five turbulence models, and one radiation model. We evaluate the performance of the adjoint implementations in terms of speed, scalability, memory usage, and accuracy. Moreover, we integrate the adjoint solvers into a gradient-based optimization framework and showcase four distinct design optimizations using DAFoam: multipoint aerodynamic optimization of a low-speed UAV wing, aerodynamic shape optimization of a transonic aircraft configuration, aerothermal optimization of a turbine internal cooling passage, and aerostructural optimization of a compressor rotor. The optimization setup for these cases, including the meshes, flow and optimization configurations, and DAFoam run scripts, are publicly available [55]. In sum, the main contribution of this work is the development of an open-source, object-oriented adjoint framework DAFoam that allows rapid adjoint solver implementations for a wide range of PDEs.

The rest of the paper is organized as follows. In Section 2, we introduce the DAFoam framework and detail the object-oriented adjoint development. The design optimization results are presented and discussed in Section 3 and we summarize our findings in Section 4.

2 Methodology

The central recipe of DAFoam's object-oriented adjoint framework is to divide the implementation into solver-agnostic and solver-specific parts. In this section, we first derive the discrete adjoint equations. Then, we elaborate on the solver-agnostic adjoint development, followed by the object-oriented interface that allows developers to specify the solver-specific implementations. To better illustrate the object-oriented adjoint development process, we use the adjoint source code for the incompressible NS equations as an example. Finally, we summarize the implemented adjoint solvers and models, and evaluate their performance in terms of speed, scalability, memory usage, and accuracy.

2.1 Adjoint equations

As mentioned above, we use the adjoint method to efficiently compute the total derivatives $\mathrm{d}f/\,\mathrm{d}\boldsymbol{x}$, where f is the objective or constraint function (e.g., drag, lift, or torque) and $\boldsymbol{x}$ is the vector of design variables (e.g., positions of control points that morph the design surface). In the discrete approach, we assume that a discretized form of governing equations is available through the primal solver, and that the design variable vector $\boldsymbol{x} \in \mathbb{R}^{n_x}$ and the state variable vector $\boldsymbol{w} \in \mathbb{R}^{n_w}$ satisfy the discrete residual equations $\boldsymbol{R}(\boldsymbol{x}, \boldsymbol{w}) = 0$, where $\boldsymbol{R} \in \mathbb{R}^{n_w}$ is the residual vector.

The functions of interest are then functions of both the design variables and the state variables: $f = f(\boldsymbol{x}, \boldsymbol{w})$. In general, we have multiple functions of interest (the objective and multiple design constraints), but in the following derivations, we consider f to be a scalar without loss of generality. As we will see later, each additional function requires the solution of another adjoint system. To obtain the total derivative $\mathrm{d}f/\mathrm{d}\boldsymbol{x}$, we apply the chain rule as follows:

$$\underbrace{\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}}_{1\times n_x} = \underbrace{\frac{\partial f}{\partial \boldsymbol{x}}}_{1\times n_x} + \underbrace{\frac{\partial f}{\partial \boldsymbol{w}}}_{1\times n_w} \underbrace{\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}\boldsymbol{x}}}_{n_w\times n_x}, \tag{1}$$

where the partial derivatives $\partial f/\partial \boldsymbol{x}$ and $\partial f/\partial \boldsymbol{w}$ are relatively cheap to evaluate because they only involve explicit computations. The total derivative $\mathrm{d}\boldsymbol{w}/\mathrm{d}\boldsymbol{x}$ matrix, on the other hand, is expensive, because $\boldsymbol{w}$ is implicitly determined by the residual equations $\boldsymbol{R}(\boldsymbol{w},\boldsymbol{x}) = 0$.

To obtain $\mathrm{d}\boldsymbol{w}/\mathrm{d}\boldsymbol{x}$, we can apply the chain rule for $\boldsymbol{R}$. We then use the fact that the governing equations should always hold, independent of the values of design variables $\boldsymbol{x}$. Therefore, the total derivative $\mathrm{d}\boldsymbol{R}/\mathrm{d}\boldsymbol{x}$ must be zero:

$$\frac{\mathrm{d}\boldsymbol{R}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}\boldsymbol{x}} = 0 \implies \underbrace{\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}\boldsymbol{x}}}_{n_w\times n_x} = -\underbrace{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^{-1}}_{n_w\times n_w} \underbrace{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}}_{n_w\times n_x}. \tag{2}$$

Substituting $\mathrm{d}\boldsymbol{w}/\mathrm{d}\boldsymbol{x}$ from Eq. (2) into Eq. (1), we get

$$\underbrace{\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}}_{1\times n_x} = \underbrace{\frac{\partial f}{\partial \boldsymbol{x}}}_{1\times n_x} - \overbrace{\underbrace{\frac{\partial f}{\partial \boldsymbol{w}}}_{1\times n_w} \underbrace{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^{-1}}_{n_w\times n_w}}^{\boldsymbol{\psi}^T} \underbrace{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}}_{n_w\times n_x}. \tag{3}$$

Now we can transpose the state Jacobian matrix $\partial \boldsymbol{R}/\partial \boldsymbol{w}$ and solve with $[\partial f/\partial \boldsymbol{w}]^T$ as the right-hand side, which yields the adjoint equation,

$$\underbrace{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^T}_{n_w\times n_w} \underbrace{\boldsymbol{\psi}}_{n_w\times 1} = \underbrace{\frac{\partial f}{\partial \boldsymbol{w}}^T}_{n_w\times 1}, \tag{4}$$

where $\boldsymbol{\psi}$ is the adjoint vector. Once we have solved this equation, we can compute the total derivative by substituting the adjoint vector $\boldsymbol{\psi}$ into Eq. (3), yielding

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \frac{\partial f}{\partial \boldsymbol{x}} - \boldsymbol{\psi}^T\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}. \tag{5}$$

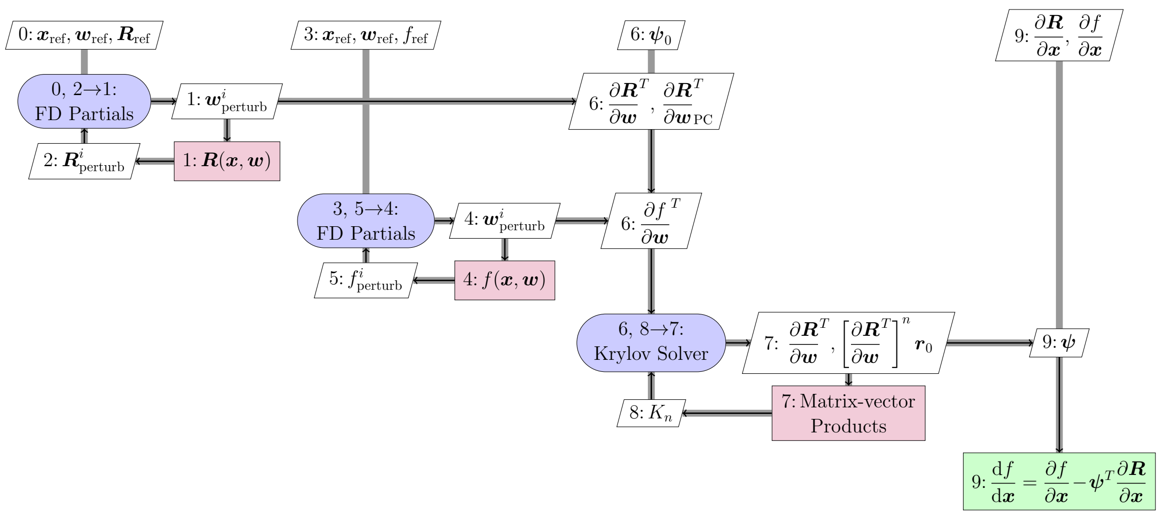
For each function of interest, we need to solve the adjoint equations only once, because the design variable is not explicitly present in Eq. (4). Therefore, its computational cost is independent of the number of design variables, but proportional to the number of functions of interest. This approach is known as the adjoint method and is advantageous for many aerospace engineering design problems where we have only a few functions of interest but may use several hundred design variables.

To summarize, a discrete adjoint implementation requires computing the partial derivatives and solving the adjoint equations and consists of four major steps:

1. Compute the partial derivatives $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$ and $[\partial f/\partial \boldsymbol{w}]^T$;
2. Solve the linear equation (4) for the adjoint vector $\boldsymbol{\psi}$;
3. Compute the partial derivatives $\partial \boldsymbol{R}/\partial \boldsymbol{x}$ and $\partial f/\partial \boldsymbol{x}$;
4. Use Eq. (5) to compute the total derivative $\mathrm{d}f/\mathrm{d}\boldsymbol{x}$.

The above four steps do not assume any specific form of the residual function $\boldsymbol{R}(\boldsymbol{w}, \boldsymbol{x})$; therefore, they are applicable for any set of discrete PDEs. In light of this observation, DAFoam implements the above four steps in a solver-agnostic manner, as detailed in Sec. 2.2. To account for solver-specific implementations, DAFoam provides high-level interfaces to specify the detailed residual function form, as elaborated on in Sec. 2.3

2.2 Solver-agnostic framework for partial derivative computation and adjoint equation solution



$0: \boldsymbol{x}_{\mathrm{ref}}, \boldsymbol{w}_{\mathrm{ref}}, \boldsymbol{R}_{\mathrm{ref}}$
$3: \boldsymbol{x}_{\mathrm{ref}}, \boldsymbol{w}_{\mathrm{ref}}, f_{\mathrm{ref}}$
$6: \boldsymbol{\psi}_0$
$9: \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}, \frac{\partial f}{\partial \boldsymbol{x}}$
0, 2→1: FD Partials
$1: \boldsymbol{w}^i_{\mathrm{perturb}}$
$6: \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^T, \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^T_{\mathrm{PC}}$
$2: \boldsymbol{R}^i_{\mathrm{perturb}}$
$1: \boldsymbol{R}(\boldsymbol{x}, \boldsymbol{w})$
3, 5→4: FD Partials
$4: \boldsymbol{w}^i_{\mathrm{perturb}}$
$6: \frac{\partial f}{\partial \boldsymbol{w}}^T$
$5: f^i_{\mathrm{perturb}}$
$4: f(\boldsymbol{x}, \boldsymbol{w})$
6, 8→7: Krylov Solver
$7: \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^T, \left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^T\right]^n \boldsymbol{r}_0$
$9: \boldsymbol{\psi}$
$8: K_n$
7: Matrix-vector Products
$9: \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \frac{\partial f}{\partial \boldsymbol{x}} - \boldsymbol{\psi}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}$

Figure 1: Process and data flow for the solver-agnostic FD Jacobian adjoint approach [33] in DAFoam.

DAFoam uses the FD Jacobian approach [33] to implement the discrete adjoint, i.e., the partial derivatives are computed using the coloring-accelerated finite-difference method and the adjoint equations are solved using a Krylov method. Figure 1 shows the process and data flow for the FD Jacobian adjoint approach. Here we use the extended design structure matrix (XDSM) representation developed by Lambe and Martins [56]. The diagonal nodes represent the modules and the off-diagonal nodes represent the data. The black lines represent the process flow in the adjoint, whereas the thick gray lines represent the data flow. The number in each node represents the execution order. The superscripts i and n are the ith color and nth iteration, respectively, and the subscripts "ref" and "perturb" are the reference and perturbed values, respectively.

As shown in Fig. 1, we use the finite-difference method to compute the partial derivatives $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$ and $[\partial f/\partial \boldsymbol{w}]^T$, accelerated by a graph coloring algorithm. We loop the processes 0−1−2−0 to compute the columns associated with the ith color in $\partial \boldsymbol{R}/\partial \boldsymbol{w}$ and then repeat the loop for all colors. Finally, we output the fully assembled $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$ to 6. Similarly, we compute $[\partial f/\partial \boldsymbol{w}]^T$ by looping the processes 3−4−5−3 and output to 6. The use of graph coloring is critical because naively computing $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$ and $[\partial f/\partial \boldsymbol{w}]^T$ using finite differences requires calling the objective and residual computation routines n_w times, one for each column in $\partial \boldsymbol{R}/\partial \boldsymbol{w}$ and $\partial f/\partial \boldsymbol{w}$. This becomes computationally prohibitive for three-dimensional problems because n_w can be more than ten million.

To reduce the computational cost, we use graph coloring [57] to exploit the sparsity of the Jacobians. To be more specific, we partition all the columns of a Jacobian matrix into different structurally orthogonal subgroups (colors), such that, in one structurally orthogonal subgroup, no two columns have a nonzero entry in a common row. With this treatment, we can simultaneously perturb multiple columns that have the same colors because no two columns (states) impact the same row (residual). We then compute their partial derivatives by calling the residual and objective computation routines only once.

