This is a preprint of the following article, which is available at: http://mdolab.engin.umich.edu
A. Yildirim, G. K. W. Kenway, C. A. Mader, and J. R. R. A. Martins, "A Jacobian-free approximate Newton–Krylov startup strategy for RANS simulations", *Journal of Computational Physics*, 397:108741, Nov. 2019.
The original article may differ from this preprint and is available at:
https://doi.org/10.1016/j.jcp.2019.06.018.

A Jacobian-free approximate Newton–Krylov startup strategy for RANS simulations

Anil Yildirim, Gaetan K. W. Kenway, Charles A. Mader, and Joaquim R. R. A. Martins
Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI, 48109

Abstract

The favorable convergence rates of Newton–Krylov-based solution algorithms have increased their popularity for computational fluid dynamics applications. Unfortunately, these methods perform poorly during the initial stages of convergence, particularly for three-dimensional Reynolds-averaged Navier–Stokes simulations. Addressing this problem requires the use of a globalization method such as pseudo-transient continuation, along with an approximate Newton–Krylov startup stage. This class of methods marches the solution in pseudo-time with a matrix-based approximate Jacobian that has a lower bandwidth and better conditioning properties compared with the exact Jacobian. However, this matrix-based approach also has shortcomings, including a large cost of computing and storing the approximate Jacobian and its preconditioner, along with the need for an updated Jacobian for every nonlinear iteration. To rectify these shortcomings, we use herein approximate residual formulations in a Jacobian-free approximate Newton–Krylov algorithm. With the approximate Jacobian, we compute the vector products by using the approximate residual computations in a matrix-free manner while forming a preconditioner based on the matrix-based approximate Jacobian. This approach keeps the approximate Jacobian up to date and mitigates the cost of forming a matrix-based Jacobian and its preconditioner at each iteration by lagging the preconditioner between nonlinear iterations. We use varying levels of approximations with the matrix-free approach and thereby demonstrate the trade-off between rate of convergence and the cost of each nonlinear iteration. The proposed implementation uses only the exact and approximate residual formulations and can therefore be generalized with minimal additional implementation effort to a range of solvers and various discretizations. The code is available under an open-source license.

1 Introduction

In multidisciplinary design optimization (MDO) environments [1], computational fluid dynamics (CFD) is often used to evaluate the aerodynamic performance of a given design and to compute the objective function and constraint gradients with respect to the design variables [2–5]. In high-fidelity aerodynamic and aerostructural design optimization processes, hundreds of successive Reynolds-averaged Navier–Stokes (RANS) simulations and adjoint solutions are required, which use meshes with up to tens of millions of cells [6–8]. These simulation-based optimizations run on massively parallel architectures and can take up to three days to complete using one thousand cores for the more complex aerostructural cases. Because the CFD solver is called repeatedly during optimization, solver efficiency is even more important than when running a few cases manually.

However, the robustness of the CFD solver affects optimization applications much more strongly because, if the solver fails, the optimization run either fails completely or, at the very least, progresses much more slowly, and a failed run represents lost computing time and requires additional human intervention. This problem is compounded by the fact that a CFD simulation is more likely to fail within an optimization run because the optimizer is likely to request the simulation of shapes that a human designer would never request [9]. The failures can be due either to the issues in the mesh deformation or to the CFD solution itself. In this work, we focus on the robustness of the CFD solver (we addressed the mesh deformation robustness in previous work [9]). In this context, obtaining a robust CFD solver at the cost of lower performance is preferable. Although each simulation might be slower, the optimizations finish sooner because they require less human intervention.

He et al. [9] studied an aerodynamic shape optimization problem that demonstrates the importance of robustness. The problem was to minimize the drag at a transonic condition while constraining lift starting from a circular shape. This required CFD simulations of many shapes a human designer would not consider, such as the shape shown in the second frame of Figure 1. Although RANS cannot accurately predict the aerodynamic performance of such designs due to the massive separation, the gradients provided the correct trends, and the optimization eventually converged to an optimal supercritical airfoil for which RANS is valid.

Newton–Krylov (NK) solvers are commonly used in CFD, where Newton's method iteratively solves the system of nonlinear equations and a Krylov subspace method solves the resulting large linear systems during each nonlinear iteration [10]. This is a desirable approach because Newton's method has favorable convergence rates and a matrix-free approach can be used for the Krylov subspace method, which greatly reduces memory requirements.

In the context of RANS simulations, if the initial guess is not in the basin of attraction, then NK-based methods often perform poorly and can diverge [11]. Correcting this problem requires the use of a globalization method [12]. Pseudo-transient continuation (PTC) [13] is the most commonly used globalization approach, where the solver strategy is initialized with a backward Euler method and the time step is ramped up as the solution converges. For smaller time steps, this method yields the favorable

Figure 1: Optimization problem studied by He et al. [9]. The optimizer started with the initial shape of a circle (first frame) and converged to a super-critical airfoil (third frame). In this process, the optimizer had to go through infeasible intermediate shapes, such as the one shown here (second frame).

stability characteristics of the backward Euler method, and as the time step ramps up, the method approaches Newton's method. Furthermore, backtracking and trust-region methods can be used to overcome the initial convergence problems with NK-type solvers [14].

Knoll and Keyes [15] comprehensively surveyed the Jacobian-free Newton–Krylov methods and their applications, and Kelley and Keyes [13] analyzed the convergence of the PTC method. Gropp et al. [16] detailed the Newton–Krylov–Schwarz (NKS) algorithms, which use the additive Schwarz method as a preconditioner (PC) for the globalized NK method with domain decomposition; this was followed by a study on the performance of NKS methods on distributed memory architectures [17]. These studies provide the foundation for the NK solver implementations used in CFD.

Hicken and Zingg [18] described a hybrid approach that used an approximate Newton–Krylov (ANK) method for efficient startup to solve the Euler equations in three dimensions, and Chisholm and Zingg [19] extended this method to two-dimensional RANS simulations. Finally, Osusky and Zingg [20] used the ANK solver for three-dimensional RANS simulations, following which they studied the performance of this approach [21]. In these studies, the ANK stage was described as a globalized NK method that operates with a matrix-based approximate Jacobian. This approximate Jacobian and its PC are lagged for a pre-determined number of iterations to alleviate the cost of computing and storing these large matrices in every iteration. Lagging these matrices for 3–5 nonlinear iterations is suitable for Euler simulations [10, 18].

However, RANS simulations require that the approximate Jacobian always be up to date [20, 21], which greatly increases the computational cost. This problem creates a critical trade-off between performance and robustness. Although a large lag can considerably accelerate the nonlinear iterations, it might cause the algorithm to diverge. For three-dimensional RANS simulations, which is the focus of this work, Osusky and Zingg [20] reported that lagging a matrix-based Jacobian does not improve solver performance and even causes the solver to diverge in some cases. In addition to

pseudo-transient continuation, Hicken and Zingg [11] studied two additional types of globalization methods based on boundary conditions and dissipation. Hicken et al. [22] also studied the dissipation-based continuation method, and Brown and Zingg [23] developed a homotopy continuation algorithm and showed that this method outperforms PTC in some cases. However, the method is relatively new compared with PTC, and further testing is required before it can replace PTC.

The high-order finite-element community also developed methods that use approximations in the Jacobian formulation to mitigate the large memory requirements and computational cost of Jacobian matrices that arise from discontinuous Galerkin discretizations. Crivellini and Bassi [24] studied the effects caused by using a reduced quadrature formulation for the numerical integrations required by the matrix-free Jacobian computations. Xia et al. [25] detailed a parallel implicit method that uses Jacobian matrices constructed with a lower-order reconstruction of the state. Both groups reported that memory and computation costs are lowered by introducing approximations into the Jacobian formulation.

Having a robust solver is also important in the field of output-based error estimation and mesh adaptation, which requires consecutive solutions of RANS and adjoint equations [26, 27]. In searching for an automated and robust CFD solver, Modisette [28] implemented a number of modifications to the baseline PTC algorithm, such as an unsteady line search and a physicality check to limit the updates computed with the underlying NK algorithm. Ceze and Fidkowski [29] studied a range of different CFL evolution methods used in the PTC algorithm and introduced the concept of constrained PTC [30], where an augmented Jacobian is used for the left-hand side to prevent non-physical updates. Burgess and Glasby [31] showed how to use methods from these two studies within the framework of the Computational Research and Engineering for Acquisition Tools and Environments.

In the present work, we use approximate residual formulations in the context of a Jacobian-free ANK algorithm, which has two main advantages: The first advantage is that the use of matrix-free operations means that the Jacobian is always up to date and contains the correct approximations. This approximate Jacobian has a lower bandwidth and is better conditioned than the exact Jacobian, so the resulting linear systems are easier to solve. This effect still holds when using the diagonal time stepping term; a linear system with an approximate Jacobian and a time stepping term is easier to solve than a linear system containing an exact Jacobian with the same time stepping term. The approximate matrix-free operations require fewer numerical operations compared with their exact counterparts, further reducing the cost of linear solutions. Because the approximate Jacobian is always up to date, we can aggressively lag its PC without the destabilizing effect of an outdated Jacobian. With the lagged PC, we can avoid forming and factorizing the PC matrix for each nonlinear iteration, which are very costly operations. Furthermore, we can monitor the performance of linear solutions with a lagged PC and automatically determine if it needs to be updated. To further improve robustness, we use both a physicality check to ensure that the updates to the state vector remain physical and a backtracking line search to pick an appropriate step size to reduce the unsteady residual norm.

The second main advantage of the proposed approach is that we build the approx-

imations for the different solver variants directly into the main residual computations used for the solver. This allows the nonlinear Newton iteration to be implemented in a general way, minimizing the effort required to implement the various approximation levels. The matrix-free formulation uses varying levels of approximate formulations without requiring further modifications. To precondition the resulting linear systems, we incompletely factorize a matrix-based approximate Jacobian to obtain the final PC. Kenway et al. [32] adopted a similar strategy for preconditioning adjoint solvers. These matrices for preconditioning are computed by using the same approximate residual computations that we use for the matrix-free operations, and we use efficient coloring acceleration techniques to reduce the cost of these operations. As a result, manual differentiation is not required, and we can use a wide range of discretization methods and options within our solver without incurring the additional implementation effort required by many implicit methods. Moreover, these residual computations directly include the full boundary conditions and overset connectivity treatments, further simplifying the implementation and improving robustness.

The high implementation cost of analytic Jacobian computations is not a roadblock in the applications of NK-type solvers, as demonstrated by numerous examples. For example, Nejat and Ollivier-Gooch [33] used an approximate analytic Jacobian with a high-order formulation for two-dimensional Euler simulations, Asgharzadeh and Borazjani [34] developed an approximate analytic Jacobian for Navier–Stokes simulations with immersed boundaries, and Cavalca et al. [35] described their approach for three-dimensional Euler simulations on unstructured meshes, where they used analytic expressions for the Jacobian computations. However, we prefer to use the approximate residual computations in a matrix-free context because this approach is generalizable and does not require further implementation upon varying the discretization method.

We use ADflow, an open source CFD solver, to demonstrate the performance of the proposed ANK method.[1] We use varying levels of approximations and study the trade-off between the nonlinear convergence rate and the cost of each nonlinear iteration for different levels of approximations. We benchmark the algorithm by applying it to a number of cases, perform tests with many variations of the ANK method, and demonstrate the effects of using a lagged PC with matrix-free operations in the solution process. The algorithm can handle unconventional aircraft configurations and is suitable for an optimization environment, which requires many successive CFD simulations for a variety of meshes.

The outline of the paper is as follows: Section 2 overviews the governing equations and the discretization method. Section 3 details the full ANK solver implementation and the choices for the default solver parameters, along with our contributions. We make two contributions to the ANK algorithm: Section 3.2 introduces a Jacobian-free formulation that uses approximate residual formulations, and Section 3.4 introduces an adaptive PC-lagging algorithm. Readers who want to implement the ANK algorithm should read all of Section 3, whereas NK experts can read only Sections 3.2 and 3.4 for a description of the proposed contributions. Section 4 describes the computational framework used in this study, the available solver algorithms, and how the ANK solver

[1] https://github.com/mdolab/adflow, accessed June 2019

fits into the overall solution process. Finally, Section 5 presents a number of results, and Section 6 summarizes the main conclusions of this work.

2 Governing Equations and Discretization

This section describes the governing equations for the three-dimensional RANS simulations and the discretization methods used to solve them.

2.1 Navier–Stokes Equations

In Cartesian coordinates, the three-dimensional Navier–Stokes (NS) equations can be written as

$$\frac{\partial}{\partial t}\int_V \mathcal{Q}dV + \oint_{\partial V} \vec{\mathcal{F}}\cdot\vec{n}dA = \oint_{\partial V} \vec{\mathcal{D}}\cdot\vec{n}dA, \tag{1}$$

where

$$\begin{aligned}
\mathcal{Q} &= \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{bmatrix}, \quad \vec{\mathcal{F}} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(E+p) \end{bmatrix}\hat{x} + \begin{bmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ v(E+p) \end{bmatrix}\hat{y} + \begin{bmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ w(E+p) \end{bmatrix}\hat{z}, \\
\vec{\mathcal{D}} &= \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ u\tau_{xx} + v\tau_{yx} + w\tau_{zx} - q_x \end{bmatrix}\hat{x} + \begin{bmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ u\tau_{xy} + v\tau_{yy} + w\tau_{zy} - q_y \end{bmatrix}\hat{y} + \begin{bmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ u\tau_{xz} + v\tau_{yz} + w\tau_{zz} - q_z \end{bmatrix}\hat{z}.
\end{aligned} \tag{2}$$

Vectors $\vec{\mathcal{F}}$ and $\vec{\mathcal{D}}$ are the convective and diffusive flux vectors, respectively. In this formulation, ρ is the density, u, v, w are the three velocity components in the x, y, z directions, respectively, E is the total energy, p is the pressure, τ_{ji} are the viscous stresses, and q_i are the heat fluxes. We use Sutherland's formula to compute laminar viscosity, and the equations are closed with the ideal gas law.

2.2 Turbulence Model

We use the one-equation model of Spalart and Allmaras [36] (SA). For brevity, we do not describe the full model. The model can be written in a simplified form as

$$\frac{\partial\tilde{\nu}}{\partial t} + u_j\frac{\partial\tilde{\nu}}{\partial x_j} = P(\tilde{\nu}) - M(\tilde{\nu}) + D(\tilde{\nu}), \tag{3}$$

where $P(\tilde{\nu})$, $M(\tilde{\nu})$, and $D(\tilde{\nu})$ are the production, destruction, and diffusion terms, respectively, and $\tilde{\nu}$ is the working variable of the SA model. Notable modifications of the baseline SA model, along with the definitions and details of the model variables, are described in the NASA Turbulence Modeling Resource.[2] Unless noted otherwise, the results presented herein were obtained by using the *SA-noft2* version.

[2] https://turbmodels.larc.nasa.gov/spalart.html, accessed June 2019

To prevent negative values for the modified vorticity $\tilde{S}$, we use the modification introduced by Allmaras et al. [37], which can be written as

$$\tilde{S} = \begin{cases} S + \bar{S}, & : \quad \bar{S} \geq -c_{v2} S \\ S + \frac{S(c_{v2}^2 + c_{v3}\bar{S})}{(c_{v3} - 2c_{v2})S - \bar{S}} & : \quad \bar{S} < -c_{v2} S, \end{cases} \tag{4}$$

where $\bar{S} = \tilde{\nu} f_{v2}/(\kappa d)^2$, $c_{v2} = 0.7$, and $c_{v3} = 0.9$.

As detailed in the next subsection, the spatial discretization of the turbulence model is accurate to first order in space, which changes the end result of the simulations and the nonlinear convergence rates compared with the use of a turbulence model that is accurate to second order. While developing a CFD solver for MDO, we saw little difference between second- and first-order-accurate turbulence discretizations. Therefore we tuned our solver with the first-order-accurate discretization for the SA turbulence model. However, the solver described in this work remains applicable to second-order-accurate discretizations for the turbulence model and even to turbulence models other than the SA model.

2.3 Spatial Discretization

We use structured multiblock meshes with overset connectivities to discretize a given domain [38]. The meshes are split into sub-blocks for load balancing and partitioned among multiple processors to make efficient use of the distributed memory architectures. We use these meshes to compute and store the cell-centered values for the density, momentum, energy, and turbulence model working variable. The vector of residuals for the n^{th} nonlinear iteration can be written as

$$\mathcal{R}_0^{(n)} = \mathcal{R}_0(\mathcal{Q}^{(n)}) = -\frac{\partial}{\partial t} \int_V \mathcal{Q} dV \bigg|_n , \tag{5}$$

where $\mathcal{Q}^{(n)}$ represents the vector of state variables.

For the finite volume formulation, we use the Jameson–Schmidt–Turkel (JST) scheme with scalar dissipation for inviscid fluxes [39]. The viscous flux gradients are computed by using the Green–Gauss approach. For structured meshes, this discretization results in a 33-point stencil. We denote the full residual computed with all these points as $\mathcal{R}_0$. The 33 points come from the 26 nearest neighbors, 6 second-level neighbors, and the cell itself.

