A Duality-Preserving Adjoint Method for Segregated Navier–Stokes Solvers

Lean Fang, Ping He*

Department of Aerospace Engineering, Iowa State University, Ames, Iowa, 50011, USA

Abstract

Adjoint methods efficiently compute gradients for systems with many inputs and have been widely used for large-scale gradient-based optimization in fluid mechanics. To ensure optimization's numerical robustness, we need to develop an adjoint solution algorithm that has a similar, if not the same, convergence rate as the primal flow solver at each optimization iteration. This consistent primal-adjoint convergence behavior is also called duality-preserving (DP). Existing DP adjoint methods are mostly for fully coupled Navier–Stokes (NS) solvers with compressible flows. However, few DP adjoints have been proposed for segregated NS solvers, e.g., the semi-implicit method for pressure-linked equations (SIMPLE) algorithm solvers widely used for incompressible flow simulations. This study will fill this gap by deriving a fixed-point segregated adjoint formulation that fully preserves the convergence behavior of primal solvers. We first rewrite the steady-state segregated NS primal solution process into a fully coupled left-preconditioned Richardson format. Then, we transpose the preconditioner matrix to obtain a DP iterative adjoint formulation. In addition to algebraic derivation, we create a new graph representation that significantly streamlines the DP adjoint development for new segregated solvers. We prove that the proposed graph representation is equivalent to primal and adjoint solutions and guarantees duality. We evaluate our proposed algorithm using three cases with increasing complexity: a miniature-sized problem, an airfoil, and a wing. To quantify duality, we compare the eigenvalues between the primal and adjoint solutions and observe excellent agreements. We also evaluate to what extent various numerical settings (e.g., inner iteration tolerances and non-dual preconditioner) impact the eigenvalue distributions. Our proposed adjoint achieves machine-precision accurate gradient computation with competitive speed. Finally, we incorporate our adjoint solver into a large-scale gradient-based optimization framework and demonstrate its capability for wing aerodynamic shape optimization. The proposed fixed-point adjoint approach has the potential to make the adjoint solution more robust and efficient for any segregated NS solvers.

Keywords: Duality-preserving discrete adjoint, segregated Navier–Stokes solvers, fixed-point iteration, gradient-based optimization, eigenvalues

1. Introduction

Computing gradients is fundamental to many numerical analyses and simulations in fluid mechanics, such as design optimization [1–3], gradient-enhanced surrogate models [4–6], inverse problems [7–9], and error estimation [10–12]. There are various options for computing gradients, including analytical, finite-difference, complex-step, and adjoint approaches. The adjoint method's gradient computation cost is independent of the number of inputs, making it efficient for optimization problems with a large number of design variables.

The adjoint method was first introduced to fluid dynamics by Pironneau [13] and then extended to gradient-based optimal design by Jameson [14]. In gradient-based optimization, we must run the adjoint solver for each iteration, so a failed or diverged adjoint may abort the entire optimization. It is

*Corresponding author
Email address: `phe@iastate.edu` (Ping He)

highly desirable to develop an adjoint algorithm that has a similar, if not the same, convergence rate as governing-equation (primal) solvers. This consistent primal-adjoint convergence behavior is also called duality preserving (DP) in the literature [15–20]. To this end, adjoint implementations must be tailored to specific partial differential equations (PDEs) and primal solvers to ensure duality. Developing DP adjoint methods for various primal PDE solvers has been an active research area in the design optimization community.

The adjoint method can be implemented in two ways: continuous and discrete [21–23]. This paper focuses on the discrete adjoint method because of its superior performance in accuracy. The original adjoint equation derived in Jameson [14] is a large-scale linear equation, and the most intuitive implementation is to solve it directly while maintaining duality. For example, Kenway et al. [21] developed a Jacobian-free Krylov adjoint method for an open-source computational fluid dynamics (CFD) solver ADflow [24]. The authors directly solved the adjoint linear equation using a generalized minimal residual (GMRES) method with an incomplete lower-upper factorization (ILU) preconditioner. Because ADflow used a Newton-Krylov method to solve the flow, the Krylov adjoint inherits the convergence of the primal solver. Similar duality was achieved by Hicken and Zingg [25] and Osusky et al. [26], who developed a Krylov adjoint method for a Newton-Krylov CFD solver. Dwight and Brezillon [27] developed a Krylov adjoint equation solution method and implemented it for an unstructured CFD solver, also known as the German Aerospace Center (DLR) TAU-Code [28].

Directly solving the large-scale adjoint equation (often ill-conditioned) requires a large amount of memory. To reduce the memory cost, one can rewrite the adjoint equation into a fixed-point iteration format to drive the adjoint residual to zero. In addition to reducing memory cost, the fixed-point adjoint can preserve duality if the primal solver also uses a fixed-point iteration scheme. For example, Giles et al. [29, 30] derived a fixed-point adjoint method with a carefully formulated left-preconditioned matrix to ensure duality. Nielsen et al. [20] then used a similar approach to develop a fixed-point adjoint solver for FUN3D, a CFD solver developed at the National Aeronautics and Space Administration (NASA). The authors demonstrated that the convergence rates between the primal and adjoint solvers were identical. Mavriplis [19] developed a left-preconditioned DP adjoint for their in-house unstructured CFD solver NSU3D. Xu et al. [31] developed a Jacobian-Trained Krylov-Implicit-Runge-Kutta algorithm to stabilize fixed-point adjoint iterations and implemented it into an open-source solver called STAMPS [32]. Fleischli et al. [16] proposed a fixed-point adjoint formulation for an in-house coupled pressure-based CFD solver. Albring et al. [18] developed a right-preconditioned fixed-point iteration adjoint approach and implemented it into an open-source CFD solver SU2 [33]. Recently, Gomes and Palacios [15] analyzed the impact of various numerical settings (e.g., linear equation convergence tolerance and their initial values) on the fixed-point adjoint for both left- and right-preconditioned methods.

All the above DP adjoint methods were developed for Navier–Stokes (NS) solvers that use a *coupled* solution strategy, i.e., flow variables are updated simultaneously within a time step. However, for incompressible flow, a *segregated* solution strategy is commonly used instead, e.g., the semi-implicit method for pressure-linked equations (SIMPLE) algorithm [34]. In this case, the flow variables are updated sequentially within a time step. Simply applying the above coupled-solver adjoint algorithms to segregated solvers no longer preserves duality. For example, adjoint methods have been implemented for segregated NS solvers in a few previous studies [35, 36], including our previous implementation in DAFoam [37, 38], an open-source discrete adjoint approach for OpenFOAM [39]. Nevertheless, the primal and adjoint convergence was inconsistent in these studies because they used linear solvers to directly solve the adjoint equation, while the primal solvers used a segregated, fixed-point iteration scheme. Akbarzadeh et al. [40] was perhaps the first to develop fixed-point iteration adjoint methods for segregated NS solvers. They rewrote the SIMPLE algorithm as a pressure-Schur-Complement scheme and derived a fixed-point adjoint formulation for laminar NS equations. In our previous work, we developed a preliminary version of DP adjoint methods for steady-state segregated Reynolds-averaged Navier–Stokes (RANS) solvers [41]. In addition to steady-state segregated NS solver, a discrete adjoint approach was proposed by Wang et al. [42] for unsteady flow solvers using the segregated fractional time step method [43]. In the above studies (and all the DP adjoint studies cited in the last paragraph), the duality was only theoretically proved or qualitatively analyzed by comparing the residual convergence rates between the primal and adjoint solutions. However, the residual convergence rate can only reflect the dominant real eigenvalue for an iterative solution process. Without evaluating the entire eigenvalue spectrum, it is unclear: (1) to what extent these adjoint methods truly preserve the duality at the discretized level, and (2) to what degree simplifications and assumptions made in these adjoint algorithms degrade the duality.

To fill this gap, this study derives a fixed-point adjoint formulation for steady-state segregated NS solvers and *quantifies* the duality between the primal and adjoint solvers. We first rewrite the segregated primal solution process into a fully coupled left-preconditioned Richardson format. Then, we transpose the preconditioner matrix to obtain a fixed-point iterative adjoint formulation that fully preserves the primal's convergence. We compare the eigenvalue distributions between the primal and adjoint solutions and quantitatively confirm the duality at the implementation level. We also evaluate to what extent various numerical settings (e.g., inner iteration tolerances and non-dual preconditioner) impact the eigenvalue distributions. In addition to algebraic derivation, we develop a new graph representation of the DP adjoint formulation that can be easily generalized for other primal segregated solvers. We will evaluate the duality, accuracy, speed, and memory usage of the proposed adjoint method using three cases with increasing complexity: a miniature-sized problem, a two-dimensional (2D) airfoil, and a three-dimensional (3D) wing. To demonstrate its robustness, we will implement our adjoint method into DAFoam to conduct wing aerodynamic shape optimization. This paper's new graph representation and eigenvalue analyses are two major steps forward compared with our previous work [41].

Note that this paper focuses on developing a duality-preserving, discrete adjoint method for steady-state, segregated finite-volume NS solvers. The term "duality-preserving" refers to the consistent convergence behavior between the primal and adjoint solvers. In other words, we aim to develop an adjoint solver that has the same convergence rate as the primal solver. Duality preserving is different from another important adjoint property called "consistency". Consistent adjoint methods develop formulations and discretization schemes for the primal and adjoint equations and their boundary conditions such that the gradients computed by the adjoint solver agree with those computed by the primal solver. Many existing adjoint studies have discussed consistency in their formulations and discretization schemes [44–50]. As will be shown in Sec 3, our proposed DP adjoint method exhibits excellent consistency.

The rest of the paper is organized as follows. In Section 2, we describe the mathematical background of the proposed adjoint approach. The adjoint performance evaluation is presented and discussed in Section 3 and we summarize our findings in Section 4.

2. Method

In this section, we first explain the segregated NS solver for primal flow simulations, followed by a general formulation for discrete adjoint gradient computation. Next, we discuss a general DP fixed-point adjoint formulation for coupled NS solvers. Then, we extend DP adjoint formulations for segregated NS solvers, which require a significant amount of algebraic work. To streamline the DP adjoint development, we propose a new graph representation approach and prove that it is equivalent to primal and adjoint solutions and guarantees duality. We also present a non-dual adjoint formulation derived based on intuition instead of rigorous math, which serves as a sanity check for our proposed adjoint.

2.1. Segregated RANS solvers for incompressible turbulent flows

The primal flow simulation is conducted by using OpenFOAM's `simpleFoam` solver, which solves steady turbulent flows governed by the incompressible NS equations:

$$\nabla \cdot \boldsymbol{U} = 0, \tag{1}$$

$$\nabla \cdot (\boldsymbol{U}\boldsymbol{U}) + \frac{1}{\rho}\nabla p - \nabla \cdot \nu_{\text{eff}}(\nabla \boldsymbol{U} + \nabla \boldsymbol{U}^T) = 0, \tag{2}$$

where $\boldsymbol{U} = [u, v, w]$ is the velocity vector, p is the pressure, ρ is the density, $\nu_{\text{eff}} = \nu + \nu_t$ with ν and ν_t being the kinematic and turbulent eddy viscosity, respectively.

To connect the turbulent viscosity to the mean flow variables and close the system, we use the Spalart–Allmaras (SA) turbulence model:

$$\nabla \cdot (\boldsymbol{U}\tilde{\nu}) - \frac{1}{\sigma}\{\nabla \cdot [(\nu + \tilde{\nu})\nabla\tilde{\nu}] + C_{b2}|\nabla\tilde{\nu}|^2\} - \beta C_{b1}\tilde{S}\tilde{\nu} + C_{w1}f_w\left(\frac{\tilde{\nu}}{d}\right)^2 = 0, \tag{3}$$

where $\tilde{\nu}$ is the modified viscosity, which can be related to the turbulent eddy viscosity via

$$\nu_t = \tilde{\nu}\frac{\chi^3}{\chi^3 + C_{v1}^3}, \quad \chi = \frac{\tilde{\nu}}{\nu}. \tag{4}$$

Refer to Spalart and Allmaras [51] for a more detailed description of the terms and parameters in the SA model.

Next, we discuss how the above governing equations are solved numerically using OpenFOAM's implementation of the SIMPLE algorithm (simpleFoam) as an example, although it can be generalized for other solvers. We follow the general steps described in Uroić [52]. Note that this paper focuses on the primal-adjoint duality, so we assume all the computations to be in serial. In the future, we will extend the DP-adjoint implementation to parallel using OpenFOAM's built-in message-passing interface (MPI). We expect it to have reasonably good scalability, similar to our previous Krylov-adjoint implementation [21, 37].

At the discretized level, the first step within an iteration of the SIMPLE algorithm is to calculate an intermediate velocity $\boldsymbol{U}^*$ by solving an under-relaxed momentum equation:

$$\boldsymbol{A}_U\boldsymbol{U}^* = \boldsymbol{b}_U - \boldsymbol{F}_p^{\text{grad}}\boldsymbol{p}^{(n)}, \tag{5}$$

where $\boldsymbol{F}_p^{\text{grad}}$ is the pressure gradient linear operator, i.e., `fvc::grad(p)` in simpleFoam. The $\boldsymbol{A}_U$ matrix is a function of $\boldsymbol{\phi}^{(n)}$ and $\tilde{\boldsymbol{\nu}}^{(n)}$, but it does not depend on $\boldsymbol{U}^{(n)}$. The $\boldsymbol{b}_U$ vector results from the non-orthogonal correction, under-relaxation, and boundary contribution. Note that from this point onward, we deal with discretized equations and variables. Therefore, variables such as $\boldsymbol{p}$ and $\tilde{\boldsymbol{\nu}}$ are treated as discrete vectors.

The second step is to calculate a pseudo-velocity variable $\hat{\boldsymbol{U}}$ and its flux $\hat{\boldsymbol{\phi}}$, i.e. `HbyA` and `phiHbyA` in simpleFoam. Note that $\hat{\boldsymbol{U}}$ has the same the boundary conditions as $\boldsymbol{U}$, and we calculate $\hat{\boldsymbol{U}}$ by performing a standard Jacobi iteration:

$$\hat{\boldsymbol{U}} = \boldsymbol{D}_U^{-1}(\boldsymbol{b}_U - \boldsymbol{Q}_U\boldsymbol{U}^*), \tag{6}$$

where $\boldsymbol{D}_U$ is the diagonal of the under-relaxed matrix $\boldsymbol{A}_U$, and $\boldsymbol{Q}_U$ is the off-diagonal part of $\boldsymbol{A}_U$. The flux $\hat{\boldsymbol{\phi}}$ is then calculated as:

$$\hat{\boldsymbol{\phi}} = \boldsymbol{F}_U^{\text{flux}}\hat{\boldsymbol{U}} + \boldsymbol{c}_U^{\text{bc}}, \tag{7}$$

where $\boldsymbol{F}_U^{\text{flux}}$ is the flux function linear operator, i.e., `fvc::flux(HbyA)`. This equation requires special attention because the left-hand side $\hat{\boldsymbol{\phi}}$ is defined at the mesh face center, and it contains boundary fields. However, the right-hand-side $\hat{\boldsymbol{U}}$ is defined at the mesh cell center, and it does not include boundary fields. To make this equation valid, we need to include the contribution of Neumann boundary conditions, such as `zeroGradient`, in the $\boldsymbol{F}_U^{\text{flux}}$ operator. In addition, we need to include the contribution of Dirichlet boundary conditions, such as `fixedValue`, in the constant vector $\boldsymbol{c}_U^{\text{bc}}$. This is typically done by calling functions to update boundary values, i.e., the `correctBoundaryConditions()` function. We use a similar treatment for other flux functions.

The third step is to calculate an intermediate pressure variable $\boldsymbol{p}^*$ by solving the pressure equation:

$$\boldsymbol{A}_p\boldsymbol{p}^* = \boldsymbol{b}_p + \boldsymbol{F}_\phi^{\text{div}}\hat{\boldsymbol{\phi}}, \tag{8}$$

where $\boldsymbol{F}_\phi^{\text{div}}$ is the divergence function linear operator on a face-stored flux, i.e., `fvc::div(phiHbyA)`.

The fourth step is the calculation of the next iteration $\boldsymbol{\phi}^{(n+1)}$ as:

$$\boldsymbol{\phi}^{(n+1)} = \hat{\boldsymbol{\phi}} - (\boldsymbol{F}_p^{\text{flux}}\boldsymbol{p}^* + \boldsymbol{c}_p^{\text{bc}}), \tag{9}$$

where $\boldsymbol{F}_p^{\text{flux}}$ is the flux function linear operator on a $\boldsymbol{p}$-like field variable, i.e., `pEqn.flux()`.

The fifth step is the explicit under-relaxation on the intermediate pressure $\boldsymbol{p}^*$, which leads to the next iteration pressure $\boldsymbol{p}^{(n+1)}$:

$$\boldsymbol{p}^{(n+1)} = \boldsymbol{p}^{(n)} + \alpha_p(\boldsymbol{p}^* - \boldsymbol{p}^{(n)}), \tag{10}$$

where $\alpha_p = 0.3$ is the under-relaxation factor for $\boldsymbol{p}$.

The sixth step is the correction of the pseudo-velocity $\hat{\boldsymbol{U}}$ by using the under-relaxed pressure $\boldsymbol{p}^{(n+1)}$, which leads to the next iteration velocity $\boldsymbol{U}^{(n+1)}$:

$$\boldsymbol{U}^{(n+1)} = \hat{\boldsymbol{U}} - \boldsymbol{D}_U^{-1}\boldsymbol{F}_p^{\text{grad}}\boldsymbol{p}^{(n+1)}. \tag{11}$$

Once all mean-flow variables are updated, we update the turbulence variable $\tilde{\boldsymbol{\nu}}$ by solving:

$$\boldsymbol{A}_{\tilde{\nu}}\tilde{\boldsymbol{\nu}}^{(n+1)} = \boldsymbol{b}_{\tilde{\nu}}, \tag{12}$$

where $\boldsymbol{b}_{\tilde{\nu}}$ arises from all the explicitly treated terms in the Spalart–Allmaras model, and $\boldsymbol{A}_{\tilde{\nu}}$ depends on $\boldsymbol{U}^{(n+1)}$, $\boldsymbol{\phi}^{(n+1)}$, and $\tilde{\boldsymbol{\nu}}^{(n)}$. The turbulence viscosity $\boldsymbol{\nu}_t$ is then corrected using the latest $\tilde{\boldsymbol{\nu}}^{(n+1)}$, which is strictly for the preparation for the next iteration, and the current primal iteration is now complete.

In outer iterations of the SIMPLE algorithm, we sequentially solve the above equations (5) to (12). Note that some of the above equations require inner iterations by solving a linear system, such as inverting the matrix $\boldsymbol{A}_U$ in Eq. (5). We will discuss the impact of the inner iteration tolerance on the primal-adjoint duality in Sec. 2.5.

A more detailed description of OpenFOAM's SIMPLE implementation can be found in Uroić [52]. The above equations (5, 6, 7, 8, 9, 10, and 11) correspond to Eqs. (2.11, 2.13, 2.14, 2.16, 2.17, 2.18, and 2.19) in [52].

2.2. General discrete adjoint formulation for computing gradients

In general, regardless of the underlying governing equations or particular primal solver, the discretized equations for the primal solver can be written in the residual form, which is a function of both design variable $\boldsymbol{x} \in \mathbb{R}^{n_x}$ and state variable $\boldsymbol{w} \in \mathbb{R}^{n_w}$:

$$\boldsymbol{R}(\boldsymbol{x}, \boldsymbol{w}) = 0. \tag{13}$$

Here the design variable is in a general form, and it can be the volume mesh coordinates, boundary conditions, parameters, or source terms in the above discrete residual equations.

The objective function of interest f depends not only on the design variables, but also on the state variables that are determined by the solution of governing equations, that is:

$$f = f(\boldsymbol{x}, \boldsymbol{w}). \tag{14}$$

To obtain the total derivative $\mathrm{d}f/\,\mathrm{d}\boldsymbol{x}$ for gradient-based optimization, we apply the chain rule as follows:

$$\underbrace{\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}}_{1\times n_x} = \underbrace{\frac{\partial f}{\partial \boldsymbol{x}}}_{1\times n_x} + \underbrace{\frac{\partial f}{\partial \boldsymbol{w}}}_{1\times n_w}\underbrace{\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}\boldsymbol{x}}}_{n_w\times n_x}, \tag{15}$$

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 $\boldsymbol{w}$ and $\boldsymbol{x}$ are implicitly determined by the residual equations $\boldsymbol{R}(\boldsymbol{w}, \boldsymbol{x}) = 0$.