Using coloring to accelerate Jacobian computation for adjoint solvers was proposed by Burdyshaw and Anderson [58], Nielsen and Kleb [59], and Lyu et al. [7], where the complex step [39] and algorithmic differentiation [40] methods were used to compute partial derivatives. Although these two methods provide accurate derivative computation, in this paper we opt to use the finite-difference method because it requires minimal modification to the primal codes, which ultimately facilitates the adjoint implementations. Moreover, its accuracy is well within the acceptable range for practical optimization problems (see the adjoint accuracy evaluation in Sec. 2.4.2).

The graph coloring for $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$ is challenging because $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$ is a $n_w \times n_w$ matrix and because OpenFOAM uses unstructured meshes, which results in an irregular sparsity pattern for $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$. In our previous work, we developed a heuristic graph coloring algorithm that runs on distributed memory systems in parallel [60]. This coloring algorithm is applicable for any mesh topology and allows us to compute $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$ by calling the residual routines between 1000 and 3000 times, independent of the mesh size and the number of CPU cores.

Using the coloring scheme to compute $[\partial f/\partial \boldsymbol{w}]^T$ requires special attention because f is typically computed based on the integration of discrete state variables over the design surface (e.g., drag and lift); therefore, $[\partial f/\partial \boldsymbol{w}]^T$ is a dense vector. To enable coloring for $[\partial f/\partial \boldsymbol{w}]^T$, we divide f into n_d discrete mesh faces on the design surface: $\partial f/\partial \boldsymbol{w} = \sum_{i=1}^{n_d} \partial f_i/\partial \boldsymbol{w}$, where f_i denotes the discrete f based on the ith mesh face on the design surface. With this treatment, we obtain a set of $\partial f_i/\partial \boldsymbol{w}$ vectors that are much sparser than $\partial f/\partial \boldsymbol{w}$. Next, we form a $n_d \times n_w$ matrix by using $\partial f_i/\partial \boldsymbol{w}$ as its ith row. We then use the coloring scheme to compute all the nonzero elements in this matrix. Finally, we sum all the rows of this matrix (i.e., $\partial f_i/\partial \boldsymbol{w}$) to obtain $\partial f/\partial \boldsymbol{w}$. He et al. [60] provides more details on this approach. The number of colors for $[\partial f/\partial \boldsymbol{w}]^T$

is at least one order of magnitude less than that for $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$.

After computing $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$ and $[\partial f/\partial \boldsymbol{w}]^T$, we use the PETSc library [61] to solve the adjoint equations (4) for the adjoint vector $\boldsymbol{\psi}$ (loop 6–7–8–6, output 9). We use the generalized minimal residual (GMRES) iterative linear equation solver. The GMRES method uses the Krylov subspace $K_i = \text{span}(r_0, Ar_0, A^2r_0, \ldots, A^{i-1}r_0)$, where $A = [\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$ is the transpose of the state Jacobian, and the initial residual is $\boldsymbol{r}_0 = [\partial f/\partial \boldsymbol{w}]^T - [\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T \boldsymbol{\psi}_0$. We assemble and store a full $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$ matrix in memory and then explicit pass it to PETSc for computing the matrix-vector products r_0, Ar_0, A^2r_0, etc. We use a nested preconditioning strategy with the additive Schwartz method as the global preconditioner and the incomplete lower and upper (ILU) factorization approach with one or two levels of fill-in for the local preconditioning. To improve convergence, we construct the preconditioner matrix $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T_{\text{PC}}$ by approximating the residuals and their linearizations [33, 60]. This strategy is effective for solving the adjoint equation, as reported in He et al. [26, 60].

No coloring scheme is needed for $\partial \boldsymbol{R}/\partial \boldsymbol{x}$ and $\partial f/\partial \boldsymbol{x}$. Instead, we use a brute-force finite-difference approach by successively perturbing the design variables and computing the perturbed residuals and objective function [33, 60]. After computing $\partial \boldsymbol{R}/\partial \boldsymbol{x}$, $\partial f/\partial \boldsymbol{x}$, and $\boldsymbol{\psi}$, we compute the total derivative $\mathrm{d}f/\,\mathrm{d}\boldsymbol{x}$ in step 9.

2.3 Solver-specific adjoint implementation

The FD Jacobian adjoint approach described in Sec. 2.2 is applicable to any primal solver. In this section, we elaborate on the DAFoam's object-oriented adjoint framework that allows developers to rapidly implement solver-specific adjoints. The DAFoam framework builds on the observation that, for different primal solvers, their adjoint implementations differ in three major aspects:

1. Elements in the residual $\boldsymbol{R}$ and state variable $\boldsymbol{w}$ vectors;
2. Stencils of Jacobians;
3. Form of residual computation routine $\boldsymbol{R} = \boldsymbol{R}(\boldsymbol{w}, \boldsymbol{x})$.

In DAFoam, we provide high-level interfaces that allow developers to easily specify the above three variations, as shown in Fig. 2. This is done by adding child classes for each primal solver and providing solver-specific implementations. To be more specific, the elements in the states and residuals are set in the `AdjointSolverRegistry` child classes, the stencils of Jacobians are specified in the `AdjointJacobianConnectivity` child classes, and the residual function computation is given in the `AdjointDerivative` child classes. OpenFOAM's primal solvers support selecting various turbulence models at runtime. To enable a similar feature for the adjoint solvers, we treat the turbulence state variable separately. The turbulence-model-related adjoint implementations are provided in the `AdjointRASModel` child classes. Once all the child classes are properly added and compiled, we can compute adjoint derivatives for any specified primal solver and model at runtime.

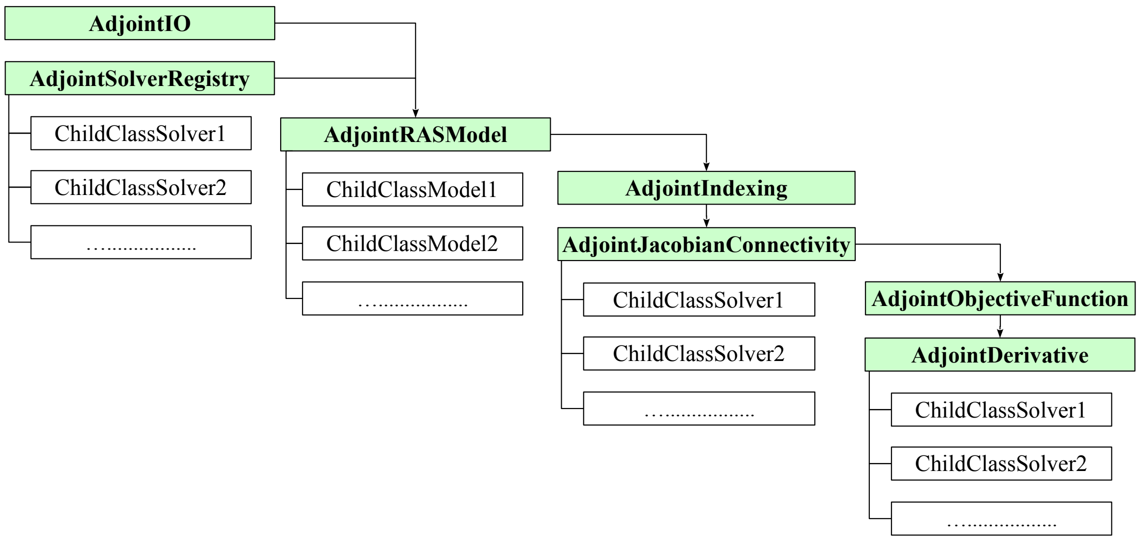


Figure 2: Object-oriented code structure for rapid discrete adjoint implementation in DAFoam. For a new adjoint solver, we need to add child classes for registering states, specifying residual connectivity, and computing residuals.

In the following, we use one of OpenFOAM's built in primal solvers (simpleFoam) as an example to illustrate the object-oriented adjoint implementation process. The governing equations for simpleFoam are the incompressible NS equations.

$$\nabla \cdot \boldsymbol{U} = 0, \tag{6}$$

$$\nabla \cdot (\boldsymbol{U}\boldsymbol{U}) + \nabla p - \nu_{\text{eff}} \nabla \cdot (\nabla \boldsymbol{U} + \nabla \boldsymbol{U}^T) = 0, \tag{7}$$

where $\boldsymbol{U}$ is the velocity vector, p is the pressure, and $\nu_{\text{eff}} = \nu + \nu_t$ is the effective kinematic viscosity with ν and ν_t being the molecular and turbulent kinematic viscosity, respectively. simpleFoam supports multiple turbulence models; however, here we use the one-equation Spalart–Allmaras (SA) turbulence model as an example:

$$\nabla \cdot (\boldsymbol{U}\tilde{\nu}) - \frac{1}{\sigma}\{\nabla \cdot [(\nu + \tilde{\nu})\nabla \tilde{\nu}] + C_{b2}|\nabla \tilde{\nu}|^2\} - C_{b1}\tilde{S}\tilde{\nu} + C_{w1} f_w \left(\frac{\tilde{\nu}}{d}\right)^2 = 0, \tag{8}$$

where $\tilde{\nu}$ is related to the turbulent viscosity ν_t via $\nu_t = \tilde{\nu}\chi^3/(\chi^3 + C_{v1}^3), \chi = \tilde{\nu}/\nu$. Spalart and Allmaras [62] includes more details on the terms and parameters in Eq. (8).

The continuity (6) and momentum (7) equations are coupled by using the semi-implicit method for pressure-linked equations (SIMPLE) algorithm [63], along with the Rhie–Chow interpolation [64]. The turbulence equation (8) is solved in a segregated manner. The finite-volume method is used to discretize the above equations on collocated meshes such that we obtain a discrete form of residual function $\boldsymbol{R}(\boldsymbol{w}, \boldsymbol{x})$. To implement the discrete adjoint method in DAFoam, we follow three steps:

1. Create a child class `AdjointSolverRegistrySimpleFoam` to specify the elements in $\boldsymbol{R}$ and $\boldsymbol{w}$. According to the governing equations (6)–(8), simpleFoam has six

Listing 1: Sample code to register residuals and states for simpleDAFoam.

```
volScalarStates.append("p");       // Register scalar volume field p
volVectorStates.append("U");       // Register vector volume field U
surfaceScalarStates.append("phi");// Register scalar surface field
  phi
this->setDerivedInfo(); // Create derived names (e.g., residuals)
```

residuals (R_u, R_v, R_w, R_p, R_{ν_t}, and R_ϕ) and six state variables (u, v, w, p, ν_t, and ϕ), where ϕ is the surface flux. The reason for treating ϕ as an independent state was explained by He et al. [60]. Therefore, we append these state names to the registry list, as shown in Listing 1. To facilitate the solver-agnostic adjoint implementation, we need to separately register the states based on their field types (i.e., volume-scalar, volume-vector, or surface-scalar variables). Then, we call `setDerivedInfo` to register derived variable names, e.g., the reference value for $\boldsymbol{U}$ (`URef`) and the residual for $\boldsymbol{U}$ (`URes`). There is no need to register turbulence states here because this is done in the child classes of `AdjointRASModel`.

2. Create a child class `AdjointJacobianConnectivitySimpleFoam` to specify how many levels of surrounding $\boldsymbol{w}$ are connected to $\boldsymbol{R}$. This information will be used by the graph coloring scheme to compute the colors. To set the connectivity, we simply assign the stencil levels to the Jacobian connectivity lists, as shown in Listing 2. Figure 3 shows the connectivity for the $\boldsymbol{U}$ residual in the case of a two-dimensional structured mesh. The $\boldsymbol{U}$ residual (`URes`) depends on $\boldsymbol{U}$, p, ν_t, and ϕ at the residual cell (level zero, denoted in green). For the one level of surrounding cells, `URes` depends on $\boldsymbol{U}$, p, and ν_t. For the two levels of surrounding cells, `URes` depends on $\boldsymbol{U}$ only. Similarly, we add connectivity information for all the other residuals. The levels of connected states for a residual can be obtained by analyzing each term's stencil in the discrete residual equations. For example, the pressure term ∇p depends on one level of surrounding p and the Laplacian term $\nu_{\text{eff}}\nabla \cdot (\nabla \boldsymbol{U} + \nabla \boldsymbol{U}^T)$ depends on one level of ν_t and two levels of $\boldsymbol{U}$. Alternatively, we can obtain the connectivity level by successively perturbing all the surrounding states for a single cell residual and evaluating which state impacts this residual. For complex residual equations, the latter is preferred. Next, we replace ν_t with the corresponding turbulence state variables in the `ResConInfo` list by calling `correctAdjStateResidualTurbCon` (e.g., replacing ν_t with $\tilde{\nu}$ for the SA model and replacing ν_t with k and ε for the k-ε model). Finally, we call `setAdjStateResidualTurbCon` to add turbulence residual connectivity. As mentioned above, we treat the turbulence-model-related adjoint implementations separately; `correctAdjStateResidualTurbCon` and `setAdjStateResidualTurbCon` are implemented in the child classes of `AdjointRASModel`.