Along with the full residual formulation, we define modified residuals that contain varying levels of approximations. The first-level approximation $\mathcal{R}_1$ omits the fourth-order dissipation fluxes that are introduced in the JST scheme. To compute the second-order dissipation fluxes when using the state $\mathcal{Q}^{(n)}$, we fix the shock sensor variable for the JST scheme, which is entropy in our case, and keep these values constant throughout the nonlinear iteration. As a result, only $\mathcal{R}_0$ computes the up-to-date entropy, whereas $\mathcal{R}_1$ uses the fixed entropy computed with $\mathcal{Q}^{(n)}$. This results in a 27-point stencil because we ignore the six second-level neighbors. The second-level of approximation $\mathcal{R}_2$ assumes that the computational mesh is perfectly orthogonal and that the gradients of velocities at faces are computed by only considering the nearest

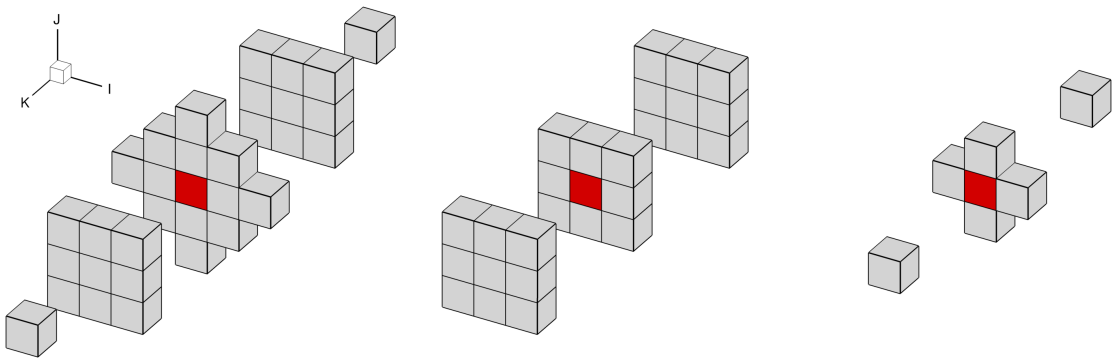

Figure 2: Exploded view of stencils used for the three levels of residual formulations. The center cells are highlighted in red.

neighbors sharing a face with the center cell, along with the approximation introduced in the previous level. The $\mathcal{R}_2$ approximation uses a seven-point stencil, which only contains the nearest neighbors sharing a face and the center cell itself.

The approximations introduced with $\mathcal{R}_2$ are commonly used to construct the PC matrix when solving the adjoint equations, or for solving the resulting linear system when Newton's method is used as a solution strategy. This is due to the lower memory requirements resulting from the reduced stencil [20, 32, 40]. Figure 2 shows the exploded view of the stencils resulting from these residual formulations.

The turbulence model is discretized by using a first-order-accurate upwind scheme for the advection term. Because this term is first-order accurate, we only consider the nearest neighbors sharing a face to compute the diffusion term (i.e., the gradients of $\tilde{\nu}$ are computed by using the orthogonal mesh assumption). As a result, the stencil for the SA model always contains seven points. The two approximate levels $\mathcal{R}_1$ and $\mathcal{R}_2$ omit the production term defined in Equation (3). Table 1 lists the approximations used with different levels of residual formulations.

Table 1: List of approximations within residual formulations.

		NS equations		SA model
Formulation	Stencil size	Advection term	Diffusion term	
$\mathcal{R}_0$	33	JST w/ scalar dissipation	2^{nd}-order accurate	1^{st}-order accurate
$\mathcal{R}_1$	27	w/o 4^{th}-order dissipation	2^{nd}-order accurate	w/o production term
$\mathcal{R}_2$	7	w/o 4^{th}-order dissipation	w/ orthogonal mesh assumption	w/o production term

Solving the RANS equations for the steady state is equivalent to finding a state vector $\mathcal{Q}$ that satisfies

$$\mathcal{R}_0(\mathcal{Q}) = 0. \tag{6}$$

Because the solver is iterative, a solution is considered to be converged when the norm of the residual vector is reduced by the specified tolerance compared with the residual

norm computed with freestream conditions, which can be written as

$$\eta_{\text{abs}} \geq \frac{||\mathcal{R}_0(\mathcal{Q}^{(n)})||_2}{||\mathcal{R}_0(\mathcal{Q}^{(\text{fs})})||_2}, \tag{7}$$

where $\mathcal{Q}^{(n)}$ is the converged solution vector at iteration n and $||\mathcal{R}_0(\mathcal{Q}^{(\text{fs})})||_2$ is the norm of the residual vector with freestream initial conditions. Although we can converge our solutions to machine precision, we select η_{abs} to be 10^{-12} to represent the typical convergence tolerances required for steady state simulations.

2.4 Variable Scaling

The state variables are normalized with respect to freestream values and, in a typical simulation, entries in the state vector $\mathcal{Q}$ have an order of magnitude of about unity. Because of the high stretching ratios used in meshes for RANS simulation, cell volumes in a given mesh can span ten orders of magnitude. This discrepancy in the numerical values causes problems with the solution algorithms, effectively pushing them toward reducing the residuals of cells with larger volumes while leaving the smaller cells unaffected. To avoid this problem, we scale the residuals of each cell by the reciprocal of the cell volume. Finally, we scale the turbulent residuals by 10^4 to ensure adequate scaling between the turbulence and mean flow residuals. The modifications presented here prevent possible numerical issues that can arise with the solution algorithm we describe below. Because the residuals are scaled within the residual formulations introduced earlier, we do not need additional considerations when solving linear systems arising from the discretization of the flow equations. This scaling strategy is similar to that proposed by Chisholm and Zingg [19].

3 Approximate Newton–Krylov Algorithm

For each iteration of the ANK solver, the update to the state vector is calculated as follows:

$$\left[\frac{\mathcal{I}}{\Delta t^{(n)}} + \left(\frac{\partial \mathcal{R}_m}{\partial \mathcal{Q}}\right)^{(n)}\right] \Delta \mathcal{Q}^{(n)} = -\mathcal{R}_0(\mathcal{Q}^{(n)}), \tag{8}$$

where the subscript m is the level of approximation used for the Jacobian matrix and the superscripts denote the values evaluated at the n^{th} iteration. The term $\mathcal{I}/\Delta t^{(n)}$ is a short hand notation for the diagonal time stepping term that we include, which is computed to give the same CFL number in every cell. We determine the time steps by selecting a global CFL number.

The rest of this section describes the important aspects of the implementation of the ANK. We highlight our contributions to the algorithm in Section 3.2, which describes the use of approximate residual computations with matrix-free operations, and in Section 3.4, which introduces an adaptive PC-lagging algorithm. For each component in the solver, we present the default solver parameters in the text and, in the last subsection, we summarize the default parameters used in this work.

3.1 Linear Solver Algorithm

To solve the linear system (8), we use the generalized minimum residual (GMRES) method [41]. This algorithm requires only the Jacobian-vector products, rather than a full Jacobian, leading to Jacobian-free Newton–Krylov (JFNK) methods [15]. In this work, we extend the JFNK method to the ANK solver.

To obtain vector products with the Jacobian for the GMRES algorithm, we use

$$\left(\frac{\partial \mathcal{R}_m}{\partial \mathcal{Q}}\right)^{(n)} v \approx \frac{\mathcal{R}_m(\mathcal{Q}^{(n)} + \epsilon v) - \mathcal{R}_m(\mathcal{Q}^{(n)})}{\epsilon}, \tag{9}$$

which is equivalent to a forward finite-difference formulation in the direction v. The subscript m denotes the residual approximation level that is used for the matrix-free operations. With this approach, we can select the residual formulation for the ANK solver during runtime with the same Jacobian-free formulation used with the JFNK methods. Finally, we determine the finite-difference step size by using [42]

$$\epsilon = \begin{cases} e_{\text{rel}} v^T Q^{(n)} / ||v||_2^2 & \text{if} \quad |v^T Q^{(n)}| > u_{\text{min}} ||v||_1 \\ e_{\text{rel}} u_{\text{min}} \text{sign}(v^T Q^{(n)}) ||v||_1 / ||v||_2^2 & \text{otherwise,} \end{cases} \tag{10}$$

where we set $u_{\text{min}} = 10^{-6}$ and $e_{\text{rel}} = 10^{-8}$.

We use a zero vector for the initial guess of the linear solver and do not use previous linear solutions as the initial guess for the next linear solution. The linear system we solve does not change drastically between nonlinear iterations. If the right-hand-side vector (i.e., the residual vector in our case) remains similar, restarting the linear solution from a previous solution would improve linear convergence. However, because we are performing linear solutions within a Newton iteration to solve a system of nonlinear equations, the subsequent solutions differ for each nonlinear iteration because the update direction that decreases the residual norm changes between nonlinear iterations. Therefore, no benefit is gained from restarting linear solutions between nonlinear updates.

3.2 Approximate Jacobian

The first main contribution of this work is to use the approximate residual computations within the matrix-free formulation to multiply the approximate Jacobian with a vector. Osusky and Zingg [20] showed that the use of an approximate Jacobian during the startup phase is an effective approach for three-dimensional RANS simulations. However, by storing the approximate Jacobian, they incur a large computational cost if it is recomputed in every iteration. One way to alleviate this computational burden is to lag the approximate Jacobian for a pre-determined number of iterations; however, this approach introduces problems involving the robustness of the solver. The optimal lag depends strongly on the problem and, for RANS simulations with complex three-dimensional geometries, there might be no choice but to recompute this approximate Jacobian for every nonlinear iteration [10, 18, 20, 21].

Our approach eliminates this robustness problem by using a matrix-free formulation for the approximate Jacobian, which we obtain by using the different levels of

approximate residual formulations introduced earlier. When using the matrix-free formulation, the approximate Jacobian in the nonlinear iterations is always up to date, without incurring the burden of formulating and storing the entries of the Jacobian matrix at each iteration. Furthermore, by using larger stencil sizes, we can incorporate the higher levels of approximate Jacobian formulations than just the first-order Jacobian, without incurring the additional memory cost of storing more entries in the matrix.

With the proposed formulation, the true Jacobian corresponds to using the matrix-free approach with $\mathcal{R}_0$. For the startup strategy, Osusky and Zingg [20] use similar approximations introduced with $\mathcal{R}_2$ to obtain a first-order Jacobian. The first-order Jacobian is also used for preconditioning in the NK or adjoint solvers [18, 32, 40]. In addition to the first-order and the true Jacobian formulations, we introduce the intermediate approximation $\mathcal{R}_1$ described in Section 2.3. Since the matrix-free formulation is completely general, we can select the level of approximation in our matrix-free operations by modifying the CFD solver options.

Section 5 investigates the effects of different levels of approximations as the main driver of the ANK solver. Little to no approximation is expected to yield a better nonlinear convergence, but with a more costly linear solution due to the increased bandwidth of the Jacobian. Conversely, a heavily approximated Jacobian should yield a lower nonlinear convergence rate, but with faster linear solutions due to the higher diagonal dominance. We use the $\mathcal{R}_1$ approximation as the default matrix-free operation in the ANK algorithm because it provides a good balance between accuracy and Jacobian stiffness.

3.3 Preconditioning

In the context of three-dimensional RANS simulations, adequate preconditioning is required for the GMRES solver to be a practical choice [15]. To faithfully represent the residual in the linear systems, we use right preconditioning, which can be written as

$$\left[\left(\frac{\mathcal{I}}{\Delta t}+\frac{\partial \mathcal{R}_m}{\partial \mathcal{Q}}\right)\mathbf{M}^{-1}\right](\mathbf{M}\Delta\mathcal{Q}) = -\mathcal{R}_0, \quad \mathbf{M}=\left(\frac{\mathcal{I}}{\Delta t}+\frac{\partial \mathcal{R}_2}{\partial \mathcal{Q}}\right), \tag{11}$$

where superscripts denoting the iteration number are omitted for simplicity. In this formulation, $\mathbf{M}^{-1}$ is not an exact inverse of $\mathbf{M}$ but is an approximate factorization. We use the additive Schwarz method (ASM) to split the problem into sub-domains, where we locally apply reverse Cuthill–McKee reordering and incomplete LU (ILU) factorization to approximate the inverse of $\mathbf{M}$. All of the algorithms mentioned here are available in the software library of the Portable, Extensible Toolkit for Scientific Computation (PETSc) [43**?** , 44], which we use in our CFD solver.

In the context of a flow or adjoint solver, an approximate Jacobian is often used to construct the PC matrix $\mathbf{M}$, resulting in lower memory requirements [20, 32, 40]. We also follow this convention for the ANK solver. To compute $\mathbf{M}$, we use the most approximate residual formulation ($\mathcal{R}_2$). We store the matrix $\mathbf{M}$ and its approximate inverse $\mathbf{M}^{-1}$. However, since we use $\mathcal{R}_2$ to build this PC matrix, the cost of storing it is low compared with more accurate Jacobian formulations that have a larger bandwidth.

In our computational framework, we compute $\partial \mathcal{R}_2 / \partial \mathcal{Q}$ by using either finite differences or algorithmic differentiation, as described by Kenway et al. [32]. We use an efficient coloring acceleration scheme described by Lyu et al. [40] to perturb the states for a finite-difference computation, or set the seeds for algorithmic differentiation. Goldfarb and Toint [45] describe the coloring scheme for a seven-point three-dimensional stencil, and we use this approach for matrix-based approximate Jacobian matrices constructed by using $\mathcal{R}_2$. This approach is generalizable because we only need the approximate residual computations and their respective stencils to perform efficient coloring of the states. In this work, we only use finite-difference computations for the PCs used with the ANK solver.

3.4 Adaptive Preconditioner Lagging

The second main contribution of this work is the adaptive PC-lagging algorithm that we developed. Forming and factorizing $\mathbf{M}$ are costly operations and are not done for each nonlinear iteration. Typically, a JFNK solver stores the factorized form of the approximate Jacobian and re-uses it for some predetermined iteration count [15]. This procedure is usually referred to as "Jacobian lagging." In an ANK solver without the matrix-free formulation, both the approximate Jacobian and its factorized form are lagged relative to the flow variables [20]. In a recent review, Witherden et al. [10] noted that the Jacobian is typically refreshed every 3–5 iterations. In our experience, the optimal lag is case dependent and determining it *a priori* is not straightforward. As a result, users typically set a lower lag, since a higher lag might destabilize the simulation. Furthermore, a single lag is used for different stages of convergence, and this lag is limited by the most problematic iterations, thereby becoming a bottleneck for the whole simulation.

Our matrix-free formulation overcomes these shortcomings. Even if the same approximate residual formulations are used for the PC and the linear system, we eliminate the robustness issues that arise due to an outdated Jacobian because the actual linear system is up to date for every nonlinear iteration, since we only lag the PC. As a result, we call this method "PC lagging" instead of "Jacobian lagging."

To determine the optimal lagging for the PC, we monitor the linear convergence for each nonlinear iteration. Generally, the PC performance deteriorates for each nonlinear iteration where it is not refreshed. As a result, the GMRES algorithm requires more iterations to reach the desired linear solution tolerance in each subsequent nonlinear iteration. Therefore, if the linear solution in a particular nonlinear iteration reaches the maximum iteration limit for the linear solver, and the linear solution in the previous nonlinear iteration was successful, then we refresh the PC. This approach provides several advantages: If the linear solution is proceeding well, we do not update the PC and avoid the unnecessary cost of forming and factorizing an approximate Jacobian. As the linear solution begins to lose effectiveness, we refresh the PC to maintain overall performance and ensure that the linear solutions converge properly.

However, even with an up-to-date PC, if the linear system fails to converge to the prescribed tolerance within the maximum iteration limit, we simply terminate the linear solution and proceed to the next nonlinear iteration to keep the computational

cost under control. This approach may yield a sequence of linear solutions that fails to reach the prescribed tolerance. Setting a good value for this tolerance is critical and we explain how to do so in the next subsection. In most cases that we have experimented with, the linear solver still achieves a relative convergence of around 0.1 before terminating due to the linear iteration limit. Although we aim for a slightly lower tolerance for the linear solver, this tolerance can also yield updates that are sufficiently accurate to achieve nonlinear convergence. In some rare cases, the linear solver stagnates and may fail to reach these more relaxed levels. In this case, the nonlinear convergence stalls due to ineffective updates. However, this behavior is caused by inadequate preconditioning, and the adaptive PC-lagging algorithm cannot avoid this problem. To overcome these failures, we restart the simulations with linear solver options that yield better PCs and higher iteration limits.

Based on the last nonlinear iteration when the PC was formed, we refresh the PC if the desired relative nonlinear convergence is reached. We set this relative convergence value at $\eta_{\text{pc}} = 0.5$. This condition aides convergence during the initial stages, when the flow field changes rapidly, and the performance of the linear solver benefits from an up-to-date PC [13]. We refer to this method as "adaptive PC lagging." Finally, as a safety check, we recompute the PC if it was lagged for $n_{\text{lag}} = 10$ nonlinear iterations. Section 3.9 presents the full lagging algorithm along with the default solver parameters.

3.5 Linear Solution Tolerance

Within each nonlinear iteration, we solve the linear system defined in Equation (8) to a specified relative tolerance:

$$\eta_{\text{lin}} \geq \frac{R_{\text{lin}}^{(n)}}{\left|\left|\mathcal{R}_0^{(n)}\right|\right|_2}, \tag{12}$$

where the linear residual $R_{\text{lin}}^{(n)}$ is defined as

$$R_{\text{lin}}^{(n)} = \left|\left|\left(\frac{\mathcal{I}}{\Delta t} + \frac{\partial \mathcal{R}_m}{\partial \mathcal{Q}}\right)^{(n)} \Delta \mathcal{Q}^{(n)} + \mathcal{R}_0^{(n)}\right|\right|_2. \tag{13}$$

The performance and robustness of the ANK solver algorithm is a strong function of η_{lin} because the cost and accuracy of each nonlinear update $\Delta \mathcal{Q}^{(n)}$ directly depend on it [46]. A low η_{lin} yields a more robust solver algorithm but leads to more expensive linear solutions. Conversely, a high η_{lin} means that the solver can perform more nonlinear iterations in a given time because each linear solution would cost less. The values for η_{lin} reported in the literature fall in the range $(10^{-4}, 10^{-1})$ [18, 20, 28]. The proposed approach involves doing many low-cost nonlinear iterations during the startup stage, so we select $\eta_{\text{lin}} = 0.05$, which is the largest value that consistently provides an adequate update without destabilizing the solver.