To solve for $\mathrm{d}\boldsymbol{w}/\,\mathrm{d}\boldsymbol{x}$, 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:

$$\frac{\mathrm{d}\boldsymbol{R}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}\boldsymbol{x}} = 0. \tag{16}$$

Rearranging the above equation, we get the linear system

$$\underbrace{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}}_{n_w \times n_w} \cdot \underbrace{\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}\boldsymbol{x}}}_{n_w \times n_x} = - \underbrace{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}}_{n_w \times n_x} . \tag{17}$$

We can then substitute the solution for $\mathrm{d}\boldsymbol{w}/\,\mathrm{d}\boldsymbol{x}$ from Eq. (17) into Eq. (15) to get

$$\underbrace{\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}}_{1 \times n_x} = \underbrace{\frac{\partial f}{\partial \boldsymbol{x}}}_{1 \times n_x} - \overbrace{\underbrace{\frac{\partial f}{\partial \boldsymbol{w}}}_{1 \times n_w} \underbrace{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^{-1}}_{n_w \times n_w}}^{\boldsymbol{\psi}^T} \underbrace{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}}_{n_w \times n_x} . \tag{18}$$

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

$$\underbrace{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^T}_{n_w \times n_w} \cdot \underbrace{\boldsymbol{\psi}}_{n_w \times 1} = \underbrace{\frac{\partial f}{\partial \boldsymbol{w}}^T}_{n_w \times 1}, \tag{19}$$

where $\boldsymbol{\psi}$ is the *adjoint state vector*.

Then, we can compute the total derivative by substituting the adjoint state vector into Eq. (18):

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \frac{\partial f}{\partial \boldsymbol{x}} - \boldsymbol{\psi}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}. \tag{20}$$

If we have multiple objective functions, for each one of them, we need to solve the adjoint equations only once, because the design variable is not explicitly present in Eq. (19). Therefore, its computational cost is independent of the number of design variables but proportional to the number of objective functions. This approach is also known as the *adjoint method* and is advantageous for practical design optimization because we typically have only a few objective functions of interest but may use hundreds or even more design variables (e.g., topology optimization).

2.3. Duality-preserving adjoint formulations for segregated NS solvers

As mentioned above, directly solving the adjoint linear equation (19) accurately and efficiently is no small endeavor. The sheer size of n_w makes it impractical to solve this linear system exactly, therefore an iterative method is preferred. Here we algebraically derive a fixed-pointed DP adjoint formulation for the segregated NS solvers. To this end, we first rewrite the segregated primal solution process into a fully coupled left-preconditioned Richardson format. Then, we transpose the preconditioner matrix to obtain an iterative adjoint formulation.

2.3.1. Left-preconditioned Richardson iteration

Based on the fixed-point DP discrete adjoint framework in [29, 30], we consider a general non-linear fixed-point primal solver written as a left-preconditioned Richardson iteration:

$$\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} + \boldsymbol{X}\boldsymbol{R}(\boldsymbol{w}^{(n)}), \tag{21}$$

where $\boldsymbol{w}^{(n)}$ is the state variable at the nth iteration, $\boldsymbol{R}(\boldsymbol{w}^{(n)})$ is the primal residual for $\boldsymbol{w} = \boldsymbol{w}^{(n)}$, and $\boldsymbol{X}$ is the primal preconditioner. In the context of segregated NS solvers, $\boldsymbol{X}$ may have a sub-block structure. Each sub-block of $\boldsymbol{X}$ can be a long string of matrix operators and may also involve matrix inverses that need to be solved approximately (e.g., inner iterations). Once the primal solver in Eq. (21) becomes sufficiently close to convergence, with Taylor Expansion at the true solution $\boldsymbol{w}^*$, it can be linearized as:

$$\boldsymbol{w}^{(n+1)} - \boldsymbol{w}^* \approx (\boldsymbol{I} + \boldsymbol{X}\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}})(\boldsymbol{w}^{(n)} - \boldsymbol{w}^*), \tag{22}$$

thus the primal iteration matrix is $\boldsymbol{I}+\boldsymbol{X}\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}$, and its eigenvalues govern the asymptotic convergence of the primal solver.

Using a simple transpose of $\boldsymbol{X}$, we can derive a DP fixed-point adjoint solver as:

$$\boldsymbol{\psi}^{(n+1)} = \boldsymbol{\psi}^{(n)} + \boldsymbol{X}^T \hat{\boldsymbol{R}}(\boldsymbol{\psi}^{(n)}), \tag{23}$$

where $\boldsymbol{\psi}^{(n)}$ is the adjoint state vector at the n^{th} iteration, and the adjoint residual $\hat{\boldsymbol{R}}(\boldsymbol{\psi})$ is defined as:

$$\hat{\boldsymbol{R}}(\boldsymbol{\psi}) = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^T \boldsymbol{\psi} - \frac{\partial f}{\partial \boldsymbol{w}}^T. \tag{24}$$

Just like $\boldsymbol{X}$, $\boldsymbol{X}^T$ can also have sub-blocks that involve long strings of operators and matrix inverse. To obtain its iteration matrix, the DP adjoint solver in Eq. (23) and Eq. (24) can be rewritten as:

$$\boldsymbol{\psi}^{(n+1)} - \boldsymbol{\psi}^* = (\boldsymbol{I} + \boldsymbol{X}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^T)(\boldsymbol{\psi}^{(n)} - \boldsymbol{\psi}^*), \tag{25}$$

where $\boldsymbol{\psi}^*$ is the true solution for the adjoint equation. The DP adjoint's iteration matrix is then $\boldsymbol{I} + \boldsymbol{X}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^T$, which shares an identical set of eigenvalues with the primal iteration matrix $\boldsymbol{I}+\boldsymbol{X}\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}$. Therefore, the primal and DP adjoint solvers theoretically share the same asymptotic convergence, albeit as we will see in a later subsection, the approximate nature of the inner iterations leads to a small deviation from the true primal-adjoint duality.

The theoretical primal-adjoint duality in Eq. (23) is advantageous. The adjoint solver runs after the convergence of the primal solver and copies the primal asymptotic convergence behavior, thus a successful primal convergence would also lead to a successful adjoint convergence. However, achieving strict primal-adjoint duality is challenging in practice because many numerical configurations and conditions (e.g., poor primal convergence) may degrade the duality, as will be discussed in Sec. 2.5 and shown in Sec. 3. The degraded duality may make some adjoint-eigenvalues extremely close to one or slightly outside the unit circle, diverging the adjoint fixed-point iteration. To avoid this, researchers have proposed various methods that can stabilize the adjoint equation solution, such as the recursive projection [53, 54], selective frequency damping [55, 56], and BoostConv methods [57, 58]. See a comprehensive review on this topic by Xu et al. [59]. We would like to highlight that the adjoint stabilization method complements the DP-adjoint solution, instead of replacing it. This is because adjoint stabilization methods work best if the eigenvalues between the primal and adjoint solutions are as close as possible, and there are only a small amount of eigenvalues outside the unit circle. In other words, adjoint stabilization methods may not apply to severely non-dual adjoint algorithms where massive adjoint-eigenvalues are outside the unit circle [59]. Developing an effective adjoint stabilization method is outside the scope of this study and will be conducted in our future work.

The adjoint formulation in Eqs. (21) and (23) has been mentioned in Gomes and Palacios [15], and it is called the left-preconditioned fixed-point adjoint formulation. But in that work, the primal solvers are fully coupled, and the primal and adjoint preconditioners (expressed approximately as $\tilde{\boldsymbol{M}}^{-1}$ and $\tilde{\boldsymbol{M}}^{-T}$) can be applied in one go, albeit with inner iterations for approximately inverting matrices. However, for a segregated primal solver such as the SIMPLE algorithm, the relationship between the current iteration and the next iteration is buried beneath the intermediate variables and sequential corrections. Therefore, we need to reformat the segregated primal solver into a left-preconditioned residual-based form outlined in Eq. (21), which requires substantial algebraic work.

2.3.2. Derivation of left preconditioner $\boldsymbol{X}$ for segregated solvers

We demonstrate how to reformat the *discretized*, segregated primal solution process into a left-preconditioned residual-based form. Within a primal iteration, there are no back-and-forth corrections between the mean-flow variables and the turbulence variable, we can treat the two parts separately. The primal preconditioner $\boldsymbol{X}$ in Eq. (21) will have a block-diagonal structure as follows:

$$\begin{bmatrix} \boldsymbol{w}_m^{(n+1)} \\ \boldsymbol{w}_t^{(n+1)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}_m^{(n)} \\ \boldsymbol{w}_t^{(n)} \end{bmatrix} + \begin{bmatrix} \boldsymbol{X}_m(\boldsymbol{w}_m^{(n)}, \boldsymbol{w}_t^{(n)}) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{X}_t(\boldsymbol{w}_m^{(n+1)}, \boldsymbol{w}_t^{(n)}) \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_m^{(n)}(\boldsymbol{w}_m^{(n)}, \boldsymbol{w}_t^{(n)}) \\ \boldsymbol{R}_t^{(n)}(\boldsymbol{w}_m^{(n+1)}, \boldsymbol{w}_t^{(n)}) \end{bmatrix}, \tag{26}$$

where the subscripts m and t denote the mean-flow and turbulence components, respectively. Both $\boldsymbol{X}_m$ and $\boldsymbol{R}_m^{(n)}$ depend on the current iteration state variables $\boldsymbol{w}_m^{(n)}$ and $\boldsymbol{w}_t^{(n)}$, while both $\boldsymbol{X}_t$ and $\boldsymbol{R}_t^{(n)}$ depend on the next iteration mean-flow $\boldsymbol{w}_m^{(n+1)}$ and the current iteration turbulence $\boldsymbol{w}_t^{(n)}$. This is because the mean-flow variables are updated first, and the turbulence variable is updated afterward within a primal iteration. Within the scope of this study, the mean-flow $\boldsymbol{w}_m$ consists of $\boldsymbol{U}$, $\boldsymbol{p}$, and $\boldsymbol{\phi}$, while the turbulence $\boldsymbol{w}_t$ is $\tilde{\boldsymbol{\nu}}$ for the Spalart–Allmaras model.

For the discretized and segregated primal solver described in Sec. 2.1 (simpleFoam), we can define the velocity residual $\boldsymbol{R}_U$ for the current iteration as:

$$\boldsymbol{R}_U^{(n)} = \boldsymbol{R}_U(\boldsymbol{U}^{(n)}, \boldsymbol{p}^{(n)}, \boldsymbol{\phi}^{(n)}, \tilde{\boldsymbol{\nu}}^{(n)}) = \boldsymbol{b}_U - \boldsymbol{F}_p^{\text{grad}} \boldsymbol{p}^{(n)} - \boldsymbol{A}_U \boldsymbol{U}^{(n)}, \tag{27}$$

By combing Eqs. (5) and (27), we can rewrite the intermediate $\boldsymbol{U}^*$ in a residual-based form:

$$\boldsymbol{U}^* = \boldsymbol{U}^{(n)} + \boldsymbol{A}_U^{-1} \boldsymbol{R}_U^{(n)}. \tag{28}$$

By substituting Eq. (27), Eq. (28) and the relationship $\boldsymbol{A}_U = \boldsymbol{D}_U + \boldsymbol{Q}_U$ into Eq. (6), we can rewrite the pseudo-velocity $\hat{\boldsymbol{U}}$ as:

$$\hat{\boldsymbol{U}} = \boldsymbol{U}^{(n)} + \boldsymbol{A}_U^{-1} \boldsymbol{R}_U^{(n)} + \boldsymbol{D}_U^{-1} \boldsymbol{F}_p^{\text{grad}} \boldsymbol{p}^{(n)}. \tag{29}$$

To avoid the need to solve a linear system in our $\boldsymbol{p}$ residual definition, we mimic Eq. (6) and Eq. (7) and introduce $\bar{\boldsymbol{U}}$ and $\bar{\boldsymbol{\phi}}$ as:

$$\bar{\boldsymbol{U}} = \boldsymbol{D}_U^{-1}(\boldsymbol{b}_U - \boldsymbol{Q}_U \boldsymbol{U}^{(n)}), \tag{30}$$

$$\bar{\boldsymbol{\phi}} = \boldsymbol{F}_U^{\text{flux}} \bar{\boldsymbol{U}} + \boldsymbol{c}_U^{\text{bc}}. \tag{31}$$

Then, we define the pressure residual $\boldsymbol{R}_p$ for the current iteration as:

$$\boldsymbol{R}_p^{(n)} = \boldsymbol{R}_p(\boldsymbol{U}^{(n)}, \boldsymbol{p}^{(n)}, \boldsymbol{\phi}^{(n)}, \tilde{\boldsymbol{\nu}}^{(n)}) = \boldsymbol{b}_p + \boldsymbol{F}_\phi^{\text{div}} \bar{\boldsymbol{\phi}} - \boldsymbol{A}_p \boldsymbol{p}^{(n)}. \tag{32}$$

By substituting the respective definitions of $\boldsymbol{R}_p^{(n)}$, $\hat{\boldsymbol{U}}$, $\hat{\boldsymbol{\phi}}$, $\bar{\boldsymbol{U}}$, and $\bar{\boldsymbol{\phi}}$ into Eq. (8), we have the intermediate pressure $\boldsymbol{p}^*$ rewritten in a residual-based form as:

$$\boldsymbol{p}^* = \boldsymbol{p}^{(n)} - \boldsymbol{A}_p^{-1} \boldsymbol{F}_\phi^{\text{div}} \boldsymbol{F}_U^{\text{flux}} \boldsymbol{D}_U^{-1} \boldsymbol{Q}_U \boldsymbol{A}_U^{-1} \boldsymbol{R}_U^{(n)} + \boldsymbol{A}_p^{-1} \boldsymbol{R}_p^{(n)}. \tag{33}$$

We can now define the $\boldsymbol{\phi}$ residual for the current iteration as:

$$\boldsymbol{R}_\phi^{(n)} = \boldsymbol{R}_\phi(\boldsymbol{U}^{(n)}, \boldsymbol{p}^{(n)}, \boldsymbol{\phi}^{(n)}, \tilde{\boldsymbol{\nu}}^{(n)}) = \bar{\boldsymbol{\phi}} - (\boldsymbol{F}_p^{\text{flux}} \boldsymbol{p}^{(n)} + \boldsymbol{c}_p^{\text{bc}}) - \boldsymbol{\phi}^{(n)}. \tag{34}$$

By substituting the respective expressions for $\boldsymbol{R}_\phi^{(n)}$, $\hat{\boldsymbol{\phi}}$, $\bar{\boldsymbol{\phi}}$, and $\boldsymbol{p}^*$ into Eq. (8), we have $\boldsymbol{\phi}^{(n+1)}$ rewritten in a residual-based form as:

$$\boldsymbol{\phi}^{(n+1)} = \boldsymbol{\phi}^{(n)} + (\boldsymbol{F}_p^{\text{flux}} \boldsymbol{A}_p^{-1} \boldsymbol{F}_\phi^{\text{div}} - \boldsymbol{I}) \boldsymbol{F}_U^{\text{flux}} \boldsymbol{D}_U^{-1} \boldsymbol{Q}_U \boldsymbol{A}_U^{-1} \boldsymbol{R}_U^{(n)} - \boldsymbol{F}_p^{\text{flux}} \boldsymbol{A}_p^{-1} \boldsymbol{R}_p^{(n)} + \boldsymbol{R}_\phi^{(n)}. \tag{35}$$

Substituting $\boldsymbol{p}^*$ in Eq. (10) with Eq. (33), we can rewrite $\boldsymbol{p}^{(n+1)}$ in a residual-based form:

$$\boldsymbol{p}^{(n+1)} = \boldsymbol{p}^{(n)} - \alpha_p \boldsymbol{A}_p^{-1} \boldsymbol{F}_\phi^{\text{div}} \boldsymbol{F}_U^{\text{flux}} \boldsymbol{D}_U^{-1} \boldsymbol{Q}_U \boldsymbol{A}_U^{-1} \boldsymbol{R}_U^{(n)} + \alpha_p \boldsymbol{A}_p^{-1} \boldsymbol{R}_p^{(n)}. \tag{36}$$

By substituting the above $\hat{\boldsymbol{U}}$ and $\boldsymbol{p}^{(n+1)}$ with Eq. (29) and Eq. (36), respectively, we have $\boldsymbol{U}^{(n+1)}$ rewritten in a residual-based form:

$$\boldsymbol{U}^{(n+1)} = \boldsymbol{U}^{(n)} + (\boldsymbol{D}_U^{-1} \boldsymbol{F}_p^{\text{grad}} \alpha_p \boldsymbol{A}_p^{-1} \boldsymbol{F}_\phi^{\text{div}} \boldsymbol{F}_U^{\text{flux}} \boldsymbol{D}_U^{-1} \boldsymbol{Q}_U + \boldsymbol{I}) \boldsymbol{A}_U^{-1} \boldsymbol{R}_U^{(n)} - \boldsymbol{D}_U^{-1} \boldsymbol{F}_p^{\text{grad}} \alpha_p \boldsymbol{A}_p^{-1} \boldsymbol{R}_p^{(n)}. \tag{37}$$

Combining Eq. (37), Eq. (36) and Eq. (35) into a matrix form, we have:

$$\begin{bmatrix}\boldsymbol{U}^{(n+1)}\\ \boldsymbol{p}^{(n+1)}\\ \boldsymbol{\phi}^{(n+1)}\end{bmatrix} = \begin{bmatrix}\boldsymbol{U}^{(n)}\\ \boldsymbol{p}^{(n)}\\ \boldsymbol{\phi}^{(n)}\end{bmatrix} + \underbrace{\begin{bmatrix}\boldsymbol{X}_{11} & \boldsymbol{X}_{12} & \boldsymbol{0}\\ \boldsymbol{X}_{21} & \boldsymbol{X}_{22} & \boldsymbol{0}\\ \boldsymbol{X}_{31} & \boldsymbol{X}_{32} & \boldsymbol{I}\end{bmatrix}}_{\boldsymbol{X}_m} \begin{bmatrix}\boldsymbol{R}_U^{(n)}\\ \boldsymbol{R}_p^{(n)}\\ \boldsymbol{R}_\phi^{(n)}\end{bmatrix}, \tag{38}$$

where:

$$\begin{aligned}
\boldsymbol{X}_{11} &= (\boldsymbol{D}_U^{-1}\boldsymbol{F}_p^{\text{grad}}\alpha_p\boldsymbol{A}_p^{-1}\boldsymbol{F}_\phi^{\text{div}}\boldsymbol{F}_U^{\text{flux}}\boldsymbol{D}_U^{-1}\boldsymbol{Q}_U + \boldsymbol{I})\boldsymbol{A}_U^{-1}\\
\boldsymbol{X}_{12} &= -\boldsymbol{D}_U^{-1}\boldsymbol{F}_p^{\text{grad}}\alpha_p\boldsymbol{A}_p^{-1}\\
\boldsymbol{X}_{21} &= -\alpha_p\boldsymbol{A}_p^{-1}\boldsymbol{F}_\phi^{\text{div}}\boldsymbol{F}_U^{\text{flux}}\boldsymbol{D}_U^{-1}\boldsymbol{Q}_U\boldsymbol{A}_U^{-1}\\
\boldsymbol{X}_{22} &= \alpha_p\boldsymbol{A}_p^{-1}\\
\boldsymbol{X}_{31} &= (\boldsymbol{F}_p^{\text{flux}}\boldsymbol{A}_p^{-1}\boldsymbol{F}_\phi^{\text{div}} - \boldsymbol{I})\boldsymbol{F}_U^{\text{flux}}\boldsymbol{D}_U^{-1}\boldsymbol{Q}_U\boldsymbol{A}_U^{-1}\\
\boldsymbol{X}_{32} &= -\boldsymbol{F}_p^{\text{flux}}\boldsymbol{A}_p^{-1}.
\end{aligned} \tag{39}$$

Note that we still solve the seemingly fully-coupled primal equation (38) in a segregated manner.

Then we can define the turbulence residual for the current iteration as:

$$\boldsymbol{R}_{\tilde{\nu}}^{(n)} = \boldsymbol{R}_{\tilde{\nu}}(\boldsymbol{U}^{(n+1)}, \boldsymbol{\phi}^{(n+1)}, \tilde{\boldsymbol{\nu}}^{(n)}) = \boldsymbol{b}_{\tilde{\nu}} - \boldsymbol{A}_{\tilde{\nu}}\tilde{\boldsymbol{\nu}}^{(n)}, \tag{40}$$

and we can rewrite $\tilde{\boldsymbol{\nu}}^{(n+1)}$ in a residual-based form as:

$$\tilde{\boldsymbol{\nu}}^{(n+1)} = \tilde{\boldsymbol{\nu}}^{(n)} + \boldsymbol{A}_{\tilde{\nu}}^{-1}\boldsymbol{R}_{\tilde{\nu}}^{(n)}, \tag{41}$$

thus the turbulence portion of the primal preconditioner in Eq. (26), is simply $\boldsymbol{X}_t = \boldsymbol{A}_{\tilde{\nu}}^{-1}$.

