AIAA SciTech Forum
11–15 & 19–21 January 2021, VIRTUAL EVENT
AIAA Scitech 2021 Forum

10.2514/6.2021-0155

Acceleration of Adjoint-Based Adaptation through Sub-Iterations for Unsteady Simulations

Kaihua Ding* and Krzysztof J. Fidkowski.†
Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI48105

Sub-iterations in adjoint-based adaptation refer to reusing primal states and fine-space adjoints, in multiple adaptive iterations. This technique has been demonstrated for steady compressible Navier-Stokes simulations, for which it achieved significant CPU time savings as compared to standard adjoint-based adaptation. In this paper, we extend the sub-iterations idea of formulating approximated and relatively accurate adaptive indicators by reusing solver information, to unsteady problems. The steady sub-iteration only needs to reconstruct in space in order to reuse the previous adaptation iteration's primal states and fine-space adjoints, while unsteady sub-iterations require both spatial and temporal reconstruction due to mismatched adapted time histories. In this paper, we present our results for simulations of the unsteady Navier-Stokes equations, with p adaptation.

I. Introduction: The Sub-Iteration Method for Steady Problems

A. Cost of Adjoint-Based Adaptation

Adjoint-based adaptation algorithms have been used in a wide range of CFD applications to achieve a faster and more accurate solution, with added benefits of obtaining tailored meshes (in the case of h or r adaptation) [1]. However, adjoint-based adaptation is computationally expensive in its most rigorous form [2].

Denote by $\mathbf{U}_H$ and $\mathbf{U}_h$ the primal solutions on the coarse and fine space, respectively. Also, let $\mathbf{R}_H$ and $\mathbf{R}_h$ denote the discrete residual vectors, both of which are functions of their respective primal states. Finally, let J_H and J_h be scalar outputs computed on the coarse and fine spaces. We assume that the output definition does not change between the coarse and fine spaces, so that $J_H\left(\mathbf{U}_H\right) = J_h\left(\mathbf{U}_h^H\right)$, where $\mathbf{U}_h^H$ is the injection of the coarse solution, $\mathbf{U}_H$, into the fine space. The standard adjoint-weighted residual error estimate of the output difference between the coarse and fine spaces reads [3–5]

$$\underbrace{J_h\left(\mathbf{U}_h^H\right) - J_h\left(\mathbf{U}_h\right)}_{\delta J} = \boldsymbol{\Psi}_h^T \delta\mathbf{R}_h = -\boldsymbol{\Psi}_h^T \mathbf{R}_h\left(\mathbf{U}_h^H\right), \tag{1}$$

where $\delta\mathbf{R}_h$ is the residual perturbation due to the difference between $\mathbf{U}_h^H$ and $\mathbf{U}_h$. The discrete fine-space adjoint, $\boldsymbol{\Psi}_h$, weights the residual perturbation, $\delta\mathbf{R}_h$, to give a linearized estimate of the output difference between solutions on the coarse and fine spaces. $\boldsymbol{\Psi}_h$ is a vector of the same size as the state and residual vectors, and it satisfies the following adjoint equation:

$$\left(\frac{\partial\mathbf{R}_h}{\partial\mathbf{U}_h}\right)^T \boldsymbol{\Psi}_h + \left(\frac{\partial J_h}{\partial\mathbf{U}_h}\right)^T = \mathbf{0}. \tag{2}$$

Although the fine-space primal state is not required in output-based error estimation, the computational and storage costs associated with Eq.(1) are not trivial, as indicated below:

$$\delta J \approx \underbrace{-\boldsymbol{\Psi}_h^T}_{\text{fine space adjoint}} \quad \underbrace{\mathbf{R}_h}_{\text{fine space residual operator}} \quad \Big(\underbrace{\mathbf{U}_h^H}_{\text{injected state}} \Big). \tag{3}$$

*R&D Engineer, ANSYS Inc.
†Associate Professor, University of Michigan, AIAA Senior Member.

For error estimation, we need to inject the state into the fine space [6–15]. A common way to construct a finer space is uniform mesh refinement, which, for steady problems, increases the spatial DOF four-fold in two dimensions and eight-fold in three dimensions. For unsteady problems, uniform refinement increases DOF eight-fold in two dimension and sixteen-fold in three dimensions. Along with the increase in DOF, computational overhead arises in the form of element geometry quantities, basis functions, mappings, etc., some or all of which are used in the calculation of the fine-space residual. Eq.(3) requires the fine-space adjoint solution, and this involves either a reconstruction or a system solve on the fine space [16–21].