With the ANK solver, we always use a fixed tolerance for the linear solution. However, Eisenstat and Walker [47] developed a method that adaptively adjusts the tolerance of the linear solver to achieve optimal nonlinear convergence while avoiding over-solving the linear system. This method is useful when using a pure Newton

method because it is tuned with respect to the expected nonlinear convergence from an update computed by using a full Jacobian. Because we introduce approximations in the Jacobian, the resulting nonlinear convergence with each iteration is less than what this method expects. As a result, the Eisenstat–Walker method predicts a higher linear solver tolerance for the next nonlinear iteration. This approach is not well suited for the startup stage because a high linear solver tolerance might lead to inaccurate updates, whereas our main goal is robustness. Therefore, we preserve a constant linear solver tolerance for robustness.

Ideally, we would want the linear solver to achieve the prescribed convergence tolerance for each nonlinear iteration. However, due to an outdated PC, the linear solver may require too many iterations to reach the prescribed tolerance. To avoid wasting computational effort, we set a maximum iteration limit for the GMRES algorithm that is sufficiently large so that most of the linear solutions converge successfully. However, the limit is not so large as to allow wasted computation during the linear solution, because taking another nonlinear step is often more beneficial. This is a problem dependent parameter, but for many practical cases of interest we find that an iteration limit of $n_{\text{lin}} = 40$ is a good default. For rare cases that repeatedly fail to reach suitable linear convergence levels, we restart the simulations with either higher iteration limits, stronger preconditioners, or both.

3.6 Solution Update

During each nonlinear iteration, the state vector is updated as

$$\mathcal{Q}^{(n+1)} = \mathcal{Q}^{(n)} + \omega^{(n)} \Delta \mathcal{Q}^{(n)}, \tag{14}$$

where $\omega^{(n)} \in [0, 1]$ is the relaxation factor. To determine a suitable value for $\omega^{(n)}$ for each iteration, we follow the approach of Modisette [28].

We use a physicality check to constrain the state by using

$$\omega_{\text{phys}} = \left[\max \left(\left\{ \left| \frac{\Delta \mathcal{Q}_{i,l}}{\theta_{\text{phys},l} \mathcal{Q}_{i,l}} \right| ; \forall \ (i \in n_{\text{cell}}) \, , \, (l \in l_{\text{phys}}) \right\} , \, 1 \right) \right]^{-1} , \tag{15}$$

where the subscripts i, l refer to the cell index and the state component index in the cell, respectively, n_{cell} is the global cell count, and l_{phys} are the variable indices subject to the physicality check. By default, we select $\theta_{\text{phys},\rho} = \theta_{\text{phys},E} = 0.2$. The physicality check for the SA working variable is modified compared with density and energy. We use $\theta_{\text{phys},\tilde{\nu}} = 0.99$ by default and only check updates that are in the negative direction. This modification enables the SA variable to increase rapidly if necessary and prevents it from taking on a negative value, which would destabilize the solution. We only check for updates to density, energy, and the SA model working variable $\tilde{\nu}$ because the momentum must be allowed to switch signs. Because the state and the update vectors are distributed among processes, we use Algorithm 1 to determine ω_{phys}.

After the physicality check, we use a line search to determine the maximum step size that would reduce the unsteady residual norm. The unsteady residual norm is defined as

$$R_u^{(n)} = \left|\left| \omega^{(n)} \frac{\mathcal{I}}{\Delta t^{(n)}} \Delta \mathcal{Q}^{(n)} + \mathcal{R}_0 \left(\mathcal{Q}^{(n)} + \omega^{(n)} \Delta \mathcal{Q}^{(n)} \right) \right|\right|_2 . \tag{16}$$

Algorithm 1 Physicality check

$\omega_{\text{local}} \leftarrow 1$ ▷ Initialize the local step to 1
for i **in** $n_{\text{cells,local}}$ **do** ▷ Loop over every cell owned by this process
  for l **in** $[l_\rho, l_E]$ **do** ▷ Loop over density and energy variables
    $\omega_{\text{local}} \leftarrow \left[\max\left(\left|\frac{\Delta\mathcal{Q}_{i,l}}{\theta_{\text{phys},l}\mathcal{Q}_{i,l}}\right|, \frac{1}{\omega_{\text{local}}}\right)\right]^{-1}$ ▷ Limit the step such that $|\Delta\mathcal{Q}_{i,l} < \theta_{\text{phys},l}\mathcal{Q}_{i,l}|$
  for $l = l_{\tilde{\nu}}$ **do** ▷ Check the SA model working variable
    if $\Delta\mathcal{Q}_{i,l} < 0$ **then** ▷ Only check negative updates
      $\omega_{\text{local}} \leftarrow \left[\max\left(\left|\frac{\Delta\mathcal{Q}_{i,l}}{\theta_{\text{phys},l}\mathcal{Q}_{i,l}}\right|, \frac{1}{\omega_{\text{local}}}\right)\right]^{-1}$ ▷ Limit the step such that $|\Delta\mathcal{Q}_{i,l} < \theta_{\text{phys},l}\mathcal{Q}_{i,l}|$
$\omega_{\text{phys}} \leftarrow \text{MPI\_Allreduce}(\omega_{\text{local}}, \min)$ ▷ Communicate to determine the global minimum ω_{local}

Starting with ω_{phys}, we backtrack to find $\omega^{(n)}$ such that

$$R_u^{(n)} \leq \left|\left|\mathcal{R}_0\left(\mathcal{Q}^{(n)}\right)\right|\right|_2 . \tag{17}$$

Algorithm 2 details the procedure of the backtracking line search. This check acts as a safety measure against failed linear solutions or bad update directions. The default value for θ_{bt} is 0.7; however, this parameter is just a generic factor for the backtracking line search algorithm.

The backtracking line search algorithm relies on the updates to the state being in the direction of descent for the unsteady residual norm. An exact Jacobian constructed with $\mathcal{R}_0$ is guaranteed to yield an update with this property. However, we cannot guarantee such a property with the approximate Jacobians constructed with $\mathcal{R}_1$ and $\mathcal{R}_2$. For most practical cases of interest, the approximate Jacobians do indeed generate updates that are in the descent direction. However, we sometimes see updates that fail to reduce the unsteady residual norm. If this happens, the solver algorithm modifies the global CFL number, as detailed in Section 3.7, and moves on to the next nonlinear iteration. This is done to increase the weight of the time stepping term with respect to the approximate Jacobian.

Finally, if these methods predict a value for $\omega^{(n)}$ that is less than a specified value ω_{min}, then $\omega^{(n)}$ is set to zero and the solver proceeds to the next nonlinear iteration without changing the state variables. In this scenario, the next nonlinear iteration changes the CFL number and refreshes the PC to avoid the bad update in the previous iteration. If the CFL number is at the lower limit, the solver simply takes the step given by $\omega^{(n)}$, even though the step size might be smaller than ω_{min}. We select the default value $\omega_{min} = 0.01$.

Algorithm 2 Backtracking line search

$\omega_u \leftarrow \omega_{\text{phys}}$ ▷ Initialize ω_u with the result from physicality check
$R_u^{(n)} \leftarrow R_u\left(\mathcal{Q}^{(n)}, \Delta\mathcal{Q}^{(n)}, \Delta t^{(n)}, \omega_u\right)$ ▷ Compute the unsteady residual norm with the current step size
if $R_u^{(n)} > \left|\left|\mathcal{R}_0\left(\mathcal{Q}^{(n)}\right)\right|\right|_2$ **then** ▷ Check if the unsteady residual has increased
  while $R_u^{(n)} > \left|\left|\mathcal{R}_0\left(\mathcal{Q}^{(n)}\right)\right|\right|_2$ **and** $\omega_u > \omega_{\text{min}}$ **do** ▷ Repeat until unsteady residual norm has decreased
    $\omega_u \leftarrow \omega_u\theta_{bt}, \quad \theta_{bt} \in (0,1)$ ▷ Backtrack to find a lower ω_u
    $R_u^{(n)} \leftarrow R_u\left(\mathcal{Q}^{(n)}, \Delta\mathcal{Q}^{(n)}, \Delta t^{(n)}, \omega_u\right)$ ▷ Compute the new unsteady residual norm
$\omega^{(n)} \leftarrow \omega_u$ ▷ Set the step size for this nonlinear iteration

3.7 Evolution of Pseudo-Time Step

We follow the approach of Ceze and Fidkowski [29] and use the monotonic variant of the residual difference method (mRDM) based on the RDM introduced by Bücker et al. [48]. In addition to the mRDM algorithm, we enforce lower and upper limits for the CFL number. The upper limit guarantees that the linear system has sufficient diagonal dominance in each nonlinear iteration. We use a constant upper limit throughout the solution process. The lower limit prevents the CFL number from going below a moderate value, given the current nonlinear convergence relative to the freestream conditions. We determine this lower limit by using the successive evolution relaxation (SER) algorithm of van Leer and Mulder [49]. Because we use this solver in an optimization framework, CFD simulations are not always started from a freestream initial condition. Instead, we use warm starts, where the solution from the previous design iteration serves to initialize the flow field. With warm starts, the solver starts with a higher CFL number than does a cold start with freestream conditions. Therefore, the lower CFL limit also serves to select an appropriate initial CFL during consecutive warm starts.

Because we lag the PC, the time step is only updated during the iterations where we update the PC. This ensures that the time stepping term included in the PC is consistent with the matrix-free operations, which improves the performance of the linear solver.

The baseline mRDM algorithm can be written as

$$\mathrm{CFL}^{(n)} = \mathrm{CFL}^{(k)} \alpha^{\gamma}, \quad \gamma = \max\left(\frac{\left|\left|\mathcal{R}_0^{(k)}\right|\right|_2 - \left|\left|\mathcal{R}_0^{(n)}\right|\right|_2}{\left|\left|\mathcal{R}_0^{(k)}\right|\right|_2}, 0 \right), \tag{18}$$

and the minimum CFL number is

$$\mathrm{CFL}_{\min}^{(n)} = \left(\frac{\left|\left|\mathcal{R}_0^{(\mathrm{fs})}\right|\right|_2}{\left|\left|\mathcal{R}_0^{(n)}\right|\right|_2} \right)^{\beta}, \tag{19}$$

where the superscript (k) refers to the last iteration during which the PC and the CFL number were updated. The term $\left|\left|\mathcal{R}_0^{(\mathrm{fs})}\right|\right|_2$ represents the total residual norm with the freestream conditions, and we select $\mathrm{CFL}^{(0)} = 5$ to initialize the CFL number. We use the default values of $\mathrm{CFL}_{\max} = 10^5$, $\alpha = 10$, and $\beta = 0.5$.

During the nonlinear iterations where the PC is updated, we determine the CFL number by considering the step size taken in the previous iteration,

$$\mathrm{CFL}^{(n)} = \begin{cases} \min\left(\mathrm{CFL}^{(k)} \alpha^{\gamma}, \mathrm{CFL}_{\max}\right), \gamma = \max\left(\frac{\left|\left|\mathcal{R}_0^{(k)}\right|\right|_2 - \left|\left|\mathcal{R}_0^{(n)}\right|\right|_2}{\left|\left|\mathcal{R}_0^{(k)}\right|\right|_2}, 0 \right), & \text{if } \omega^{(n-1)} > \omega_{\mathrm{ramp}} \\ \max\left(\mathrm{CFL}^{(k)}, \mathrm{CFL}_{\min}^{(n)}\right), & \text{else if } \omega_{\mathrm{ramp}} \geq \omega^{(n-1)} > \omega_{\min} \\ \max\left(\theta_{\mathrm{red}} \mathrm{CFL}^{(k)}, \mathrm{CFL}_{\min}^{(n)}\right), & \text{else.} \end{cases} \tag{20}$$

This formulation uses the mRDM algorithm to ramp up the CFL number if the previous update was not heavily limited and enforces the upper and lower limits to the CFL

number. The variable $\omega_{\text{ramp}} = 0.4$ determines the minimum step size required to increase the CFL number, and $\omega_{\min} = 0.01$ is the minimum step size below which the CFL number is reduced by $\theta_{\text{red}} = 0.5$. This formulation is used only during iterations in which we refresh the PC. For any other nonlinear iteration in which the PC is lagged, we retain the CFL number from the previous iteration.

3.8 Inclusion of Turbulence Model

The inclusion of the turbulence model in the set of mean flow equations introduces new issues in the nonlinear solution process. These issues arise largely because of the discrepancy in the scaling of variables and residuals between the SA model working variable $\tilde{\nu}$ and the flow variables ρ, ρu, ρv, ρw, E. Osusky and Zingg [20] addressed this issue by introducing automatic scaling of the turbulence working variable with respect to the maximum value obtained in the computational domain. In addition to the scaling, the convergence characteristics of these variables also differ significantly, because the SA model residual often varies wildly during the initial stages of convergence. Chauhan et al. [50] showed that the optimal algorithm for converging tightly coupled systems might change depending on the convergence stage. Initially, a Gauss–Seidel approach might be faster, whereas Newton-type algorithms are expected to converge faster toward the final stages. The characteristics of the SA model convergence also depend on implementation details and the selected variant of the model, further increasing complexity and making it more difficult to find a set of parameters that works for a wide range of cases.

The proposed approach is to decouple the turbulence variable from the flow variables during the ANK stage and to update $\tilde{\nu}$ after updating the flow variables in each nonlinear iteration in a nonlinear block Gauss–Seidel fashion. This approach allows us to form two separate linear systems: one for the flow variables consisting of 5×5 block entries and a second one for the turbulence model variable only. The algorithms described previously determine the update for the flow variables with the frozen turbulence assumption. After the flow variables are updated, a similar process is repeated for the turbulence model, where we assume that the flow variables are frozen and obtain an update vector for the turbulence model.

This approach has two advantages: First of all, by decoupling the flow variables from the turbulence model variable, we recover the favorable convergence characteristics of the flow variables. This way, the turbulence model residuals are allowed to vary without adversely affecting the convergence of the flow variables. Second, decoupling the turbulence variable overcomes the challenges caused by the scaling of variables and residuals. This improves the conditioning of the linear systems under consideration, which reduces the cost of the linear solution. Furthermore, we can monitor the linear solver performance of the turbulence solver and only update the PC used for the turbulence model while keeping a lagged PC for the flow variables because the PC for the turbulence model often requires more frequent updates. However, we still use a fully coupled approach for the final stage of convergence with our NK solver. Once the initial transients settle, the scaling introduced in the discretization section suffices to couple the turbulence variable to the flow variables.

3.9 The Complete ANK Algorithm

Algorithm 3 details the complete ANK solver. The main nonlinear loop (line 7) iterates the states until the relative convergence η_{ANK} of the ANK solver is achieved. We use the subscript f for quantities composed only of flow variables (ρ, ρe, ρv, ρw, E) and of t for the turbulence model variable $\tilde{\nu}$. The lack of subscript means that the term contains entries with all the variables in the state vector. The superscript n gives the current iteration and k gives the last iteration when the flow PC was updated.

Algorithm 3 is the approach we use by default, but many other variations of this algorithm are possible. For example, a version that couples flow and turbulence would omit the operations with subscript t and solve the flow and turbulence variables as a single linear system. With the Jacobian-free approach, we can also select the level of approximation to use for the matrix-free operations in the linear solutions. As noted in lines 14 and 23 of Algorithm 3, we use $\mathcal{R}_1$ by default; however, the same algorithm can be used with $\mathcal{R}_0$ or $\mathcal{R}_2$ for the matrix-free operations.

Many tunable parameters appear in the ANK algorithm, and the default set of parameters might require modifications for certain simulations where the solver fails. Generating a default set of parameters that would work optimally with every CFD simulation is essentially impossible. Given a set of default parameters that are close to optimal, users can make the necessary adjustments to the CFD solver for poorly converging cases instead of modifying the case itself [51]. A critical trade-off thus exists between robustness and performance. Although conservative settings that yield extremely robust algorithms can be obtained, they would yield highly sub-optimal performance for many well-behaving cases. Conversely, a failed solution that requires user intervention decreases the efficiency of the overall optimization process and slows the design procedure.

Having many tunable parameters in a solver algorithm is beneficial in terms of robustness and flexibility. Not having tunable parameters, or having fewer tunable parameters, might seem advantageous because of the low cost of the initial setup. However, with these solvers, users cannot modify the algorithms when a simulation fails. Conversely, solvers with tunable parameters are more flexible in handling these failed cases, because these parameters can be modified on a case-by-case basis for failed simulations.

Table 2 lists the default values for the solver parameters. In Table 2, we grouped the options with respect to the components in the solver algorithm to facilitate the correspondence of these parameters to other CFD codes that might not have this exact set of parameters. These parameters are tuned for three-dimensional RANS simulations of full or partial aircraft configurations in transonic conditions with meshes composed of millions to tens of millions of cells, which give mesh sizes that are appropriate for studies of aerodynamic shapes and for aerostructural optimization of aircraft geometries [6, 8, 52, 53]. The parameter values listed in Table 2 are also expected to yield satisfactory results for subsonic and two-dimensional aeronautical applications; however, meshes that are much smaller or much larger can yield sub-optimal performance, so minor adjustments to the parameters might be necessary.