2.3.3. Duality-preserving adjoint as a precondition matrix transpose

Given the block-diagonal structure of the primal preconditioner in Eq. (26), we can simply transpose the preconditioner to obtain the DP adjoint formulation :

$$\begin{bmatrix}\boldsymbol{\psi}_m^{(n+1)}\\ \boldsymbol{\psi}_t^{(n+1)}\end{bmatrix} = \begin{bmatrix}\boldsymbol{\psi}_m^{(n)}\\ \boldsymbol{\psi}_t^{(n)}\end{bmatrix} + \begin{bmatrix}\boldsymbol{X}_m^T & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{X}_t^T\end{bmatrix}\begin{bmatrix}\hat{\boldsymbol{R}}_m^{(n)}\\ \hat{\boldsymbol{R}}_t^{(n)}\end{bmatrix}, \tag{42}$$

where the adjoint residual $\hat{\boldsymbol{R}}^{(n)} = \hat{\boldsymbol{R}}(\boldsymbol{\psi}^{(n)})$ follows the definition in Eq. (24), and the subscripts m and t denote the mean-flow and turbulence components, respectively.

Recall that in the primal formulation in Eq. (26), $\boldsymbol{X}_m$ and $\boldsymbol{R}_m^{(n)}$ depend on $\boldsymbol{w}_m^{(n)}$ and $\boldsymbol{w}_t^{(n)}$ while $\boldsymbol{X}_t$ and $\boldsymbol{R}_t^{(n)}$ depend on $\boldsymbol{w}_m^{(n+1)}$ and $\boldsymbol{w}_t^{(n)}$. The adjoint solver starts after the convergence of the primal solver, which implies $\boldsymbol{w}_m^{(n+1)} = \boldsymbol{w}_m^{(n)}$. Hence we utilize the converged state variables $\boldsymbol{w}_m^{(n)}$ and $\boldsymbol{w}_t^{(n)}$ for the adjoint formulation and forgo $\boldsymbol{w}_m^{(n+1)}$. We will discuss the effects and handling of such discrepancies between the primal and adjoint solvers in detail in Sec. 2.5.

By applying the transpose of $\boldsymbol{X}_m$, we obtain the mean-flow portion of the DP fixed-point adjoint:

$$\begin{bmatrix}\boldsymbol{\psi}_U^{(n+1)}\\ \boldsymbol{\psi}_p^{(n+1)}\\ \boldsymbol{\psi}_\phi^{(n+1)}\end{bmatrix} = \begin{bmatrix}\boldsymbol{\psi}_U^{(n)}\\ \boldsymbol{\psi}_p^{(n)}\\ \boldsymbol{\psi}_\phi^{(n)}\end{bmatrix} + \begin{bmatrix}\boldsymbol{X}_{11}^T & \boldsymbol{X}_{21}^T & \boldsymbol{X}_{31}^T\\ \boldsymbol{X}_{12}^T & \boldsymbol{X}_{22}^T & \boldsymbol{X}_{32}^T\\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\hat{\boldsymbol{R}}_U^{(n)}\\ \hat{\boldsymbol{R}}_p^{(n)}\\ \hat{\boldsymbol{R}}_\phi^{(n)}\end{bmatrix}, \tag{43}$$

and the turbulence portion of the DP adjoint solver is:

$$\boldsymbol{\psi}_{\tilde{\nu}}^{(n+1)} = \boldsymbol{\psi}_{\tilde{\nu}}^{(n)} + \tilde{\boldsymbol{A}}_{\tilde{\nu}}^{-T}\hat{\boldsymbol{R}}_{\tilde{\nu}}^{(n)}, \tag{44}$$

where $\tilde{\boldsymbol{A}}_{\tilde{\nu}}$ is an approximate version of $\boldsymbol{A}_{\tilde{\nu}}$ constructed from the converged $\boldsymbol{U}^{(n)}$, $\boldsymbol{\phi}^{(n)}$ and $\tilde{\boldsymbol{\nu}}^{(n)}$ instead of $\boldsymbol{U}^{(n+1)}$, $\boldsymbol{\phi}^{(n+1)}$ and $\tilde{\boldsymbol{\nu}}^{(n)}$.

We now have a complete and explicitly expressed formulation of a DP fixed-point adjoint solver for the SIMPLE algorithm with Eq. (43) and Eq. (44). The seemingly fully-coupled expression in Eq. (43) can then be broken down into sequential steps similar to those of the original primal solver, albeit this process can be tedious and time-consuming. The sequential steps of a DP adjoint iteration will be shown in the next subsection when we develop an equivalent but more streamlined graph representation approach for DP adjoint formulation and implementation.

2.4. Duality-preserving adjoint as a reverse graph (DARG)

The DP adjoint formulation derived in the last subsection treats the segregated and sequential primal solver and its DP adjoint as seemingly fully coupled. It first packs all sequential steps of a primal iteration into a large and highly complex preconditioner matrix $\boldsymbol{X}$. Then, it unpacks the equally large and complex matrix transpose $\boldsymbol{X}^T$ and breaks it down into sequential steps again for DP adjoint implementation. This pack-then-unpack procedure, while having the merit of explicitly ensuring that the adjoint is duality-preserving, can be quite tedious and time-consuming for the DP adjoint implementation. Thus, in this subsection, we present an equivalent but more streamlined approach called duality-preserving adjoint as a reverse graph (DARG). This approach takes advantage of the segregated and sequential nature and can directly produce a segregated DP adjoint solver from its segregated primal counterpart without the above tedious pack-and-unpack procedure. A similar graph representation approach was proposed for evaluating the direct and adjoint linearized dynamics of compressible flow solvers by De Pando et al. [60], and a rigorous mathematical proof for the validity of such a graph representation method can be found in their Appendix A.

Recall that for each iteration, the original segregated primal solver in Eqs. (5) to (11) sequentially solves for a series of intermediate variables before arriving at the updated state variables. Hence, the primal preconditioner $\boldsymbol{X}$ in Eq. (21) should also be understood as a process of sequentially applying intermediate operators leading to a series of intermediate variables that we have not identified at this point. Once $\boldsymbol{X}$ is properly interpreted as an aforementioned sequential process, we can accordingly construct a graph representation for $\boldsymbol{X}$. Then, the reverse graph of $\boldsymbol{X}$ with transposed intermediate operators is equivalent to the $\boldsymbol{X}^T$ operator. The DP adjoint implementation as sequential steps can be directly read off from the $\boldsymbol{X}^T$ graph. In the following, we will elaborate on the proposed DARG approach.

We first show the prerequisite algebraic work that interprets $\boldsymbol{X}$ as a sequential process. In essence, we identify intermediate variables that the linear mapping $\boldsymbol{X}$ propagates through by performing rudimentary operations. These operations subtract the original primal equations from their reformatted residual definitions. For example, subtracting Eq. (5) with Eq. (27) leads to:

$$\boldsymbol{\beta}_{\text{pri}} = \boldsymbol{A}_U^{-1}\boldsymbol{R}_U^{(n)}, \tag{45}$$

where the newly identified intermediate variable $\boldsymbol{\beta}_{\text{pri}} = \boldsymbol{U}^* - \boldsymbol{U}^{(n)}$ is calculated from the $\boldsymbol{U}$-component of the input $\boldsymbol{R}^{(n)}$. Note that $\boldsymbol{X}$ is essentially a linear mapping from the primal residuals $\boldsymbol{R}^{(n)} = \boldsymbol{R}(\boldsymbol{w}^{(n)})$ to the update on the state variables $\Delta\boldsymbol{w} = \boldsymbol{w}^{(n+1)} - \boldsymbol{w}^{(n)}$. Therefore, we expect all the intermediate variables to be also written as Δ operators. Similarly, subtracting Eq. (6) with Eq. (30) gives us:

$$\boldsymbol{\gamma}_{\text{pri}} = -\boldsymbol{D}_U^{-1}\boldsymbol{Q}_U\boldsymbol{\beta}_{\text{pri}}, \tag{46}$$

where $\boldsymbol{\gamma}_{\text{pri}} = \hat{\boldsymbol{U}} - \bar{\boldsymbol{U}}$ results from the upstream $\boldsymbol{\beta}_{\text{pri}}$. Unlike in the last subsection, here we do not substitute the right-hand side $\boldsymbol{\beta}_{\text{pri}}$ in Eq. (46) with Eq. (45). This is because we want to preserve the relationship between $\boldsymbol{\gamma}_{\text{pri}}$ and $\boldsymbol{\beta}_{\text{pri}}$, which will be reflected in the graph representation of $\boldsymbol{X}$.

We will continue this process of deriving equations in which the left-hand sides are newly identified Δ intermediate variables that can be calculated from the right-hand side upstream variables. By subtracting Eq. (7) with Eq. (31), we have:

$$\boldsymbol{\delta}_{\text{pri}} = \boldsymbol{F}_U^{\text{flux}}\boldsymbol{\gamma}_{\text{pri}}, \tag{47}$$

Table 1: The terms in the primal preconditioner $\boldsymbol{X}$ in Eq. (39) match the paths in the DARG graph (Fig. 1 left) one-to-one.

Terms in matrix	Paths in graph
$\boldsymbol{X}_{11}$	$\boldsymbol{R}_U \rightarrow \boldsymbol{\beta}_{\text{pri}} \rightarrow \boldsymbol{\omega}_{\text{pri}} \rightarrow \Delta\boldsymbol{U}$
	$\boldsymbol{R}_U \rightarrow \boldsymbol{\beta}_{\text{pri}} \rightarrow \boldsymbol{\gamma}_{\text{pri}} \rightarrow \boldsymbol{\delta}_{\text{pri}} \rightarrow \boldsymbol{\varepsilon}_{\text{pri}} \rightarrow \boldsymbol{\eta}_{\text{pri}} \rightarrow \boldsymbol{\kappa}_{\text{pri}} \rightarrow \boldsymbol{\omega}_{\text{pri}} \rightarrow \Delta\boldsymbol{U}$
$\boldsymbol{X}_{12}$	$\boldsymbol{R}_p \rightarrow \boldsymbol{\varepsilon}_{\text{pri}} \rightarrow \boldsymbol{\eta}_{\text{pri}} \rightarrow \boldsymbol{\kappa}_{\text{pri}} \rightarrow \boldsymbol{\omega}_{\text{pri}} \rightarrow \Delta\boldsymbol{U}$
$\boldsymbol{X}_{21}$	$\boldsymbol{R}_U \rightarrow \boldsymbol{\beta}_{\text{pri}} \rightarrow \boldsymbol{\gamma}_{\text{pri}} \rightarrow \boldsymbol{\delta}_{\text{pri}} \rightarrow \boldsymbol{\varepsilon}_{\text{pri}} \rightarrow \boldsymbol{\eta}_{\text{pri}} \rightarrow \boldsymbol{\kappa}_{\text{pri}} \rightarrow \Delta\boldsymbol{p}$
$\boldsymbol{X}_{22}$	$\boldsymbol{R}_p \rightarrow \boldsymbol{\varepsilon}_{\text{pri}} \rightarrow \boldsymbol{\eta}_{\text{pri}} \rightarrow \boldsymbol{\kappa}_{\text{pri}} \rightarrow \Delta\boldsymbol{p}$
$\boldsymbol{X}_{31}$	$\boldsymbol{R}_U \rightarrow \boldsymbol{\beta}_{\text{pri}} \rightarrow \boldsymbol{\gamma}_{\text{pri}} \rightarrow \boldsymbol{\delta}_{\text{pri}} \rightarrow \boldsymbol{\xi}_{\text{pri}} \rightarrow \Delta\boldsymbol{\phi}$
	$\boldsymbol{R}_U \rightarrow \boldsymbol{\beta}_{\text{pri}} \rightarrow \boldsymbol{\gamma}_{\text{pri}} \rightarrow \boldsymbol{\delta}_{\text{pri}} \rightarrow \boldsymbol{\varepsilon}_{\text{pri}} \rightarrow \boldsymbol{\eta}_{\text{pri}} \rightarrow \boldsymbol{\xi}_{\text{pri}} \rightarrow \Delta\boldsymbol{\phi}$
$\boldsymbol{X}_{32}$	$\boldsymbol{R}_p \rightarrow \boldsymbol{\varepsilon}_{\text{pri}} \rightarrow \boldsymbol{\eta}_{\text{pri}} \rightarrow \boldsymbol{\xi}_{\text{pri}} \rightarrow \Delta\boldsymbol{\phi}$
$\boldsymbol{I}$	$\boldsymbol{R}_\phi \rightarrow \boldsymbol{\xi}_{\text{pri}} \rightarrow \Delta\boldsymbol{\phi}$

where $\boldsymbol{\delta}_{\text{pri}} = \hat{\boldsymbol{\phi}} - \bar{\boldsymbol{\phi}}$. Subtracting Eq. (8) with Eq. (32) leads to:

$$\boldsymbol{\eta}_{\text{pri}} = \boldsymbol{A}_p^{-1}\boldsymbol{\varepsilon}_{\text{pri}}, \tag{48}$$

where $\boldsymbol{\eta}_{\text{pri}} = \boldsymbol{p}^* - \boldsymbol{p}^{(n)}$, and the right-hand-side term $\boldsymbol{\varepsilon}_{\text{pri}} = \boldsymbol{F}_\phi^{\text{div}}\boldsymbol{\delta}_{\text{pri}} + \boldsymbol{R}_p^{(n)}$. By subtracting Eq. (9) with Eq. (34), we have:

$$\boldsymbol{\xi}_{\text{pri}} = \boldsymbol{\delta}_{\text{pri}} - \boldsymbol{F}_p^{\text{flux}}\boldsymbol{\eta}_{\text{pri}} + \boldsymbol{R}_\phi^{(n)}, \tag{49}$$

where $\boldsymbol{\xi}_{\text{pri}} = \boldsymbol{\phi}^{(n+1)} - \boldsymbol{\phi}^{(n)}$. Rearranging terms in Eq. (10) leads to:

$$\boldsymbol{\kappa}_{\text{pri}} = \alpha_p\boldsymbol{\eta}_{\text{pri}}, \tag{50}$$

where $\boldsymbol{\kappa}_{\text{pri}} = \boldsymbol{p}^{(n+1)} - \boldsymbol{p}^{(n)}$. By plugging Eq. (5), Eq. (6), and the relationship $\boldsymbol{A}_U = \boldsymbol{D}_U + \boldsymbol{Q}_U$ into Eq. (11), we have:

$$\boldsymbol{\omega}_{\text{pri}} = \boldsymbol{\beta}_{\text{pri}} - \boldsymbol{D}_U^{-1}\boldsymbol{F}_p^{\text{grad}}\boldsymbol{\kappa}_{\text{pri}} \tag{51}$$

where $\boldsymbol{\omega}_{\text{pri}} = \boldsymbol{U}^{(n+1)} - \boldsymbol{U}^{(n)}$. For the turbulence portion of the primal solver, subtracting Eq. (12) with Eq. (40) leads to:

$$\tilde{\boldsymbol{\nu}}^{(n+1)} - \tilde{\boldsymbol{\nu}}^{(n)} = \boldsymbol{A}_{\tilde{\nu}}^{-1}\boldsymbol{R}_{\tilde{\nu}}^{(n)}. \tag{52}$$

When viewed together, Eqs. (45) to (52) reveal how the primal preconditioner $\boldsymbol{X}$ sequentially applies linear operators to propagate intermediate variables within each primal iteration.

We can then construct a graph representation for $\boldsymbol{X}$, as shown in Fig. 1 left. Each vertex in the graph represents a variable encountered during the propagation by $\boldsymbol{X}$. The linear mapping $\boldsymbol{X}$'s input and output are $\boldsymbol{R}^{(n)}$ and $\Delta\boldsymbol{w}$, respectively. The other vertices are the aforementioned intermediate variables. As shown above, we rename the intermediate variables to Greek symbols with the subscript "pri". Each directed edge carries an aforementioned intermediate linear operator; it points from the upstream variable that the intermediate operator applies to, and it points to the resulting downstream variable. Each non-input variable shown in the graph is calculated as the summation of all of its adjacent upstream operator-variable products. Thus, this graph representation is equivalent to the sequential calculations in Eqs. (45) to (52). Furthermore, as shown in Table 1, the directed paths starting from input variables and ending at output variables also match the terms in Eq. (39) one-to-one. This indicates that the graph representation for $\boldsymbol{X}$ is truly equivalent to its matrix form counterpart.

The graph representation for the DP adjoint preconditioner $\boldsymbol{X}^T$ can then be easily constructed as the reverse graph with transposed intermediate operators (see Fig. 1 right). We also rename the DP adjoint's intermediate variables and change the subscript to "adj". Note that while the primal preconditioner's intermediate variables arise from equations with physical significance, their DP adjoint counterparts are purely mathematical.

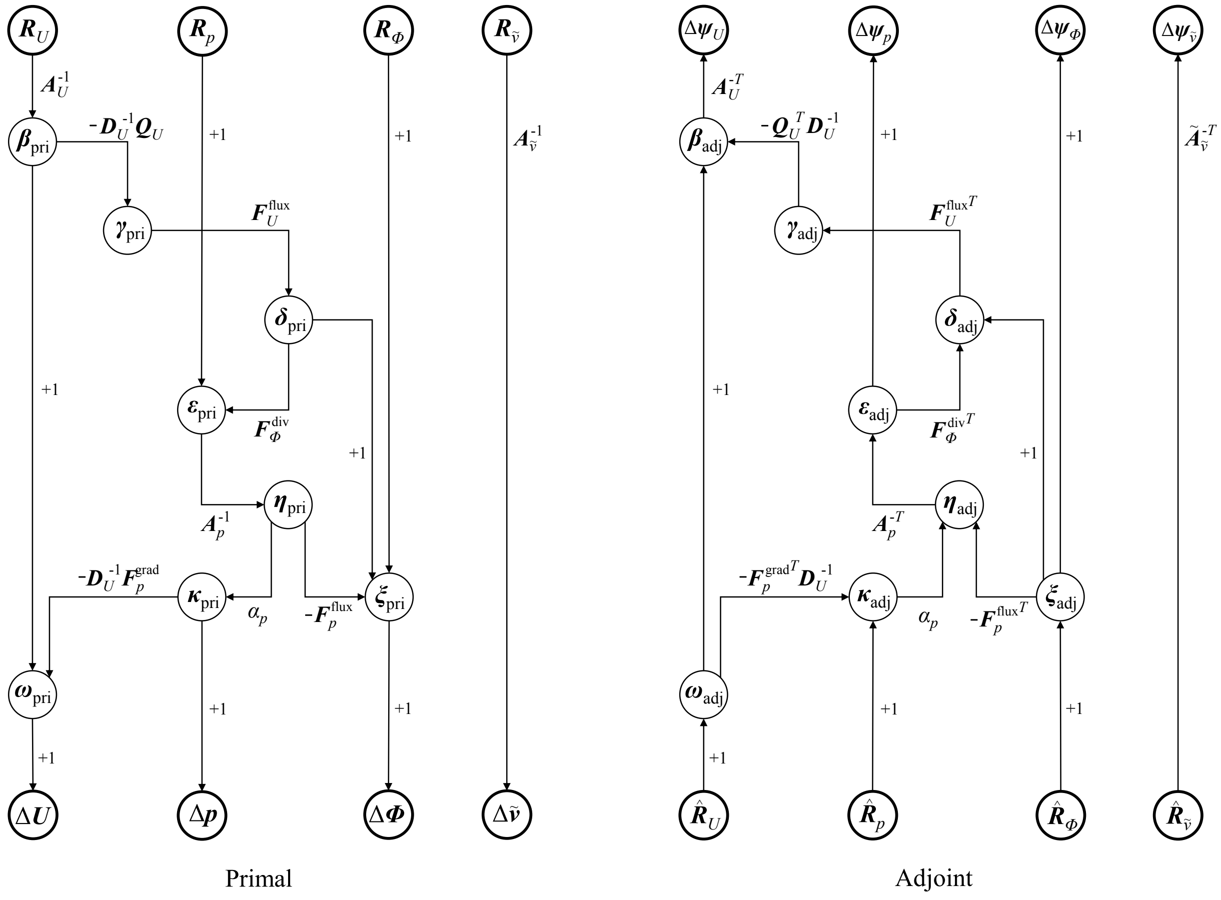

Figure 1: Duality-preserving adjoint as a reverse graph (DARG). Shown on the left is the graph representation for the primal preconditioner $\boldsymbol{X}$ which results from a rudimentary algebraic reformat of the segregated primal solver (execution order: top to bottom). On the right is the graph representation for the DP adjoint preconditioner $\boldsymbol{X}^T$, which is constructed as the reverse graph of $\boldsymbol{X}$ with transposed intermediate operators (execution order: bottom to top). A sequential DP adjoint implementation can be readily read off from the graph for $\boldsymbol{X}^T$.