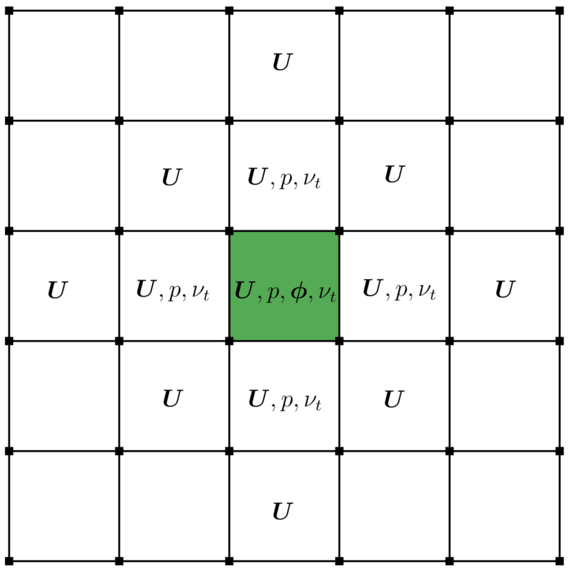
3. Create a child class `AdjointDerivativeSimpleFoam` and provide a function to compute $\boldsymbol{R}$ based on $\boldsymbol{w}$ and $\boldsymbol{x}$. Generally, this task is time-consuming because

Listing 2: Sample code to specify the momentum residual connectivity for simpleDAFoam.

```
ResConInfo.set
(
  "URes",
  {
    {"U","p","nut","phi"},  // Level zero connected states
    {"U","p","nut"},        // Level one connected states
    {"U"}                   // Level two connected states
  }
);
....                        // Connectivity for other residuals

// replace "nut" in ResConInfo with the corresponding turbulence
// state variables for the selected turbulence model
adjRAS.correctAdjStateResidualTurbCon(ResConInfo["URes"]);
....

// add residual connectivity for the selected turbulence model
adjRAS.setAdjStateResidualTurbCon(ResConInfo);
```



$\boldsymbol{U}$
$\boldsymbol{U}$
$\boldsymbol{U}, p, \nu_t$
$\boldsymbol{U}$
$\boldsymbol{U}$
$\boldsymbol{U}, p, \nu_t$
$\boldsymbol{U}, p, \boldsymbol{\phi}, \nu_t$
$\boldsymbol{U}, p, \nu_t$
$\boldsymbol{U}$
$\boldsymbol{U}$
$\boldsymbol{U}, p, \nu_t$
$\boldsymbol{U}$
$\boldsymbol{U}$

Figure 3: Connectivity of the $\boldsymbol{U}$ residual for a two-dimensional structured mesh. The residual cell is in green.

the residual functions in a primal solver typically include low-level implementation details and can become complex as more governing equations are involved. Fortunately, each primal solver in OpenFOAM has standardized high-level residual computation routines that we can reuse to facilitate the adjoint implementations. Listing 3 shows the code corresponding to the $\boldsymbol{U}$ residual computation as an example. Note that this listing illustrates the idea of reusing the primal codes for residual computation; the actual calcResiduals function in simpleDAFoam is slightly different. In the primal solver simpleFoam, a finite-volume matrix (FVM) object `UEqn` has been already created containing all the relevant terms (pressure, divergence, deviatoric, and Laplacian) for the $\boldsymbol{U}$ equation (lines 4 to 10). Then simpleFoam solves the linear equation `UEqn&U=0` to obtain $\boldsymbol{U}$ (line 12). In the adjoint implementation, we reuse the `UEqn` from the primal solver (lines 20 to 26) and perform a matrix-vector product `UEqn&U` to compute the `URes` (line 27), where '&" denotes the matrix-vector product operation in OpenFOAM. This strategy allows us to rapidly identify and construct the residual computation functions without specific knowledge of the low-level implementations of primal solvers. We need to specify residual computation codes for all other residuals. In addition, we need to update intermediate variables that depend on the state variables and are used in the residual computation (for example, the density needs to be updated based on the updated pressure and temperature). For simpleDAFoam, we do not need to update any intermediate variable (lines 34 to 37).

In summary, we can follow the above steps to rapidly implement the discrete adjoint method for existing or new steady-state primal solvers in OpenFOAM. We can also follow a similar route (register state names, set connectivity, and provide residual functions) to implement the adjoint for different turbulence models in the `AdjointRASModel` child classes. We need to add or modify only a few hundred lines of source code. The major differences in adjoint implementations between different primal solvers and models are the details of the residual computation functions. As mentioned above, we can reuse the OpenFOAM's built-in FVM matrices, a high-level interface to construct the linear equation matrices. This convenient feature allows us to rapidly construct the corresponding residual computation functions without specific knowledge of the low-level implementation details.

2.4 Performance evaluation

Using DAFoam's object-oriented adjoint framework shown in Sec. 2.3, we implement the adjoint method for eight primal solvers, five turbulence models, and one radiation model, as listed in Table 1. These solvers and models include a wide range of disciplines (aerodynamics, heat transfer, structures, and radiation) and flow conditions (incompressible, subsonic, transonic, as well as full and transitional turbulence). The naming convention is to add "DA" (discrete adjoint) to the original name of a primal solver in OpenFOAM. Taking simpleDAFoam as an example, it is based on the OpenFOAM's built-in steady-state incompressible solver simpleFoam. We can use simpleDAFoam to simulate the flow and compute the adjoint derivatives in a gradient-based optimization framework.

2.4.1 Speed, scalability, and memory usage

Now we evaluate the speed, scalability, and memory usage of the above adjoint implementations. We use Case 3 from the AIAA Aircraft Design Optimization Discussion Group

Listing 3: Sample code to compute flow residuals for simpleDAFoam.

```
// ************************************************************
// Original U equation solution code in simpleFoam for reference
{
  fvScalarMatrix UEqn                          // U FVM matrix
  (
      fvm::div(phi,U)                          // Divergence term
    + fvc::grad(p)                             // Pressure term
    - fvm::laplacian(nuEff,U)                  // Laplacian term
    - fvc::div(nuEff*dev2(T(fvc::grad(U)))) // Deviatoric term
  );
  UEqn.relax();                                // Set under-relaxation
  UEqn.solve();                // Solve the equation UEqn&U=0 to get U
}
// ************************************************************

// Residual computation function in simpleDAFoam
void calcResiduals()
{
  // Reuse the FVM matrix UEqn from the primal solver
  fvVectorMatrix UEqn
  (
      fvm::div(phi,U)                          // Divergence term
    + fvc::grad(p)                             // Pressure term
    - fvm::laplacian(nuEff,U)                  // Laplacian term
    - fvc::div(nuEff*dev2(T(fvc::grad(U)))) // Deviatoric term
  );
  // Matrix-vector product for the U residual
  volVectorField URes = UEqn&U;

  ...                                  // Compute other residuals
}

// Update any intermediate variables that depend on the state
// variables and are used in the residual computation
void updateIntermediateVariables()
{
  // no intermediate variable for simpleDAFoam
}
```

(ADODG)[3], which consists of an unswept rectangular wing with a NACA 0012 airfoil profile. We run the flow simulations and adjoint computation using simpleDAFoam at Reynolds number 10^6 and Mach number 0.15. The objective function is the drag coefficient (C_D) and the nominal flight condition is at lift coefficient (C_L) of 0.375. The design variables are the twists (γ) at eight spanwise locations. We used this configuration to benchmark the performance of DAFoam adjoint solvers in Kenway et al. [33]. We generate a fine structured mesh with 10 141 696 cells and the computational domain

[3]http://mdolab.engin.umich.edu/content/aerodynamic-design-optimization-workshop

Table 1: Summary of implemented adjoint solvers and turbulence models.

Adjoint solvers	Governing Equations
laplacianDAFoam	Laplacian equation
simpleDAFoam	Incompressible NS equations
simpleTDAFoam	Incompressible NS equations with heat transfer
buoyantBoussinesqSimpleDAFoam	Incompressible NS equations with heat transfer, buoyancy, and radiation
rhoSimpleDAFoam	Compressible NS equations (subsonic)
rhoSimpleCDAFoam	Compressible NS equations (transonic)
buoyantSimpleDAFoam	Compressible NS equations with heat transfer, buoyancy, and radiation
solidDisplacementDAFoam	Linear elastic equation

Turbulence Models	Description
SpalartAllmaras	Spalart–Allmaras one-equation model
kEpsilon	Standard $k-\varepsilon$ two-equation model
realizableKE	Realizable $k-\varepsilon$ model
kOmegaSST	Menter $k-\omega$ SST two equation model
kOmegaSSTLM	Langtry–Menter four equation transitional model based on kOmegaSST

Table 2: Wall-clock runtime for the flow and adjoint computation with increasing number of CPU cores. The mesh contains 10 141 696 cells. The adjoint computation scales well up to 1536 cores.

Nodes	Cores	Flow runtime (s)	Adjoint runtime (s)	Adjoint/flow
8	192	810 (100.0%)	1809 (100.0%)	2.2
16	384	433 (93.5%)	976 (92.7%)	2.3
32	768	242 (83.7%)	481 (94.0%)	2.0
64	1536	156 (64.9%)	262 (86.3%)	1.7

extends 20 chords from the surface. The primal solver runs for 3000 steps, at which point the residuals drop 13 orders of magnitudes and stall. For the adjoint computation, we set the relative residual tolerance for the adjoint linear equation solution to 10^{-6}, which is a typical value in an optimization process. There is no need converge the residual of adjoint linear equations any more than this because the adjoint total derivatives are accurate only up to four significant digits (Table 3) due to the errors in the finite-difference-based partial derivative computation (Sec. 2.2).

Table 2 shows the speed and scalability of flow and adjoint computations. All the simulations are conducted on Stampede 2 [65] using the Skylake nodes. The Skylake nodes are equipped with an Intel Xeon Platinum 8160 CPU running at 2.1 GHz, and each node has 48 CPU cores and 196 GB of memory. The values in parentheses are the parallel efficiencies, defined as $192t_{192}/(nt_n) \times 100\%$, where t is the wall-clock runtime and n is the number of CPU cores. The adjoint derivative computation scales better than the flow simulation, especially when using more CPU cores. For example, with 1536 CPU cores, the parallel efficiencies for the flow and adjoint are 65% and 86%, respectively. In addition, the runtime ratio between the adjoint and flow solutions varies between 1.7 to 2.2, which is within the acceptable range for performing practical optimization. The runtimes for the partial derivative computation and adjoint linear equation solution are similar. In terms of memory usage, the flow solution takes 73.2 GB memory, while the adjoint computation requires 1146.2 GB memory. The large peak memory requirement is the current bottle neck of our adjoint implementation, which is in part because we explicitly form and store the transpose state Jacobian matrix $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$ and its preconditioner matrix $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T_{\mathrm{PC}}$ in memory. Due to the use of SIMPLE algorithm and Rhie–Chow interpolation, the pressure residual depends on three levels of surrounding cells in simpleDAFoam [33, 60]. In contrast, for density-based flow solvers such as ADflow [33, 41] and SU2 [37], the flow residuals depend at most on two levels of surrounding cells. The larger stencil in simpleDAFoam results in a denser $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$ matrix, which requires a large amount of memory to store. In addition, the denser preconditioner matrix $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T_{PC}$ causes a larger memory overhead when using a nonzero fill-in level in the ILU factorization. To alleviate the large memory requirement, we can use the Jacobian-free GMRES adjoint solution strategy detailed by Kenway et al. [33], where the $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$ matrix is not explicitly computed or stored. For ADflow, the Jacobian-free approach saved up to 30% memory, compared with the FD Jacobian method [33]. Implementing the Jacobian-free approach in DAFoam and optimizing its performance will be conducted in the future work.

2.4.2 Adjoint accuracy

To verify the adjoint derivative accuracy, we consider all the implemented adjoint solvers and models. For the flow solvers, we use the ADODG Case 3 with a coarse mesh of 102 912 cells. The average y^+ is 32.6; therefore, we use wall functions for all turbulence models. The Mach numbers for the incompressible, subsonic, and transonic conditions are 0.15, 0.5, and 0.7, respectively. As mentioned above, the adjoint solvers compute derivatives at all eight spanwise locations; however, we show only one representative derivative at the 40% spanwise location, $\mathrm{d}C_D/\,\mathrm{d}\gamma_{40\%}$, for all solvers and models.