A few scenarios exist in which the user would need to modify this default set

Algorithm 3 ANK solver

$n \leftarrow 1$ ▷ Initialize nonlinear iteration counter
$n_{\mathrm{pc},f} \leftarrow 0$ ▷ Initialize iteration counter for flow PC lagging
$k \leftarrow 0$ ▷ Initialize the last iteration for which the PC was updated
$\mathrm{CFL}^{(0)} \leftarrow 5.0$ ▷ Initialize CFL number
$\omega^{(0)} \leftarrow 1.0$ ▷ Initialize step factor
$\mathcal{R}_0^{(0)} \leftarrow \mathcal{R}_0(\mathcal{Q}^{(0)})$ ▷ Compute initial residual
while $||\mathcal{R}_0^{(n)}||_2 > \eta_{\mathrm{ANK}}||\mathcal{R}_0^{(0)}||_2$ **do** ▷ Iterate until relative convergence is achieved
  if $\mathrm{mod}\ (n_{\mathrm{pc},f}, n_{\mathrm{lag}}) = 0$ **or** $||\mathcal{R}^{(n-1)}||_2 < \eta_{\mathrm{pc}}||\mathcal{R}^{(k)}||_2$ **then** ▷ Check if the PC needs to be updated
    $\mathrm{CFL}^{(n)} \leftarrow$ Equation (20) ▷ Update CFL number
    $\mathbf{M}_f^{-1} \leftarrow \left(\frac{\mathcal{I}}{\Delta t^{(n)}} + \frac{\partial \mathcal{R}_2}{\partial \mathcal{Q}^{(n)}}\right)_f^{-1}$ ▷ Update flow PC using $\mathcal{R}_2$ and factorize it
    $k \leftarrow n$ ▷ Record the iteration number when the flow PC was updated
    $n_{\mathrm{pc},f} \leftarrow 0$ ▷ Reset flow PC lag counter
    $n_{\mathrm{pc},t} \leftarrow 0$ ▷ Reset turbulence PC lag counter
  $\Delta \mathcal{Q}_f^{(n)} \leftarrow \mathrm{solve}\left(\frac{\mathcal{I}}{\Delta t^{(n)}} + \frac{\partial \mathcal{R}_1}{\partial \mathcal{Q}^{(n)}}\right)_f \Delta \mathcal{Q}_f^{(n)} = -\mathcal{R}_{0,f}^{(n)}$ ▷ Determine update to flow variables
  $\omega_{\mathrm{phys},f} \leftarrow$ Algorithm 1 ▷ Apply physicality check for ρ and E
  $\omega_f^{(n)} \leftarrow$ Algorithm 2 ▷ Backtrack to reduce unsteady flow residual norm
  if $\omega_f^{(n)} < \omega_{\mathrm{min}}$ **and** $\mathrm{CFL}^n > \mathrm{CFL}_{\mathrm{min}}^n$ **then** ▷ Check if flow step is too small
    $\omega_f^{(n)} \leftarrow 0$ ▷ Reject step; try again with lower CFL
  $\mathcal{Q}_f^{(n+1)} \leftarrow \mathcal{Q}_f^{(n)} + \omega_f^{(n)} \Delta \mathcal{Q}_f^{(n)}$ ▷ Update flow variables
  if $n_{\mathrm{pc},t} = 0$ **then** ▷ Check if turbulence PC needs to be updated
    $\mathbf{M}_t^{-1} \leftarrow \left(\frac{\mathcal{I}}{\Delta t^{(n)}} + \frac{\partial \mathcal{R}_2}{\partial \mathcal{Q}^{(n)}}\right)_t^{-1}$ ▷ Update turbulence PC using $\mathcal{R}_2$ and factorize it
  $\mathcal{R}_{0,t}^{(n)} \leftarrow \mathcal{R}_{0,t}(\mathcal{Q}_f^{(n+1)}, \mathcal{Q}_t^{(n)})$ ▷ Compute turbulence residual with updated flow variables
  $\Delta \mathcal{Q}_t^{(n)} \leftarrow \mathrm{solve}\left(\frac{\mathcal{I}}{\Delta t^{(n)}} + \frac{\partial \mathcal{R}_1}{\partial \mathcal{Q}^{(n)}}\right)_t \Delta \mathcal{Q}_t^{(n)} = -\mathcal{R}_{0,t}^{(n)}$ ▷ Determine update to turbulence model variables
  $\omega_{\mathrm{phys},t} \leftarrow$ Algorithm 1 ▷ Apply physicality check for $\tilde{\nu}$ only
  $\omega_t^{(n)} \leftarrow$ Algorithm 2 ▷ Backtrack to reduce unsteady turbulence residual norm
  if $\omega_t^{(n)} < \omega_{\mathrm{min}}$ **then** ▷ Check if turbulence step is too small
    $\omega_t^{(n)} \leftarrow 0$ ▷ Reject step; try again with an updated turbulence PC
  $\mathcal{Q}_t^{(n+1)} \leftarrow \mathcal{Q}_t^{(n)} + \omega_t^{(n)} \Delta \mathcal{Q}_t^{(n)}$ ▷ Update turbulence variable
  $\mathcal{R}_0^{(n+1)} \leftarrow \mathcal{R}_0(\mathcal{Q}^{(n+1)})$ ▷ Update all residuals for next iteration
  $n_{\mathrm{pc},f} \leftarrow n_{\mathrm{pc},f} + 1$ ▷ Increment flow PC lag counter
  $n_{\mathrm{pc},t} \leftarrow n_{\mathrm{pc},t} + 1$ ▷ Increment turbulence PC lag counter
  if $\omega_f^{(n)} < \omega_{\mathrm{min}}$ **or** $(\eta_{\mathrm{lin}}^{(n)} > \eta_{\mathrm{lin}}$ **and** $\eta_{\mathrm{lin}}^{(n-1)} \leq \eta_{\mathrm{lin}})$ **then** ▷ Check if the flow update broke down
    $n_{\mathrm{pc},f} \leftarrow 0$ ▷ Update flow PC in next iteration
  if $\omega_t^{(n)} < \omega_{\mathrm{min}}$ **or** $(\eta_{\mathrm{lin},t}^{(n)} > \eta_{\mathrm{lin}}$ **and** $\eta_{\mathrm{lin},t}^{(n-1)} \leq \eta_{\mathrm{lin}})$ **then** ▷ Check if the turbulence update broke down
    $n_{\mathrm{pc},t} \leftarrow 0$ ▷ Update turbulence PC in next iteration
  $n \leftarrow n + 1$ ▷ Increment nonlinear iteration counter

of parameters. For example, cases that contain massive separation or heavy shocks yield particularly ill-conditioned Jacobian matrices. In these cases, the default set of parameters may not be suitable, so users can create a stronger linear solver either by increasing the iteration limit, increasing the preconditioner strength, or both. As another example, the algorithm used for the terminal convergence might fail even after a relative convergence of 10^{-5}. When this happens, users can lower the η_{ANK} parameter to achieve a more robust overall solver.

4 Computational Framework

This work uses the CFD code ADflow, which is available under an open-source license.[3] ADflow is based on SUmb [54], which is an explicit solver that uses a Runge–Kutta time-marching scheme. The development of ADflow involved adding to SUmb the ANK solver described herein, a Jacobian-free adjoint solver [32], an overset mesh capability [38], and a Python interface to facilitate aerodynamic shape optimization [2] and multidisciplinary studies [8]. ADflow solves Euler, laminar Navier–Stokes, and Reynolds-averaged Navier–Stokes equations in steady, unsteady, and time-spectral modes and on both multiblock structured and overset meshes. The code is written in Fortran 90 and wrapped in Python for interfacing with the MDO of aircraft configurations with the high-fidelity framework [4]. ADflow has a discrete adjoint implementation [32] which has been effective in aerodynamic shape optimization studies [2, 7, 9, 55–59], as well as aerostructural optimization studies [5, 60–62]. We use the PETSc software library [43**?** , 44] to implement the GMRES algorithm, preconditioning techniques, and matrix-free computations.

Instead of using a single solver algorithm for the entire simulation, ADflow switches between algorithms depending on the relative convergence of the residual norm. This is done to ensure that we obtain the best convergence rate attainable at different stages of convergence. The relative convergence can be monitored by using the relative reduction in the residual norm going into iteration n with respect to the freestream value; namely,

$$\eta_{\text{rel}}^{(n)} = \frac{||\mathcal{R}_0^{(n)}||_2}{||\mathcal{R}_0^{(\text{fs})}||_2}. \tag{21}$$

[3] https://github.com/mdolab/adflow, accessed June 2019

Table 2: Default values for the tunable parameters.

Component	Description	Symbol	Value
Linear solver	Approximation level for matrix-free operations	$\mathcal{R}_m$	$\mathcal{R}_1$
	ILU fill level for PCs	–	2
	ASM overlap for PCs	–	1
	Linear solution convergence tolerance	η_{lin}	0.05
	Iteration limit for the GMRES solver	n_{lin}	40
PC lagging	Relative convergence for PC update	η_{pc}	0.5
	Maximum PC lag value	n_{lag}	10
Step size	Maximum allowed change for ρ and E variables	$\theta_{\text{phys},\rho} = \theta_{\text{phys},E}$	0.20
	Maximum allowed decrease for $\tilde{\nu}$	$\theta_{\text{phys},\tilde{\nu}}$	0.99
	Backtracking line search factor	θ_{bt}	0.7
Pseudo-transient continuation	RDM factor for CFL number	α	10
	SER exponent for minimum CFL number	β	0.5
	Initial CFL number	$\text{CFL}^{(0)}$	5
	Maximum CFL number	CFL_{max}	10^5
	CFL cutback factor	θ_{red}	0.5
	Step size for CFL ramp threshold	ω_{ramp}	0.4
	Minimum allowed step size	ω_{min}	0.01
Nonlinear convergence	Relative convergence tolerance with ANK solver	η_{ANK}	10^{-5}
	Absolute convergence tolerance for simulations	η_{abs}	10^{-12}

ADflow contains the following solvers:

- RK: A five-stage fourth-order accurate low-memory Runge–Kutta scheme. This is the only solver algorithm carried over from SUmb; all other algorithms were developed with ADflow.
- D3ADI: Diagonalized diagonally dominant alternating direction implicit scheme of Klopfer et al. [63].
- ANK: The solver algorithm proposed and described herein.
- NK: A fully coupled Jacobian-free Newton–Krylov algorithm. We formulate the PC for this solver in the same way as for the ANK solver. However, the NK solver always uses the exact residual formulation ($\mathcal{R}_0$) for the matrix-free operations. With this method, we do not include a time stepping term in the left-hand side of Equation (8), effectively yielding a CFL number of infinity. The solver uses the algorithm introduced by Eisenstat and Walker [47] to determine the linear solution tolerance for each nonlinear iteration.

ADflow uses the RK or D3ADI methods as smoothers in a multigrid solution procedure. These solvers are efficient for structured multiblock meshes but are not applicable to overset meshes. The ANK solver provides good convergence rates for both multiblock and overset meshes, making it the startup algorithm of choice for overset meshes. The NK solver yields the best convergence rate of all four algorithms in ADflow, provided that the initial guess is inside the basin of attraction. However, if the initial guess is far from the solution, the NK solver performs poorly and can diverge. As a result, we need to use one of the other algorithms as a startup method. In this work, for the sake of brevity, we focus on the ANK and NK algorithms. However, many combinations of these methods are possible because ADflow can switch between solver algorithms during each nonlinear iteration, which is not a common feature in CFD solvers.

We converge our simulations with the ANK solver until the residual norm decreases by five orders of magnitude ($\eta_{\text{rel}} = 10^{-5}$). Witherden et al. [10] suggested that $\eta_{\text{rel}} = 10^{-4}$ is suitable for stability with the NK solver algorithm. Although this value is usually a good guess for the NK solver, selecting a lower value for this relative convergence results in a smoother transition to the NK solver and prevents potential failures during the switch.

5 Results

This section presents the results for benchmarking the performance of the default solver algorithm and the range of possible ANK solver variations. All computational times are reported in *TauBendch* work units (WU).[4] For each computational architecture, we define one WU as the average result from 10 TauBench runs, with the

[4] http://www.ipacs-benchmark.org/index.php?s=download&unterseite=taubench, accessed June 2019

instructions described in the Third International Workshop on High-Order CFD Methods guidelines.[5] We ran the test 10 times on each architecture with the command `mpirun -np 1 ./TauBench -n 250000 -s 10` and report the average values as one WU for each architecture. Unless noted otherwise, we use the set of default parameter values listed in Table 2. We report the performance metrics for the ANK solver stage separately from the overall solver cost. The results for different stages are noted along with their respective relative convergences η_{rel}. Unless noted otherwise, results reported for $\eta_{\text{rel}} = 10^{-5}$ correspond to the ANK stage, while results for $\eta_{\text{rel}} = 10^{-12}$ correspond to the total values within a simulation. The performance metrics of interest are the number of nonlinear iterations (NIs), the cumulative number of linear iterations (LIs), and the WUs or thousands of WUs (kWUs) required to reach the stated relative convergence tolerances.

5.1 Baseline Performance of Solver Variations

To demonstrate the algorithm, we examine the performance with different ANK solver variations by using the ONERA M6 wing geometry. Recently, Gleize et al. [64] presented their computational studies on the M6 wing, and they fully describe the geometry that we use in this work. The computational mesh is a multiblock mesh with about eight million cells. The flow conditions are $M = 0.84$, $\text{Re} = 11.7 \times 10^6$, reference temperature $T_{\text{ref}} = 310.93$ K, and angle of attack $\alpha = 3.06^\circ$.

For these tests, we use six Ivybridge nodes on NASA's Pleiades supercomputer. Each node contains 20 cores, so each test uses 120 cores in total. One WU corresponds to 6.24 s of CPU time. We converge the simulations to $\eta_{\text{rel}} = 10^{-5}$ with the ANK solver and then switch to the NK solver for the rest of the convergence until $\eta_{\text{rel}} = 10^{-12}$.

We tested the three different approximation levels $\mathcal{R}_0$–$\mathcal{R}_2$ and tested each approximation level with both the decoupled and coupled turbulence method, for a total of six tests. The decoupled method is equivalent to Algorithm 3, and the coupled method marches both the flow variables and turbulence model variable in a coupled manner. After obtaining an update vector by solving the single linear system in the coupled mode, we use the physicality check and backtracking line search algorithms again to obtain the final update vector. From this perspective, the decoupled mode is analogous to a nonlinear block Gauss–Seidel approach where the blocks are updated by using Newton's method, whereas the coupled mode is representative of a stand-alone Newton's method.

Figure 3 shows the convergence for each variant of the ANK solver, and the corresponding numerical values are listed in Table 3. During the initial two-orders-of-magnitude convergence, each case performs almost identically, which we attribute to the CFL number being relatively low in this region, so the approximations have minimal effect. The decoupled mode using $\mathcal{R}_0$ performs the best in terms of the number of nonlinear iterations. However, because of the increased cost of the linear solution, this case performs worse in terms of wall time than the decoupled cases with approximate residual formulations ($\mathcal{R}_1$ and $\mathcal{R}_2$).

[5] https://www.grc.nasa.gov/hiocfd/guidelines/, accessed June 2019

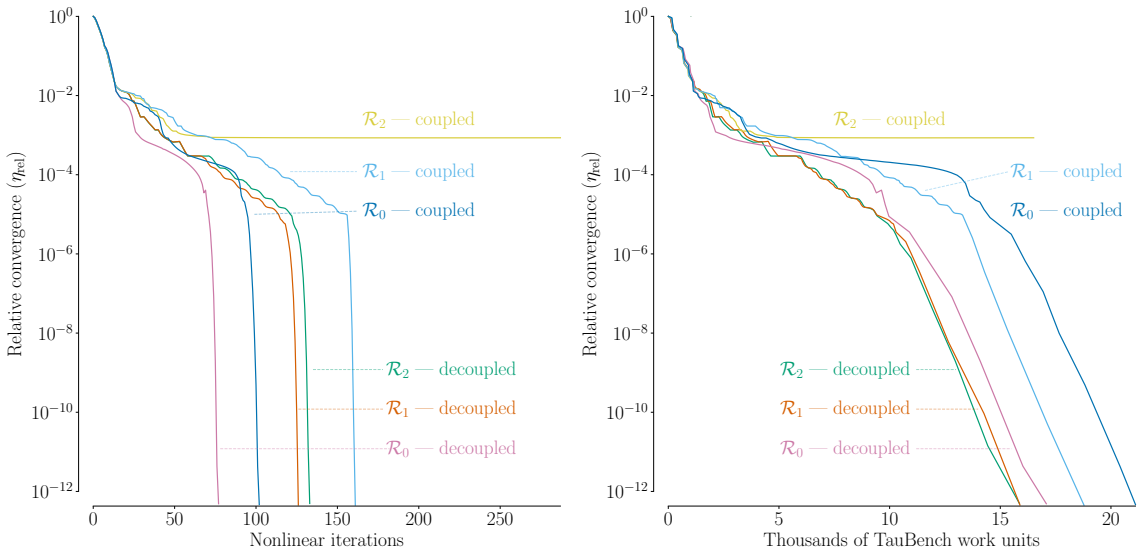

Figure 3: Relative convergence versus number of nonlinear iterations and thousands of TauBench WUs for the ONERA M6 wing. The $\mathcal{R}_m$ symbols denote the residual formulation used in the matrix-free operations. "Coupled" and "decoupled" represent how the turbulence model is marched in pseudo-time. While they require more nonlinear iterations, tests that use approximate residual formulations converge faster.