As shown in Fig. 1 right, a DP adjoint iteration begins with the evaluation of the adjoint residuals $\hat{\boldsymbol{R}}_U^{(n)}$, $\hat{\boldsymbol{R}}_p^{(n)}$, $\hat{\boldsymbol{R}}_\phi^{(n)}$, $\hat{\boldsymbol{R}}_{\tilde{\nu}}^{(n)}$ using reverse-mode automatic differentiation (AD). It then applies the DP adjoint preconditioner $\boldsymbol{X}^T$ which leads to the update on the adjoint state vector $\Delta\boldsymbol{\psi}_U$, $\Delta\boldsymbol{\psi}_p$, $\Delta\boldsymbol{\psi}_\phi$, $\Delta\boldsymbol{\psi}_{\tilde{\nu}}$. Note that the below DARG representations use top-to-bottom and bottom-to-top execution orders of the intermediates vertices for the primal and adjoint solutions, respectively. To implement the DARG representation into actual adjoint code, we use the following sequential steps:

$$
\begin{aligned}
1: &\quad \boldsymbol{\omega}_{\text{adj}} = \hat{\boldsymbol{R}}_U^{(n)}, \\
2: &\quad \boldsymbol{\xi}_{\text{adj}} = \hat{\boldsymbol{R}}_\phi^{(n)}, \\
3: &\quad \Delta\boldsymbol{\psi}_\phi = \boldsymbol{\xi}_{\text{adj}}, \\
4: &\quad \boldsymbol{\kappa}_{\text{adj}} = \hat{\boldsymbol{R}}_p^{(n)} - \boldsymbol{F}_p^{\text{grad}^T} \boldsymbol{D}_U^{-1} \boldsymbol{\omega}_{\text{adj}}, \\
5: &\quad \boldsymbol{\eta}_{\text{adj}} = \alpha_p \boldsymbol{\kappa}_{\text{adj}} - \boldsymbol{F}_p^{\text{flux}^T} \boldsymbol{\xi}_{\text{adj}}, \\
6: &\quad \boldsymbol{\varepsilon}_{\text{adj}} = \boldsymbol{A}_p^{-T} \boldsymbol{\eta}_{\text{adj}}, \\
7: &\quad \Delta\boldsymbol{\psi}_p = \boldsymbol{\varepsilon}_{\text{adj}}, \\
8: &\quad \boldsymbol{\delta}_{\text{adj}} = \boldsymbol{F}_\phi^{\text{div}^T} \boldsymbol{\varepsilon}_{\text{adj}} + \boldsymbol{\xi}_{\text{adj}}, \\
9: &\quad \boldsymbol{\gamma}_{\text{adj}} = \boldsymbol{F}_U^{\text{flux}^T} \boldsymbol{\delta}_{\text{adj}}, \\
10: &\quad \boldsymbol{\beta}_{\text{adj}} = \boldsymbol{\omega}_{\text{adj}} - \boldsymbol{Q}_U^T \boldsymbol{D}_U^{-1} \boldsymbol{\gamma}_{\text{adj}}, \\
11: &\quad \Delta\boldsymbol{\psi}_U = \boldsymbol{A}_U^{-T} \boldsymbol{\beta}_{\text{adj}}, \\
12: &\quad \Delta\boldsymbol{\psi}_{\tilde{\nu}} = \tilde{\boldsymbol{A}}_{\tilde{\nu}}^{-T} \hat{\boldsymbol{R}}_{\tilde{\nu}}^{(n)}.
\end{aligned}
\tag{53}
$$

The transposed operators that arise from the left-hand side coefficients, including $\boldsymbol{Q}_U^T$, $\boldsymbol{A}_p^T$, $\boldsymbol{A}_U^T$, and $\tilde{\boldsymbol{A}}_{\tilde{\nu}}^T$, are constructed by swapping the upper and lower coefficient arrays. The matrix inverse for $\boldsymbol{A}_p^{-T}$, $\boldsymbol{A}_U^{-T}$, and $\tilde{\boldsymbol{A}}_{\tilde{\nu}}^{-T}$ is then handled by OpenFOAM's built-in Gauss–Seidel solver. The rest of the transposed operator-variable products above (e.g. $\boldsymbol{F}_p^{\text{flux}^T} \boldsymbol{\xi}_{\text{adj}}$) are evaluated with reverse-mode AD. Note that the DARG adjoint solution uses exactly the same number of numerical operators as the primal solution. Therefore, we expect that the adjoint's computational speed will be similar to the primal's.

Our DARG method for DP adjoint formulation can be generalized for any segregated NS solvers and turbulence models. It can be summarized as the following major steps:

1. Reformat the segregated primal solver through rudimentary algebraic operations. This interprets the primal preconditioner $\boldsymbol{X}$ as a sequential process of applying intermediate operators.
2. Construct the graph representation for $\boldsymbol{X}$.
3. Reverse the graph for $\boldsymbol{X}$ and transpose the intermediate operators on the edges. This produces the graph representation for the DP adjoint preconditioner $\boldsymbol{X}^T$.
4. Read off the sequential calculations from the graph for $\boldsymbol{X}^T$. This produces the sequential implementaion for the DP adjoint.

2.5. Special considerations for primal-adjoint duality and adjoint accuracy

Although the proposed DP adjoint solver is derived from the transpose of the primal solver, several sources of minor discrepancies in their asymptotic convergence behaviors do exist.

The first source is the non-linearity in the primal solvers. As shown in Eq. (22), we use the linear portion of Taylor's expansion when deriving the corresponding adjoint solver. The effects of higher-order terms may not be negligible if the primal solver has not sufficiently converged, e.g., a relative tolerance of greater than 10^{-5} for the primal residuals. Thus the convergence level of the primal solver affects the asymptotic convergence behavior of the proposed adjoint solver. This effect vanishes to zero as the primal solver converges tightly, e.g., a relative tolerance of less than 10^{-10} for the primal residuals.

The second source of discrepancies comes from the treatment of the turbulence residual and left-hand side matrix. As mentioned earlier, we use $\tilde{\boldsymbol{R}}_{\tilde{\nu}}^{(n)} = \boldsymbol{R}_{\tilde{\nu}}(\boldsymbol{U}^{(n)}, \boldsymbol{\phi}^{(n)}, \tilde{\boldsymbol{\nu}}^{(n)})$ and $\tilde{\boldsymbol{A}}_{\tilde{\nu}}(\boldsymbol{U}^{(n)}, \boldsymbol{\phi}^{(n)}, \tilde{\boldsymbol{\nu}}^{(n)})$ instead of $\boldsymbol{R}_{\tilde{\nu}}^{(n)} = \boldsymbol{R}_{\tilde{\nu}}(\boldsymbol{U}^{(n+1)}, \boldsymbol{\phi}^{(n+1)}, \tilde{\boldsymbol{\nu}}^{(n)})$ and $\boldsymbol{A}_{\tilde{\nu}}(\boldsymbol{U}^{(n+1)}, \boldsymbol{\phi}^{(n+1)}, \tilde{\boldsymbol{\nu}}^{(n)})$ when formulating the adjoint solver. This deviation once again depends on the primal solver's convergence tolerance, and it will also vanish to zero as the primal solver converges tightly.

In this study, we focus on flow problems of academic interest that can converge tightly. Therefore, the two sources of discrepancies mentioned above are inconsequential within the scope of the current work. However, flow problems encountered in industrial applications may not converge tightly and may exhibit limit cycle oscillations (LCO), and in those scenarios, the two aforementioned sources of discrepancies may become significant. The topic of LCO and the related techniques to converge the adjoint solver are beyond the scope of this study, and a more detailed discussion can be found in [31].

The third source of discrepancy is due to the inner linear iterations. Note that Gomes and Palacios [15] discussed the impact of inner iteration on primal-adjoint consistency for left- and right-preconditioned adjoint methods. They suggested that a tight inner iteration is needed for right-preconditioned adjoint methods (e.g., the one used in SU2 [18]) to ensure consistency. However, the choice of inner iteration tolerance has little impact on the consistency for left-preconditioned adjoint methods (e.g., the one proposed in this paper). However, they did not further discuss whether the inner iteration tolerance impacts the primal-adjoint duality. A similar fixed-point left-preconditioned adjoint formulation was proposed by Nielsen et al. [20] for coupled NS solvers; however, the impact of inner iteration tolerance on primal-adjoint duality was not analyzed. Here we demonstrate why the primal-adjoint duality discrepancy is challenging to avoid in practice, and why a particular choice of internal initial guess can avoid or lead to an additional source of error in adjoint accuracy. We also show that with the correct choice of internal initial guess, solving internal linear equations iteratively leads to a perturbation on the left-preconditioner matrix. Therefore, it has no effect on the adjoint accuracy, but it causes a deviation in asymptotic convergence.

We use the Gauss-Seidel method as an example, and the same framework can also work with other iterative methods. Consider a generic, possibly non-linear iteration:

$$\boldsymbol{M}\boldsymbol{w}^{(n+1)} = \underbrace{\boldsymbol{M}\boldsymbol{w}^{(n)} - \boldsymbol{R}(\boldsymbol{w}^{(n)})}_{\boldsymbol{b}}, \tag{54}$$

or equivalently in a left-preconditioned Richardson form:

$$\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} - \boldsymbol{M}^{-1}\boldsymbol{R}(\boldsymbol{w}^{(n)}). \tag{55}$$

We normally treat Eq. (54) as a linear equation $\boldsymbol{M}\boldsymbol{w} = \boldsymbol{b}$, and solve it with an iterative method, e.g. Gauss-Seidel, using $\boldsymbol{v}^{(0)}$ as the initial guess. In general, after N_i internal iterations, the approximate solution is:

$$\boldsymbol{v}^{(N_i)} = \boldsymbol{v}^* + \boldsymbol{G}^{N_i}(\boldsymbol{v}^{(0)} - \boldsymbol{v}^*), \tag{56}$$

where $\boldsymbol{v}^*$ is the true solution, $\boldsymbol{G}$ is the iteration matrix, and in the case of Gauss-Seidel, $\boldsymbol{G} = -(\boldsymbol{D_M} + \boldsymbol{L_M})^{-1}\boldsymbol{U_M}$, and $\boldsymbol{D_M}$, $\boldsymbol{L_M}$ and $\boldsymbol{U_M}$ are the diagonal, lower, and upper breakdown of the left-hand side matrix $\boldsymbol{M}$. Plugging $\boldsymbol{v}^{(0)} = \boldsymbol{w}^{(n)}$, $\boldsymbol{v}^* = \boldsymbol{w}^{(n)} - \boldsymbol{M}^{-1}\boldsymbol{R}(\boldsymbol{w}^{(n)})$, and $\boldsymbol{v}^{(N_i)} = \boldsymbol{w}^{(n+1)}$ into Eq. (56), we get:

$$\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} - (\boldsymbol{I} - \boldsymbol{G}^{N_i})\boldsymbol{M}^{-1}\boldsymbol{R}(\boldsymbol{w}^{(n)}). \tag{57}$$

Thus the internal iterations effectively perturb the true left-preconditioner $\boldsymbol{M}^{-1}$ as $(\boldsymbol{I} - \boldsymbol{G}^{N_i})\boldsymbol{M}^{-1}$. They add a small perturbation to the asymptotic convergence behavior, which only vanishes to zero if the internal iteration number N_i goes to infinity, but they do not introduce any error to the converged solution.

However, for a residual-based solver outlined in Eq. (55), such as the reformatted primal solver and the proposed DP adjoint solver in this study, we first approximately solve $\boldsymbol{M}^{-1}\boldsymbol{R}(\boldsymbol{w}^{(n)})$ as the solution to $\boldsymbol{M}\boldsymbol{x} = -\boldsymbol{R}(\boldsymbol{w}^{(n)})$ then add $\boldsymbol{w}^{(n)}$ to get $\boldsymbol{w}^{(n+1)}$. Therefore, the use of $\boldsymbol{v}^{(0)} = \boldsymbol{w}^{(n)}$ as the initial guess for the internal iterations would lead to:

$$\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} - (\boldsymbol{I} - \boldsymbol{G}^{N_i})\boldsymbol{M}^{-1}\boldsymbol{R}(\boldsymbol{w}^{(n)}) - \boldsymbol{G}^{N_i}\boldsymbol{w}^{(n)}, \tag{58}$$

which includes an additional error term $-\boldsymbol{G}^{N_i}\boldsymbol{w}^{(n)}$ that only vanishes to zero when the internal iteration number N_i goes to infinity, and this is highly undesirable.

Instead of using $\boldsymbol{v}^{(0)} = \boldsymbol{w}^{(n)}$ as the initial guess for the internal iterations, we choose $\boldsymbol{v}^{(0)} = \boldsymbol{0}$, and this leads to the exact same expression in Eq. (57) thus avoiding the additional error term in Eq. (58). To validate the mathematical derivation shown above, we compare the unmodified SIMPLE solver that uses $\boldsymbol{v}^{(0)} = \boldsymbol{w}^{(n)}$ as the internal initial guess to its reformatted residual-based equivalent that uses $\boldsymbol{v}^{(0)} = \mathbf{0}$ as the internal initial guess, and they share an identical convergence history (not shown in this paper).

With the correct choice of internal initial guess, the inner iterations for the primal and adjoint solvers will not lead to an additional source of error. Instead, they will lead to a small perturbation in the asymptotic convergence behavior. It is generally not possible to make the perturbation terms in the primal and adjoint solvers match in a way that also makes the above perturbation consistent, as shown in Eq. (57). In theory, if we allow the internal linear solver to converge tightly, then the deviation caused by the perturbation terms can be negligible, but this would incur a high computational cost with no benefit to accuracy (see a detailed discussion in Sec. 3.4). Thus, empirically, we ask the primal and adjoint's internal iterations to converge loosely to a relative tolerance of 10^{-2}. We will evaluate the impact of inner iteration convergence on the primal and adjoint's eigenvalue spectrum and computational speed in the results section.

2.6. Non-dual adjoint for segregated solvers

To demonstrate the necessity of carefully formulating the DP adjoint preconditioner, we use a non-dual fixed-point adjoint formulation as a sanity check. Instead of rigorously deriving the left-preconditioner matrix to preserve duality, we use the primal equations' left-hand side coefficient matrices to form a block-diagonal matrix as its substitute. Then, we transpose this block-diagonal matrix to get the non-dual adjoint preconditioner (X^T_{nd}), as shown in Eq. (59).

$$\underbrace{\begin{bmatrix} \boldsymbol{A}_U^T & & & \\ & \boldsymbol{A}_p^T & & \\ & & -\boldsymbol{I}_\phi & \\ & & & \tilde{\boldsymbol{A}}_{\tilde{\nu}}^T \end{bmatrix}}_{X^T_{\text{nd}}} \Delta\boldsymbol{\psi} = -\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^T \boldsymbol{\psi} + \frac{\partial f}{\partial \boldsymbol{w}}^T, \tag{59}$$

where $\Delta\boldsymbol{\psi}$ has four components $\Delta\boldsymbol{\psi}_U$, $\Delta\boldsymbol{\psi}_p$, $\Delta\boldsymbol{\psi}_\phi$, and $\Delta\boldsymbol{\psi}_{\tilde{\nu}}$. $\boldsymbol{A}_U$, $\boldsymbol{A}_p^T$, and $\tilde{\boldsymbol{A}}_{\tilde{\nu}}^T$ are the discretization matrices defined in Eqs. (5), (8), and (12). We solve Eq. (59) using the block Gauss-Seidel method. To stabilize the solution, we use an under-relaxation factor of 0.3 for updating the pressure adjoint variable. No under-relaxation is used for the other variables.

This non-dual adjoint formulation is simple and requires much less effort for its implementation than the proposed DP adjoint. A similar adjoint preconditioner formulation was used in a fully coupled NS solver [19]. However, we will demonstrate in the result section that this non-dual adjoint formulation may diverge due to the inconsistent eigenvalue spectrum between the adjoint and segregated primal solvers, although it can converge for simple cases.

3. Results and Discussion

We present four distinct cases to evaluate the proposed DP adjoint solver. We start with a miniature-sized problem that validates the theoretical primal-adjoint duality. We then show a 2D airfoil case and proceed to a more challenging 3D wing case. We evaluate the adjoint's duality, accuracy, speed, and memory usage. The duality is quantified by comparing the eigenvalue distributions between the primal and adjoint, and we use the Arnoldi method (see Appendix for details) to extract the iteration matrix eigenvalues. Finally, we conduct a complete wing aerodynamic shape optimization with the proposed DP adjoint and then compare the optimization result to that obtained from the well-established Krylov-adjoint approach to demonstrate its robustness.

3.1. A miniature-sized flow problem: primal-adjoint duality validation

We first validate the theoretical primal-adjoint duality using a simple 2D channel flow case. The flow is incompressible and turbulent. The channel is square-shaped, the left side is the inlet, the right side is the outlet, and both the top and the bottom are no-slip walls. We use only 3 cells in both the

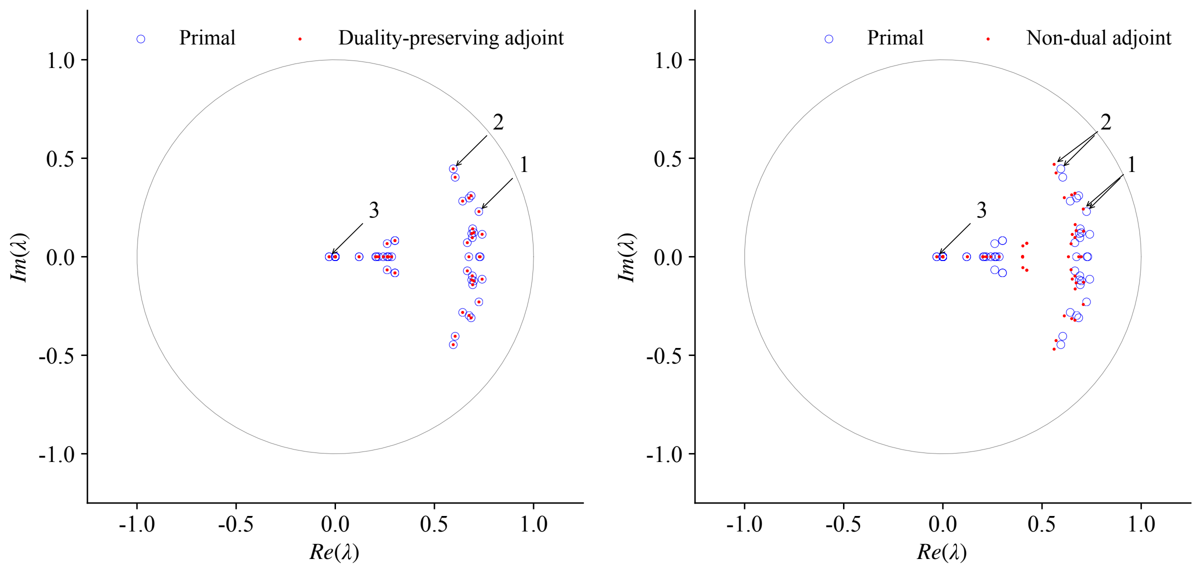

Figure 2: Eigenvalue distributions for the miniature-sized problem. Left: comparison between the primal and duality-preserving adjoint. Right: comparison between the primal and non-dual adjoint. The representative eigenvalues are labeled and shown in Table 2. We observe excellent agreement between the primal and duality-preserving adjoint, while the non-dual adjoint has discernible discrepancies.

Table 2: Three representative eigenvalues for the primal, duality-preserving adjoint, and non-dual adjoint solvers (refer to Fig. 2 for their locations). A complex conjugate pair of eigenvalues count as one eigenvalue. The proposed duality-preserving adjoint matches the primal for almost all eight significant digits (max relative error: <0.01%). However, the non-dual adjoint matches only one significant digit (max relative error: 5.5%).

Label	Primal	Duality-preserving adjoint	Non-dual adjoint
1	$0.72444132 \pm 0.22944547\ i$	$0.72444132 \pm 0.2294454\underline{6}\ i$	$0.7\underline{0808883} \pm 0.2\underline{4211570}\ i$
2	$0.59491033 \pm 0.44624963\ i$	$0.59491033 \pm 0.44624963\ i$	$0.5\underline{6102889} \pm 0.4\underline{6904806}\ i$
3	$-0.03107663 \pm 0.00000000\ i$	$-0.03107663 \pm 0.00000000\ i$	$-0.03\underline{078003} \pm 0.00000000\ i$

streamwise and vertical directions. We let the primal and adjoint inner iterations converge tightly to about 1×10^{-14}. For the outer iteration, the primal and adjoint converge to 1×10^{-10} when the change in the state variables between two iterations is negligible. To measure the inner iteration convergence, we use the L2 norm of linear system residuals, i.e., Eqs. (5), (8), and (12). For the outer iterations, we use the L2 norm of the flow residuals, i.e., Eqs. (27), (32), (34), and (40), to measure convergence.