The adjoint-weighted residual error estimate is typically paired with mesh adaptation, and the mesh is successively refined to reduce the error. Often, the mesh is refined incrementally; for example, using hanging-node element subdivision of a fixed-fraction of elements with the highest error. In such cases, many adaptive iterations may be required to sufficiently reduce the output error, and at each iteration, the adjoint-weighted residual calculation must be repeated in order to obtain adaptive indicators to drive the adaptation process.

B. Sub-iteration Algorithm for Steady Problems

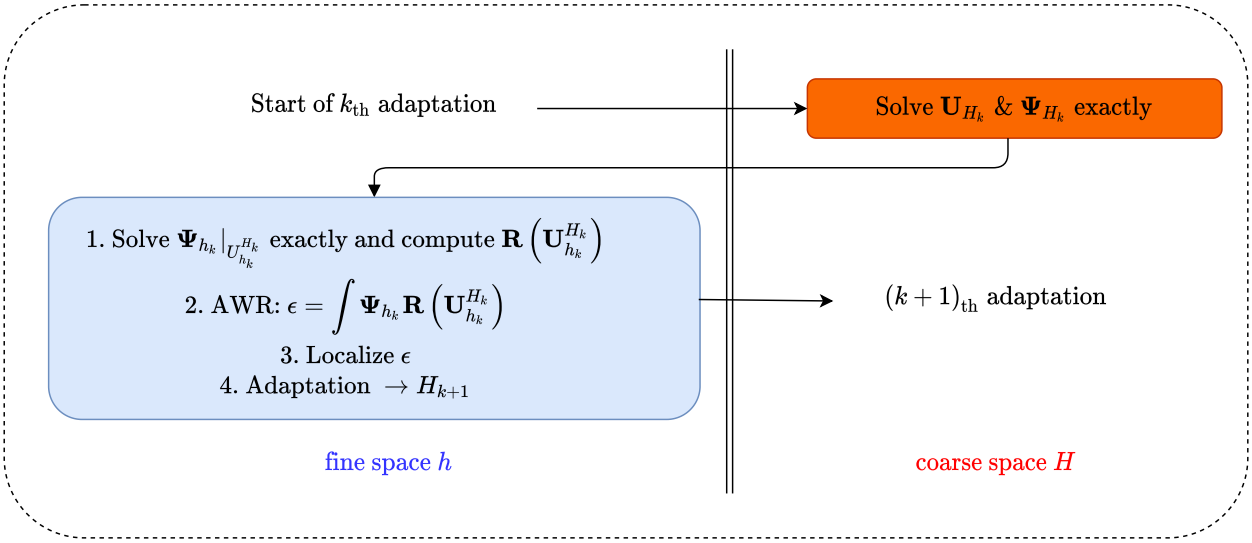
To address the cost of solving the adjoint on a finer discretization, the sub-iteration algorithm is formulated. For steady simulations with sub-iterations, we reuse primal states $\mathbf{U}_{H_{k-1}}$ solution through a smoothing process [16–21] to obtain $\mathbf{U}_{H_k}$ and solve the adjoint $\mathbf{\Psi}_{H_k}$ exactly to secure an accurate error estimate. k is the index for the k^{th} adaptation iteration. Because of the lack of a convergence requirement for $\mathbf{U}_{H_k}$ (obtained by smoothing $\mathbf{U}_{H_{k-1}}$), the accuracy of error estimates tends to deteriorate for prolonged sub-iteration adaptation cycles. Thus, standard adaptation cycles are interspersed with sub-iteration adaptation cycles, so that the sub-iteration algorithm still produces reasonable adaptive indicators with the added benefit of accurate error estimates produced by the standard adaptation cycles. The sub-iteration algorithm scheme can be expressed as shown in Fig. 1(b). For comparison, the standard adjoint-based adaptation algorithm is shown in Fig. 1(a). The comparison between Fig. 1(a) and Fig. 1(b), shows that the sub-iteration and standard algorithm's difference lies in their respective treatments of the current space primal, $\mathbf{U}_H$, and the fine-space adjoint, $\mathbf{\Psi}_h$. The sub-iteration algorithm approximates both $\mathbf{U}_{H_k}$ and $\mathbf{\Psi}_{h_k}$, while the standard adjoint-based adaptation algorithm solves for these 2 quantities exactly.