For the structural analysis solver (solidDisplacementDAFoam), we use the Rotor 67 case (an axial compressor rotor) [66] with a rotational speed of 840 rad s^{-1}. We generate a triangular unstructured mesh with 91 475 cells. The objective function is an aggregation of the von Mises stresses (σ_v^a) of the rotor subjected to the centrifugal force, and the design variable is the rotational speed. To compute σ_v^a, we use the Kreisselmeier–Steinhauser (KS) function to aggregate σ_v over all the mesh cells [67] such that σ_v^a is a conservative approximation of the maximal von Mises stress.

For the heat transfer solver (laplacianDAFoam), we use a flange geometry with 5712 cells. The heat transfer is imposed by adding a 200 K temperature difference between the internal and external boundaries. The objective function is the average temperature $\overline{T}$ in the flange, and the design variable is the temperature at the internal boundary T_i.

We directly compute the total derivatives using the finite-difference method and use them as the reference derivative values. To control the errors in the finite-difference method, we perform step size studies by adding various perturbation magnitudes (ranging from 10^{-5} to 10^{-2} with an interval of 0.1) to the design variable and then compare the reference total derivative values. We conduct similar step size studies for the partial derivatives in the adjoint computation; the tested step size magnitudes range from 10^{-9} to 10^{-4} with an interval of 0.1. The best step size varies depending on cases; however, 10^{-3} and 10^{-7} are the most common perturbation step sizes for the reference total derivatives and the partial derivatives computation, respectively.

Finally, we evaluate the accuracy of adjoint derivative computation, as shown in Table 3. For all the implemented solvers and models, there is good agreement between the adjoint derivatives and the reference values computed using the brute-force finite-difference method mentioned above. The average relative error is less than 0.1%. This level of error is acceptable for performing gradient-based optimization, as shown in our previous studies [12, 26, 60].

3 Results and Discussion

In this section, we integrate the implemented adjoint solvers and models into a gradient-based optimization framework described in our previous work [60]. We then perform four distinct optimizations that cover a wide range of disciplines, configurations, and flow conditions: 1. Multipoint aerodynamic optimization for a low-speed UAV wing, 2. Trimmed aerodynamic optimization for a transonic aircraft configuration, 3. Aerothermal optimization for a turbine internal cooling passage, 4. Aerostructural optimization for an axial compressor rotor.

The main objective of this section is to demonstrate the benefit of having the flexibility to rapidly implement the discrete adjoint method. Therefore, a comprehensive optimization analysis that evaluates the impacts of different flow conditions, configurations, turbulence models, and mesh densities for each of these applications is outside the scope of the paper.

3.1 Multipoint aerodynamic optimization of a low-speed UAV wing

In this subsection, we perform a multipoint aerodynamic optimization for a low-speed UAV wing. The goal is to demonstrate DAFoam's multipoint optimization capability for incompressible conditions. We use the adjoint solver simpleDAFoam, and the governing equations are the incompressible NS equations, as shown in Eqs. (6) and (7). We use the Spalart–Allmaras turbulence model (8) for all the optimizations in this paper.

The wing geometry is taken from a multi-mission UAV prototype called Odyssey [68]. The wing planform is rectangular with an aspect ratio of 8.57, a span of 4.572 m, and an Eppler 214 airfoil. There is no twist or sweep in the baseline wing geometry. For the

Table 3: The adjoint derivatives for all the implemented solvers and turbulence models reasonably match the reference values. The average error is less than 0.1%.

ADODG Case 3 aerodynamics, Mach number 0.15, $\mathrm{d}C_D/\,\mathrm{d}\gamma_{40\%} \times 10^{-4}$

Solver	Turbulence Model	Reference	Adjoint	Error
simpleDAFoam	Spalart–Allmaras	−7.4141337	−7.414$\underline{3066}$	0.002%
	$k-\varepsilon$	−7.6683056	−7.668$\underline{4497}$	0.002%
	Realizable $k-\varepsilon$	−7.5622189	−7.56$\underline{48494}$	0.035%
	$k-\omega$ SST	−7.4261818	−7.42$\underline{54367}$	0.010%
	$k-\omega$ SSTLM	−7.3058416	−7.3$\underline{105428}$	0.064%
simpleTDAFoam	Spalart–Allmaras	−7.4041615	−7.404$\underline{3562}$	0.003%
buoyantBoussinesqSimpleDAFoam	Spalart–Allmaras	−7.1222484	−7.122$\underline{3115}$	0.001%

ADODG3 wing aerodynamics, Mach number 0.5, $\mathrm{d}C_D/\,\mathrm{d}\gamma_{40\%} \times 10^{-4}$

Solver	Turbulence Model	Reference	Adjoint	Error
rhoSimpleDAFoam	Spalart–Allmaras	-7.9530799	-7.95$\underline{13823}$	0.021%
	$k-\varepsilon$	-8.2006894	-8.2$\underline{124952}$	0.144%
	Realizable $k-\varepsilon$	-8.1720642	-8.172$\underline{4885}$	0.005%
	$k-\omega$ SST	-8.0006496	-8.0$\underline{162433}$	0.195%
	$k-\omega$ SSTLM	-7.9764931	-7.9$\underline{875446}$	0.138%
buoyantSimpleDAFoam	Spalart–Allmaras	-7.5714023	-7.57$\underline{09535}$	0.006%

ADODG3 wing aerodynamics, Mach number 0.7, $\mathrm{d}C_D/\,\mathrm{d}\gamma_{40\%} \times 10^{-3}$

Solver	Turbulence Model	Reference	Adjoint	Error
rhoSimpleCDAFoam	Spalart–Allmaras	-1.4533128	-1.45$\underline{24513}$	-0.059%

Rotor 67 blade rotating at ω=840 rad/s, $\mathrm{d}\sigma_v^a/\,\mathrm{d}\omega \times 10^4$

Solver	Turbulence Model	Reference	Adjoint	Error
solidDisplacementDAFoam	–	-9.6106984	-9.610$\underline{9359}$	0.002%

Flange heat transfer, $\mathrm{d}\overline{T}/\,\mathrm{d}T_i \times 10^{-1}$

Solver	Turbulence Model	Reference	Adjoint	Error
laplacianDAFoam	–	3.5956891	3.595689$\underline{0}$	<0.001%

CFD, we use a structured hexahedral mesh with 548 352 cells identical to that of our previous work [69, Fig. 5]. We use ANSYS ICEM-CFD to generate the surface mesh, and then we extrude the surface mesh to the volume mesh using a hyperbolic mesh

Table 4: Multipoint aerodynamic optimization setup for the low-speed UAV wing, which has 129 design variables and 416 constraints.

	Function or variable	Description	Quantity
minimize	$f = \sum_{i=1}^{3} w^i {C_D}^i$	Weighted drag coefficients	
with respect to	Δz	Deformation of FFD points in the vertical direction	120
	γ	Twist	6
	α	Angle of attack	3
		Total design variables	**129**
subject to	C_L=0.6, 0.75, or 0.9	Lift-coefficient constraints for each flight condition	3
	$t \geq 0.5 t_{\text{baseline}}$	Minimum-thickness constraint	400
	$V \geq V_{\text{baseline}}$	Minimum-volume constraint	1
	$\Delta z_{\text{LE}}^{\text{upper}} = -\Delta z_{\text{LE}}^{\text{lower}}$	Fixed leading-edge constraint	6
	$\Delta z_{\text{TE}}^{\text{upper}} = -\Delta z_{\text{TE}}^{\text{lower}}$	Fixed trailing-edge constraint	6
	$-0.5 \text{ m} < \Delta z < 0.5 \text{ m}$	Design variable bounds	
		Total constraints	**416**

marching method [70] implemented in the pyHyp package [4]. The average y^+ is 33.7. The simulation domain extends to 30 chord lengths. The Mach number is 0.074 and the Reynolds number is 9×10^5.

In our previous work, we had performed a single point aerodynamic optimization for the same UAV wing [60]. In this paper, we use a similar design variable and constraint setup but consider multipoint optimization, as detailed in Table 4. We select three flight conditions with C_L=0.6, 0.75 (nominal), and 0.9. The objective is the weighted C_D with weights of 0.25, 0.5, and 0.25 for the three flight conditions, respectively.

All the optimizations are conducted using MACH [5], an open-source framework for high-fidelity gradient-based optimization. We use the free-form deformation (FFD) method to parameterize the design surface [71] through the pyGeo package[6]. For volume mesh deformation, we use an analytic inverse-distance method [72] through the IDWarp package[7]. As in our previous work we use 120 free-form deformation (FFD) points to control the local wing shape at six spanwise locations [69, Fig. 5]. In addition, the twists at these six spanwise locations are selected to be the design variables along with the angle of attack at the three flight conditions. The root twist is fixed. The total number of design variables is 129. We constrain the lift coefficients for each flight condition. In addition, we limit the local wing thickness to be greater than 50% of the baseline thickness. Finally, we constrain the total volume of the optimized wing to be greater than or equal to that of the baseline wing, and the leading and trailing edges of the wing

[4]https://github.com/mdolab/pyhyp
[5]https://github.com/mdolab/mach-aero
[6]https://github.com/mdolab/pygeo
[7]https://github.com/mdolab/idwarp

are fixed. In total, we have 416 constraints for this case. We use SNOPT [73] as the optimizer for all the optimizations in this paper through the pyOptSparse interface [74][8]. We set the optimality and feasibility tolerances to 10^{-6}. We run the optimizations until the optimizer either reaches the above tolerances or aborts due to numerical difficulties to further improve the design. We also manually terminate the optimizations if the objective function changes less than 0.001% in five steps.

The comparison of pressure, spanwise lift, twist, and maximum thickness distributions between the baseline and optimized designs is shown in Fig. 4. The optimization converges in 25 steps, achieving a 3.7% drag reduction. This is lower than the single-point optimization (5.6% drag reduction) reported in our previous work, which is expected [60]. However, compared with the previous single-point optimization [60, Fig. 16], the leading edge is less sharp at all spanwise locations, which is better for off-design performance. Similar to the single-point optimization, the optimized design achieves the desired elliptical lift distribution by fine-tuning the twist, thickness, and camber distribution.

3.2 Trimmed aerodynamic optimization of a transonic aircraft configuration

In this subsection, we perform a trimmed aerodynamic shape optimization for an aircraft wing-body-tail configuration. The goal is to demonstrate DAFoam's optimization capability for complex aircraft configurations at transonic conditions, similarly to our previous efforts [23, 24, 75, 76] using a dedicated RANS solver ADflow [33, 41]. We use the adjoint solver rhoSimpleCDAFoam, and the governing equations are the compressible NS equations:

$$\nabla \cdot (\rho \boldsymbol{U}) = 0, \tag{9}$$

$$\nabla \cdot (\rho \boldsymbol{U}\boldsymbol{U}) + \nabla p - \mu_{\text{eff}} \nabla \cdot (\nabla \boldsymbol{U} + \nabla \boldsymbol{U}^T) = 0, \tag{10}$$

$$\nabla \cdot (\rho e \boldsymbol{U}) + \nabla \cdot (0.5\rho |U|^2 \boldsymbol{U} + p\boldsymbol{U}) - \alpha_{\text{eff}} \nabla \cdot (\nabla e) = 0, \tag{11}$$

where ρ is the density, e is the internal energy, μ is the dynamic viscosity, and α is the thermal diffusivity. The heat and mechanical source terms are ignored. The governing equations (9)–(11) are solved using the compressible form of the SIMPLEC algorithm [77], which is a modified SIMPLE algorithm for compressible flows.

The aircraft geometry is obtained from the Common Research Model (CRM), which is representative of a modern transonic commercial transport aircraft with a size similar to that of a Boeing 777. This configuration is also known as the Drag Prediction Workshop 4 (DPW4) geometry [78] and was studied in our previous work [23, 24, 75, 76]. The Mach number is 0.85 and the Reynolds number is 5×10^6. We generate unstructured hexahedral meshes with a total of 872 404 cells using OpenFOAM's built-in utility snappyHexMesh , as shown in Fig. 5. The average y^+ is 307.8.