This miniature-sized problem is desirable for two reasons. First, the inner and outer iterations can easily converge tightly to machine precision, thus effectively removing the small deviations as contributing factors to the primal-adjoint duality (refer to Sec. 2.5). Secondly, the Arnoldi method can retrieve all eigenvalues of the iteration matrix after less than 100 iterations. Therefore, the eigenvalues for this case can be considered as the exact eigenvalues instead of approximate ones.

As shown in Fig. 2 left, the eigenvalue distributions for the primal and DP adjoint solvers visually overlap with no discernible deviation. Furthermore, as seen in Table 2, the highlighted 3 representative eigenvalues also have about 8 matching digits. This excellent agreement is practically all one can hope for with the use of the finite difference in the linearized primal Arnoldi iterations (refer to the Appendix).

The residual convergence plots for the primal and adjoint solvers, shown in Fig. 3, are also consistent with the findings on the iteration matrix eigenvalues. The overall slopes of the convergence trajectories are visually similar between the primal (Fig. 3 top) and DP adjoint (Fig. 3 mid). This indicates that the dominant eigenvalues for the two solvers should have close values of modulus, see Table 2. Additionally,

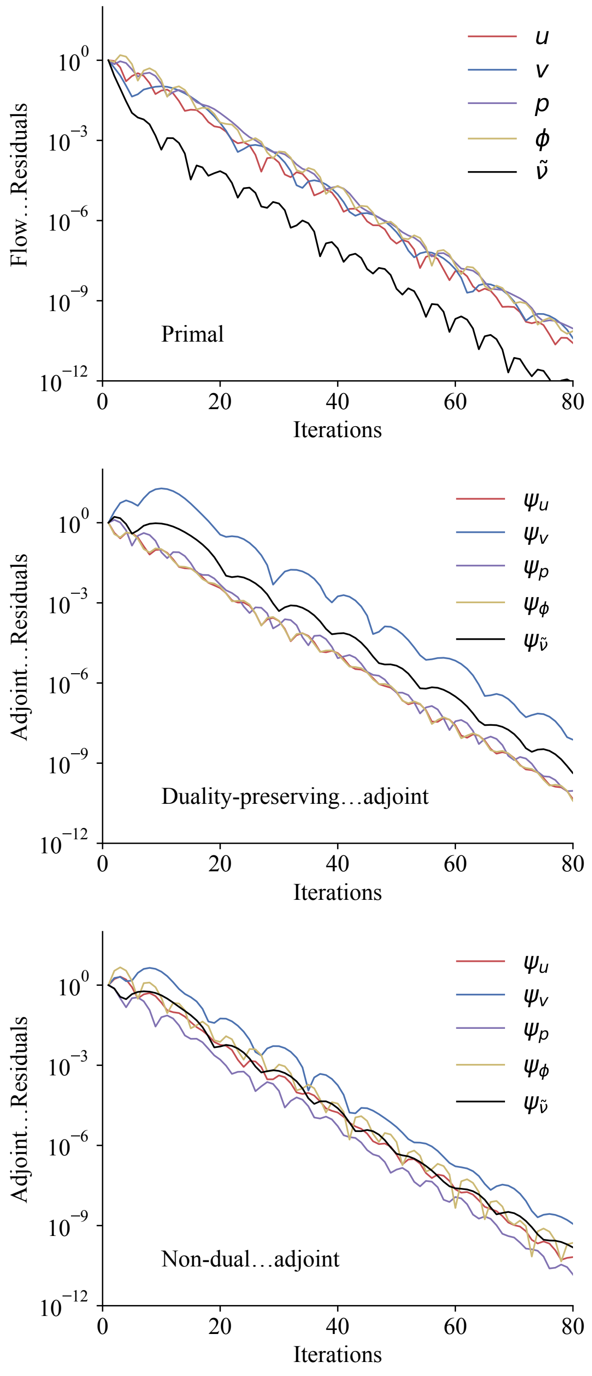

Figure 3: Convergence history of primal and adjoint residuals for the miniature-sized problem. Top: primal. Mid: duality-preserving adjoint. Bot: non-dual adjoint. Tight inner iteration convergence case. Both duality-preserving and non-dual adjoint have a similar convergence rate as the primal solver.

for both primal and DP adjoint, the convergence trajectories always oscillate, which is indicative of a non-real dominant eigenvalue conjugate pair, consistent with that shown in Table 2.

As for the non-dual adjoint, its iteration matrix eigenvalue distribution deviates from that of the primal solver, but it still converges for this problem (Fig. 3 bot). As shown in Fig. 2 right, there are

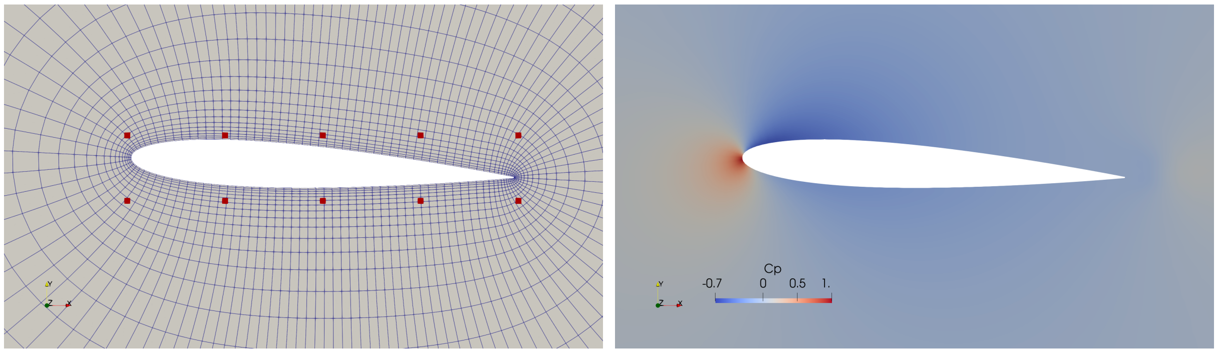

Figure 4: Left: the structured mesh generated by pyHyp and free-form deformation (FFD) points (denoted in red). Right: the pressure distribution for the airfoil case.

discernible discrepancies between the eigenvalue distributions for the primal and non-dual adjoint, but the overall patterns are still visually similar. According to Table 2, the highlighted 3 representative eigenvalues for the non-dual adjoint share only 1 matching digit with their primal counterparts, and the max relative error is 5.5%.

In summary, we demonstrate that the proposed DP adjoint fully preserves the eigenvalue spectrum of the primal solver using a miniature-sized problem. Although the non-dual adjoint's residual convergence rate is similar to the primal solver, its eigenvalues can match only one significant digit.

3.2. NACA0012 airfoil: 2D aerodynamics

We now consider 2D flow over the NACA0012 airfoil, which is of common academic interest but still relatively simple. The flow is incompressible and turbulent. The Reynolds number is 1.0×10^4, and the angle of attack is 3 degrees. Fig. 4 shows the mesh and free-form deformation (FFD) control points for this case. We use the pyHyp package [61] to generate a structured mesh with 4,000 cells. The computational domain extends 20 chord lengths. The average y^+ is 1.3, therefore a wall function is not used. The functions of interest are the drag (C_D) and lift (C_L) coefficients. We use the FFD approach to parameterize the airfoil geometry, and the design variable is the twist angle (γ). The twist is changed by rotating all the FFD control points using the pyGeo package [62].

We quantify the level of primal-adjoint duality by evaluating the iteration matrix eigenvalues obtained through the Arnoldi method. In the previous miniature-sized flow problem, we can compute exact eigenvalues after a small number of Arnoldi iterations. However, in this airfoil case, exact eigenvalue extraction is no longer viable; thus, we run the Arnoldi method for only 1000 iterations which results in 1000 approximate eigenvalues for the airfoil problem. We ensure that the outskirt of the 1000 approximate eigenvalues has well converged by visually and quantitatively comparing them to those obtained from the first 500 Arnoldi iterations. In particular, we check the numerical values of the representative eigenvalues to make sure that they have at least 10 matching digits between the 1000-Arnoldi-iteration and 500-Arnoldi-iteration results. The primal and adjoint outer iterations are allowed to converge tightly to 10^{-10}, thus only the inner iterations remain as a contributing factor to the level of primal-adjoint duality. Therefore, we investigate the asymptotic convergence for two scenarios: 1) the inner iterations converge tightly to 10^{-14}, and 2) the inner iterations converge loosely to a relative tolerance of 10^{-2}. The tight inner convergence scenario is adherent to the theoretical primal-adjoint duality, thus it serves as a benchmark. For the more practical loose inner convergence scenario, some deviation from the true primal-adjoint duality is inevitable.

As shown in Fig. 5 left, the outskirts of iteration matrix eigenvalue distributions visually overlap between the primal and DP adjoint. This can be further confirmed in Table 3, where the representative eigenvalues have 4 to 7 matching digits. This is a high-level agreement given that we use finite differences in the linearized primal Arnoldi iterations. For the loose inner convergence case (Fig. 6 left), the eigenvalue distributions have visual discrepancies between the primal and DP adjoint solvers, as expected. However, the overall patterns are still close between the two, and they also differ slightly from the pattern of the tight

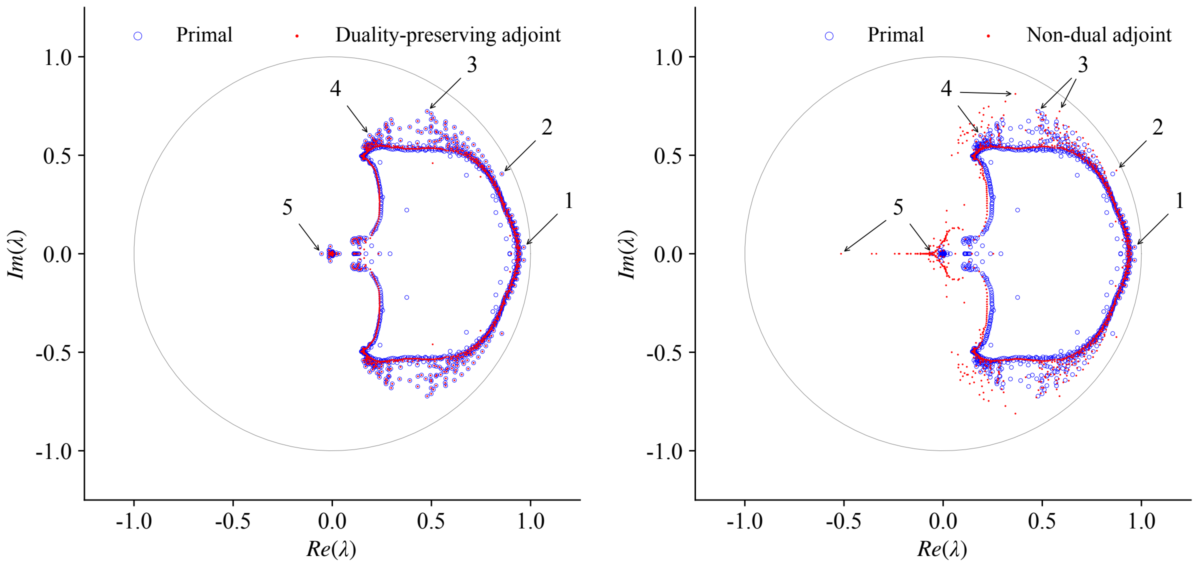

Figure 5: Eigenvalue distributions for the 2D airfoil problem with *tight* inner iteration tolerance. Left: comparison between the primal and duality-preserving adjoint. Right: comparison between the primal and non-dual adjoint. The representative eigenvalues are labeled and shown in Table 3. We observe excellent agreement between the primal and duality-preserving adjoint, while the non-dual adjoint has large discrepancies, especially in the $Re(\lambda) < 0$ region.

Table 3: Five representative eigenvalues for the primal, duality-preserving adjoint, and non-dual adjoint solvers with *tight* inner iteration tolerance (2D airfoil; refer to Fig. 5 for their locations). A complex conjugate pair of eigenvalues count as one eigenvalue. The proposed duality-preserving adjoint matches the primal for 4 to 7 significant digits (max relative error: <0.01%). However, the non-dual adjoint has significantly different eigenvalues (max relative error: >100%).

	Primal	Duality-preserving adjoint	Non-dual adjoint
1	$0.96573819 \pm 0.03282698\ i$	$0.965738\underline{32} \pm 0.032826\underline{74}\ i$	$0.96\underline{494297} \pm 0.03\underline{345844}\ i$
2	$0.85518127 \pm 0.40500358\ i$	$0.8551\underline{7800} \pm 0.40500\underline{457}\ i$	$0.8\underline{7393316} \pm 0.4\underline{2276886}\ i$
3	$0.47847839 \pm 0.72265136\ i$	$0.478478\underline{05} \pm 0.722651\underline{58}\ i$	$0.\underline{58696326} \pm 0.722\underline{75575}\ i$
4	$0.18867190 \pm 0.59983501\ i$	$0.18867\underline{222} \pm 0.59983\underline{552}\ i$	$0.\underline{36456250} \pm 0.\underline{81131675}\ i$
5	$-0.05411324 \pm 0.00000000\ i$	$-0.05411324 \pm 0.00000000\ i$	$-0.\underline{51482627} \pm 0.00000000\ i$

Table 4: Five representative eigenvalues for the primal, duality-preserving adjoint, and non-dual adjoint solvers with *loose* inner iteration tolerance (2D airfoil; refer to Fig. 6 for their locations). A complex conjugate pair of eigenvalues count as one eigenvalue. Both duality-preserving and non-dual adjoint eigenvalues deviate from the primal's, but the duality-preserving adjoint has a better agreement. The max relative errors for the DP- and non-dual adjoint solutions are 18.4% and >100%.

	Primal	Duality-preserving adjoint	Non-dual adjoint
1	$0.96678151 \pm 0.03071720\ i$	$0.966\underline{69765} \pm 0.03\underline{237728}\ i$	$0.96\underline{545747} \pm 0.03\underline{322350}\ i$
2	$0.84966219 \pm 0.41005026\ i$	$0.8\underline{5062451} \pm 0.4\underline{0055657}\ i$	$0.8\underline{5310765} \pm 0.4\underline{3768112}\ i$
3	$0.50949257 \pm 0.73771737\ i$	$0.\underline{66705557} \pm 0.\underline{69006810}\ i$	$0.\underline{66665597} \pm 0.\underline{69208577}\ i$
4	$0.18966167 \pm 0.60305324\ i$	$0.1\underline{9952557} \pm 0.60\underline{278607}\ i$	$0.18\underline{706726} \pm 0.\underline{70946428}\ i$
5	$-0.05471746 \pm 0.00000000\ i$	$-0.054\underline{11095} \pm 0.00000000\ i$	$-0.\underline{51561329} \pm 0.00000000\ i$

inner convergence scenario shown in Fig. 5 left. Furthermore, Table 4 indicates that the representative eigenvalues for the loose inner convergence scenario may still share up to 2 matching digits between the primal and adjoint, but the level of agreement varies. The max relative error for the loose tolerance case

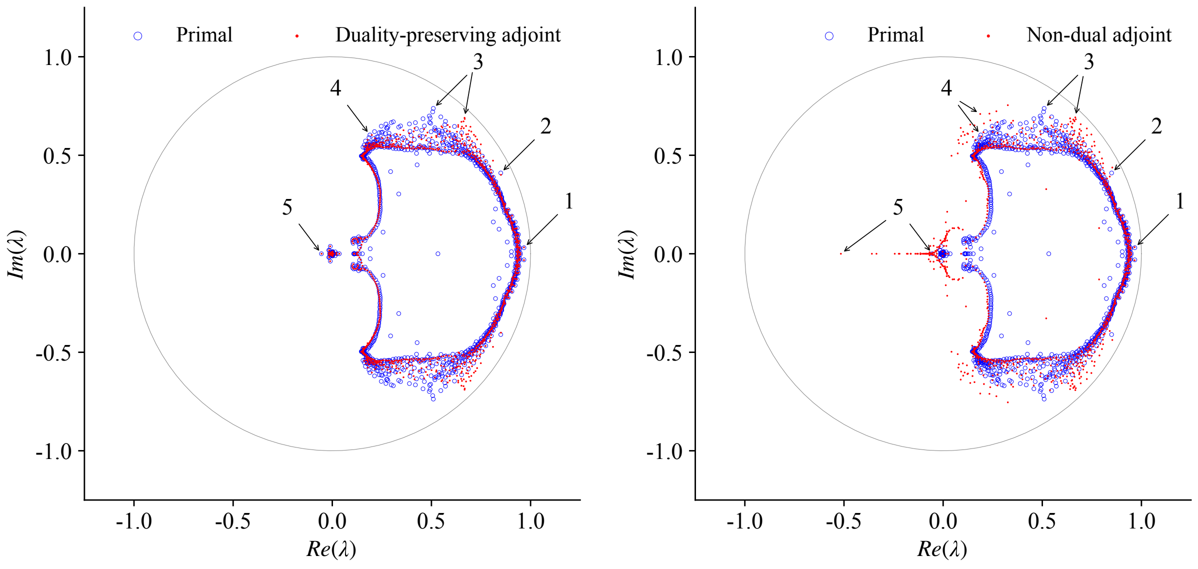

Figure 6: Eigenvalue distributions for the 2D airfoil problem with *loose* inner iteration tolerance. Left: comparison between the primal and duality-preserving adjoint. Right: comparison between the primal and non-dual adjoint. The representative eigenvalues are labeled and shown in Table 4. The duality-preserving adjoint's eigenvalue distribution only slightly differs from the primal, while the non-dual adjoint has large discrepancies, especially in the $Re(\lambda) < 0$ region.

Table 5: Verification of adjoint gradient accuracy (airfoil case). γ is the twist angle. The adjoint derivatives match the reference values (forward-mode AD) by 10 digits. All the gradients have been multiplied by 10^3.

Case	Gradients	Reference	Adjoint
Airfoil	$\mathrm{d}C_D/\mathrm{d}\gamma$	2.918037335493	2.918037335$\underline{604}$
	$\mathrm{d}C_L/\mathrm{d}\gamma$	30.26037720144	30.26037720$\underline{387}$

is 18.4%. As expected, the convergence rates between the primal and DP adjoint (loose inner convergence scenario) are similar, as shown in Fig. 7 top and mid.

The intuitively formulated non-dual adjoint solver still converges successfully for the airfoil case in both the tight and loose inner convergence scenarios (Fig. 7 bot). As shown in Fig. 5 right and Fig. 6 right, the eigenvalue distributions for the non-dual adjoint still share visual similarities with the primal references. However, they have some extra clusters of eigenvalues (e.g., in the $Re(\lambda) < 0$ region), which turn out to be inconsequential because they are well within the unit circle. Table 3 and Table 4 indicate that in both scenarios, the largest positive real eigenvalues for the non-dual adjoint have 3 matching digits with the respective primal references. However, its other representative eigenvalues may hardly match a representative primal eigenvalue. The overall similarity and noticeable difference between the non-dual adjoint and primal eigenvalue distributions are expected because it is meant to somewhat mimic the asymptotic convergence of the primal solver rather than replicating it.