Similar trends occur with the coupled mode when we use the exact residual formulation $\mathcal{R}_0$. Between $\eta_{\text{rel}} = 10^{-2}$ and 10^{-4}, the convergence rate slows down, whereas the cases with approximate residual formulations do a better job in this region. Overall, the decoupled cases perform better in terms of wall time. Among the decoupled cases, those that use the approximations in the matrix-free operations ($\mathcal{R}_1$ and $\mathcal{R}_2$) perform better than when $\mathcal{R}_0$ is used. These results demonstrate the advantages of marching the flow and turbulence variables separately and of using approximate residual formulations in matrix-free operations.

5.2 Effect of Lagged Preconditioner

To study the effects of lagging the PC, we use the test case introduced in the previous subsection with the identical setup and the default ANK solver (Algorithm 3). To monitor these effects, we consider the eigenvalue spectra of the preconditioned matrix-free operators. These spectral analyses are similar to those used by Qin et al. [65] to compare the stand-alone and preconditioned linear systems in the context of hypersonic viscous simulations. An effective PC is expected to cluster the eigenvalues and thus improve the linear solver performance. As we lag the PC during nonlinear iterations and it becomes outdated, we expect the favorable clustering properties to diminish. By monitoring how the eigenvalue spectra change as the solver takes nonlinear steps without updating the PC, we demonstrate how the lagging algorithm affects the linear solver performance.

The ANK solver goes through three different phases to reach $\eta_{\text{rel}} = 10^{-5}$, as shown

in Figure 3. During the initial phase, which we define as going from a freestream initial conditions to $\eta_{\text{rel}} = 10^{-2}$, the global CFL number is relatively small. Here, all six solver variations performed almost identically, both in terms of convergence per nonlinear iteration count and wall time. In this stage, strong transients exist with large changes to the flow field, especially near the no-slip surfaces that represent the aircraft geometry. The intermediate stage can be defined as the convergence from $\eta_{\text{rel}} = 10^{-2}$ to about 10^{-4}. During this stage, the CFL number is relatively large and the far-field boundary conditions interact with the no-slip surfaces. In this stage, the turbulence model variables go through strong transients. The final stage of the startup can be defined as the convergence from the intermediate stage to $\eta_{\text{rel}} = 10^{-5}$. Here, the CFL number is typically at its enforced upper limit (CFL_{max}) and most flow features have already been settled. Kelley and Keyes [13] described similar stages of convergence.

To faithfully represent the effects of a lagged PC during different convergence stages, we converge the solution to $\eta_{\text{rel}} = 10^{0}$, 10^{-3}, and 10^{-5} by using the default ANK algorithm and then lag the PC for ten nonlinear iterations while monitoring the eigenvalue spectra at each nonlinear iteration. Because computing the true eigenvalues would be very expensive for these systems, we consider the approximations to the eigenvalues obtained by using the Arnoldi iteration method to produce the orthogonal projection of the linear system onto the Krylov subspace. During the spectral analysis process, the updates to the state vector are determined by solving the linear system to $\eta_{\text{lin}} = 0.05$, as done in the baseline algorithm. However, we further converge the linear systems to $\eta_{\text{lin}} = 10^{-8}$ to get better approximations to the eigenvalue spectra of the preconditioned operators. We chose not to converge the linear systems to machine precision ($\eta_{\text{lin}} = 10^{-16}$) because, at these levels, the finite-difference formulation (9) introduces large truncation errors in the eigenvalue approximations, yielding inconsistent results.

The process for computing the eigenvalue spectra of the matrix-free operators as

Table 3: ONERA M6 wing tests. For each case, we list the number of nonlinear iterations (NI), cumulative number of linear iterations (LI), and thousands of work units (kWU) required to reach the target convergence. The approximate formulations perform better overall in spite of requiring more nonlinear iterations. The coupled approach for the turbulence model requires both more nonlinear iterations and more work units.

			$\eta_{\text{rel}} = 10^{-5}$			$\eta_{\text{rel}} = 10^{-12}$		
Test	Approximation Level	Turbulence	NI	LI	kWU	NI	LI	kWU
1	$\mathcal{R}_2$	Decoupled	122	2184	9.44	133	2391	15.86
2	$\mathcal{R}_1$	Decoupled	114	1967	9.42	126	2176	15.89
3	$\mathcal{R}_0$	Decoupled	71	2139	9.97	77	2371	17.08
4	$\mathcal{R}_2$	Coupled		—			—	
5	$\mathcal{R}_1$	Coupled	156	2723	13.30	161	2905	18.78
6	$\mathcal{R}_0$	Coupled	95	2758	14.54	102	2974	21.11

we lag the respective PCs consists of the following steps:

1. Converge the system to η_{target} by using the default ANK solver.
2. Fix the CFL number to the CFL number from the last iteration and update the PCs for linear systems containing the flow and turbulence variables.
3. Increase the linear iteration limit to 300 to ensure that the linear systems are converged to $\eta_{\text{lin}} = 10^{-8}$ without any restarts to the GMRES algorithm.
4. Compute the update vector for the flow variables $\Delta\mathcal{Q}_f$ by solving the linear system to $\eta_{\text{lin}} = 0.05$.
5. Further solve the linear system to $\eta_{\text{lin}} = 10^{-8}$ and obtain approximations to the eigenvalues.
6. Update the state vector by using the default methods defined in Algorithms 1 and 2.
7. Repeat steps 4–6 for the turbulence model.
8. Without updating the PCs, go back to Step 4 and repeat for ten nonlinear iterations.

Using this methodology, we study the evolution of the eigenvalue spectra at three different relative convergence levels ($\eta_{\text{target}} = 10^0, 10^{-3}, 10^{-5}$), where $\eta_{\text{target}} = 10^0$ refers to initiating the procedure from freestream conditions, whereas the default algorithm is used to reach the latter two targets. For each η_{target}, we start from a freestream condition and run independent tests for all three convergence levels to eliminate any cross effects due to taking ten nonlinear iterations without lagging the PCs in the convergence history.

Figures 4–6 demonstrate how the eigenvalue spectra evolve as the PCs are lagged for the linear systems that we solve at each iteration. The figures show plots of the spectra evaluated at iterations 1–5 and 10 after the PC update. The results under flow and turbulence represent the eigenvalues of the linear systems used to march the flow and turbulence variables, respectively. From a numerical perspective, the clustering of eigenvalues is preferred because it reflects better linear convergence, whereas a wider distribution of eigenvalues on the complex plane characterizes ill-conditioned operators.

Figure 4 shows that, during the initial stages of convergence, the PC for the flow variables becomes outdated faster than the PC for the turbulence model. As the flow field undergoes significant changes, the discrepancies between the actual linear system and its PC increase faster than the linear system for the turbulence model. This behavior is characterized by the spreading of eigenvalues after ten nonlinear iterations without updating the PC. In contrast, the eigenvalue spectrum of the linear system for the turbulence model spreads less, which means that, after ten iterations, the PC for the turbulence model is more effective than the PC for the flow variables.

Figure 5 shows how the eigenvalue spectra change when the PC is lagged during the intermediate stages of convergence ($\eta_{\text{target}} = 10^{-3}$). During this stage, the spectrum

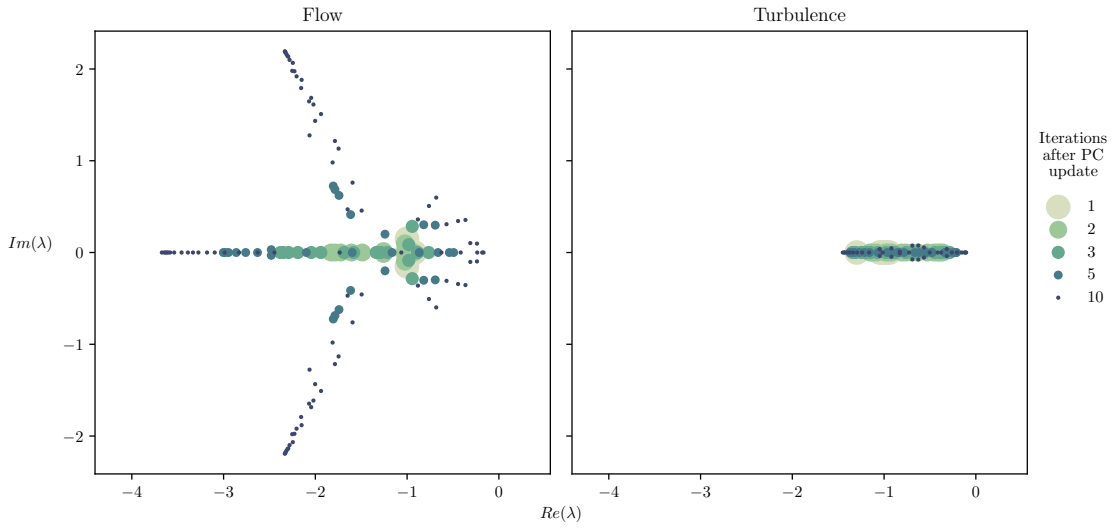

Figure 4: Evolution of the eigenvalue spectra as the respective PCs are lagged at $\eta_{\text{target}} = 10^0$ and CFL = 5.0. In the initial stages of convergence, changes in the flow variables are larger compared with the turbulence model. Therefore, the PC for the flow variables becomes outdated more rapidly than the PC for the turbulence model.

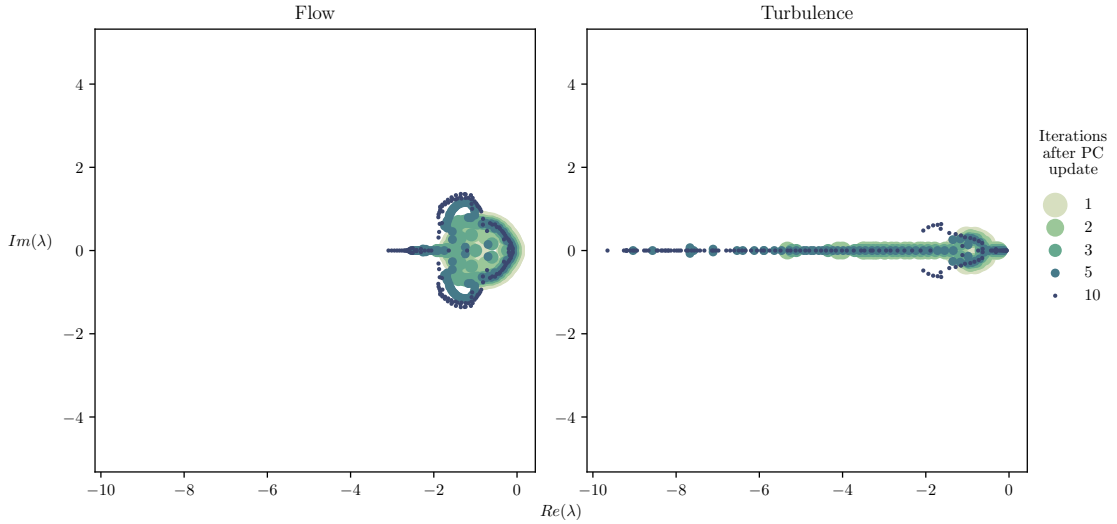

Figure 5: Evolution of the eigenvalue spectra as the respective PCs are lagged at $\eta_{\text{target}} = 10^{-3}$ and CFL = 387.1. In the intermediate stages of the startup, the turbulence model undergoes large changes, and the PC for this system loses effectiveness after a few iterations.

for the turbulence model spreads out faster than the spectrum for the flow variables. This shows that the turbulence model goes through larger changes during this stage of convergence, so its PC is more rapidly outdated.

Finally, Figure 6 shows the evolution of the eigenvalue spectra during the final stages of convergence with the ANK solver ($\eta_{\text{target}} = 10^{-5}$). During this stage, most

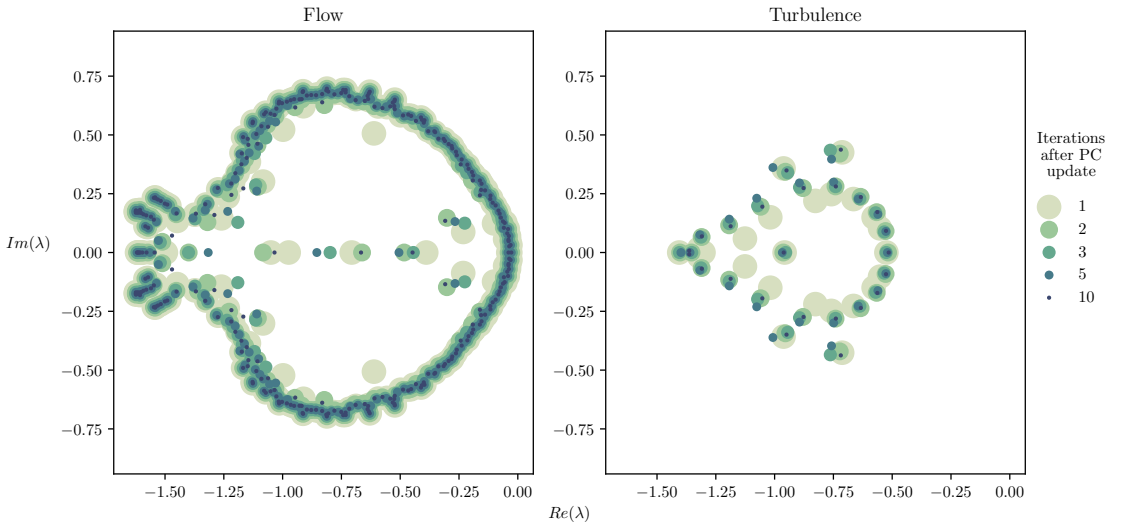

Figure 6: Evolution of the eigenvalue spectra as the respective PCs are lagged at $\eta_{\text{target}} = 10^{-5}$ and CFL $= 10^5$. During the final stage of the startup, both flow and turbulence model variables undergo small changes, so the respective PCs maintain their effectiveness despite being lagged.

flow features have already been resolved, so the updates to the state vectors are small. As a result, lagging the PCs do not change the eigenvalue spectra as much as in the earlier stages of convergence. This shows that, during this stage, lagging the PC does not hinder the performance of the linear solver because the discrepancies between the matrix-free operations and the PCs remain relatively small.

The different trends that appear in the evolution of eigenvalue spectra during different stages of convergence demonstrate the need for adapting the PC lagging strategy as a function of convergence stage. If a single lag were selected for the complete startup process, it would be set by the most problematic stage (i.e., the initial stage for flow variables and the intermediate stage for turbulence variables). Having an adaptive PC-lagging algorithm alleviates this limiting factor. Furthermore, the results show that the eigenvalue spectra for the flow and turbulence model variables evolve differently in the different stages of convergence, which demonstrates that the use of a decoupled algorithm might be more beneficial because the PCs for the respective linear systems can be updated separately as needed. In contrast, a coupled approach would require more frequent PC updates because, at different stages of convergence, different variables act as the bottleneck hindering the performance of the linear solver.

5.3 Generalization of Matrix-Free Approximate Jacobian Method to Other Discretizations

By default, we use the JST discretization scheme with scalar dissipation to compute the inviscid fluxes, as stated in Section 2.3. However, the approximate matrix-free approach can be generalized to other discretization methods. To demonstrate this, we test the algorithm by using the JST scheme with matrix dissipation [66] and the

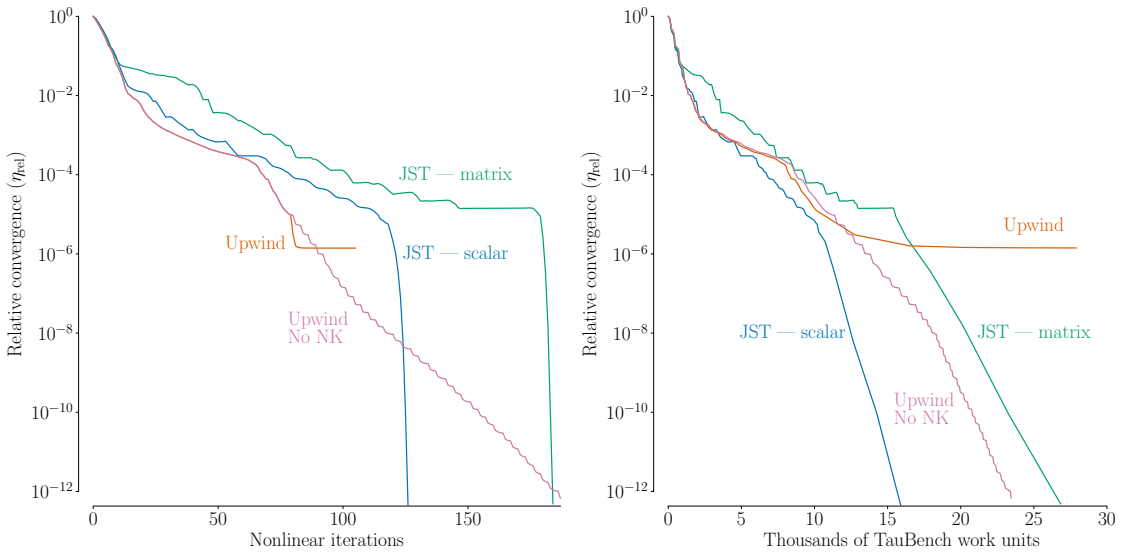

Figure 7: Relative convergence versus number of nonlinear iterations and thousands of TauBench WUs for the ONERA M6 wing. We used three different discretization methods for the inviscid flux computations to demonstrate the applicability of the proposed algorithm to various discretization methods. The upwind method with the Roe flux fails during the NK stage; however, the same case without the switch to the NK solver converges successfully.

upwind method using the Roe flux [67]. The test case is identical to that in Section 5.1. We only test the algorithm with decoupled turbulence. In addition to the JST scheme with matrix dissipation and the upwind scheme, we also include for comparison the results of using $\mathcal{R}_1$ and decoupled turbulence from Section 5.1.