Table 5 summarizes the trimmed aerodynamic optimization setup. The objective function is C_D. The design variable and constraint setup is similar to our previous

[8]https://github.com/mdolab/pyoptsparse

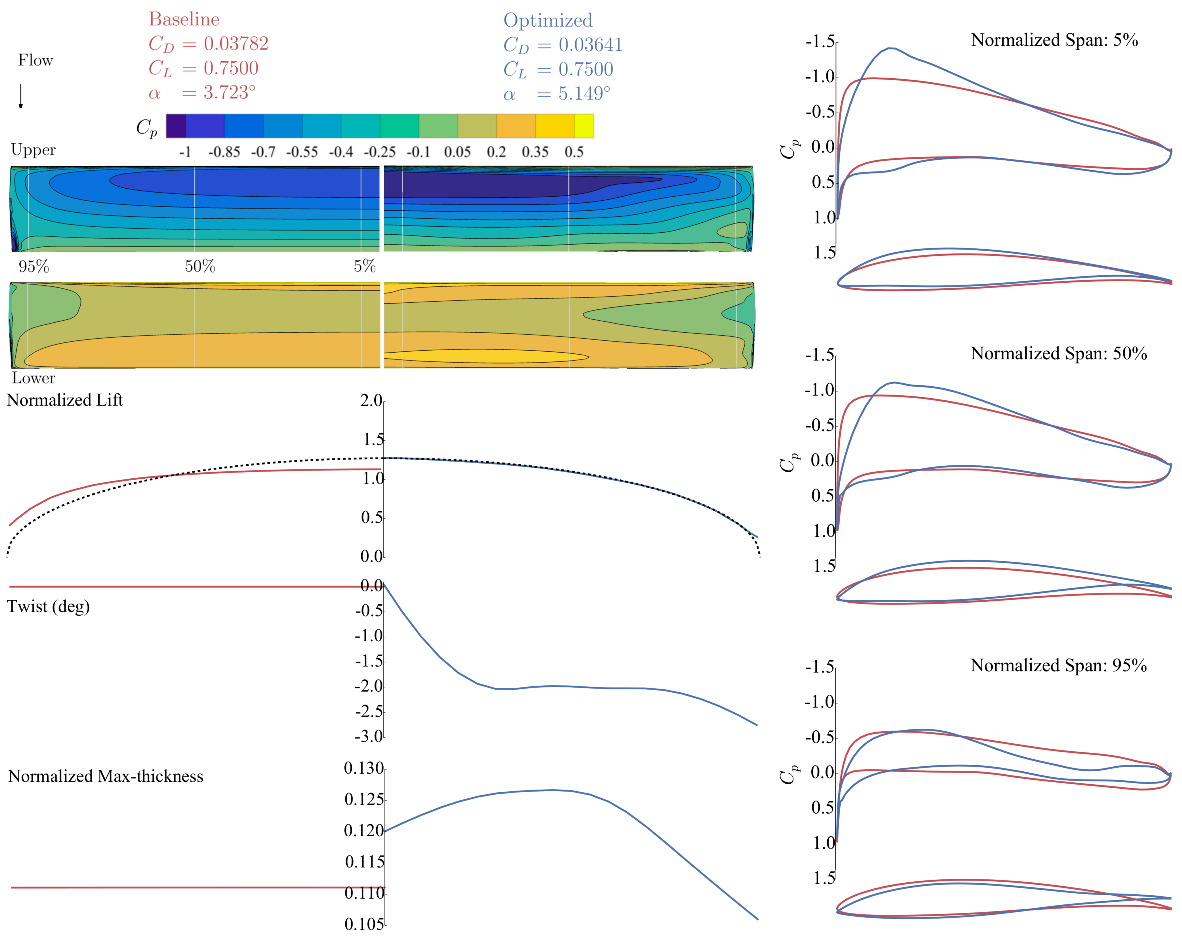


Figure 4: Multipoint aerodynamic optimization results for the UAV wing. We achieve the desired elliptical lift distribution. The drag is reduced by 3.7%, and the constraints are satisfied.

work [75, 76]. We use 216 FFD points to control the local wing shape at nine spanwise locations (Fig. 5). In addition, the wing twists at these nine spanwise locations are selected to be the design variables, along with the tail rotation and the angle of attack. The root twist is fixed. The total number of design variables is 227. We constrain the lift coefficients to be equal to 0.5 and the pitching moment to be equal to zero. In addition, we limit the local wing thickness to be greater than 20% of the baseline thickness. Finally, we constrain the total volume of the optimized wing to be greater than or equal to that of the baseline wing, and the leading and trailing edges of the wing are fixed. In total, we have 771 constraints for this case.

The comparison of pressure, spanwise lift, wing twist, tail rotation, and maximal-thickness distributions between the baseline and optimized geometries is shown in Fig. 6. The optimization converges in 31 steps, achieving 3.6% drag reduction. This is partially achieved by fine tuning the wing shape and twist distribution to change the spanwise lift closer to the desired elliptical distribution. In addition, the shock wave at the upper wing surface is eliminated in the optimized design, as confirmed by the sectional

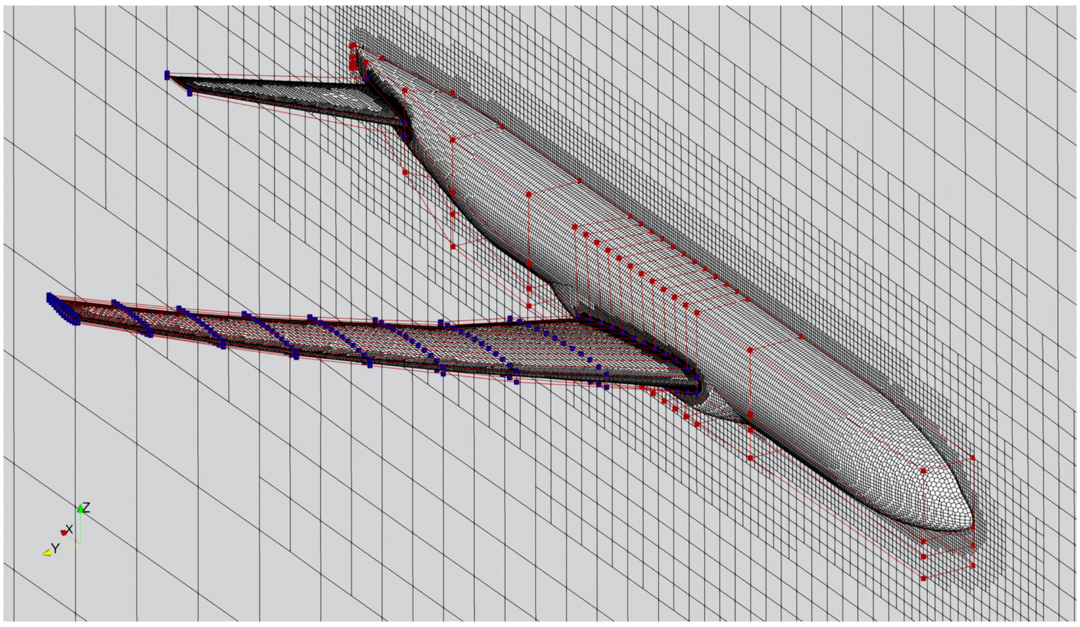


Figure 5: Unstructured hexahedral mesh for the CRM wing-body-tail case. The squares are the FFD control points to morph the design surface geometry. Only the blue FFD points are allowed to move.

Table 5: Trimmed aerodynamic optimization setup for the CRM wing-body-tail configuration, which has 227 design variables and 771 constraints.

	Function or variable	Description	Quantity
minimize	C_D	Drag coefficients	
with respect to	Δz	Displacement of FFD points in the vertical direction	216
	γ	Wing twist	9
	η_{tail}	Tail rotation	1
	α	Angle of attack	1
		Total design variables	**227**
subject to	C_L=0.5	Lift-coefficient constraint	1
	C_M^y=0	Pitching moment constraint	1
	$t \geq 0.2t_{\text{baseline}}$	Minimum-thickness constraint	750
	$V \geq V_{\text{baseline}}$	Minimum-volume constraint	1
	$\Delta z_{\text{LE}}^{\text{upper}} = -\Delta z_{\text{LE}}^{\text{lower}}$	Fixed leading-edge constraint	9
	$\Delta z_{\text{TE}}^{\text{upper}} = -\Delta z_{\text{TE}}^{\text{lower}}$	Fixed trailing-edge constraint	9
	$-10 \text{ m} < \Delta z < 10 \text{ m}$	Design variable bounds	
		Total constraints	**771**

pressure distributions. By adjusting the tail rotation, the pitching moment changes from -0.08427 (baseline) to -0.00036 (optimized) and the pitching moment constraint is satisfied.

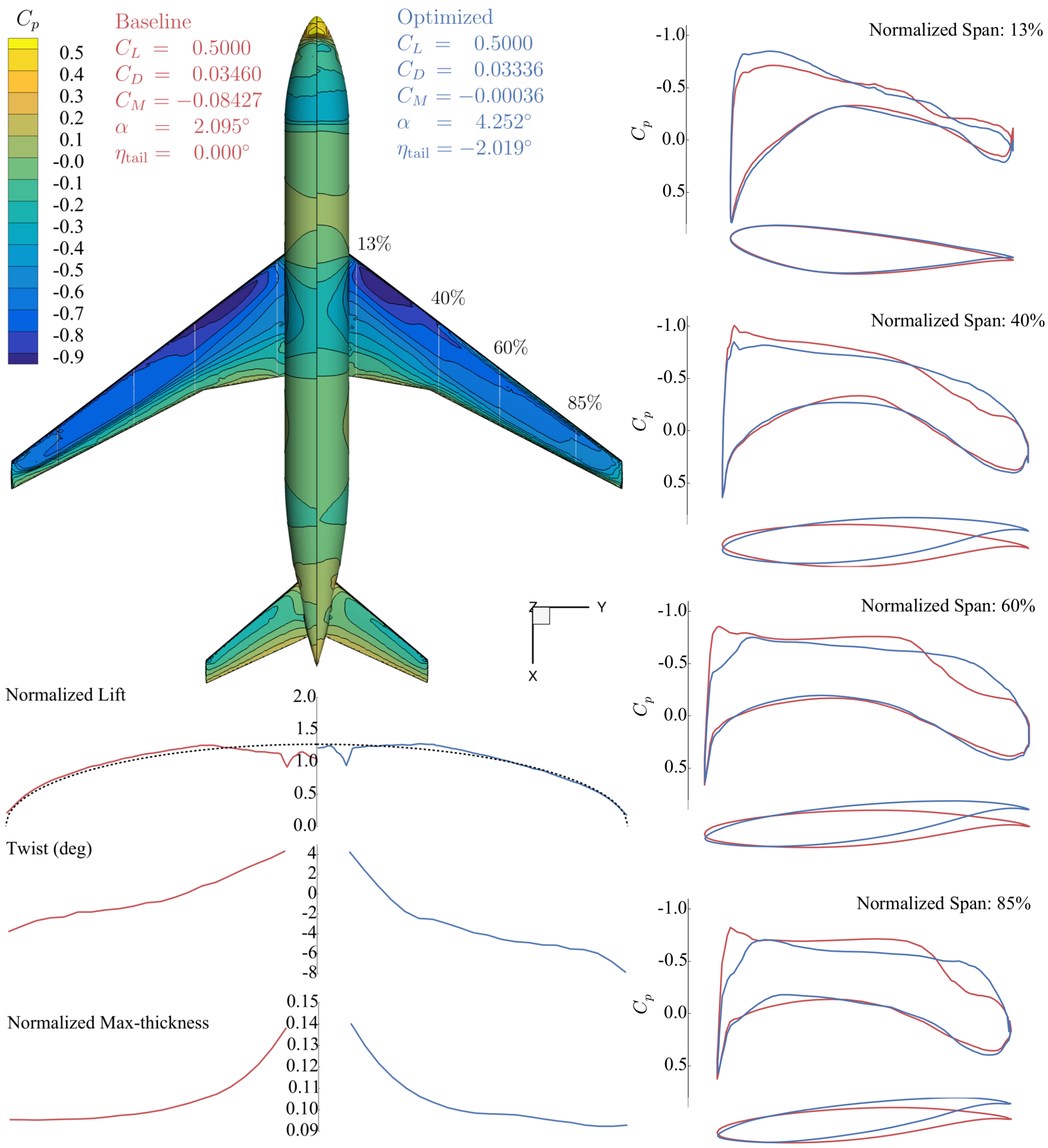


Figure 6: Trimmed aerodynamic optimization results for the CRM wing-body-tail configuration. The drag is reduced by 3.6%, and the constraints are satisfied.

3.3 Aerothermal optimization of a turbine internal cooling passage

In this subsection, we perform an aerothermal optimization for a turbine internal cooling channel. The goal is to demonstrate DAFoam's flexibility to handle complex PDEs that involve multiple disciplines. We use the adjoint solver buoyantBoussinesqSimpleDAFoam, and the governing equations are the incompressible NS equations, coupled to heat transfer, buoyancy, and radiation equations:

$$\nabla \cdot \boldsymbol{U} = 0, \tag{12}$$

Table 6: Aerothermal optimization setup for the turbine internal cooling channel, which has 113 design variables and 38 constraints.