The adjoint gradient computation accuracy is a critical aspect of design optimization. It is highly desirable to have accurate adjoint gradient computation that is consistent with the primal solver, a feature also known as consistent adjoint [46, 63]. Inaccurate or inconsistent gradients may mislead the optimizer in finding new search directions, which results in suboptimal or infeasible designs. We verify the accuracy of the proposed DP adjoint for the NACA0012 airfoil case. We allow both the primal and adjoint solvers to converge tightly to a relative tolerance of 10^{-14}, and the inner convergence for both the primal and adjoint is 10^{-2} (relative tolerance). After the primal convergence, the adjoint solver solves the adjoint equation and calculates an intermediate gradient of the objective function, i.e. the drag coefficient (C_D),

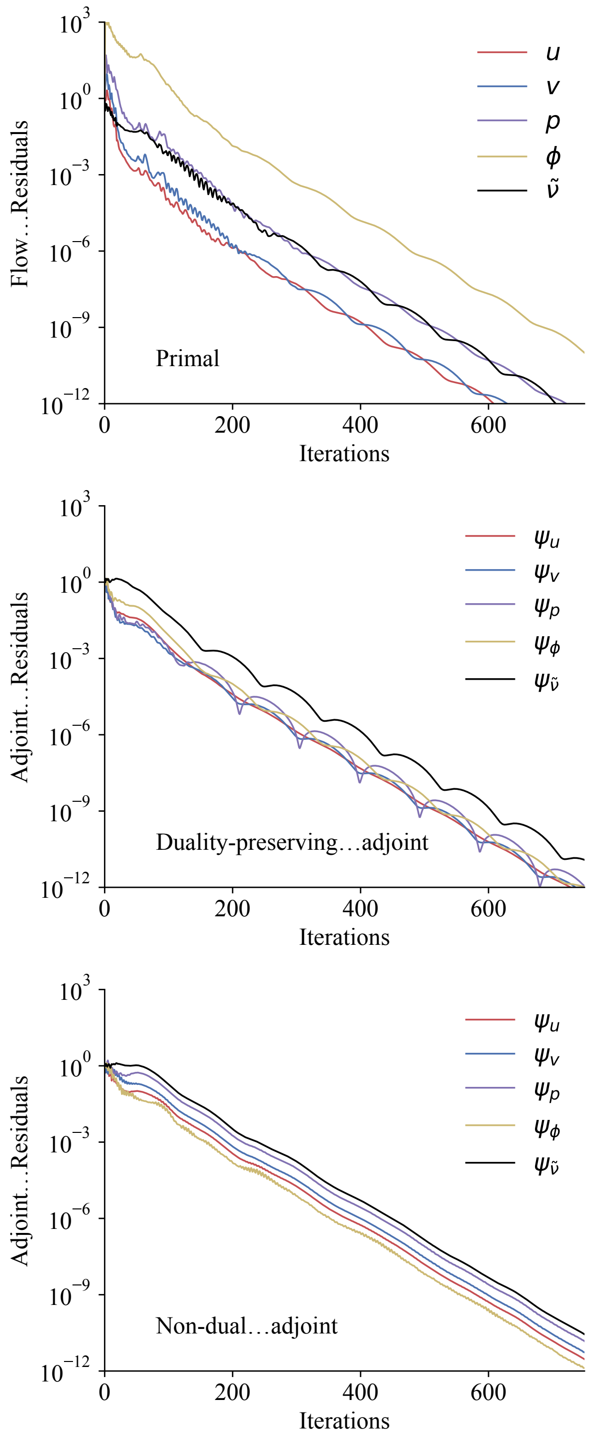

Figure 7: Convergence history of flow (top), duality-preserving adjoint (mid), and non-dual adjoint (bot) residuals for the airfoil case. The inner iteration converges loosely to a relative tolerance of 10^{-2}. Both duality-preserving and non-dual adjoint have a similar convergence rate as the primal solver.

with respect to the vertex coordinates of the volume mesh (x_v). This intermediate gradient ($\mathrm{d}C_D/\,\mathrm{d}x_v$) is then passed to the pyGeo [62] and IDWarp [61] modules to calculate the final gradient with respect to the

Table 6: Comparison of computational and memory cost between the primal and adjoint solvers (airfoil case). The adjoint computation is faster than the flow simulation and its memory cost is about six times higher.

	Speed (s)	Memory (GB)
Primal	16.9	0.06
Adjoint	12.4	0.35
Adjoint/Primal	0.73	5.8

design variable, i.e. the twist angle (γ). pyGeo uses the free-form deformation method to parameterize the design surface geometry and computes the gradient of the surface mesh coordinates with respect to the design variable (e.g., $\mathrm{d}x_s/\,\mathrm{d}\gamma$). IDWarp uses an inverse-distance weighted approach to deform the volume mesh coordinates smoothly and computes the gradient of the volume mesh coordinates with respect to the surface mesh coordinates (i.e., $\mathrm{d}x_v/\,\mathrm{d}x_s$). In other words, the adjoint solver, IDWarp, and pyGeo compute $\mathrm{d}C_D/\,\mathrm{d}x_v$, $\mathrm{d}x_v/\,\mathrm{d}x_s$, and $\mathrm{d}x_s/\,\mathrm{d}\gamma$, respectively, and the final gradient is computed as $\mathrm{d}C_D/\,\mathrm{d}\gamma = [\mathrm{d}C_D/\,\mathrm{d}x_v][\mathrm{d}x_v/\,\mathrm{d}x_s][\mathrm{d}x_s/\,\mathrm{d}\gamma]$. A similar approach is used to compute gradients for C_L. The final gradient is then compared with the forward-mode AD reference value for the verification of adjoint accuracy, and as indicated in Table 5, the adjoint gradients match the reference values by 10 digits. It indicates that our adjoint gradient computation algorithm is fully consistent with the primal solution process. Note that our DP adjoint uses a residual-based formulation and is more accurate than non-residual-based adjoint methods for coupled [18] and segregated [40] NS solvers.

We also evaluate the relative computational cost of the proposed DP adjoint solver. We first allow the primal solver to converge to a relative tolerance of 10^{-10}, then we make the DP adjoint converge to a relative tolerance of 10^{-5}. This is a typical setup in design optimization because the computational cost of tight adjoint convergence significantly outweighs the benefit of more accurate gradients. In other words, decreasing the adjoint convergence from 1×10^{-5} to 1×10^{-10} will practically have the same optimization results; however, the computational cost may increase by one-fold. The inner convergence for both the primal and adjoint is 10^{-2} (relative tolerance). The computation is performed on a local workstation with an Intel Xeon W-1370 CPU, and the primal and adjoint solvers are run in serial. As indicated in Table 6, the runtime and memory usage ratios between the adjoint and primal are 0.73 and 5.8, respectively. As mentioned before, our DP adjoint has the exact same number of numerical operators as the primal. Therefore, the computational speeds are similar between the primal and adjoint solvers. Our adjoint needs more memory than primal for two reasons: (1) we use a discrete adjoint approach which is known to requires a relatively large amount of memory than continuous adjoints, e.g., it needs to compute matrix-vector products using AD [21]. (2) we use an operator overloading tool called CoDiPack [64] to differentiate the primal solver OpenFOAM. The operator overloading is known to require more memory than other AD tools such as source code transformation. Overall, the memory ratio is acceptable given that most of modern computers are equipped with sufficient memory for large-scale optimization.

In summary, we demonstrate our proposed DP adjoint fully preserves the duality for a more complex case (airfoil). Moreover, it computes machine-precision accurate gradients with a competitive speed. The non-dual adjoint still converges for this case, although its eigenvalue distribution exhibits noticeable discrepancies from the primal.

3.3. ADODG3 wing: 3D aerodynamics

Next, we consider a 3D wing case, which is significantly more complex than the previous 2D airfoil case. Figure 8 shows the numerical mesh, FFD control points, and pressure distribution. The geometry is an unswept rectangular wing with a NACA0012 airfoil cross-section and an aspect ratio of 6.12, obtained from the AIAA aerodynamic design optimization discussion group case 3 (ADODG-3). The flow condition is incompressible and turbulent, with the Reynolds number being 6.7×10^4 and the pitch angle being 3 degrees. The computational domain extends 20 chords from the wing surface, and we simulate only half of the wing geometry along with the symmetrical boundary condition. The structured mesh has $100,000$

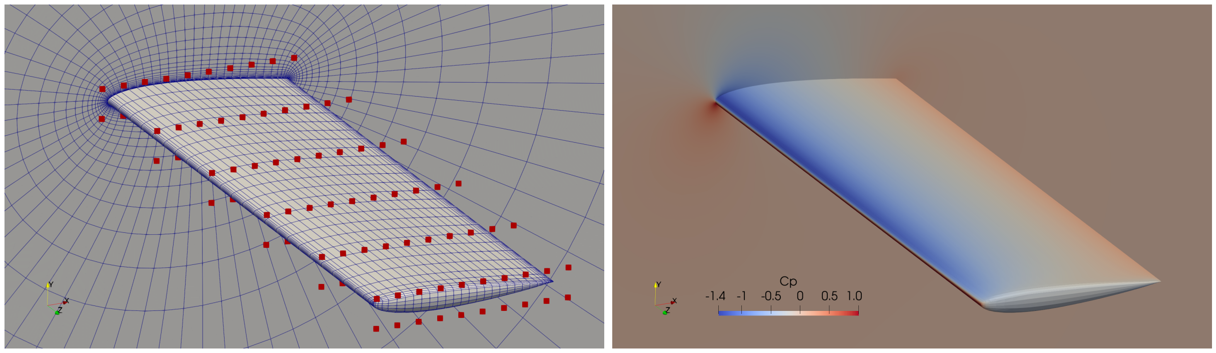

Figure 8: On the left is the structured hexahedral mesh for ADODG-3, a Low-speed rectangular wing. The red dots are the FFD control points that parameterize the wing geometry. On the right is the pressure distribution.

Table 7: Five representative eigenvalues for the primal, duality-preserving adjoint, and non-dual adjoint solvers with *tight* inner iteration tolerance (3D wing; refer to Fig. 9 for their locations). A complex conjugate pair of eigenvalues count as one eigenvalue. The proposed duality-preserving adjoint matches the primal for six to seven significant digits (max relative error: <0.01%). However, the non-dual adjoint has significantly different eigenvalues (max relative error: 55.7%).

Label	Primal	Duality-preserving adjoint	Non-dual adjoint
1	$0.96527383 \pm 0.00000000\ i$	$0.965273\underline{41} \pm 0.00000000\ i$	$\underline{1.1143630} \pm \underline{0.33342687}\ i$
2	$0.90056594 \pm 0.31844184\ i$	$0.90056594 \pm 0.31844184\ i$	$0.\underline{56887082} \pm 0.\underline{73420516}\ i$
3	$0.51604036 \pm 0.68033669\ i$	$0.5160403\underline{4} \pm 0.680336\underline{73}\ i$	$0.\underline{19560927} \pm 0.\underline{87869942}\ i$
4	$0.23404422 \pm 0.69349890\ i$	$0.234044\underline{17} \pm 0.6934989\underline{3}\ i$	$\underline{-0.08291052} \pm 0.\underline{90756559}\ i$
5	$-0.97844034 \pm 0.00000000\ i$	$-0.97844034 \pm 0.00000000\ i$	$-0.9\underline{6855333} \pm 0.00000000\ i$

cells. The average y^+ is 1.2, therefore a wall function is not used. The functions of interest are C_D or C_L, and the design variables are six twist angles along the wing spanwise locations.

Again, we investigate the level of primal-adjoint duality for the DP adjoint by studying the iteration matrix eigenvalues obtained through the Arnoldi method. The numerical setup for the Arnoldi method is the same as the airfoil case. Similar to what we found for the airfoil case, the outskirts of eigenvalue distributions visually overlap (Fig. 9 left) for the wing case with tight inner convergence. In addition, the representative eigenvalues have at least 6 matching digits (Table 7). The loose inner convergence scenario (Fig. 10 left), on the other hand, demonstrates that the eigenvalue distributions exhibit only minor visual discrepancies between the primal and adjoint solvers. Their overall patterns are also visually close to that of the tight inner convergence scenario shown in Fig. 9 left. Furthermore, Table 8 indicates that the representative eigenvalues for the loose inner convergence scenario may still share up to 3 matching digits between the primal and adjoint, but the level of agreement varies. The max relative error is 9.7% for the loose tolerance case. The residual convergence rates are similar between the primal and DP adjoint (Fig. 11 top and mid).

Although the DP adjoint converges well for this case, outlier scenarios may exist when the primal's dominant eigenvalue is extremely close to one or slightly above one. These scenarios can happen when the primal's convergence is poor due to complex flow conditions (e.g., large flow separation). The DP adjoint's eigenvalues (perturbed by loose inner convergence) may lie outside of the unit circle. In practical design optimization, we typically backtrack the line search step and re-run the primal to avoid these extreme cases. Another option is to use stabilization methods [59] for fixed-point iteration, as mentioned above. The DP adjoint stabilization is outside the scope of this paper and will be implemented in our future work.

Unlike in the previous 2D airfoil case, the intuitively formulated non-dual adjoint solver fails to converge for the ADODG-3 wing case in both the tight and loose inner convergence scenarios (Fig. 11 bot). As shown in Fig. 9 right and Fig. 10 right, the eigenvalue distribution for the non-dual adjoint

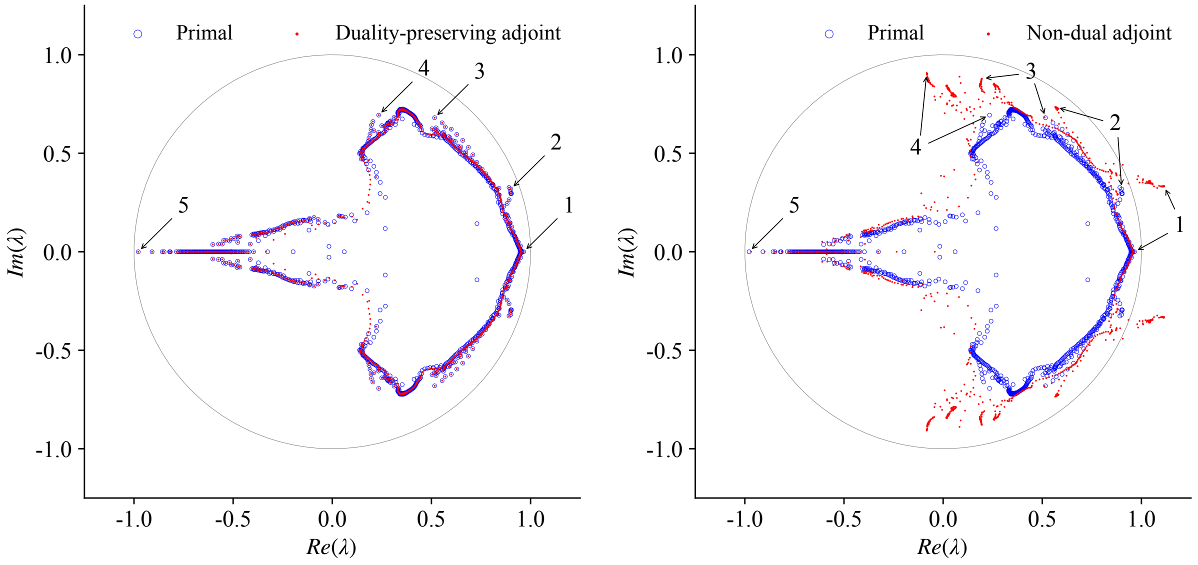

Figure 9: Eigenvalue distributions for the 3D wing problem with *tight* inner iteration tolerance. Left: comparison between the primal and duality-preserving adjoint. Right: comparison between the primal and non-dual adjoint. The representative eigenvalues are labeled and shown in Table 7. We observe excellent agreement between the primal and duality-preserving adjoint, while the non-dual adjoint has eigenvalues outside of the unit circle.

has only a small part of its outskirt matching the respective primal reference. However, it has multiple clusters of large eigenvalues that significantly deviate from the primal reference. Some of those clusters lie outside of the unit circle, which is the root cause of the divergence. Table 7 and Table 8 also qualitatively confirm the graphical observation as there is indeed one representative eigenvalue pair that has a modulus larger than one. Combining the observations for all three cases, we conclude that our proposed adjoint

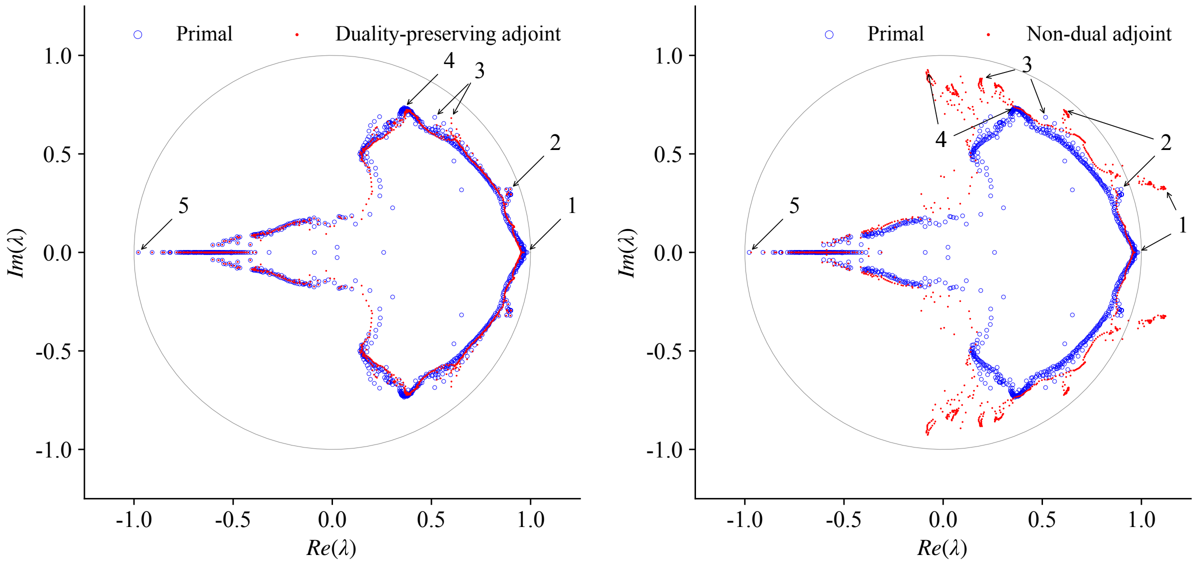

Figure 10: Eigenvalue distributions for the 3D wing problem with *loose* inner iteration tolerance. Left: comparison between the primal and duality-preserving adjoint. Right: comparison between the primal and non-dual adjoint. The representative eigenvalues are labeled and shown in Table 8. The duality-preserving adjoint's eigenvalue distribution only slightly differs from the primal, while the non-dual adjoint has eigenvalues outside of the unit circle.

Table 8: Five representative eigenvalues for the primal, duality-preserving adjoint, and non-dual adjoint solvers with *loose* inner iteration tolerance (3D wing; refer to Fig. 10 for their locations). A complex conjugate pair of eigenvalues count as one eigenvalue. The proposed duality-preserving adjoint's eigenvalues deviate from the primal's (max relative error: 9.7%). However, the non-dual adjoint has significantly different eigenvalues (max relative error: 52.1%).

Label	Primal	Duality-preserving adjoint	Non-dual adjoint
1	$0.98186665 + 0.00000000\,i$	$0.96\underline{614630} + 0.00000000\,i$	$\underline{1.1209257} \pm 0.\underline{32474444}\,i$
2	$0.89721952 \pm 0.32020700\,i$	$0.89\underline{822099} \pm 0.32\underline{127935}\,i$	$0.\underline{60884086} \pm 0.\underline{72370287}\,i$
3	$0.51656571 \pm 0.68592093\,i$	$0.\underline{60003196} \pm 0.68\underline{175256}\,i$	$0.\underline{19577796} \pm 0.\underline{88245046}\,i$
4	$0.36225654 \pm 0.73516467\,i$	$0.3\underline{7263817} \pm 0.7\underline{2283739}\,i$	$\underline{-0.07802425} \pm 0.\underline{92629808}\,i$
5	$-0.97807130 + 0.00000000\,i$	$-0.978\underline{42716} + 0.00000000\,i$	$-0.9\underline{6856232} + 0.00000000\,i$

Table 9: Verification of adjoint gradient accuracy (wing case). γ is the twist angle, and the subscript denotes the spanwise twist section. The adjoint derivatives match the reference values (forward-mode AD) by eight to ten digits. All the gradients have been multiplied by 10^3.

Case	Gradient	Reference	Adjoint
Wing	$\mathrm{d}C_D/\mathrm{d}\gamma_1$	0.20523085809	$0.205230858\underline{18}$
	$\mathrm{d}C_D/\mathrm{d}\gamma_2$	0.29254555514	$0.292545555\underline{26}$
	$\mathrm{d}C_D/\mathrm{d}\gamma_3$	0.32973600328	$0.329736003\underline{44}$
	$\mathrm{d}C_D/\mathrm{d}\gamma_4$	0.32509347007	$0.325093470\underline{15}$
	$\mathrm{d}C_D/\mathrm{d}\gamma_5$	0.26357778101	$0.2635777810\underline{4}$
	$\mathrm{d}C_D/\mathrm{d}\gamma_6$	0.11375193254	0.11375193254
	$\mathrm{d}C_L/\mathrm{d}\gamma_1$	11.2133034230	$11.2133034\underline{308}$
	$\mathrm{d}C_L/\mathrm{d}\gamma_2$	15.6691363611	$15.6691363\underline{760}$
	$\mathrm{d}C_L/\mathrm{d}\gamma_2$	16.8094293381	$16.8094293\underline{518}$
	$\mathrm{d}C_L/\mathrm{d}\gamma_2$	15.3426677083	$15.3426677\underline{152}$
	$\mathrm{d}C_L/\mathrm{d}\gamma_2$	11.1072840966	$11.10728409\underline{98}$
	$\mathrm{d}C_L/\mathrm{d}\gamma_2$	4.25863393980	$4.2586339\underline{4066}$

Table 10: Comparison of computational and memory cost between the primal and adjoint solvers (wing case). The adjoint is only slightly slower than the primal, but its memory cost is much higher.

	Speed (s)	Memory (GB)
Primal	162	0.22
Adjoint	352	4.7
Adjoint/primal	2.2	21

algorithm fully preserves the segregated primal solver's eigenvalue spectrum and converges robustly, while a non-dual adjoint formulation may diverge for complex flow cases.