To use the approximate matrix-free approach with the additional discretization methods, we first need to define approximate residual formulations that have a structure similar to those we built for the JST scheme with scalar dissipation (see Table 1). For the JST scheme with matrix dissipation, we adopt the same modification we defined for the case with scalar dissipation. For the upwind method, the exact residuals are computed with a second-order-accurate reconstruction of the states on the faces for flux computations. For $\mathcal{R}_1$ and $\mathcal{R}_2$, we simply use a first-order reconstruction of the states on the faces. The Roe flux itself is not modified for $\mathcal{R}_1$ and $\mathcal{R}_2$. The approximations for both discretization methods result in the same stencils that we defined for the JST scheme with scalar dissipation.

We tuned the default solver parameters for the JST scheme with scalar dissipation; this set of parameters achieves the desired convergence during the startup stage for both the JST scheme with matrix dissipation and the upwind method. Once the code switches to the NK solver with the upwind method, convergence stalls because the linear solver repeatedly fails to converge. We reran this case without switching to the NK solver, and the ANK solver successfully converges the system to $\eta_{\text{rel}} = 10^{-12}$, as presented in Figure 7 and Table 4. This result shows that the approximations in the

matrix-free formulation improve convergence by making the resulting linear systems easier to solve.

5.4 Comparison with Matrix-Based Algorithm

We now discuss the performance characteristics of the matrix-based and matrix-free solver variants for a range of Jacobian lags. For all cases described in this section, we use the coarse overset mesh of the Common Research Model (CRM) wing-body (WB) configuration, which was used in the 6$^{\text{th}}$ Drag Prediction Workshop (DPW6) [68]. Coder et al. [69] describe the overset mesh used for the DPW6 test cases, which has 14 million cells. For these tests, we use eight Ivybridge nodes on NASA's Pleiades supercomputer. Each node contains 20 cores, and each test uses 160 cores in total. One WU corresponds to 6.24 s.

A total of 18 tests are performed with this case, where the flow conditions are $M = 0.85$, $\text{Re} = 5 \times 10^6$, reference temperature $T_{\text{ref}} = 310.93$ K, and angle of attack $\alpha = 2.4^\circ$. The first seven tests are executed with the matrix-based ANK algorithm (denoted MB) that uses the $\mathcal{R}_2$ approximations. The matrix-based algorithm uses the approximate Jacobian formed to create the PC matrix as the main driver in the solution process. With this algorithm, we lag both the Jacobian and its PC. The next seven tests represent the matrix-free ANK algorithm (denoted MF) that again uses the same approximation level $\mathcal{R}_2$ as the matrix-based algorithms. Finally, we conduct two tests each with the matrix-free algorithm using $\mathcal{R}_0$ and $\mathcal{R}_1$. With all matrix-free algorithms, we only lag the PC because the Jacobian is approximated by using the matrix-free approach introduced in Equation (9). With these tests, we handle the turbulence model separately, as detailed in Algorithm 3. The simulations are converged to $\eta_{\text{rel}} = 10^{-5}$ with the ANK solver and then converged to $\eta_{\text{rel}} = 10^{-12}$ with the NK solver.

Table 5 details the full range of tests conducted with this case, along with the relevant performance metrics. With these tests, the adaptive lagging process (denoted

Table 4: ONERA M6 wing tests using different discretization methods for the inviscid fluxes. For each case, we list the number of nonlinear iterations (NI), cumulative number of linear iterations (LI), and thousands of work units (kWU) required to reach the target convergence. The ANK solver converges cases with different discretizations. For the upwind case, the NK solver fails, but the ANK solver reaches 10^{-12} relative convergence, which shows that the Jacobian approximations improve robustness.

		$\eta_{\text{rel}} = 10^{-5}$			$\eta_{\text{rel}} = 10^{-12}$		
Discretization Method	η_{ANK}	NI	LI	kWU	NI	LI	kWU
JST with scalar dissipation	10^{-5}	114	1967	9.42	126	2176	15.89
JST with matrix dissipation	10^{-5}	179	3570	15.56	184	3943	26.83
Upwind	10^{-5}	79	1973	10.59		—	
Upwind	No NK	79	1973	11.18	187	4327	23.45

adaptive) corresponds to Algorithm 3. Furthermore, a constant lag corresponds to omitting the criteria for refreshing the Jacobian or the PC when the desired relative convergence is reached. The results reveal the following two trends:

First, for larger Jacobian lags, the matrix-free algorithm with the $\mathcal{R}_2$ approximations outperforms the matrix-based algorithm with respect to the number of nonlinear iterations required to reach a relative convergence of $\eta_{\text{rel}} = 10^{-5}$, which is expected for the matrix-free formulation. However, the matrix-based algorithm clearly outperforms the matrix-free algorithm in terms of computational time because each matrix-vector multiplication with the matrix-based approximate Jacobian is 10% to 20% cheaper than the finite-difference approximation used in the matrix-free approach. If implemented correctly, the two algorithms with a Jacobian or PC always up to date should only differ due to the truncation errors introduced with the respective finite-difference computations. This similarity is demonstrated by the results of Tests 1 and 8, which

Table 5: Tests with the coarse overset mesh of the CRM WB configuration. For each case, we list the number of nonlinear iterations (NI), cumulative number of linear iterations (LI), and thousands of work units (kWU) required to reach the target convergence. The results demonstrate the advantage of lagging the PC.

			$\eta_{\text{rel}} = 10^{-5}$			$\eta_{\text{rel}} = 10^{-12}$		
Test	ANK Variant	Jacobian/PC Lag	NI	LI	kWU	NI	LI	kWU
1	MB-$\mathcal{R}_2$	1	191	3242	42.23	213	3908	83.88
2	MB-$\mathcal{R}_2$	2	412	7046	59.76	428	7609	95.08
3	MB-$\mathcal{R}_2$	3	195	3305	23.66	220	3961	80.43
4	MB-$\mathcal{R}_2$	4	263	3855	27.47	287	4548	70.33
5	MB-$\mathcal{R}_2$	5	270	3801	25.27	293	4449	65.51
6	MB-$\mathcal{R}_2$	10	635	7135	39.42	652	7706	75.09
7	MB-$\mathcal{R}_2$	Adaptive	290	4039	23.31	314	4748	67.09
8	MF-$\mathcal{R}_2$	1	190	3232	52.42	212	3895	93.82
9	MF-$\mathcal{R}_2$	2	199	3385	59.98	223	4047	100.96
10	MF-$\mathcal{R}_2$	3	180	3151	41.89	200	3775	80.42
11	MF-$\mathcal{R}_2$	4	186	3239	30.62	208	3884	70.89
12	MF-$\mathcal{R}_2$	5	240	3947	34.36	260	4615	75.67
13	MF-$\mathcal{R}_2$	10	197	3688	30.21	218	4323	69.93
14	MF-$\mathcal{R}_2$	Adaptive	277	4486	35.88	298	5192	79.91
15	MF-$\mathcal{R}_1$	10	102	2971	27.74	120	3563	64.52
16	MF-$\mathcal{R}_1$	Adaptive	106	2850	26.96	123	3456	64.64
17	MF-$\mathcal{R}_0$	10	71	1775	16.65	87	2448	58.3
18	MF-$\mathcal{R}_0$	Adaptive		—			—	

show that the two different methods require an almost identical number of nonlinear iterations to achieve $\eta_{\text{rel}} = 10^{-5}$. The small discrepancy is due to the accumulation of finite-difference truncation errors over thousands of linear iterations.

Second, with the approximate formulations, the solver requires more nonlinear iterations, as demonstrated in Tests 7, 14, 16, and 17. The trade-off here is between the number of nonlinear iterations and the cost of each nonlinear iteration. Tests 7 and 16, despite having a different number of nonlinear iterations, require an almost identical number of WUs to reach the same level of convergence. Test 18 fails because the adaptive PC-lagging algorithm results in too frequent updates to the PC, which in turn prematurely increases the CFL number to unstable values.

Figure 8 plots the relative convergence versus the number of nonlinear iterations and versus the number of WUs required to achieve $\eta_{\text{rel}} = 10^{-12}$ with the tests that use the adaptive PC-lagging algorithm. The $\mathcal{R}_0$ variant fails due to the high CFL number obtained after consecutive PC updates, and the $\mathcal{R}_1$ approximation requires fewer nonlinear iterations compared with $\mathcal{R}_2$. When using the $\mathcal{R}_2$ formulation, the convergence oscillates, especially toward the final stages of the startup. During these iterations, the backtracking line search limits the step sizes to achieve a stable solver. However, the approximations still introduce some jumps in the total residual norm. The line search only aims to reduce the unsteady residual norm and, because of the time stepping term in this formulation, the total residual norm can increase, as it does here.

Comparing the computational cost across different cases in terms of TauBench WUs demonstrates the performance gains due to the approximate Jacobian. Across all tests done with the automatic lagging algorithm, the solver performs similarly despite requiring a different total number of nonlinear iterations. This discrepancy is due to the lower cost per linear iteration of the approximate formulations and the approximations that improve the conditioning of the linear system by increasing the diagonal dominance of the Jacobian. Because this case uses an overset mesh, we could generate high-quality component meshes while preserving orthogonality in the layers. As a result, the approximations introduced with $\mathcal{R}_2$ do not degrade the performance. Despite being faster than the matrix-free algorithm with $\mathcal{R}_2$, we do not use the matrix-based algorithm in our production runs where we perform full-scale optimization because this algorithm can destabilize the solver in the more challenging cases. Also, the matrix-based method with Jacobian lagging is stabilized by limiting the step size for each nonlinear iteration with the physicality check and the backtracking line search. For the matrix-free approaches, these methods act as a safety net for bad solution updates and are not as limiting as in the matrix-based case.

5.5 Effects of Different Levels of Approximations

We now discuss the effects of different levels of approximations in the matrix-free formulation. Compared with overset meshes, structured multiblock meshes tend to have more skewness in the layers due to the geometrical constraints when generating a continuous structured mesh for a wing-body configuration. As a result, the orthogonal mesh approximation in $\mathcal{R}_2$ breaks down sooner, necessitating more accurate residual

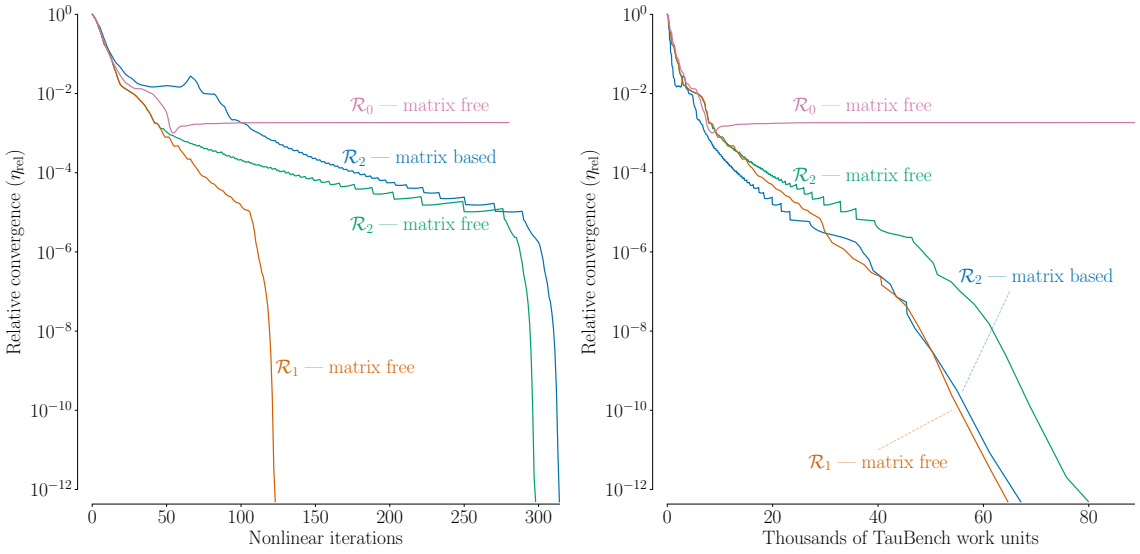

Figure 8: Relative convergence versus number of nonlinear iterations and thousands of TauBench WUs for the overset mesh of the CRM WB configuration. All the results presented here use the adaptive PC-lagging algorithm. A trade-off exists between computational cost and convergence rate per nonlinear iteration.

formulations. To demonstrate this effect, we use the coarse multiblock mesh of the CRM WB configuration used for the DPW6 case [68], which has 32 million cells. This geometry is identical to that in the previous subsection, with similar boundary layer resolution, as defined by the DPW6 meshing guidelines [68]. The flow conditions are the same as the overset case defined in the previous set of results and tests are performed with the full ANK solver (Algorithm 3). These tests use the adaptive PC-lagging algorithm, along with the matrix-free formulation for each approximation level. These tests were done on the 20 Ivybridge nodes on the Pleiades supercomputer, so the TauBench WUs are the same as for the previous cases. We study six cases in all, with all three approximation levels ($\mathcal{R}_0$–$\mathcal{R}_2$) for the matrix-free operations, and we test each approximation level with two different switching tolerances for the NK solver; namely, $\eta_{\text{ANK}} = 10^{-5}$ and 10^{-6}. We also impose a limit to the cumulative linear iteration count and terminate the simulations when they reach 10 000 linear iterations.

Table 6 lists the test cases and the corresponding results. The tests with $\mathcal{R}_0$ and $\mathcal{R}_2$ fail to reach either of the switch tolerances, so we only include tests with the default switch tolerance because the remaining two cases follow identical convergence paths.

The test with the $\mathcal{R}_1$ and the default parameters achieve the default switching tolerance of $\eta_{\text{ANK}} = 10^{-5}$; however, the NK solver performs poorly after the switch. Conversely, the NK solver has no difficulty converging to $\eta_{\text{rel}} = 10^{-12}$ when using a lower switching tolerance of $\eta_{\text{ANK}} = 10^{-6}$ along with $\mathcal{R}_1$. This shows that the criterion for switching to the NK solver is not straightforward and a more robust method is required, as mentioned in Section 3.9.

Because mesh orthogonality is not preserved on this multiblock mesh, the $\mathcal{R}_2$ ap-

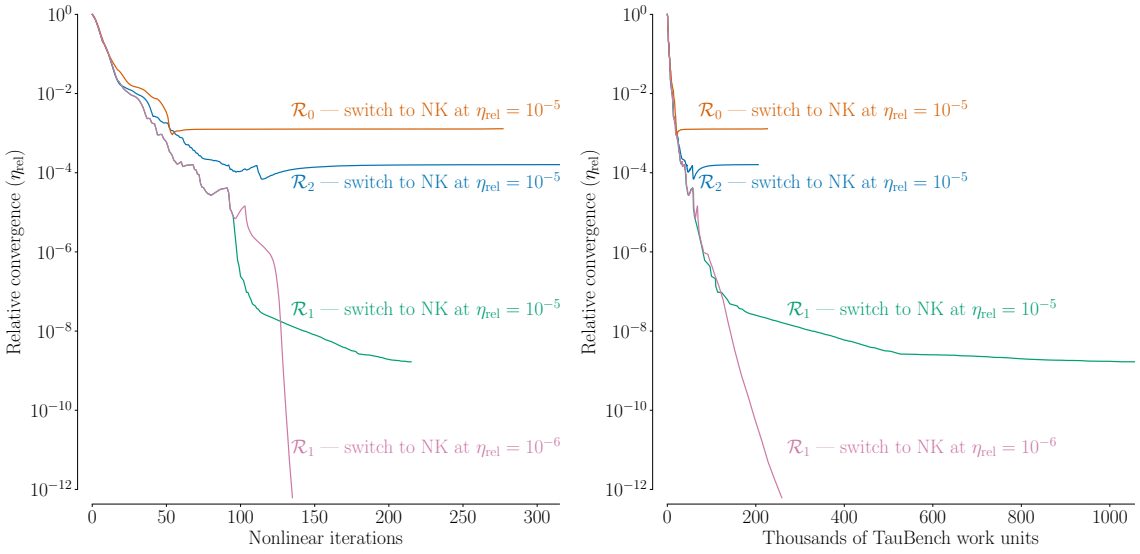

Figure 9: Relative convergence versus number of nonlinear iterations and thousands of TauBench WUs with the multiblock mesh of the CRM WB configuration. The cases presented here use the matrix-free formulation and the adaptive PC-lagging algorithm. The tests with $\mathcal{R}_0$ and $\mathcal{R}_2$ fail. Delaying the switch to the NK solver from $\eta_{\text{rel}} = 10^{-5}$ to $\eta_{\text{rel}} = 10^{-6}$ improves the performance of the NK solver.

proximation level breaks down faster than the $\mathcal{R}_1$ approximation level. Moreover, the test with $\mathcal{R}_0$ also fails because the linear solutions in this test fail due to the increased stiffness introduced by the additional terms.