	Function or variable	Description	Quantity
Minimize	f	Weighted C_{PL} and $\overline{Nu}$	
with respect to	Δx, Δy, and Δz	Displacement of FFD points	**113**
subject to	g_z^{sym}=0	Zero slope at symmetry plane	29
	$\Delta y_{in1} + \Delta y_{in2} > t_{\text{min}}$	Non-overlapping inner walls	9
	bound (Δx, Δy, Δz)	Design-variable bounds to confine the design surfaces within the bounding box	
		Total constraints	**38**

$$\nabla \cdot (\boldsymbol{UU}) + \nabla p - \nu_{\text{eff}} \nabla \cdot (\nabla \boldsymbol{U} + \nabla \boldsymbol{U}^T) - \rho g = 0. \tag{13}$$

$$\nabla \cdot (T\boldsymbol{U}) - \alpha_{\text{eff}} \nabla \cdot (\nabla T) + \frac{aG - 4e\sigma T^4}{\rho C_p} = 0, \tag{14}$$

$$\gamma \nabla \cdot (\nabla G) - aG + 4e\sigma T^4 = 0, \tag{15}$$

where g is the gravitational acceleration, T is the temperature, C_p is the specific heat at constant pressure, and σ is the Stefan–Boltzmann constant. We use the P1 radiation model [79], where a and e are the absorption and emission coefficients, respectively, and G is the incident radiation intensity. The emission contribution and scattering are ignored.

The baseline geometry is a U-bend channel benchmark, representative of a section of serpentine internal cooling passages [80]. It has a square cross section with a hydraulic diameter $D_h = 0.075$ m, and upstream section going from $x = 0$ (inlet) to $x = 10D_h$, a 180°-bend section, and a downstream section from $x = 10D_h$ back to $x = 0$ (outlet), refer to [26, Fig.4]. We generate a structured mesh with 409 600 cells using OpenFOAM's built-in utility blockMesh. The average y^+ is 1.3. The Reynolds number is 4.2×10^4, based on D_h. In our previous work, we performed aerothermal optimization for the same U-bend channel [26]. In this subsection, we consider a case that adds buoyancy and radiation effects. We added the gravitational acceleration $g = (-9.81, 0, 0)$ in the streamwise direction and use a 10 K temperature difference between the main stream and the walls to drive heat transfer.

The objective function is a combination of total pressure loss coefficient C_{PL} and average Nusselt number $\overline{Nu}$, and their weights are 0.2 and -0.8, respectively. We use 63 FFD points to morph the bend section and have 113 degrees of freedom (design variables) in total [26, Fig. 5]. To ensure practical shape, we impose 38 geometry constraints, as shown in Table 6. More detailed optimization configurations can be found in He et al. [26].

Figure 7 compares the velocity and Nusselt number of the baseline and optimized designs. The optimization converges in 34 steps. We obtain simultaneous improvement for aerodynamics (C_{PL} decreases by 20.5%) and heat transfer (Nusselt number increases

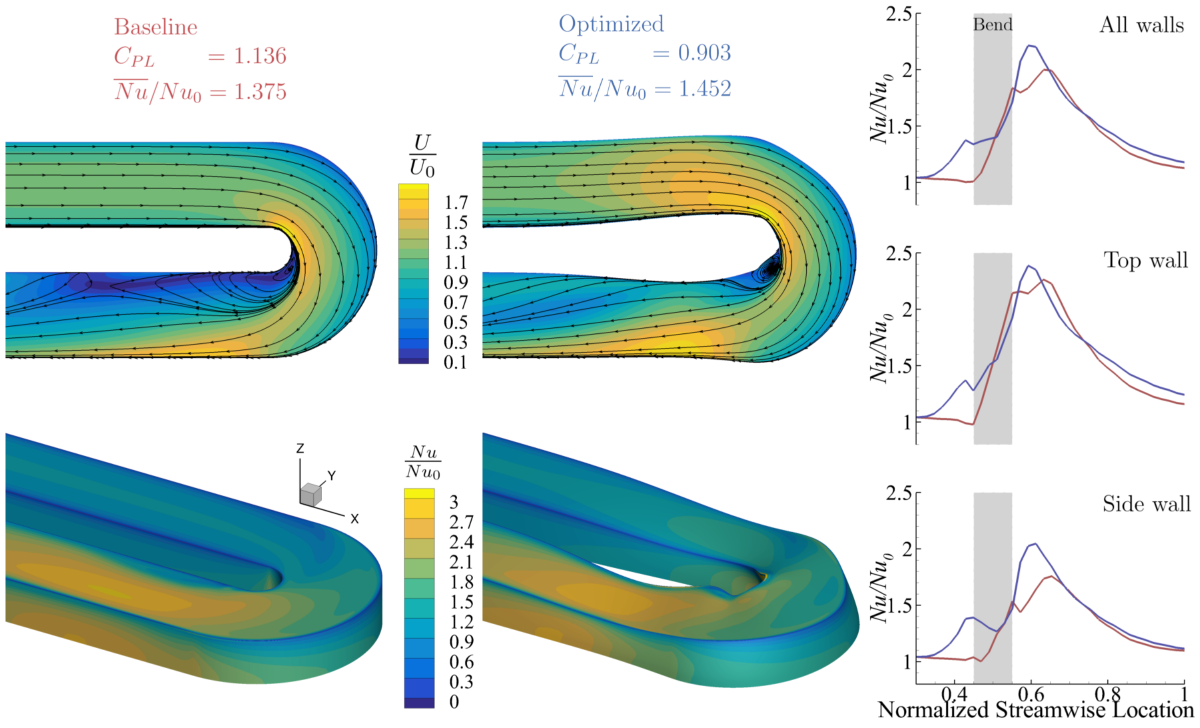


Figure 7: Aerothermal optimization results for the turbine internal cooling channel, where a simultaneous improvements for the aerodynamics and heat transfer are achieved.

by 5.6%). Similar to what we observed in the previous work [26], the aerodynamic loss reduction is primarily achieved by creating a smoother U-bend section that reduces the flow separation, as evident in the velocity contours and streamlines comparison. In addition, the channel shrinks before and after the U-bend section, which increases the velocity magnitude and the convective heat transfer, as shown in the local Nusselt number contours and the streamwise Nusselt number distributions. However, we observe a smaller separation bubble in the baseline design compared with the previous work [26, Fig. 11a], for which buoyancy and radiation were not included. In addition, we obtain higher reduction in aerodynamic loss (20.5% compared with 11.7% [26]), although a small separation region is present in the optimized design.

3.4 Aerostructural optimization of an axial compressor rotor

In this subsection, we perform an aerostructural optimization for an axial compressor rotor. The goal is to demonstrate DAFoam's capability to integrate two adjoint solvers (flow and structural analyses) for multidisciplinary design optimization. These two adjoint solvers are rhoSimpleDAFoam and solidDisplacementDAFoam. The governing equations for rhoSimpleDAFoam are the compressible NS equations, written in the multiple-reference-frame form:

$$\nabla \cdot (\rho \boldsymbol{U}_a) = 0, \tag{16}$$

$$\nabla \cdot (\rho \boldsymbol{U}_r \boldsymbol{U}_a) + \boldsymbol{\omega} \times \boldsymbol{U}_a + \nabla p - \mu_{\text{eff}} \nabla \cdot (\nabla \boldsymbol{U}_a + \nabla \boldsymbol{U}_a^T) = 0, \tag{17}$$

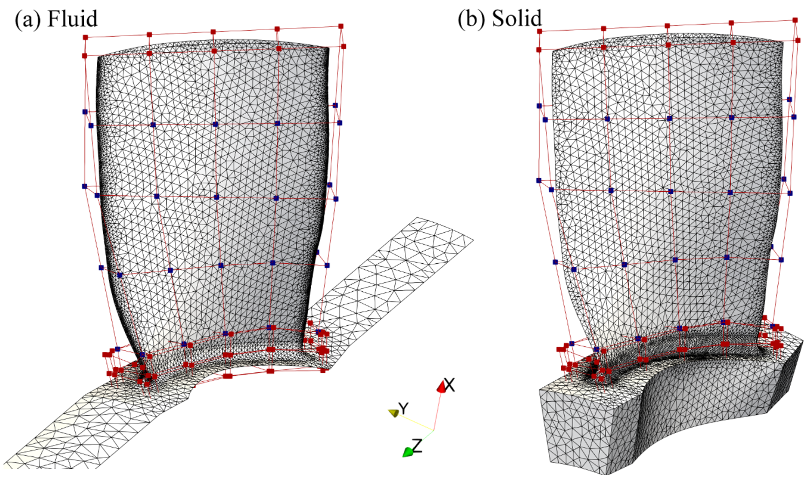


Figure 8: Unstructured triangular meshes for the Rotor 67 case. The squares are the FFD control points to morph the design surface geometry. Only the blue FFD points are allowed to move.

$$\nabla \cdot (\rho e \boldsymbol{U}_a) + \nabla \cdot (0.5\rho |U|_a^2 \boldsymbol{U}_a + p\boldsymbol{U}_a) - \alpha_{\text{eff}} \nabla \cdot (\nabla e) = 0, \tag{18}$$

where $\boldsymbol{U}_a$ and $\boldsymbol{U}_r$ are the absolute and relative velocities, respectively, and they are related through $\boldsymbol{U}_a = \boldsymbol{U}_r + \boldsymbol{\omega} \times \boldsymbol{x}$ with $\boldsymbol{\omega}$ being the rotational speed vector and $\boldsymbol{x}$ being the cell-center coordinate vector. The above governing equations are solved using the compressible form of SIMPLE algorithm based on the absolute velocity in the stationary frame; however, the flux for the convective term in the momentum equation (17) is computed using the relative velocity in the rotating frame.

The governing equations for solidDisplacementDAFoam are the linear elastic equations with the centrifugal force:

$$\frac{\partial^2 \rho \boldsymbol{D}}{\partial t^2} - \nabla \cdot (\mu \nabla \boldsymbol{D} + \mu \nabla \boldsymbol{D}^T + \lambda \boldsymbol{I} \text{tr}[\nabla \boldsymbol{D}]) - \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{x}) = 0, \tag{19}$$

where $\boldsymbol{D}$ is the structural displacement, t is the time, $\boldsymbol{I}$ is a 3×3 identity matrix, "tr" denotes the trace of a matrix, and μ and λ are the Lamé constants for the material. The governing equations are solved by time marching the solutions until the steady-state is reached. At each time step, the three components of structural displacement vector are solved in a segregated manner.

The baseline geometry is Rotor 67, which has 22 blades with a tip radius R_T of 255.4 mm and an aspect ratio of 1.56 [66]. We consider the no tip-clearance configuration and run the simulations at 50% design speed (840 rad/s) with a total pressure ratio $p_0^{out}/p_0^{in} = 1.1$. The inlet absolute Mach number is 0.29, and the chordwise Reynolds number is 8.5×10^5. We simulate only one blade with rotationally periodic boundaries conditions. We find that the contribution of the pressure force to the maximum stress is one order of magnitude lower than that of the centrifugal force. Therefore, the pressure load is ignored, and the only external force for the blade structure is the centrifugal force due to rotation. In addition, the maximum blade structural displacement is less

Table 7: Aerostructural optimization setup for the Rotor 67 case, which has 120 design variables and 3 constraints.

	Function or variable	Description	Quantity
minimize	C_M^z	Normalized torque	
with respect to	Δx, Δy, Δz	FFD displacement in the x, y, and z directions	**120**
subject to	$\dot{m} = \dot{m}_{\text{baseline}}$	Constant mass flow rate	1
	$p_0^{\text{out}}/p_0^{\text{in}} = [p_0^{\text{out}}/p_0^{\text{in}}]_{\text{baseline}}$	Constant total pressure ratio	1
	$\sigma_v^a \leq [\sigma_v^a]_{\text{baseline}}$	von Mises stress constraint	1
	-2 mm $< (\Delta x, \Delta y, \Delta z) <$ 2 mm	Design variable bounds	
		Total constraints	**3**

than 0.01 mm. Therefore, we assume that the blade is rigid and thus the fluid-structure interaction is ignored. As shown in Fig. 8, we use ANSYS ICEM-CFD to generate a triangular unstructured mesh with 361 256 cells for the fluid domain with an average y^+ of 224.4, and a 94 711 cell mesh for the solid domain.

Table 7 summarizes the aerostructural optimization setup. The objective function is the normalized torque C_M^z. We use 40 FFD points to morph the shape of the rotor blade, as shown in Fig. 8. These FFD points can move in the x, y, and z directions, for a total of 120 design variables. The FFD point displacements between the fluid and solid domains are identical. To ensure that the optimized design satisfies the aerodynamic coupling for the other compressor components, we constrain the mass flow rate $\dot{m}$ and total pressure ratio to be equal to their baseline values, similar to Wang et al. [81]. In addition, we impose a stress constraint to force σ_v^a to be less than or equal to the baseline value to avoid structure failure. As mentioned before, σ_v^a is a conservative approximation of the maximum von Mises stress.