Finally, we verify the gradient computation accuracy of the proposed DP adjoint. The numerical setup is the same as the one used in the airfoil case. The adjoint gradients match the reference value by 8 to 10 digits for the ADODG-3 wing case, as shown in Table 9. We also evaluate the relative computational and memory cost of the proposed DP adjoint solver, as shown in Table 10. The runtime and memory ratio between the adjoint and primal are 2.2 and 21, respectively.

3.4. Balance between the duality and speed

As shown in the previous two subsections, our proposed adjoint algorithm preserves the duality well when a tight inner iteration tolerance (10^{-14}) is used. However, the primal-adjoint duality degrades

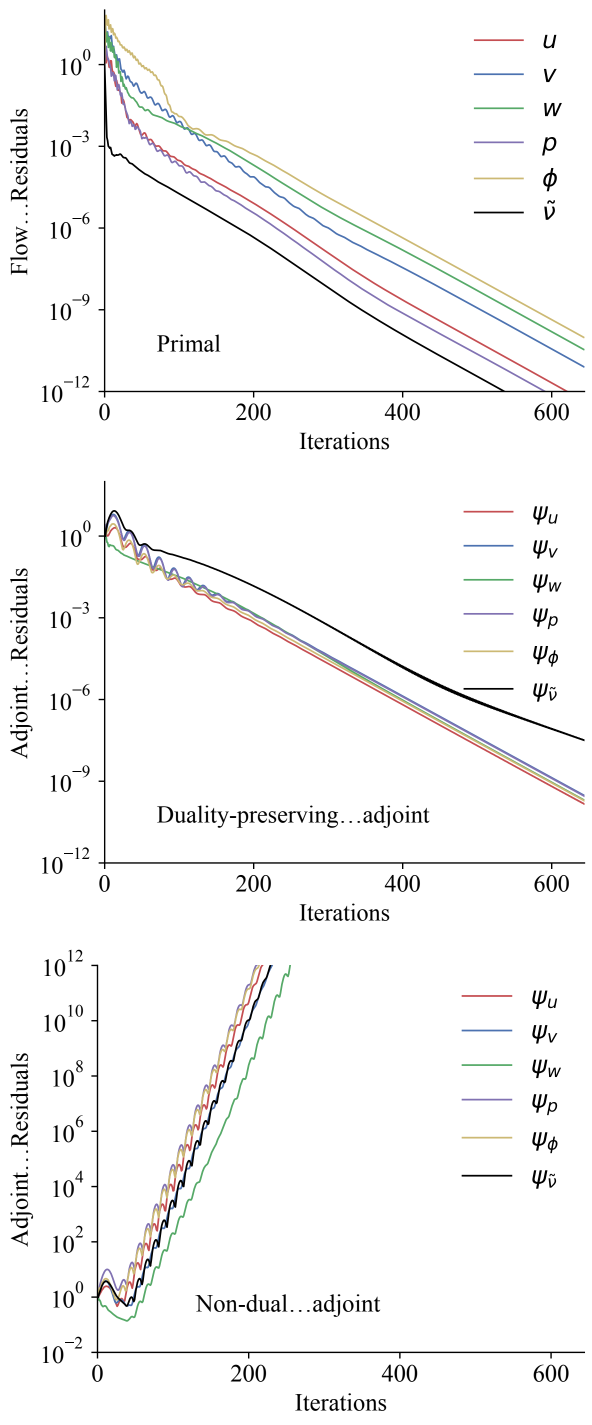

Figure 11: Convergence history of primal and adjoint residuals for the 3D wing case. Top: primal. Mid: duality-preserving adjoint. Bot: non-dual adjoint. Loose inner iteration convergence case. The duality-preserving has a similar convergence rate as the primal solver. However, the non-dual adjoint diverges.

if a loose inner iteration tolerance (10^{-2}) is used instead. Although it provides benefits in duality, a tight inner iteration tolerance is not necessary because it will increase the computational cost for both primal and adjoint solvers. In practice, one should always use a loose inner tolerance as the default for

Table 11: Primal and adjoint runtime with various relative tolerances for inner iterations.

Case	Tolerance	Runtime, s (Primal)	Extra cost (Primal)	Runtime, s (Adjoint)	Extra cost (Adjoint)
Airfoil	10^{-2}	16.9	–	12.4	–
	10^{-4}	18.0	6.5%	13.3	7.3%
	10^{-6}	18.7	10.7%	15.4	24.2%
	10^{-8}	19.0	12.4%	18.4	48.4%
Wing	10^{-2}	162	–	352	–
	10^{-4}	237	46.3%	548	55.7%
	10^{-6}	285	75.9%	857	143.5%
	10^{-8}	313	93.2%	1190	237.5%

the DP-adjoint solution. However, it is possible that the primal solver converges but the adjoint solver diverges, due to the duality discrepancy caused by the loose inner tolerance. In this case, one needs to trade the adjoint speed for duality by tightening the inner tolerance, such that the adjoint solver can better preserve the convergence behavior of the primal solver. This section discusses the balance between duality and speed of the proposed adjoint algorithm.

To quantify the balance, we ran primal and adjoint simulations with various relative tolerances for inner iterations (10^{-2} to 10^{-8}). The absolute tolerance of inner iterations is set to 10^{-10} for all the cases. In addition, we require the outer iteration to converge to 10^{-10} for all simulations. We then record the runtime for both the airfoil and wing cases, and the data are shown in Table 11. As expected, both the primal and adjoint runtime increases when the inner iteration tolerance is tightened. For the airfoil case, the extra computational cost to maintain duality is relatively low. For example, the inner tolerance of 10^{-4} requires only 6.5% more runtime than the 10^{-2} case. However, for the wing case, the extra computational cost for tightening the inner iteration tolerance is significantly higher. 55.7% more runtime is needed when tightening the inner iteration tolerance from 10^{-2} to 10^{-4}. This high extra computational cost is mainly caused by the rapid increase in the number of inner iterations for the pressure linear equation solver. In this study, we use OpenFOAM's built-in Gauss-Seidel (GS) coupled with the generalized geometric-algebraic multi-grid (GAMG) method as the linear equation solver for the pressure and pressure-adjoint variables. For the wing case, the GAMG+GS solver needs 20, 130, and 250 iterations to converge the pressure equation residual by 2, 4, and 6 orders of magnitudes, respectively. We speculate this issue is not shown in the airfoil case because of the smaller mesh size and 2D nature of the flow. To alleviate the above issue, we can use a Krylov method to solve the pressure equation instead. Krylov solvers converge quadratically and are expected to perform better when a tight inner iteration tolerance is required; however, this topic is beyond the scope of this study.

Another observation from Table 11 is that the extra computational cost for the primal solver is much lower than that for the adjoint solver when tightening the inner iteration tolerance. This is mainly because the primal inner solver uses the flow solutions from the previous outer iteration as the initial guess. Therefore, the initial residual for the primal inner iteration decreases as the simulation goes on, and the inner iterations reach the absolute tolerance threshold before reaching the relative tolerance threshold, especially near the end of the simulations. Our proposed DP-adjoint solver cannot benefit from the above because it always uses a zero initial guess for the inner iteration (see the detailed discussion for this choice in Sec. 2.5). Therefore, the adjoint inner iteration needs to always satisfy the prescribed relative tolerance.

3.5. Incorporating the DP adjoint into gradient-based wing aerodynamic optimization

To further evaluate the proposed DP adjoint's numerical robustness, we incorporate it into a gradient-based optimization framework and conduct wing aerodynamic optimization. The goal is to evaluate whether the DP adjoint can robustly converge when the geometry and flow conditions change during the optimization.

Table 12: Formulation of the wing optimization problem. We use 126 design variables and 80 constraints.

	Function/Variable	Description	Quantity
Minimize	C_D	Drag coefficient of the wing	
with respect to	$0 \leq \alpha \leq 10$	Angle of attack	1
	$-10 \leq \gamma \leq 10$	Twists of each non-root FFD sections	5
	$-1.0 \leq \Delta y \leq 1.0$	Vertical displacements of FFD points	120
		Total Design Variables	**126**
subject to	$C_L = 0.375$	Lift constraint	1
	$V_{\text{bl}} \leq V$	Minimum volume constraint	1
	$0.5 \cdot t_{\text{bl}} \leq t \leq 3 \cdot t_{\text{bl}}$	Minimum thickness constraint	60
	$\Delta y_{\text{LE, upper}} = -\Delta y_{\text{LE, lower}}$	Fixed leading edge constraint	6
	$\Delta y_{\text{TE, upper}} = -\Delta y_{\text{TE, lower}}$	Fixed trailing edge constraint	6
	$0.8 \cdot r_{\text{LE, bl}} \leq r_{\text{LE}} \leq 3 \cdot r_{\text{LE, bl}}$	Leading edge radius constraint	6
		Total Constraint Functions	**80**

Table 13: Summary of the wing aerodynamic shape optimization results. The DP- and Krylov-adjoint methods reduce the drag by 9.7% and 9.9%, respectively, and all constraints are satisfied.

	Baseline Design	Optimized Design (DP-adjoint)	Optimized Design (Krylov-adjoint)
C_D	0.02187	0.01975	0.01970
C_L	0.3750	0.3750	0.3751
α	5.021°	3.499°	3.668°
Optimality	1.7×10^{-3}	8.9×10^{-5}	7.5×10^{-5}
Feasibility	1.8×10^{-6}	8.2×10^{-6}	7.9×10^{-5}

We use the ADODG-3 wing geometry as the baseline design. The flow is incompressible and turbulent with the Reynolds number being 6.7×10^4. Table 12 shows the detailed optimization configuration. The objective function is C_D. The design variables are the angle of attack (α), the wing twist angles (γ) at the five spanwise FFD sections (the wing root is fixed), and the vertical displacement (Δy) of the 120 FFD points. We have 126 design variables in total. In terms of constraints, we maintain a fixed target C_L of 0.375. Moreover, we fix the leading and trailing edges of the wing by constraining the relevant FFD control point movement. To ensure a practical design, we also apply geometric constraints on the wing's volume, thickness, and leading-edge radius. We implement our proposed DP adjoint into the DAFoam gradient computation toolbox [37, 38] and couple it with the OpenMDAO/Mphys framework [65] for aerodynamic optimization. We use the sparse nonlinear optimizer (SNOPT) in the pyOptSparse [66] module, and we deploy pyGeo and IDWarp modules to handle the geometry parameterization and volume mesh deformation during optimization. As a comparison, we also run an optimization with the exact same setup except that we use the DAFoam's default Krylov adjoint method [21] to compute gradients.

A comprehensive analysis of the optimization result is beyond the scope of this paper, so we provide only a brief summary here. Table 13 shows the summary of optimization results. We first analyze the optimization results from the DP-adjoint approach. The wing optimization converges after 16 major iterations, and the objective function (C_D) drops by 9.7% while the target C_L is maintained at 0.375. The baseline design's optimality and feasibility are 1.7×10^{-3} and 1.6×10^{-6}, respectively. For the optimized design, the optimality drops to 8.9×10^{-5} and the feasibility remains at a low level. This indicates that the optimization converges well. Moreover, the angle of attack reduces from 5.021° to 3.499°. The optimization reduces the angle of attack and creates a cambered airfoil shape to reduce the aerodynamic drag, which can be seen more clearly in Fig. 12. The cambered shape moves the peak pressure load afterward and makes the pressure distribution more evenly distributed along the chord.

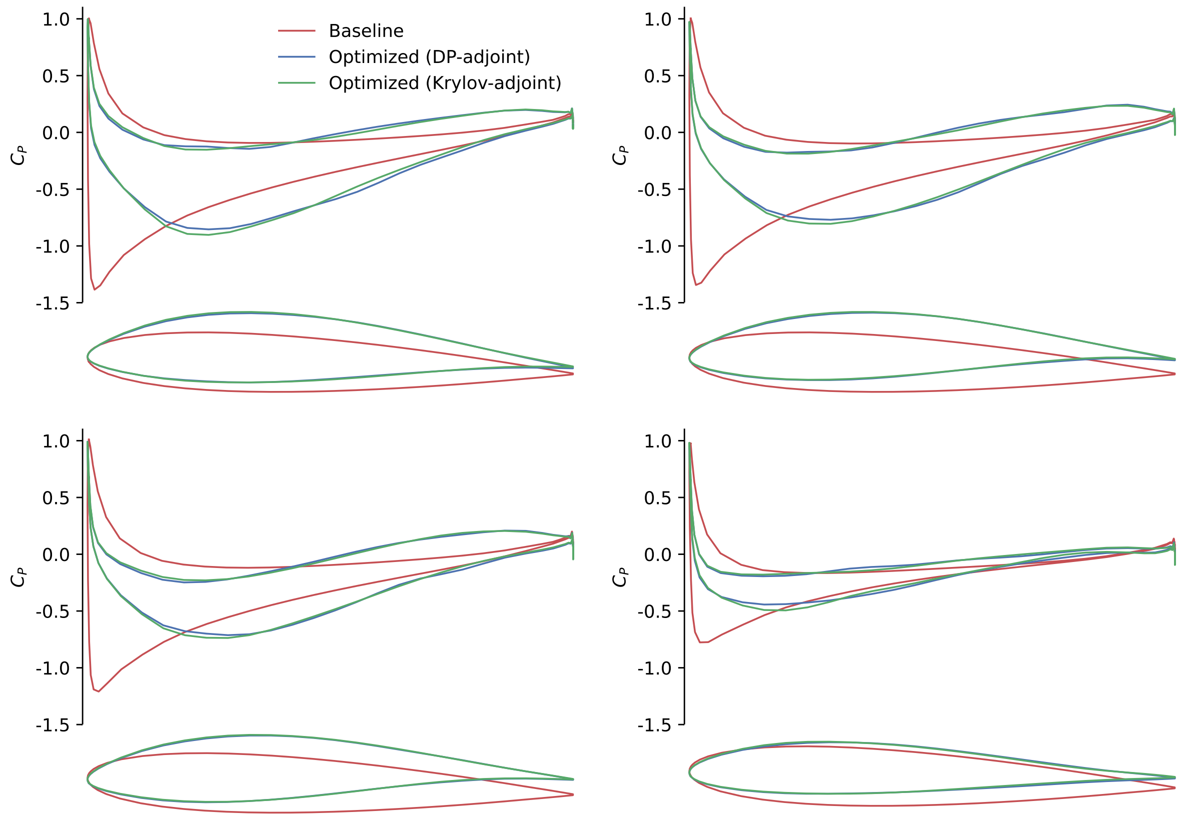

Figure 12: Comparison of C_p distribution and airfoil shape at different wing spanwise sections between the baseline and optimized designs. Top left: 3% span. Top right: 31% span. Bottom left: 63% span. Bottom right: 94% span. The baseline, DP-adjoint optimized, and Krylov-adjoint optimized designs are denoted in red, blue, and green, respectively.

This trend is consistent with our previous aerodynamic optimization results that used a Krylov-based adjoint solver [37, 38, 67]. To further confirm this, we compare the optimization results obtained from the DP- and Krylov-adjoint approaches. The Krylov-adjoint approach achieves a slightly higher drag reduction (9.9%; Table 13) and slightly different pressure distributions (Fig. 12) than the DP-adjoint approach. However, the optimized sectional airfoil geometries are practically the same between the DP- and Krylov-adjoint approaches. Overall, the optimization results between the DP- and Krylov-adjoint approaches are similar.

4. Conclusion

This paper derives a duality-preserving (DP) adjoint formulation that shares the same asymptotic convergence with the primal segregated Navier–Stokes solvers. The adjoint formulation derivation starts with rewriting the segregated primal solution process into a fully coupled left-preconditioned Richardson format. Then, we transpose the preconditioner matrix to obtain a fixed-point adjoint formulation for the segregated primal solver. The transpose operation maintains the iteration matrix eigenvalues and thus preserves the duality.

In addition to the algebraic derivation, we create a new graph representation called duality-preserving adjoint as a reverse graph (DARG). DARG uses vertices and edge-directed operators to represent the segregated numerical computation processes for the primal solver. Then, it reverses the graph with transposed operators to construct the DP adjoint. We proved that the DARG representation is equivalent to the above preconditioner transpose approach, and its implementation is much more streamlined and generalizable.

We analyze the impact of various numerical settings on the adjoint duality and accuracy, and find the initial values of the inner iteration require special attention. Naively using non-zero initial values from a previous outer iteration will introduce errors to the adjoint state vector and deteriorate the adjoint accuracy. To avoid this, we should always use zero initial values for inner iterations. Moreover, we observe that true duality can only be achieved if the inner iteration converges tightly. However, in practice, we allow the primal and adjoint inner iterations to converge only to a relative tolerance of 10^{-2} or 10^{-3}, which introduces additional perturbations to the preconditioner matrix.

To quantify the duality, we use the Arnoldi method to compute the iteration matrix eigenvalues for the primal and DP adjoint solvers. As a sanity check, we also use a non-dual adjoint formulation derived based on intuition instead of rigorous math. We consider three benchmark cases with increasing complexity: a miniature-sized problem, a 2D airfoil, and a 3D wing. We find that the DP adjoint's eigenvalue distributions agree reasonably well with the primal for all three cases. Using a loose inner iteration tolerance perturbs the eigenvalue distributions by a small amount; however, all the perturbed eigenvalues are still within the unit circle, and the DP adjoint converges well for all three cases. Although the convergence rate of the non-dual adjoint is similar to the primal for the miniature-sized and airfoil problems, its eigenvalues significantly differ from the primal solver. For the most complicated wing case, the non-dual adjoint diverges, and parts of its eigenvalues lie outside the unit circle. The eigenvalue tests demonstrate that the proposed DP adjoint formulation can robustly preserve the duality and converge the adjoint.

In addition to the duality, we evaluate the accuracy, speed, and memory usage of the DP adjoint algorithm. Our proposed adjoint exhibits accurate gradient computation that matches the forward-mode automatic differentiation references by eight to ten significant digits. This accurate gradient is critical for design optimization because it allows optimization to converge tightly and robustly. Moreover, the proposed adjoint algorithm exhibits competitive speed, with the adjoint-primal runtime ratio being less than 2.2. The DP-adjoint solution requires up to 21 times more memory than the primal solution, which is primarily caused by the use of an operator overloading AD tool to compute matrix-vector products. However, such memory usage is still acceptable because most modern workstations or high-performance computing (HPC) systems are equipped with sufficient memory. To further demonstrate the robustness of the DP adjoint computation, we incorporate it into a gradient-based optimization framework and conduct a wing aerodynamic shape optimization. The optimization converges well, and the objective function (drag) reduces by 9.7%.

The proposed fixed-point adjoint approach has the potential to make the adjoint solution more robust and memory efficient for any segregated NS solvers. In the future, we will extend the adjoint approach for unsteady flow problems.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Number 2223676. The computations were partially done in the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation Grant number ACI-1548562.

Appendix: Computing eigenvalues using the Arnold method

Suppose we have a general square matrix operator $\boldsymbol{A}$, which can be the iteration matrix for the primal, DP adjoint, or non-dual adjoint solvers. The Arnoldi process starts with a vector $\boldsymbol{v}$ and builds an orthogonal basis for the Krylov subspace $\boldsymbol{K}_m(\boldsymbol{A}, \boldsymbol{v})$, where m is the number of Arnoldi iterations. As a by-product, this process generates an upper Hessenberg matrix $\boldsymbol{H} \in \mathbb{R}^{m\times m}$, whose eigenvalues (also known as Ritz values) can approximate those of $\boldsymbol{A}$. If the number of Arnoldi iterations m equals the size of $\boldsymbol{A}$, all eigenvalues of $\boldsymbol{A}$ can be obtained as the Ritz values. However, for most practical cases, m is much smaller than the size of $\boldsymbol{A}$, and the Ritz value distribution tends to capture the outskirt of the eigenvalue distribution for $\boldsymbol{A}$. This behavior is suitable for our analysis because the asymptotic convergence of a specific solver is governed by the outskirt eigenvalues of its iteration matrix, and the interior eigenvalues are inconsequential.

With the Arnoldi method, the aforementioned matrix operator $\boldsymbol{A}$ does not need to be treated explicitly, and we can view it as a linear operator. Suppose the linear operator $\boldsymbol{A}$ takes a vector input $\boldsymbol{x}$ and generates a vector output $\boldsymbol{y}$, i.e., $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, where $\boldsymbol{A}$ is the respective iteration matrix. Just like the state variable $\boldsymbol{w}$ and the adjoint state vector $\boldsymbol{\psi}$, $\boldsymbol{x}$ or $\boldsymbol{y}$ has 4 components that correspond to $\boldsymbol{U}$, $\boldsymbol{p}$, $\boldsymbol{\phi}$, and $\tilde{\boldsymbol{\nu}}$. Thus we name the 4 components of $\boldsymbol{x}$ accordingly as $\boldsymbol{x}_U$, $\boldsymbol{x}_p$, $\boldsymbol{x}_\phi$, and $\boldsymbol{x}_{\tilde{\nu}}$; similarly, $\boldsymbol{y}$ has 4 components $\boldsymbol{y}_U$, $\boldsymbol{y}_p$, $\boldsymbol{y}_\phi$, and $\boldsymbol{y}_{\tilde{\nu}}$. This linear operation is performed once for every Arnoldi iteration, and we can use the existing code of the respective solver for its implementation in the Arnoldi process.