Figure 9 shows the convergence history for the tests considered here. The relative convergence plotted versus TauBench WUs shows the negative effect of a premature switch to the NK solver. The NK solver uses a stronger PC to converge the linear systems more tightly at the cost of more expensive linear iterations. However, in this case, both the nonlinear convergence and the linear solver performance suffer

Table 6: Tests with the multiblock mesh of the CRM WB configuration. For each case, we list the number of nonlinear iterations (NI), cumulative number of linear iterations (LI), and thousands of work units (kWU) required to reach the target convergence. The results show the robustness problems that arise with $\mathcal{R}_0$ and $\mathcal{R}_2$. Furthermore, the NK solver fails when the switch is done prematurely, whereas a lower switch value solves this problem.

		$\eta_{\text{rel}} = 10^{-5}$			$\eta_{\text{rel}} = 10^{-6}$			$\eta_{\text{rel}} = 10^{-12}$		
Approximation level	η_{ANK}	NI	LI	kWU	NI	LI	kWU	NI	LI	kWU
MF-$\mathcal{R}_2$	10^{-5}		—			—			—	
MF-$\mathcal{R}_1$	10^{-5}	95	2484	61.24	98	2654	84.48		—	
MF-$\mathcal{R}_0$	10^{-5}		—			—			—	
MF-$\mathcal{R}_1$	10^{-6}	95	2484	62.27	119	3494	82.02	135	4801	258.75

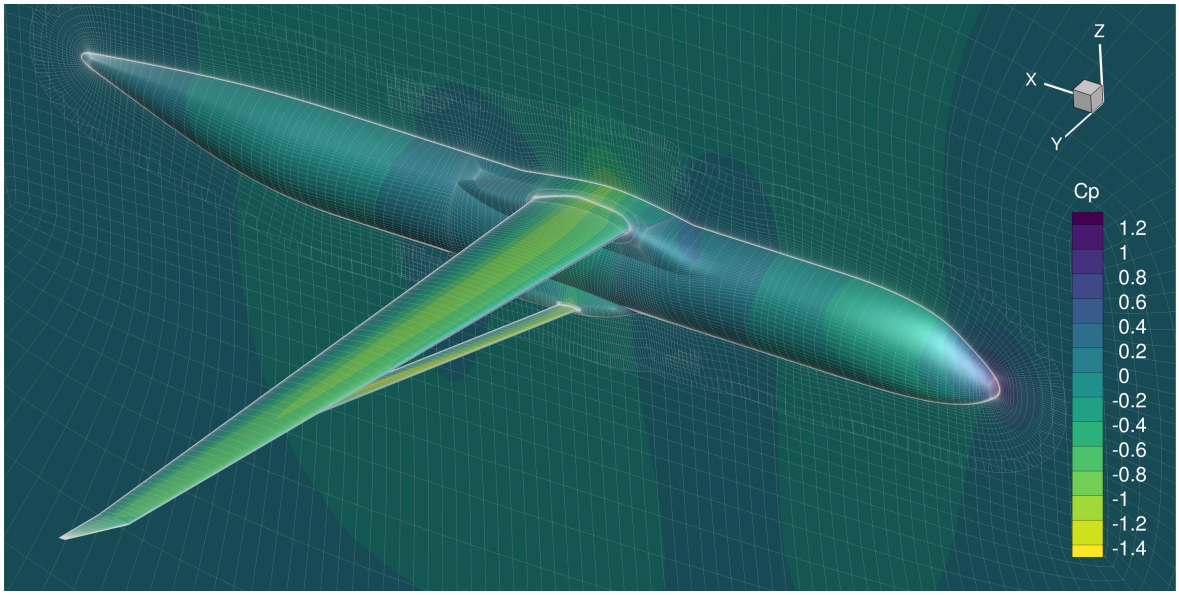

Figure 10: Strut-braced wing PADRI configuration. Coefficient of pressure contours of the solution and overset mesh active cells. The complex overset connectivity structure makes it practically impossible to obtain coarser levels of this mesh.

from premature switching, resulting in a considerably longer runtime (up to the 10 000 iteration limit). These results justify the use of the $\mathcal{R}_1$ approximation as default with the approximate matrix-free formulation. Although the exact formulation ($\mathcal{R}_0$) gives better convergence rates for some portion of the ANK stage, the algorithm with $\mathcal{R}_1$ is more robust and can prevent failures in badly behaved cases such as this one.

5.6 Effects of Coupling the Turbulence Model Equation

By default, we march the turbulence model in pseudo-time, separate from the flow variables as described in Algorithm 3. However, the trade-off between improvements in convergence when handling the equations in a coupled manner and the increased stiffness of the linear systems with the coupled mode is not straightforward. To further investigate this trade-off, we consider the strut-braced wing (SBW) configuration used in the Platform for Aircraft Drag Reduction Innovation (PADRI) test case[6] shown in Figure 10. In this configuration, the wing-strut junction creates complex flow features: a standing shock and a separation region aft of the shock, as shown in Figure 11. The presence of separation increases the importance of the coupling between flow variables and the turbulence model.

Secco et al. [56, 57] optimized the aerodynamic shape of the SBW geometry through extensive use of overset meshes, and we use for this configuration the same baseline mesh they generated, which contains 6.4 million cells. The overset mesh for this configuration contains complex connectivities, as shown in Figure 10 along with the coefficient of pressure contours of the solution. As a result of this complex overset structure, gen-

[6] http://congress.cimne.com/padri-2017/frontal/Topics.asp, accessed June 2019

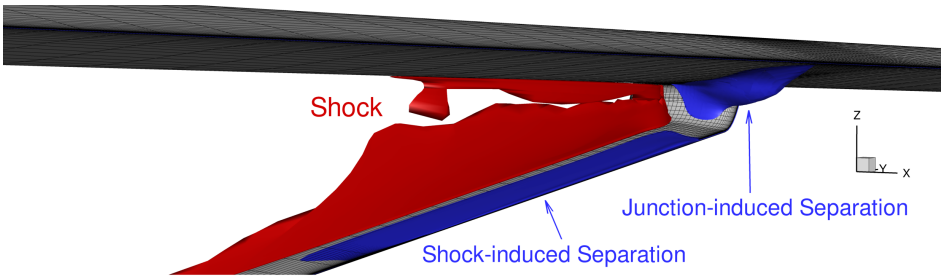

Figure 11: Converged flow solution near the wing-strut junction, emphasizing the standing shock and the separation regions.

erating coarser levels of this mesh is practically impossible. Therefore we cannot use a multigrid startup strategy and have to rely on the ANK solver for robust startup. In addition to demonstrating the effects of coupling the turbulence model, this test case shows the capabilities of the solver algorithm to treat unconventional configurations such as the SBW. The flow conditions for this case are $M = 0.72$, $\mathrm{Re} = 2.3 \times 10^7$, reference temperature $T_{\mathrm{ref}} = 228.71$ K, and angle of attack $\alpha = 1.0°$. For this case, along with the baseline turbulence model introduced earlier, we use the quadratic constitutive relation introduced by Spalart [70]. With this modification, the turbulence model is fully classified as *SA-noft2-QCR2000*. These tests were run on the Skylake nodes of the Stampede2 supercomputer. We used one node per simulation, which contains 48 cores, and one WU on these processors corresponds to 2.98 s.

We conducted two tests: one with the default solver using the decoupled turbulence method defined in Algorithm 3 and a second test with the coupled solver variant, where the turbulence model variable is coupled to the flow variables in the linear system we consider in each nonlinear iteration. Although the default algorithm converges without difficulty, the coupled variation stalls before reaching $\eta_{\mathrm{rel}} = 10^{-5}$, as shown by Table 7 and Figure 12. Because of the complex flow features near the wing-strut junction, the coupled algorithm fails to find an update vector that decreases the unsteady residual norm. As a result, the coupled solver stalls because the backtracking line search returns the minimum step size for the remaining iterations. This example shows that, even for a case with tight coupling in the flow and turbulence model variables, having a decoupled solver is beneficial when handling regions where the residual norms fluctuate.

Table 7: SBW configuration tests. For each case, we list the number of nonlinear iterations (NI), cumulative number of linear iterations (LI), and thousands of work units (kWU) required to reach the target convergence. Convergence fails when the turbulence model variable is coupled to the flow variables in the implicit formulation.

	$\eta_{\mathrm{rel}} = 10^{-5}$			$\eta_{\mathrm{rel}} = 10^{-12}$		
Turbulence	NI	LI	kWU	NI	LI	kWU
Decoupled	76	915	7.72	95	1356	22.7
Coupled		—			—	

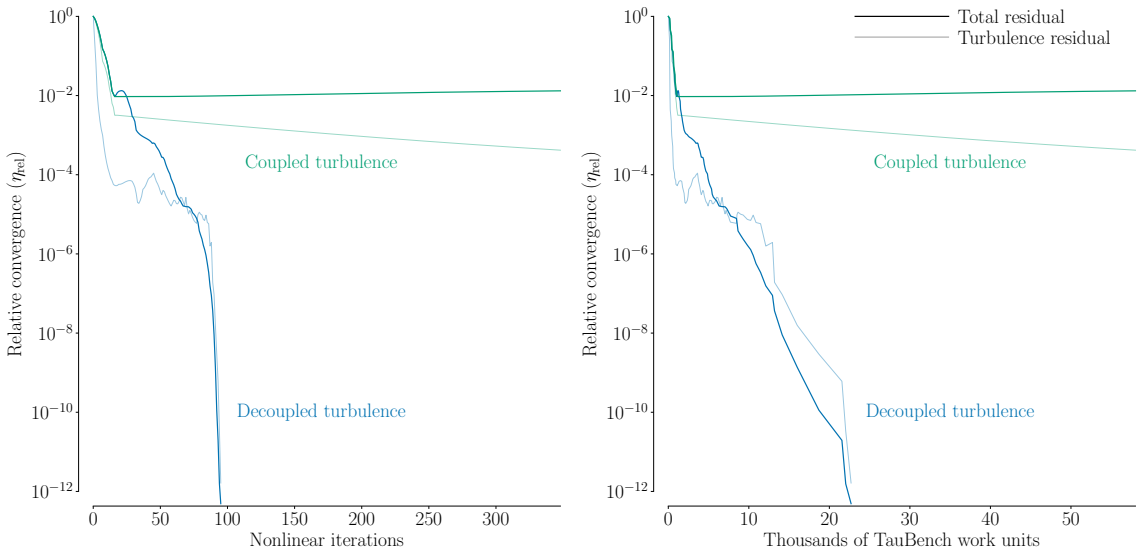

Figure 12: Relative convergence versus number of nonlinear iterations and thousands of TauBench WUs for the SBW configuration. The default algorithm converges without difficulty, but the coupled variant stalls before reaching $\eta_{\text{rel}} = 10^{-5}$.

5.7 Comparison with Multigrid Startup

The final group of tests uses CFD solutions to train surrogate models through a data-driven approach, which requires thousands of consecutive simulations of two-dimensional airfoils. Li et al. [71] developed a fast, data-driven approach for airfoil analysis and optimization by using surrogate models and used more than 100 000 CFD simulations to train their surrogate model. To demonstrate the effectiveness of the proposed solver and compare with a multigrid startup procedure, we study the convergence of a large set of consecutive CFD solutions with a subset of cases from the work of Li et al. [71]. We focus on the set of subsonic airfoils obtained from the Webfoil Database.[7] This set consists of 1172 airfoil shapes obtained from the database and pre-processed to obtain a smooth representation of the airfoils with mode shapes and weights. For the details of this process, we refer the reader to the work done by Li et al. [71]. All of these discretizations are structured two-dimensional geometries for which creating coarser mesh levels is straightforward. We use these cases to compare the performance of the ANK solver with that of a multigrid startup method.

We did four sets of tests, each considering all 1172 airfoil shapes, and studied the overall performance of the solver algorithms. All airfoil meshes have the same size (35 840 cells). In the first three sets of tests, we use the 3w, 4w, and 5w multigrid schemes to reach $\eta_{\text{rel}} = 10^{-5}$ relative convergence. With the multigrid approach, the solver initializes the solution at the coarsest mesh level and moves through the finer mesh levels as it reduces the residual norm by two orders of magnitude on each level. After the solver reaches the finest mesh level, it executes the prescribed multigrid solution cycle by using the D3ADI method as the smoother [63]. The CFL number

[7] http://webfoil.engin.umich.edu, accessed June 2019

with the multigrid tests is fixed at 5.0. In the fourth set of tests, we use the ANK solver to reach $\eta_{\text{rel}} = 10^{-5}$ relative convergence. For these tests, we reduce the ILU fill level for the PCs in the ANK solver to unity and preserve the remaining default parameters. The flow conditions are $M = 0.45$, $\text{Re} = 6.5 \times 10^6$, reference temperature $T_{\text{ref}} = 310.93$ K, and angle of attack $\alpha = 2.5°$. The solutions are converged to $\eta_{\text{rel}} = 10^{-12}$ by using the NK solver after the startup stage. Again, we used one Skylake node of the Stampede2 supercomputer for each set of tests. Due to the embarrassingly parallel nature of running a large number of small cases, we distributed 1172 airfoil cases among 48 processors of a node, so each simulation used one Skylake core, where one WU corresponds to 2.98 s.

Table 8 lists the arithmetic mean and standard deviation of the performance metrics for each set of successful tests, along with the number of failed cases for each test. To compute these statistics, we omit the results from the failed cases. In these tests, four cases fail on each method due to failures in the mesh-generation algorithm, which is external to ADflow. The remaining failed cases represent cases where the flow solver terminates because it reaches 5000 cumulative linear iterations. For the multigrid methods, the simulations are terminated when they reach 5000 nonlinear iterations, where each multigrid cycle on the finest mesh is counted as one linear iteration. For each method, the solver converges more than 99% of all 1172 airfoil shapes, which are representative of a large design space of subsonic airfoil shapes. The average WUs required to reach $\eta_{\text{rel}} = 10^{-5}$ is reduced as the solver uses coarser levels in the multigrid

Table 8: Results with 1172 airfoil shapes. For each case, we list the number of nonlinear iterations (NI), cumulative number of linear iterations (LI), and thousands of work units (kWU) required to reach the target convergence. The methods 3w, 4w, and 5w represent the respective multigrid startup strategy. AM and SD stand for arithmetic mean and standard deviation of the respective metrics over the set of successful simulations with the given startup method. We report only the number of solver failures and omit the four failures due to mesh generation for each method. The ANK solver converges almost all airfoils with convergence rates comparable to a multigrid approach, even though the ANK solver only requires the finest grid level.

			$\eta_{\text{rel}} = 10^{-5}$			$\eta_{\text{rel}} = 10^{-12}$		
Method	Failures	Value	NI	LI	WU	NI	LI	WU
3w	0	AM	604.86	604.86	78.88	616.39	780.13	93.47
		SD	129.02	129.02	13.14	130.44	173.32	15.17
4w	5	AM	197.42	197.42	29.62	204.65	340.92	41.28
		SD	45.94	45.94	5.74	47.16	78.25	7.22
5w	0	AM	160.91	160.91	27.09	167.35	294.47	37.85
		SD	29.64	29.64	3.80	30.94	61.60	5.20
ANK	3	AM	35.16	381.63	24.54	46.79	536.22	35.66
		SD	10.69	110.79	6.24	12.03	130.46	7.17

scheme, and the best performance is achieved with the ANK solver.

However, the residual computations on which the ANK and NK solvers operate are vectorized, whereas the residual computations used with the multigrid methods are not. This is due to the fact that it is easier to implement vectorized residual computations within the ANK and NK solvers because we can treat these computations as black boxes. Conversely, implementing vectorization for multigrid methods is more challenging because details related to the coarse grid representations of the state must be addressed. Due to vectorization, there is an acceleration of about 1.5 overall in the ANK and NK solvers compared with the non-vectorized versions. The acceleration is moderate because the residual computations are only responsible for half of the computational cost of the ANK solver. Running the same tests with the ANK solver and disabling the vectorization yields an average of 35.67 WUs to reach $\eta_{\text{rel}} = 10^{-5}$, which places the ANK solver performance between the performance levels of the 3w and 4w multigrid methods. These results show that the ANK solver performs at a level similar to that of a multigrid startup method, while only requiring the finest mesh level. This enables the use of the ANK solver with overset meshes, whereas we typically cannot generate coarser levels of the overset meshes for a multigrid startup because of connectivity problems.

6 Conclusion

In this work, we describe two contributions to approximate Newton-based solvers. First, we introduce the use of approximate residual computations for a matrix-free approximate Jacobian formulation. Second, we describe in detail our adaptive PC-lagging algorithm. The proposed matrix-free approach enables the algorithm to use varying levels of approximations and eliminates the lack of robustness that arises when the Jacobian is lagged with matrix-based approaches. We describe in detail our implementation and explain how the approach is generalizable to a wide range of discretization methods. We test the proposed algorithm with a range of problems of interest whose results justify our implementation choices. The algorithm can handle challenging three-dimensional geometries and is shown to be suitable for a data-driven study that requires over one hundred thousand successive RANS simulations.

By using the open-source CFD solver ADflow, we demonstrate the trade-off between the nonlinear convergence rate and the cost of each linear solution and study a range of approximations in the matrix-free operations. When approximate residual formulations are used in the matrix-free operations, the solver requires more nonlinear iterations than when using the exact formulation, but the cost of each nonlinear iteration is reduced. As a result, the use of the approximate formulations offsets the cost of additional nonlinear iterations and yields higher nonlinear convergence rates during the initial stages of convergence.

It remains unclear which approximation level performs best, despite having tested a wide range of cases that would not only showcase our default approach but also demonstrate the trade-offs between different cases. In spite of not achieving the best performance for all cases, the approximation level $\mathcal{R}_1$ successfully converges all cases

in this study, along with many other test cases that we used while developing the solver. To prevent the interruption of repeated automated runs, such as those required in a design optimization loop, our priority is solver robustness. This approach also reduces the cost of manual intervention, which is generally more costly than additional computational time. Therefore, we select $\mathcal{R}_1$ as the default approximation level in the matrix-free operations.