Figure 9 shows the comparison of pressure and stress distributions between the baseline and optimized designs. The optimization converges in 20 steps and achieves a 3.2% reduction in torque. This is primarily achieved by fine tuning the blade shape and thus the pressure distribution. For example, we observe that the aft-loaded pressure distribution in the baseline design is shifted forward, especially at the mid span. In addition, the change in blade shape is a compromise between aerodynamic and structural considerations; the mass flow rate, total pressure ratio, and maximal stress constraints are well satisfied in the aerostructural optimization.

4 Conclusion

In this paper, we propose DAFoam: an open-source, object-oriented framework to rapidly implement the discrete adjoint method for any of the steady-state primal solvers in OpenFOAM. This can be accomplished by adding or modifying only a few hundred lines of source code. The central recipe in the proposed approach is to use a generalized framework for partial derivative computation and adjoint equation solution, and then

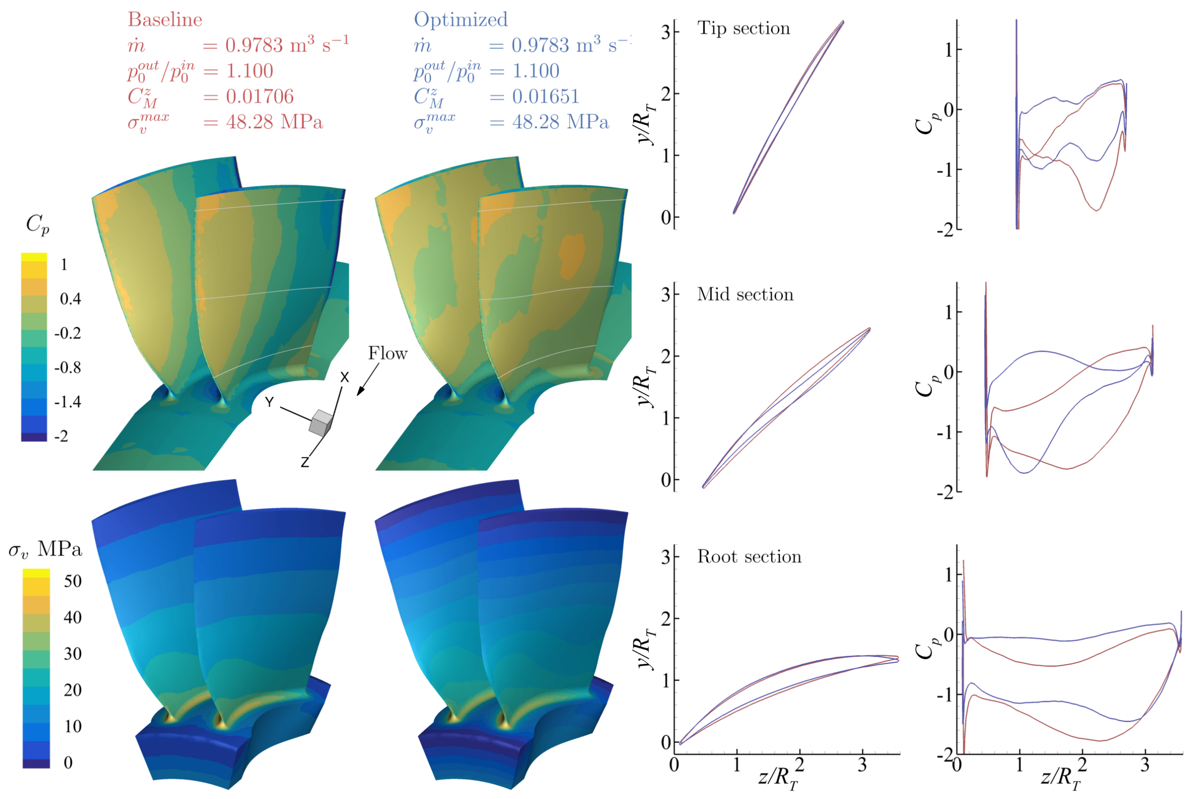


Figure 9: Aerostructural optimization results for the Rotor 67 case. The torque is reduced by 3.2%, and the constraints are satisfied.

provide a high-level interface to add the solver-specific implementations.

For the solver-agnostic adjoint implementation, DAFoam uses the finite-difference method to compute the partial derivatives, accelerated by a heuristic parallel graph coloring scheme. The adjoint equations are then solved using the GMRES method. For the solver-specific part, DAFoam provides an object-oriented interface that requires developers to specify only the functions to compute the residuals along with names of state variables and stencil levels. The residuals are computed by reusing the FVM matrix objects that have been already defined in the OpenFOAM primal solvers. This convenient feature allows users to easily construct the residual functions without specific knowledge of their low-level implementations.

Using the above strategy, we implement the adjoint method for eight primal solvers, five turbulence models, and one radiation model. These adjoint implementations exhibit excellent scalability with up to 10 million cells and 1536 CPU cores. The runtime ratio between adjoint and flow computation ranges from 1.7 to 2.2, and the average error in the adjoint derivatives is less than 0.1%.

Finally, we integrate the implemented adjoint solvers and models into a gradient-based optimization framework MACH and perform four distinct design optimizations that involve aerodynamics, heat transfer, structures, and radiation. We obtain performance improvements for all these four cases: 3.7% drag reduction for the multipoint aerodynamic optimization of the UAV wing, 3.6% drag reduction for the aerodynamic optimization of CRM wing-body-tail configuration, 20.5% reduction in aerodynamic loss and 5.6% increase in heat transfer for the aerothermal optimization of the U-bend cooling passage, and 3.2% torque reduction for the aerostructural optimization of the Rotor 67 blade. The optimization setup for these cases, including the meshes, flow and optimization configurations, and run scripts, are publicly available [55].

DAFoam is available under an open-source license and is a powerful tool for the high-fidelity multidisciplinary design optimization of engineering systems such as aircraft, ground vehicles, marine vessels, and turbomachinery.

Acknowledgments

The computations were done in the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation Grant No. ACI-1548562.

References

[1] Pironneau, O., "On Optimum Profiles in Stokes Flow," *Journal of Fluid Mechanics*, Vol. 59, No. 01, 1973, pp. 117–128. doi:10.1017/S002211207300145X.

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

[3] Nielsen, E. J., and Anderson, W. K., "Aerodynamic Design Optimization on Unstructured Meshes Using the Navier–Stokes Equations," *AIAA Journal*, Vol. 37, No. 11, 1999, pp. 1411–1419. doi:10.2514/2.640.

[4] Jameson, A., "Aerodynamic shape optimization using the adjoint method," Lecture series, Von Karman Institute for Fluid Dynamics, Rode Saint Genèse, Belgium, 2003.

[5] Mavriplis, D. J., "Discrete Adjoint-Based Approach for Optimization Problems on Three-Dimensional Unstructured Meshes," *AIAA Journal*, Vol. 45, No. 4, 2007, pp. 741–750. doi:10.2514/1.22743.

[6] Mader, C. A., Martins, J. R. R. A., Alonso, J. J., and van der Weide, E., "ADjoint: An Approach for the Rapid Development of Discrete Adjoint Solvers," *AIAA Journal*, Vol. 46, No. 4, 2008, pp. 863–873. doi:10.2514/1.29123.

[7] Lyu, Z., Kenway, G. K. W., and Martins, J. R. R. A., "Aerodynamic Shape Optimization Investigations of the Common Research Model Wing Benchmark," *AIAA Journal*, Vol. 53, No. 4, 2015, pp. 968–985. doi:10.2514/1.J053318.

[8] Secco, N. R., Jasa, J. P., Kenway, G. K. W., and Martins, J. R. R. A., "Component-based Geometry Manipulation for Aerodynamic Shape Optimization with Overset Meshes," *AIAA Journal*, Vol. 56, No. 9, 2018, pp. 3667–3679. doi:10.2514/1.J056550.

[9] Bons, N. P., He, X., Mader, C. A., and Martins, J. R. R. A., "Multimodality in Aerodynamic Wing Design Optimization," *AIAA Journal*, Vol. 57, No. 3, 2019, pp. 1004–1018. doi:10.2514/1.J057294.

[10] 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.

[11] Kröger, J., Kühl, N., and Rung, T., "Adjoint volume-of-fluid approaches for the hydrodynamic optimisation of ships," *Ship Technology Research*, Vol. 65, No. 1, 2018, pp. 47–68. doi:10.1080/09377255.2017.1411001.

[12] He, P., Filip, G., Martins, J. R. R. A., and Maki, K. J., "Design Optimization for Self-Propulsion of a Bulk Carrier Hull Using a Discrete Adjoint Method," *Computers & Fluids*, Vol. 192, 2019, p. 104259. doi:10.1016/j.compfluid.2019.104259.

[13] Zhang, P., Lu, J., Song, L., and Feng, Z., "Study on continuous adjoint optimization with turbulence models for aerodynamic performance and heat transfer in turbomachinery cascades," *International Journal of Heat and Mass Transfer*, Vol. 104, 2017, pp. 1069–1082. doi:10.1016/j.ijheatmasstransfer.2016.08.103.

[14] Gkaragkounis, K., Papoutsis-Kiachagias, E., and Giannakoglou, K., "The continuous adjoint method for shape optimization in Conjugate Heat Transfer problems with turbulent incompressible flows," *Applied Thermal Engineering*, Vol. 140, 2018, pp. 351–362. doi:10.1016/j.applthermaleng.2018.05.054.

[15] Leader, M. K., Chin, T. W., and Kennedy, G. J., "High-Resolution Topology Optimization with Stress and Natural Frequency Constraints," *AIAA Journal*, Vol. 57, No. 8, 2019. doi:10.2514/1.J057777.

[16] Anderson, E. M., Bhuiyan, F. H., Mavriplis, D. J., and Fertig, R. S., "Adjoint-Based High-Fidelity Structural Optimization of Wind-Turbine Blade for Load Stress Minimization," *AIAA Journal*, Vol. 57, No. 9, 2019. doi:10.2514/1.J057756.

[17] Martins, J. R. R. A., and Lambe, A. B., "Multidisciplinary Design Optimization: A Survey of Architectures," *AIAA Journal*, Vol. 51, No. 9, 2013, pp. 2049–2075. doi:10.2514/1.J051895.

[18] Martins, J. R. R. A., and Hwang, J. T., "Review and Unification of Methods for Computing Derivatives of Multidisciplinary Computational Models," *AIAA Journal*, Vol. 51, No. 11, 2013, pp. 2582–2599. doi:10.2514/1.J052184.

[19] Hwang, J. T., and Martins, J. R. R. A., "A computational architecture for coupling heterogeneous numerical models and computing coupled derivatives," *ACM Transactions on Mathematical Software*, Vol. 44, No. 4, 2018, p. Article 37. doi:10.1145/3182393.

[20] Gray, J. S., Hwang, J. T., Martins, J. R. R. A., Moore, K. T., and Naylor, B. A., "OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization," *Structural and Multidisciplinary Optimization*, Vol. 59, No. 4, 2019, pp. 1075–1104. doi:10.1007/s00158-019-02211-z.

[21] Martins, J. R. R. A., Alonso, J. J., and Reuther, J. J., "High-Fidelity Aerostructural Design Optimization of a Supersonic Business Jet," *Journal of Aircraft*, Vol. 41, No. 3, 2004, pp. 523–530. doi:10.2514/1.11478.

[22] Kenway, G. K. W., Kennedy, G. J., and Martins, J. R. R. A., "Scalable Parallel Approach for High-Fidelity Steady-State Aeroelastic Analysis and Derivative Computations," *AIAA Journal*, Vol. 52, No. 5, 2014, pp. 935–951. doi:10.2514/1.J052255.

[23] Brooks, T. R., Kenway, G. K. W., and Martins, J. R. R. A., "Benchmark Aerostructural Models for the Study of Transonic Aircraft Wings," *AIAA Journal*, Vol. 56, No. 7, 2018, pp. 2840–2855. doi:10.2514/1.J056603.

[24] Burdette, D. A., and Martins, J. R. R. A., "Design of a Transonic Wing with an Adaptive Morphing Trailing Edge via Aerostructural Optimization," *Aerospace Science and Technology*, Vol. 81, 2018, pp. 192–203. doi:10.1016/j.ast.2018.08.004.

[25] Garg, N., Kenway, G. K. W., Martins, J. R. R. A., and Young, Y. L., "High-fidelity Multipoint Hydrostructural Optimization of a 3-D Hydrofoil," *Journal of Fluids and Structures*, Vol. 71, 2017, pp. 15–39. doi:10.1016/j.jfluidstructs.2017.02.001.

[26] He, P., Martins, J. R. R. A., Mader, C. A., and Maki, K., "Aerothermal Optimization of a Ribbed U-Bend Cooling Channel Using the Adjoint Method," *International Journal of Heat and Mass Transfer*, Vol. 140, 2019, pp. 152–172. doi:10.1016/j.ijheatmasstransfer.2019.05.075.