For the primal solver, the iteration matrix is $\boldsymbol{A} = \boldsymbol{I} + \tilde{\boldsymbol{X}}\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}$, where $\tilde{\boldsymbol{X}}$ is the perturbed version of the theoretical primal preconditioner $\boldsymbol{X}$ due to the inner iterations (see Sec. 2.5). If the inner iteration converges tightly, $\tilde{\boldsymbol{X}} = \boldsymbol{X}$. Then, the linear operation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ becomes:

$$\boldsymbol{y} = \boldsymbol{x} + \tilde{\boldsymbol{X}}\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}\boldsymbol{x}. \tag{60}$$

We use central finite difference to evaluate the matrix-vector product $\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}\boldsymbol{x}$:

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}\boldsymbol{x} \approx \frac{1}{2\varepsilon}(\boldsymbol{R}(\boldsymbol{w} + \varepsilon\boldsymbol{x}) - \boldsymbol{R}(\boldsymbol{w} - \varepsilon\boldsymbol{x})), \tag{61}$$

where $\varepsilon = 1 \times 10^{-6}$ is the step size. The components of the state variable $\boldsymbol{w}$, i.e., $\boldsymbol{U}$, $\boldsymbol{p}$, $\boldsymbol{\phi}$, and $\tilde{\boldsymbol{\nu}}$, can have drastically different scales, thus to effectively perturb $\boldsymbol{w}$ as $\boldsymbol{w} \pm \varepsilon\boldsymbol{x}$ in Eq. 61, we introduce an appropriate diagonal matrix scaling $\boldsymbol{D}_\alpha$ to both $\boldsymbol{x}$ and $\boldsymbol{y}$, i.e., $\boldsymbol{x} = \boldsymbol{D}_\alpha\boldsymbol{x}'$ and $\boldsymbol{y} = \boldsymbol{D}_\alpha\boldsymbol{y}'$. The scaling factors for $\boldsymbol{U}$, $\boldsymbol{p}$, $\boldsymbol{\phi}$, and $\tilde{\nu}$ are U_0, $0.5{U_0}^2$, $U_0 S_f$, and $\tilde{\nu}_0$, respectively. Here the superscript 0 denotes the free-stream condition, and S_f is the surface area for each mesh face.

The scaling introduced above effectively changes the linear operation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ into $\boldsymbol{y}' = \boldsymbol{D}_\alpha^{-1}\boldsymbol{A}\boldsymbol{D}_\alpha\boldsymbol{x}'$, which still retains the same set of eigenvalues. Thus after the scaling, the linear operation in Eq. 60 becomes:

$$\boldsymbol{y}' = \boldsymbol{x}' + \boldsymbol{D}_\alpha^{-1}\tilde{\boldsymbol{X}}\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}\boldsymbol{D}_\alpha\boldsymbol{x}', \tag{62}$$

where $\tilde{\boldsymbol{X}}$ is a linear operator (perturbed preconditioner) similar to the primal solution process from Eq. 45 to Eq. 52. The only difference is that, we use $\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}\boldsymbol{x}$ as the input instead of $\boldsymbol{R}$.

For the DP adjoint solver, the iteration matrix is $\boldsymbol{A} = \boldsymbol{I} + \widetilde{\boldsymbol{X}^T}\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^T$, where $\widetilde{\boldsymbol{X}^T}$ is the perturbed version of the DP adjoint preconditioner $\boldsymbol{X}^T$ due to inner iterations. Then, the linear operation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ becomes:

$$\boldsymbol{y} = \boldsymbol{x} + \widetilde{\boldsymbol{X}^T}\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^T\boldsymbol{x}. \tag{63}$$

We do not apply any scaling to the linear operation in Eq. 63. The matrix-vector product $\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^T\boldsymbol{x}$ is computed using the reverse-mode AD. Similarly, $\widetilde{\boldsymbol{X}^T}$ represents the adjoint solution process in Eq. 53. Note that we fix the number of inner iterations in $\tilde{\boldsymbol{X}}$ and $\widetilde{\boldsymbol{X}^T}$, instead of letting the inner iteration converge to a prescribed tolerance. This treatment ensures the linear operator $\boldsymbol{A}$ (iteration matrix) is identical for each Arnoldi iteration. Also, note that we apply the Arnoldi method to the reformatted primal solver, instead of the original SIMPLE solver. This is because the original, segregated SIMPLE algorithm is not readily compatible with the Arnoldi method for iteration matrix eigenvalue extraction. We use a similar Arnoldi iteration approach for the non-dual adjoint solver. Algorithm 1 summarizes the Arnoldi iteration process.

The Arnoldi processes for all three aforementioned solvers use an all-one vector as the starting vector. Once an Arnoldi process is completed and a corresponding upper Hessenberg matrix is generated, we use MATLAB's standard QR algorithm to find all Ritz values.

Algorithm 1 Arnoldi process

```
Input: Arbitrary starting vector $\boldsymbol{v}$
Output: Upper Hessenberg matrix $\boldsymbol{H}$
 1: Initialize $\boldsymbol{H} \in \mathbb{R}^{m \times m}$ as an all-zero matrix
 2: $\boldsymbol{x}_1 = \boldsymbol{v} / \parallel \boldsymbol{v} \parallel$
 3: for $i = 1 : m$ do
 4:     $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_i$
 5:     for $j = 1 : i$ do
 6:         $\boldsymbol{H}_{j,i} = \boldsymbol{x}_i^T \boldsymbol{y}$
 7:         $\boldsymbol{y} = \boldsymbol{y} - \boldsymbol{H}_{j,i}\boldsymbol{x}_j$
 8:     if $i < m$ then
 9:         $\boldsymbol{H}_{i+1,i} = \parallel \boldsymbol{y} \parallel$
10:         $\boldsymbol{x}_{i+1} = \boldsymbol{y} / \boldsymbol{H}_{i+1,i}$
```

References

References

[1] J. R. R. A. Martins, A. Ning, Engineering design optimization, Cambridge University Press, 2021.

[2] L. Abergo, M. Morelli, A. Guardone, Aerodynamic shape optimization based on discrete adjoint and RBF, Journal of Computational Physics (2023) 111951. Publisher: Elsevier.

[3] A. Jameson, L. Martinelli, N. A. Pierce, Optimum aerodynamic design using the Navier–Stokes equations, Theoretical and Computational Fluid Dynamics 10 (1998) 213–237. doi:10.1007/s001620050060.

[4] M. A. Bouhlel, J. R. R. A. Martins, Gradient-enhanced kriging for high-dimensional problems, Engineering with Computers 1 (2019) 157–173. doi:10.1007/s00366-018-0590-x.

[5] M. A. Bouhlel, S. He, J. R. R. A. Martins, Scalable gradient-enhanced artificial neural networks for airfoil shape design in the subsonic and transonic regimes, Structural and Multidisciplinary Optimization 61 (2020) 1363–1376. doi:10.1007/s00158-020-02488-5.

[6] S. Ulaganathan, I. Couckuyt, T. Dhaene, J. Degroote, E. Laermans, Performance study of gradient-enhanced Kriging, Engineering with computers 32 (2016) 15–34. Publisher: Springer.

[7] B. Kaltenbacher, A. Schlintl, Fractional time stepping and adjoint based gradient computation in an inverse problem for a fractionally damped wave equation, Journal of Computational Physics 449 (2022) 110789. Publisher: Elsevier.

[8] E. J. Parish, K. Duraisamy, A paradigm for data-driven predictive modeling using field inversion and machine learning, Journal of computational physics 305 (2016) 758–774. Publisher: Elsevier.

[9] D. Givoli, A tutorial on the adjoint method for inverse problems, Computer Methods in Applied Mechanics and Engineering 380 (2021) 113810. Publisher: Elsevier.

[10] K. J. Fidkowski, D. L. Darmofal, Review of output-based error estimation and mesh adaptation in computational fluid dynamics, AIAA Journal 49 (2011) 673–694.

[11] L. Shi, Z. J. Wang, Adjoint-based error estimation and mesh adaptation for the correction procedure via reconstruction method, Journal of Computational Physics 295 (2015) 261–284. Publisher: Elsevier.

[12] B. Cockburn, S. Xia, An adjoint-based adaptive error approximation of functionals by the hybridizable discontinuous Galerkin method for second-order elliptic equations, Journal of Computational Physics 457 (2022) 111078. Publisher: Elsevier.

[13] O. Pironneau, On optimum profiles in Stokes flow, Journal of Fluid Mechanics 59 (1973) 117–128. doi:10.1017/S002211207300145X, publisher: Cambridge Univ Press.

[14] A. Jameson, Aerodynamic design via control theory, Journal of Scientific Computing 3 (1988) 233–260. doi:10.1007/BF01061285.

[15] P. Gomes, R. Palacios, Pitfalls of discrete adjoint fixed-points based on algorithmic differentiation, AIAA Journal (2021) 1–6. Publisher: American Institute of Aeronautics and Astronautics.

[16] B. Fleischli, L. Mangani, A. Del Rio, E. Casartelli, A discrete adjoint method for pressure-based algorithms, Computers & Fluids 227 (2021) 105037. Publisher: Elsevier.

[17] A. Rubino, M. Pini, P. Colonna, T. Albring, S. Nimmagadda, T. Economon, J. Alonso, Adjoint-based fluid dynamic design optimization in quasi-periodic unsteady flow problems using a harmonic balance method, Journal of Computational Physics 372 (2018) 220–235.

[18] T. A. Albring, M. Sagebaum, N. R. Gauger, Efficient aerodynamic design using the discrete adjoint method in SU2, 17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference (2016) 13–17. doi:10.2514/6.2016-3518, iSBN: 978-1-62410-439-8.

[19] D. J. Mavriplis, Multigrid solution of the discrete adjoint for optimization problems on unstructured meshes, AIAA Journal 44 (2006) 42–50.

[20] E. J. Nielsen, J. Lu, M. A. Park, D. L. Darmofal, An implicit, exact dual adjoint solution method for turbulent flows on unstructured grids, Computers and Fluids 33 (2004) 1131–1155. doi:10.1016/j.compfluid.2003.09.005, iSBN: 1757864881.

[21] G. K. Kenway, C. A. Mader, P. He, J. R. Martins, Effective adjoint approaches for computational fluid dynamics, Progress in Aerospace Sciences 110 (2019) 100542. doi:10.1016/j.paerosci.2019.05.002, publisher: Pergamon.

[22] J. E. V. Peter, R. P. Dwight, Numerical sensitivity analysis for aerodynamic optimization: A survey of approaches, Computers and Fluids 39 (2010) 373–391. doi:10.1016/j.compfluid.2009.09.013.

[23] M. B. Giles, N. A. Pierce, An introduction to the adjoint approach to design, Flow, Turbulence and Combustion 65 (2000) 393–415. doi:10.1023/a:1011430410075, iSBN: 1386-6184.

[24] C. A. Mader, G. K. W. Kenway, A. Yildirim, J. R. R. A. Martins, ADflow—An open-source computational fluid dynamics solver for aerodynamic and multidisciplinary optimization, Journal of Aerospace Information Systems (2020). doi:10.2514/1.I010796.

[25] J. E. Hicken, D. W. Zingg, Aerodynamic optimization algorithm with integrated geometry parameterization and mesh movement, AIAA Journal 48 (2010) 400–413. doi:10.2514/1.44033.

[26] L. Osusky, H. Buckley, T. Reist, D. W. Zingg, Drag minimization based on the Navier—Stokes equations using a Newton—Krylov approach, AIAA Journal 53 (2015) 1555–1577. doi:10.2514/1.J053457.

[27] R. P. Dwight, J. Brezillon, Effect of approximations of the discrete adjoint on gradient-based optimization, AIAA Journal 44 (2006) 3022–3031.

[28] T. Gerhold, M. Galle, O. Friedrich, J. Evans, T. Gerhold, M. Galle, O. Friedrich, J. Evans, Calculation of complex three-dimensional configurations employing the DLR-TAU-code, in: 35th aerospace sciences meeting and exhibit, 1997, p. 167.

[29] M. B. Giles, On the iterative solution of adjoint equations, Automatic Differentiation of Algorithms: From Simulation to Optimization (2002) 145–151.

[30] M. Giles, N. Pierce, M. Giles, N. Pierce, Adjoint equations in CFD-Duality, boundary conditions and solution behaviour, in: 13th computational fluid dynamics conference, 1997, p. 1850.

[31] S. Xu, D. Radford, M. Meyer, J.-D. Muller, Stabilisation of discrete steady adjoint solvers, Journal of Computational Physics 299 (2015) 175–195. doi:10.1016/j.jcp.2015.06.036, publisher: Elsevier.

[32] J.-D. Mueller, J. Hueckelheim, O. Mykhaskiv, J.-D. Müller, O. Mykhaskiv, J. Hückelheim, STAMPS: a finite-volume solver framework for adjoint codes derived with source-transformation AD, in: 2018 Multidisciplinary analysis and optimization conference, 2018, p. 2928. doi:10.2514/6.2018-2928.

[33] T. D. Economon, F. Palacios, S. R. Copeland, T. W. Lukaczyk, J. J. Alonso, SU2: An open-source suite for multiphysics simulation and design, AIAA Journal 54 (2016) 828–846. doi:10.2514/1.j053813.

[34] S. V. Patankar, D. B. Spalding, A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows, International Journal of Heat and Mass Transfer 15 (1972) 1787–1806. doi:10.1016/0017-9310(72)90054-3.

[35] C. B. Dilgen, S. B. Dilgen, D. R. Fuhrman, O. Sigmund, B. S. Lazarov, Topology optimization of turbulent flows, Computer Methods in Applied Mechanics and Engineering 331 (2018) 363–393. Publisher: Elsevier.

[36] C. M. Okubo Jr, L. F. N. Sá, C. Y. Kiyono, E. C. N. Silva, A discrete adjoint approach based on finite differences applied to topology optimization of flow problems, Computer Methods in Applied Mechanics and Engineering 389 (2022) 114406. Publisher: Elsevier.

[37] P. He, C. A. Mader, J. R. R. A. Martins, K. J. Maki, DAFoam: An open-source adjoint framework for multidisciplinary design optimization with OpenFOAM, AIAA Journal 58 (2020) 1304–1319. doi:10.2514/1.J058853.

[38] P. He, C. A. Mader, J. R. R. A. Martins, K. J. Maki, An aerodynamic design optimization framework using a discrete adjoint approach with OpenFOAM, Computers & Fluids 168 (2018) 285–303. doi:10.1016/j.compfluid.2018.04.012.

[39] H. G. Weller, G. Tabor, H. Jasak, C. Fureby, A tensorial approach to computational continuum mechanics using object-oriented techniques, Computers in Physics 12 (1998) 620–631. Publisher: AIP Publishing.

[40] S. Akbarzadeh, Y. Wang, J. D. Mueller, Fixed point discrete adjoint of simple-like solvers, in: 22nd AIAA Computational Fluid Dynamics Conference, 2015, p. 2750.

[41] L. Fang, P. He, A consistent fixed-point discrete adjoint method for segregated Navier–Stokes solvers, in: AIAA AVIATION 2022 forum, 2022, p. 4000.

[42] M. Wang, Q. Wang, T. A. Zaki, Discrete adjoint of fractional-step incompressible navier-stokes solver in curvilinear coordinates and application to data assimilation, Journal of Computational Physics 396 (2019) 427–450.

[43] J. Kim, P. Moin, Application of a fractional-step method to incompressible navier-stokes equations, Journal of computational physics 59 (1985) 308–323.

[44] N. Kühl, J. Kröger, M. Siebenborn, M. Hinze, T. Rung, Adjoint complement to the volume-of-fluid method for immiscible flows, Journal of Computational Physics 440 (2021) 110411.

[45] A. Stück, T. Rung, Adjoint complement to viscous finite-volume pressure-correction methods, Journal of Computational Physics 248 (2013) 402–419.

[46] J. E. Hicken, J. Li, O. Sahni, A. A. Oberai, Adjoint consistency analysis of residual-based variational multiscale methods, Journal of Computational Physics 255 (2013) 396–406. Publisher: Elsevier.

[47] R. Hartmann, T. Leicht, Generalized adjoint consistent treatment of wall boundary conditions for compressible flows, Journal of Computational Physics 300 (2015) 754–778.

[48] M. Kontou, X. Trompoukis, V. Asouti, K. Giannakoglou, On the discretization of the continuous adjoint to the euler equations in aerodynamic shape optimization, in: International Conference on Adaptive Modeling and Simulation, 2023.

[49] E. M. Papoutsis-Kiachagias, K. C. Giannakoglou, Continuous adjoint methods for turbulent flows, applied to shape and topology optimization: Industrial applications, Archives of Computational Methods in Engineering 23 (2016) 255–299. doi:10.1007/s11831-014-9141-9, iSBN: 1134-3060.

[50] S. K. Nadarajah, The discrete adjoint approach to aerodynamic shape optimization, Stanford University, 2003.

[51] P. Spalart, S. Allmaras, A one-equation turbulence model for aerodynamic flows, in: 30th aerospace sciences meeting and exhibit, 1992. doi:10.2514/6.1992-439.

[52] T. Uroić, Implicitly coupled finite volume algorithms, Ph.D. thesis, University of Zagreb. Faculty of Mechanical Engineering and Naval Architecture, 2019.

[53] G. M. Shroff, H. B. Keller, Stabilization of unstable procedures: the recursive projection method, SIAM Journal on numerical analysis 30 (1993) 1099–1120.

[54] R. P. Dwight, J. Brezillon, Efficient and robust algorithms for solution of the adjoint compressible navier–stokes equations with applications, International journal for numerical methods in fluids 60 (2009) 365–389.

[55] E. Åkervik, L. Brandt, D. S. Henningson, J. Hœpffner, O. Marxen, P. Schlatter, Steady solutions of the navier-stokes equations by selective frequency damping, Physics of fluids 18 (2006).

[56] F. Richez, M. Leguille, O. Marquet, Selective frequency damping method for steady rans solutions of turbulent separated flows around an airfoil at stall, Computers & Fluids 132 (2016) 51–61.

[57] V. Citro, P. Luchini, F. Giannetti, F. Auteri, Efficient stabilization and acceleration of numerical simulation of fluid flows by residual recombination, Journal of Computational Physics 344 (2017) 234–246.

[58] A. Dicholkar, F. Zahle, N. N. Sørensen, Convergence enhancement of simple-like steady-state rans solvers applied to airfoil and cylinder flows, Journal of Wind Engineering and Industrial Aerodynamics 220 (2022) 104863.

[59] S. Xu, J. Zhao, H. Wu, S. Zhang, J.-D. Müller, H. Huang, M. Rahmati, D. Wang, A review of solution stabilization techniques for RANS CFD solvers, Aerospace 10 (2023) 230. Publisher: MDPI.

[60] M. F. De Pando, D. Sipp, P. J. Schmid, Efficient evaluation of the direct and adjoint linearized dynamics from compressible flow solvers, Journal of Computational Physics 231 (2012) 7739–7755.

[61] N. R. Secco, G. K. W. Kenway, P. He, C. A. Mader, J. R. R. A. Martins, Efficient mesh generation and deformation for aerodynamic shape optimization, AIAA Journal 59 (2021) 1151–1168. doi:10.2514/1.J059491.

[62] G. K. W. Kenway, G. J. Kennedy, J. R. R. A. Martins, A CAD-Free approach to high-fidelity aerostructural optimization, in: Proceedings of the 13th AIAA/ISSMO multidisciplinary analysis optimization conference, Fort Worth, TX, 2010. doi:10.2514/6.2010-9231.

[63] R. Hartmann, Adjoint consistency analysis of discontinuous Galerkin discretizations, SIAM Journal on Numerical Analysis 45 (2007) 2671–2696. Publisher: SIAM.

[64] N. R. G. M. Sagebaum T. Albring, High-performance derivative computations using CoDiPack, ACM Transactions on Mathematical Software (TOMS) 45 (2019).

[65] J. S. Gray, J. T. Hwang, J. R. R. A. Martins, K. T. Moore, B. A. Naylor, OpenMDAO : An open-source framework for multidisciplinary design , analysis , and optimization, Structural and Multidisciplinary Optimization (2019).

[66] N. Wu, G. K. W. Kenway, C. A. Mader, J. Jasa, J. R. R. A. Martins, pyOptSparse: A Python framework for large-scale constrained nonlinear optimization of sparse systems, AIAA Journal (2020). doi:`10.21105/joss.02564`.

[67] H. U. Koyuncuoglu, P. He, Simultaneous wing shape and actuator parameter optimization using the adjoint method, Aerospace Science and Technology 130 (2022) 107876. Publisher: Elsevier.