We describe in detail our adaptive PC-lagging algorithm and demonstrate the effects of using a lagged PC by studying how the eigenvalue spectra of the preconditioned systems evolve as we lag the PC. As expected, the PC effectiveness deteriorates as it is lagged between nonlinear iterations. However, the rate at which the eigenvalue spectra expand depends on the nonlinear convergence stage. Furthermore, we show that the eigenvalue spectra resulting from linearization of the flow variables and the turbulence model variable yield different characteristics during different stages of convergence.

The matrix-free operations use the varying levels of approximate residual formulations directly in the underlying linear solver algorithm. We compute the matrix-based approximate Jacobian for the PCs by using finite differences along with coloring acceleration techniques. As a result, no manual differentiation is required during the solver implementation, and any change in the underlying residual formulation is directly reflected in the implicit formulation. This approach greatly reduces the implementation effort and the ANK solver can be generalized to a wide range of discretization methods.

7 Acknowledgments

This work was supported by the National Science Foundation (award number 1435188). Computations were performed in the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by the National Science Foundation grant number ACI-1548562, and the NASA High-End Computing (HEC) Program through the NASA Advanced Supercomputing (NAS) Division at Ames Research Center. Special thanks to Ney R. Secco for the SBW materials, Jichao Li for the airfoil database resources, Krzysztof Fidkowski for his valuable suggestions, and Jason Hicken for reviewing an early version of this paper.

References

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

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

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

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

[5] Kenway, G. K. W., and Martins, J. R. R. A., "Multipoint High-Fidelity Aerostructural Optimization of a Transport Aircraft Configuration," *Journal of Aircraft*, Vol. 51, No. 1, 2014, pp. 144–160. doi:10.2514/1.C032150.

[6] Kenway, G. K. W., and Martins, J. R. R. A., "Multipoint Aerodynamic Shape Optimization Investigations of the Common Research Model Wing," *AIAA Journal*, Vol. 54, No. 1, 2016, pp. 113–128. doi:10.2514/1.J054154.

[7] Yu, Y., Lyu, Z., Xu, Z., and Martins, J. R. R. A., "On the Influence of Optimization Algorithm and Starting Design on Wing Aerodynamic Shape Optimization," *Aerospace Science and Technology*, Vol. 75, 2018, pp. 183–199. doi:10.1016/j.ast.2018.01.016.

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

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

[10] Witherden, F. D., Jameson, A., and Zingg, D. W., "The Design of Steady State Schemes for Computational Aerodynamics," *Handbook of Numerical Analysis*, Vol. 18, 2017, pp. 303–349. doi:10.1016/bs.hna.2016.11.006.

[11] Hicken, J., and Zingg, D., "Globalization Strategies for Inexact-Newton Solvers," *19th AIAA Computational Fluid Dynamics*, American Institute of Aeronautics and Astronautics, 2009. doi:10.2514/6.2009-4139.

[12] Allgower, E., and Georg, K., *Introduction to Numerical Continuation Methods*, Classics in Applied Mathematics, Society for Industrial and Applied Mathematics, 2003. doi:10.1137/1.9780898719154.

[13] Kelley, C., and Keyes, D., "Convergence Analysis of Pseudo-Transient Continuation," *SIAM Journal on Numerical Analysis*, Vol. 35, No. 2, 1998, pp. 508–523. doi:10.1137/S0036142996304796.

[14] Bellavia, S., and Berrone, S., "Globalization strategies for Newton–Krylov methods for stabilized FEM discretization of Navier–Stokes equations," *Journal of Computational Physics*, Vol. 226, No. 2, 2007, pp. 2317–2340. doi:10.1016/j.jcp.2007.07.021.

[15] Knoll, D. A., and Keyes, D. E., "Jacobian-free Newton–Krylov methods: a survey of approaches and applications," *Journal of Computational Physics*, Vol. 193, No. 2, 2004, pp. 357–397. doi:10.1016/j.jcp.2003.08.010.

[16] Gropp, W., Keyes, D., McInnes, L. C., and Tidriri, M. D., "Globalized Newton-Krylov-Schwarz Algorithms and Software for Parallel Implicit CFD," *The International Journal of High Performance Computing Applications*, Vol. 14, No. 2, 2000, pp. 102–136. doi:10.1177/109434200001400202.

[17] Gropp, W. D., Kaushik, D. K., Keyes, D. E., and Smith, B. F., "High-performance parallel implicit CFD," *Parallel Computing*, Vol. 27, No. 4, 2001, pp. 337–362. doi:10.1016/S0167-8191(00)00075-2.

[18] Hicken, J. E., and Zingg, D. W., "Parallel Newton-Krylov Solver for the Euler equations Discretized Using Simultaneous Approximation Terms," *AIAA Journal*, Vol. 46, No. 11, 2008, pp. 2773–2786. doi:10.2514/1.34810.

[19] Chisholm, T. T., and Zingg, D. W., "A Jacobian-free Newton–Krylov algorithm for compressible turbulent fluid flows," *Journal of Computational Physics*, Vol. 228, No. 9, 2009, pp. 3490–3507. doi:10.1016/j.jcp.2009.02.004.

[20] Osusky, M., and Zingg, D. W., "Parallel Newton–Krylov–Schur Flow Solver for the Navier–Stokes Equations," *AIAA Journal*, Vol. 51, No. 12, 2013, pp. 2833–2851. doi:10.2514/1.J052487.

[21] Brown, D. A., and Zingg, D. W., "Performance of a Newton–Krylov–Schur Algorithm for Solving Steady Turbulent Flows," *AIAA Journal*, Vol. 54, No. 9, 2016, pp. 2645–2658. doi:10.2514/1.J054513.

[22] Hicken, J., Buckley, H., Osusky, M., and Zingg, D., "Dissipation-based continuation: a globalization for inexact-Newton solvers," *20th AIAA Computational Fluid Dynamics Conference*, American Institute of Aeronautics and Astronautics, 2011. doi:10.2514/6.2011-3237.

[23] Brown, D. A., and Zingg, D. W., "A monolithic homotopy continuation algorithm with application to computational fluid dynamics," *Journal of Computational Physics*, Vol. 321, No. Supplement C, 2016, pp. 55–75. doi:10.1016/j.jcp.2016.05.031.

[24] Crivellini, A., and Bassi, F., "An implicit matrix-free Discontinuous Galerkin solver for viscous and turbulent aerodynamic simulations," *Computers & Fluids*, Vol. 50, No. 1, 2011, pp. 81–93. doi:10.1016/j.compfluid.2011.06.020.

[25] Xia, Y., Luo, H., Frisbey, M., and Nourgaliev, R., "A set of parallel, implicit methods for a reconstructed discontinuous Galerkin method for compressible flows on 3D hybrid grids," *Computers & Fluids*, Vol. 98, 2014, pp. 134–151. doi:10.1016/j.compfluid.2014.01.023.

[26] Fidkowski, K. J., and Darmofal, D. L., "Review of Output-Based Error Estimation and Mesh Adaptation in Computational Fluid Dynamics," *AIAA Journal*, Vol. 49, No. 4, 2011, pp. 673–694. doi:10.2514/1.J050073.

[27] Hartmann, R., Held, J., Leicht, T., and Prill, F., "Error Estimation and Adaptive Mesh Refinement for Aerodynamic Flows," *ADIGMA - A European Initiative on the Development of Adaptive Higher-Order Variational Methods for Aerospace Applications*, Notes on Numerical Fluid Mechanics and Multidisciplinary Design, Springer, Berlin, Heidelberg, 2010, pp. 339–353. doi:10.1007/978-3-642-03707-8_24.

[28] Modisette, J. M., "An automated reliable method for two-dimensional Reynolds-Averaged Navier-Stokes simulations," Ph.D. thesis, Massachusetts Institute of Technology, 2011. URL http://hdl.handle.net/1721.1/68406.

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

[30] Ceze, M., and Fidkowski, K. J., "Constrained pseudo-transient continuation," *International Journal for Numerical Methods in Engineering*, Vol. 102, No. 11, 2015, pp. 1683–1703. doi:10.1002/nme.4858.

[31] Burgess, N., and Glasby, R. S., "Advances in Numerical Methods for CREATE-AV Analysis Tools," *52nd Aerospace Sciences Meeting*, AIAA SciTech Forum, American Institute of Aeronautics and Astronautics, 2014. doi:10.2514/6.2014-0417.

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

[33] Nejat, A., and Ollivier-Gooch, C., "A high-order accurate unstructured finite volume Newton–Krylov algorithm for inviscid compressible flows," *Journal of Computational Physics*, Vol. 227, No. 4, 2008, pp. 2582–2609. doi:10.1016/j.jcp.2007.11.011.

[34] Asgharzadeh, H., and Borazjani, I., "A Newton–Krylov method with an approximate analytical Jacobian for implicit solution of Navier–Stokes equations on staggered overset-curvilinear grids with immersed boundaries," *Journal of Computational Physics*, Vol. 331, 2017, pp. 227–256. doi:10.1016/j.jcp.2016.11.033.

[35] Cavalca, D. F., Bringhenti, C., Campos, G. B., Tomita, J. T., and Silva, O. F. R., "Development and convergence analysis of an effective and robust implicit Euler solver for 3D unstructured grids," *Journal of Computational Physics*, Vol. 367, 2018, pp. 399–415. doi:10.1016/j.jcp.2018.04.005.

[36] Spalart, P., and Allmaras, S., "A One-Equation Turbulence Model for Aerodynamic Flows," *Lé Recherche Aerospatiale*, Vol. 1, 1994, pp. 5–21. doi:10.2514/6.1992-439.

[37] Allmaras, S. R., Johnson, F. T., and Spalart, P. R., "Modifications and Clarifications for the Implementation of the Spalart-Allmaras Turbulence Model," Big Island, Hawaii, 2012. URL http://www.iccfd.org/iccfd7/assets/pdf/papers/ICCFD7-1902_paper.pdf.

[38] Kenway, G. K. W., Secco, N., Martins, J. R. R. A., Mishra, A., and Duraisamy, K., "An Efficient Parallel Overset Method for Aerodynamic Shape Optimization," *Proceedings of the 58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, AIAA SciTech Forum*, Grapevine, TX, 2017. doi:10.2514/6.2017-0357.

[39] Jameson, A., Schmidt, W., and Turkel, E., "Numerical solution of the Euler equations by finite volume methods using Runge Kutta time stepping schemes," *14th Fluid and Plasma Dynamics Conference*, American Institute of Aeronautics and Astronautics, 1981. doi:10.2514/6.1981-1259.

[40] Lyu, Z., Kenway, G. K., Paige, C., and Martins, J. R. R. A., "Automatic Differentiation Adjoint of the Reynolds-Averaged Navier–Stokes Equations with a Turbulence Model," *21st AIAA Computational Fluid Dynamics Conference*, San Diego, CA, 2013. doi:10.2514/6.2013-2581.

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

[42] Brown, P., and Saad, Y., "Hybrid Krylov Methods for Nonlinear Systems of Equations," *SIAM Journal on Scientific and Statistical Computing*, Vol. 11, No. 3, 1990, pp. 450–481. doi:10.1137/0911026.

[43] Balay, S., Buschelman, K., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H., "PETSc Web page," , 2009. URL http://www.mcs.anl.gov/petsc.

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

[45] Goldfarb, D., and Toint, P. L., "Optimal estimation of Jacobian and Hessian matrices that arise in finite difference calculations," *Mathematics of Computation*, Vol. 43, No. 167, 1984, pp. 69–88. doi:10.1090/S0025-5718-1984-0744925-5.

[46] Dembo, R. S., Eisenstat, S. C., and Steihaug, T., "Inexact Newton Methods," *SIAM Journal on Numerical Analysis*, Vol. 19, No. 2, 1982, pp. 400–408. doi:10.1137/0719025.

[47] Eisenstat, S., and Walker, H., "Choosing the Forcing Terms in an Inexact Newton Method," *SIAM Journal on Scientific Computing*, Vol. 17, No. 1, 1996, pp. 16–32. doi:10.1137/0917003.

[48] Bücker, H. M., Pollul, B., and Rasch, A., "On CFL evolution strategies for implicit upwind methods in linearized Euler equations," *International Journal for Numerical Methods in Fluids*, Vol. 59, No. 1, 2009, pp. 1–18. doi:10.1002/fld.1798.

[49] Van Leer, B., and Mulder, W. A., "Relaxation Methods for Hyperbolic Conservation Laws," {*Numerical Methods for the Euler Equations of Fluid Dynamics, F. Angrand, A. Dervieux, J. A. Desideri, and R. Glowinski, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA*, 1985, pp. 312–333. URL https://www.researchgate.net/publication/265780696_Relaxation_Methods_for_Hyperbolic_Conservation_Laws.

[50] Chauhan, S. S., Hwang, J. T., and Martins, J. R. R. A., "An automated selection algorithm for nonlinear solvers in MDO," *Structural and Multidisciplinary Optimization*, Vol. 58, No. 2, 2018, pp. 349–377. doi:10.1007/s00158-018-2004-5.

[51] Osusky, M., and Zingg, D. W., "Steady three-dimensional turbulent flow computations with a parallel Newton-Krylov-Schur algorithm," *52nd Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, 2014. doi:10.2514/6.2014-0242.

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

[53] Bons, N. P., He, X., Mader, C. A., and Martins, J. R. R. A., "Multimodality in Aerodynamic Wing Design Optimization," *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Denver, CO, 2017. doi:10.2514/6.2017-3753.

[54] van der Weide, E., Kalitzin, G., Schluter, J., and Alonso, J. J., "Unsteady Turbomachinery Computations Using Massively Parallel Platforms," *Proceedings of the 44th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, 2006. doi:10.2514/6.2006-421, AIAA 2006-0421.

[55] Madsen, M. H. A., Zahle, F., Sørensen, N. N., and Martins, J. R. R. A., "Multipoint high-fidelity CFD-based aerodynamic shape optimization of a 10 MW wind turbine," *Wind Energy Science*, Vol. 4, 2019, pp. 163–192. doi:10.5194/wes-4-163-2019.

[56] Secco, N. R., and Martins, J. R. R. A., "RANS-based Aerodynamic Shape Optimization of a Strut-braced Wing with Overset Meshes," *Journal of Aircraft*, Vol. 56, No. 1, 2019, pp. 217–227. doi:10.2514/1.C034934.

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

[58] Liem, R. P., Martins, J. R. R. A., and Kenway, G. K., "Expected Drag Minimization for Aerodynamic Design Optimization Based on Aircraft Operational Data," *Aerospace Science and Technology*, Vol. 63, 2017, pp. 344–362. doi:10.1016/j.ast.2017.01.006.

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

[60] Brooks, T. R., Martins, J. R. R. A., and Kennedy, G. J., "High-fidelity Aerostructural Optimization of Tow-steered Composite Wings," *Journal of Fluids and Structures*, Vol. 88, 2019, pp. 122–147. doi:10.1016/j.jfluidstructs.2019.04.005.

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

[62] Liem, R. P., Kenway, G. K. W., and Martins, J. R. R. A., "Multimission Aircraft Fuel Burn Minimization via Multipoint Aerostructural Optimization," *AIAA Journal*, Vol. 53, No. 1, 2015, pp. 104–122. doi:10.2514/1.J052940.

[63] Klopfer, G., Hung, C., Wijngaart, R. V. d., and Onufer, J., "A diagonalized diagonal dominant alternating direction implicit (D3ADI) scheme and subiteration correction," *29th AIAA, Fluid Dynamics Conference*, American Institute of Aeronautics and Astronautics, 1998. doi:10.2514/6.1998-2824.

[64] Gleize, V., Dumont, A., Mayeur, J., and Destarac, D., "RANS simulations on TMR test cases and M6 wing with the Onera elsA flow solver (Invited)," *53rd AIAA Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, 2015. doi:10.2514/6.2015-1745.

[65] Qin, N., Xu, X., and Richards, B. E., "Newton-like methods for fast high resolution simulation of hypersonic viscous flows," *Computing Systems in Engineering*, Vol. 3, No. 1, 1992, pp. 429–435. doi:10.1016/0956-0521(92)90128-6.

[66] Swanson, R. C., and Turkel, E., "On central-difference and upwind schemes," *Journal of Computational Physics*, Vol. 101, No. 2, 1992, pp. 292–306. doi:10.1016/0021-9991(92)90007-L.

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

[68] Tinoco, E. N., Brodersen, O., Keye, S., and Laflin, K., "Summary of Data from the Sixth AIAA CFD Drag Prediction Workshop: CRM Cases 2 to 5," *55th AIAA Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, 2017. doi:10.2514/6.2017-1208.

[69] Coder, J. G., Hue, D., Kenway, G., Pulliam, T. H., Sclafani, A. J., Serrano, L., and Vassberg, J. C., "Contributions to the Sixth Drag Prediction Workshop Using Structured, Overset Grid Methods," *Journal of Aircraft*, 2017, pp. 1–14. doi:10.2514/1.C034486.

[70] Spalart, P. R., "Strategies for turbulence modelling and simulations," *International Journal of Heat and Fluid Flow*, Vol. 21, No. 3, 2000, pp. 252–263. doi:10.1016/S0142-727X(00)00007-2.

[71] Li, J., Bouhlel, M. A., and Martins, J. R. R. A., "Data-based Approach for Fast Airfoil Analysis and Optimization," *AIAA Journal*, Vol. 57, No. 2, 2019, pp. 581–596. doi:10.2514/1.J057129.