[27] Gray, J. S., Mader, C. A., Kenway, G. K. W., and Martins, J. R. R. A., "Modeling Boundary Layer Ingestion Using a Coupled Aeropropulsive Analysis," *Journal of Aircraft*, Vol. 55, No. 3, 2018, pp. 1191–1199. doi:10.2514/1.C034601.

[28] Gray, J. S., and Martins, J. R. R. A., "Coupled Aeropropulsive Design Optimization of a Boundary Layer Ingestion Propulsor," *The Aeronautical Journal*, Vol. 123, No. 1259, 2019, pp. 121–137. doi:10.1017/aer.2018.120.

[29] Jonsson, E., Riso, C., Lupp, C. A., Cesnik, C. E. S., Martins, J. R. R. A., and Epureanu, B. I., "Flutter and Post-Flutter Constraints in Aircraft Design Optimization," *Progress in Aerospace Sciences*, Vol. 109, 2019, p. 100537. doi:10.1016/j.paerosci.2019.04.001.

[30] Thomas, J. P., and Dowell, E. H., "Discrete Adjoint Approach for Nonlinear Unsteady Aeroelastic Design Optimization," *AIAA Journal*, 2019, pp. 1–9. doi:10.2514/1.J057504.

[31] Giles, M. B., and Pierce, N. A., "An Introduction to the Adjoint Approach to Design," *Flow, Turbulence and Combustion*, Vol. 65, 2000, pp. 393–415. doi:10.1023/A:1011430410075.

[32] Peter, J. E. V., and Dwight, R. P., "Numerical Sensitivity Analysis for Aerodynamic Optimization: A Survey of Approaches," *Computers and Fluids*, Vol. 39, No. 3, 2010, pp. 373–391. doi:10.1016/j.compfluid.2009.09.013.

[33] 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*, 2019. doi:10.1016/j.paerosci.2019.05.002.

[34] Anderson, W. K., and Venkatakrishnan, V., "Aerodynamic Design Optimization on Unstructured Grids with a Continuous Adjoint Formulation," *Computers and Fluids*, Vol. 28, No. 4, 1999, pp. 443–480. doi:10.1016/S0045-7930(98)00041-3.

[35] Othmer, C., "A continuous adjoint formulation for the computation of topological and surface sensitivities of ducted flows," *International Journal for Numerical Methods in Fluids*, Vol. 58, No. 8, 2008, pp. 861–877. doi:10.1002/fld.1770.

[36] Papoutsis-Kiachagias, E., and Giannakoglou, K., "Continuous adjoint methods for turbulent flows, applied to shape and topology optimization: industrial applications," *Archives of Computational Methods in Engineering*, Vol. 23, No. 2, 2016, pp. 255–299. doi:10.1007/s11831-014-9141-9.

[37] Economon, T. D., Palacios, F., Copeland, S. R., Lukaczyk, T. W., and Alonso, J. J., "SU2: An Open-Source Suite for Multiphysics Simulation and Design," *AIAA Journal*, Vol. 54, No. 3, 2016, pp. 828–846. doi:10.2514/1.j053813.

[38] Nadarajah, S., and Jameson, A., "A Comparison of the Continuous and Discrete Adjoint Approach to Automatic Aerodynamic Optimization," *Proceedings of the 38th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, 2000. doi:10.2514/6.2000-667.

[39] 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.

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

[41] 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.

[42] Nemec, M., Aftosmis, M., Murman, S., and Pulliam, T., "Adjoint formulation for an embedded-boundary Cartesian method," *43rd AIAA Aerospace Sciences Meeting and Exhibit*, 2005, p. 877. doi:10.2514/6.2005-877.

[43] Giles, M. B., Duta, M. C., Müller, J.-D., and Pierce, N. A., "Algorithm Developments for Discrete Adjoint Methods," *AIAA Journal*, Vol. 41, No. 2, 2003, pp. 198–205. doi:10.2514/2.1961.

[44] Hicken, J. E., and Zingg, D. W., "A simplified and flexible variant of GCROT for solving nonsymmetric linear systems," *SIAM Journal on Scientific Computing*, Vol. 32, No. 3, 2010, pp. 1672–1694. doi:10.1137/090754674.

[45] Hicken, J. E., and Zingg, D. W., "Aerodynamic Optimization Algorithm with Integrated Geometry Parameterization and Mesh Movement," *AIAA Journal*, Vol. 48, No. 2, 2010, pp. 400–413. doi:10.2514/1.44033.

[46] Osusky, L., Buckley, H., Reist, T., and Zingg, D. W., "Drag Minimization Based on the Navier—Stokes Equations Using a Newton—Krylov Approach," *AIAA Journal*, Vol. 53, No. 6, 2015, pp. 1555–1577. doi:10.2514/1.J053457.

[47] Mavriplis, D. J., "Time Dependent Adjoint Methods for Single and Multi-disciplinary Problems," Lecture series, chapter 1, Von Karman Institute for Fluid Dynamics, 2015.

[48] Towara, M., "Discrete Adjoint Optimization with OpenFOAM," Ph.D. thesis, RWTH Aachen University, 2018.

[49] Müller, J.-D., Mykhaskiv, O., and Hückelheim, J., "STAMPS: a Finite-Volume Solver Framework for Adjoint Codes Derived with Source-Transformation AD," *AIAA Multidisciplinary Analysis and Optimization Conference*, 2018. doi:10.2514/6.2018-2928.

[50] Albring, T. A., Sagebaum, M., and Gauger, N. R., "Efficient aerodynamic design using the discrete adjoint method in SU2," *17th AIAA/ISSMO multidisciplinary analysis and optimization conference*, 2016, p. 3518. doi:10.2514/6.2016-3518.

[51] Dwight, R. P., and Brezillon, J., "Effect of Approximations of the Discrete Adjoint on Gradient-Based Optimization," *AIAA Journal*, Vol. 44, No. 12, 2006, pp. 3022–3031. doi:10.2514/1.21744.

[52] Mohammadi, B., "A new optimal shape design procedure for inviscid and viscous turbulent flows," *International Journal for Numerical Methods in Fluids*, Vol. 25, No. 2, 1997, pp. 183–203. doi:10.1002/(SICI)1097-0363(19970730)25:2<183::AID-FLD545>3.0.CO;2-U.

[53] Towara, M., and Naumann, U., "A discrete adjoint model for OpenFOAM," *Procedia Computer Science*, Vol. 18, 2013, pp. 429–438. doi:10.1016/j.procs.2013.05.206.

[54] Weller, H. G., Tabor, G., Jasak, H., and Fureby, C., "A tensorial approach to computational continuum mechanics using object-oriented techniques," *Computers in Physics*, Vol. 12, No. 6, 1998, pp. 620–631. doi:10.1063/1.168744.

[55] He, P., Mader, C. A., Martins, J. R. R. A., and Maki, K. J., *DAFoam Multidisciplinary Design Optimization Setup*, Mendeley Data, 2019. doi:10.17632/cg4n68bm9v.

[56] 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, 2012, pp. 273–284. doi:10.1007/s00158-012-0763-y.

[57] Gebremedhin, A. H., Manne, F., and Pothen, A., "What Color Is Your Jacobian? Graph Coloring for Computing Derivatives," *SIAM Review*, Vol. 47, No. 4, 2005, pp. 629–705. doi:10.1137/S0036144504444711.

[58] Burdyshaw, C. E., and Anderson, W. K., "A general and extensible unstructured mesh adjoint method," *Journal of Aerospace Computing, Information, and Communication*, Vol. 2, No. 10, 2005, pp. 401–413. doi:10.2514/1.15932.

[59] Nielsen, E. J., and Kleb, W. L., "Efficient Construction of Discrete Adjoint Operators on Unstructured Grids Using Complex Variables," *AIAA Journal*, Vol. 44, No. 4, 2006, pp. 827–836. doi:10.2514/1.15830.

[60] He, P., Mader, C. A., Martins, J. R. R. A., and Maki, K. J., "An Aerodynamic Design Optimization Framework Using a Discrete Adjoint Approach with OpenFOAM," *Computers & Fluids*, Vol. 168, 2018, pp. 285–303. doi:10.1016/j.compfluid.2018.04.012.

[61] Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., May, D. A., McInnes, L. C., Mills, R. T., Munson, T., Rupp, K., Sanan, P., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H., "PETSc Users Manual," Tech. Rep. ANL-95/11 - Revision 3.10, Argonne National Laboratory, 2018. URL `http://www.mcs.anl.gov/petsc`.

[62] Spalart, P., and Allmaras, S., "A One-Equation Turbulence Model for Aerodynamic Flows," *30th Aerospace Sciences Meeting and Exhibit*, 1992. doi:10.2514/6.1992-439.

[63] Patankar, S. V., and Spalding, D. B., "A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows," *International Journal of Heat and Mass Transfer*, Vol. 15, No. 10, 1972, pp. 1787–1806. doi:10.1016/0017-9310(72)90054-3.

[64] Rhie, C., and Chow, W. L., "Numerical study of the turbulent flow past an airfoil with trailing edge separation," *AIAA Journal*, Vol. 21, No. 11, 1983, pp. 1525–1532. doi:10.2514/3.8284.

[65] Stanzione, D., Barth, B., Gaffney, N., Gaither, K., Hempel, C., Minyard, T., Mehringer, S., Wernert, E., Tufo, H., Panda, D., and Teller, P., "Stampede 2: The Evolution of an XSEDE Supercomputer," *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, ACM, New York, NY, USA, 2017, pp. 15:1–15:8. doi:10.1145/3093338.3093385.

[66] Strazisar, A. J., Wood, J. R., Hathaway, M. D., and Suder, K. L., "Laser anemometer measurements in a transonic axial-flow fan rotor," 1989.

[67] Lambe, A. B., Martins, J. R. R. A., and Kennedy, G. J., "An Evaluation of Constraint Aggregation Strategies for Wing Box Mass Minimization," *Structural and Multidisciplinary Optimization*, Vol. 55, No. 1, 2017, pp. 257–277. doi:10.1007/s00158-016-1495-1.

[68] Ryaciotaki-Boussalis, H., and Guillaume, D., "Computational and Experimental Design of a Fixed-Wing UAV," *Handbook of Unmanned Aerial Vehicles*, Springer, 2015, pp. 109–141. doi:10.1007/978-90-481-9707-1_121.

[69] He, P., Mader, C. A., Martins, J. R. R. A., and Maki, K. J., "An object-oriented framework for rapid discrete adjoint development using OpenFOAM," *AIAA Science and Technology Forum and Exposition*, 2019. doi:10.2514/6.2019-1210.

[70] Chan, W. M., and Steger, J. L., "Enhancements of a three-dimensional hyperbolic grid generation scheme," *Applied Mathematics and Computation*, Vol. 51, No. 2–3, 1992, pp. 181–205. doi:10.1016/0096-3003(92)90073-A.

[71] 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.

[72] 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.

[73] 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.

[74] Perez, R. E., Jansen, P. W., and Martins, J. R. R. A., "pyOpt: A Python-Based Object-Oriented Framework for Nonlinear Constrained Optimization," *Structural and Multidisciplinary Optimization*, Vol. 45, No. 1, 2012, pp. 101–118. doi:10.1007/s00158-011-0666-3.

[75] Chen, S., Lyu, Z., Kenway, G. K. W., and Martins, J. R. R. A., "Aerodynamic Shape Optimization of the Common Research Model Wing-Body-Tail Configuration," *Journal of Aircraft*, Vol. 53, No. 1, 2016, pp. 276–293. doi:10.2514/1.C033328.

[76] Kenway, G. K. W., and Martins, J. R. R. A., "Buffet Onset Constraint Formulation for Aerodynamic Shape Optimization," *AIAA Journal*, Vol. 55, No. 6, 2017, pp. 1930–1947. doi:10.2514/1.J055172.

[77] Van Doormaal, J., and Raithby, G., "Enhancements of the SIMPLE method for predicting incompressible fluid flows," *Numerical heat transfer*, Vol. 7, No. 2, 1984, pp. 147–163. doi:10.1080/01495728408961817.

[78] Vassberg, J. C., DeHaan, M. A., Rivers, S. M., and Wahls, R. A., "Development of a Common Research Model for Applied CFD Validation Studies," 2008. doi:10.2514/6.2008-6919, aIAA 2008-6919.

[79] Modest, M. F., *Radiative heat transfer*, Academic press, 2013.

[80] Verstraete, T., "The VKI U-Bend Optimization Test Case," Tech. rep., 2016. URL `http://aboutflow.sems.qmul.ac.uk/events/munich2016/benchmark/testcase1/`.

[81] Wang, D., He, L., Li, Y., and Wells, R., "Adjoint Aerodynamic Design Optimization for Blades in Multistage Turbomachines—Part II: Validation and Application," *Journal of Turbomachinery*, Vol. 132, No. 2, 2010, p. 021012. doi:10.1115/1.3103928.