Fig. 1(a) and Fig. 1(b), show what a single adaptation iteration looks like for the standard and sub-iteration algorithms. In practice, error estimation and adaptation consists of multiple adaptation iterations, i.e., many Fig. 1(a) and Fig. 1(b) connected in tandem, depicted in Fig 2. Here, we address a single standard adaptation iteration (Fig. 1(a)) as a standard block and a single sub-iteration adaptation iteration (Fig. 1(b)) as a sub-iteration block. In Fig 2, standard algorithm only consists of standard adaptation blocks. On the other hand, for sub-iteration algorithms, standard adaptation blocks are interspersed with sub-iteration adaptation blocks. The designation "sub-iteration 1" signifies that we intersperse 1 sub-iteration block per standard adaptation block; "sub-iteration 2" means that we intersperse 2 sub-iteration blocks per standard adaptation block; "sub-iteration 3" means that we intersperse 3 sub-iteration blocks per standard adaptation block; and, "sub-iteration s" means s sub-iteration blocks per standard adaptation block. Per our parametric studies on s ([2]), we do not recommend using a large number for s as adaptive indicator accuracy deteriorates with large s. For all steady problems tested, sub-iteration 1 and sub-iteration 2, have proven to be optimal. The details of sub-iteration algorithm for steady simulation can be found in [2].

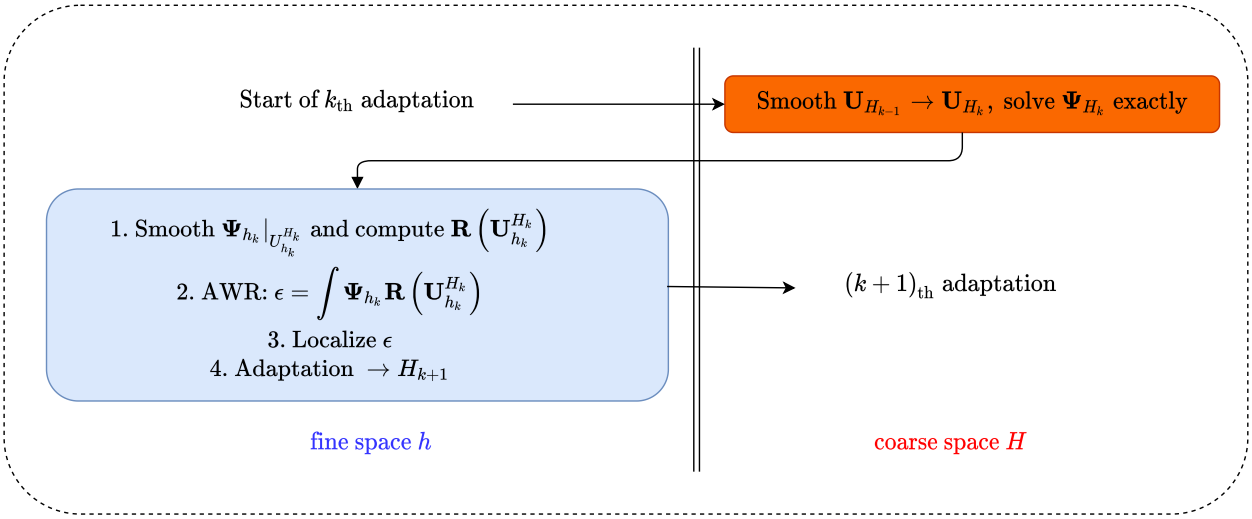
In addition, we show one of the test cases, which we presented in [2] for the steady sub-iteration algorithm. This test case is a NACA 0012 airfoil at a free-stream Mach number of 0.5, angle of attack of 2°, Prandtl number of 0.71, and Reynolds number of 5000. Our output of interest is the drag coefficient. The initial mesh consists of 234 quadrilaterals, curved with a quartic geometry representation. The fixed adaptation fraction for all adaptation strategies is the same, $f = 0.1$, and the approximation order is $p = 2$.

Figure 3 shows a comparison of the various adaptive strategies for this case. Figure 3(c) shows that the methods with sub-iterations exhibit a similar output error convergence behavior with "degrees of freedom" (DOF) compared to standard adaptation. However, as shown in Fig. 3(d)*, sub-iterations show an advantage in CPU time over standard adaptation. The runs with two and three sub-iterations converge the fastest. As the number of sub-iteration increases, from 3 to 5, the error-estimate-corrected curves converge progressively slower. As the number of sub-iterations rises, the accuracy of adaptive indicators gradually deteriorates and sub-iteration based adaptations becomes less effective.

*Timings were performed on a workstation with dual socket Intel Xeon 2.1 GHz 8-core processors and 96GB total RAM.



(a) Standard adaptation block



(b) Sub-iteration adaptation block

Fig. 1 Schematic of the error estimation and adaptation iteration, for both adjoint-based standard and sub-iteration algorithms, for steady simulations. The primal states and fine-space adjoints are reused in the sub-iterations, where they are only smoothed via an inexpensive iterative solver, e.g., 5 element block Jacobi iterations, thereby saving computational time compared to the standard approach, in which both primal states and fine-space adjoints are re-solved at every adaptive iteration.

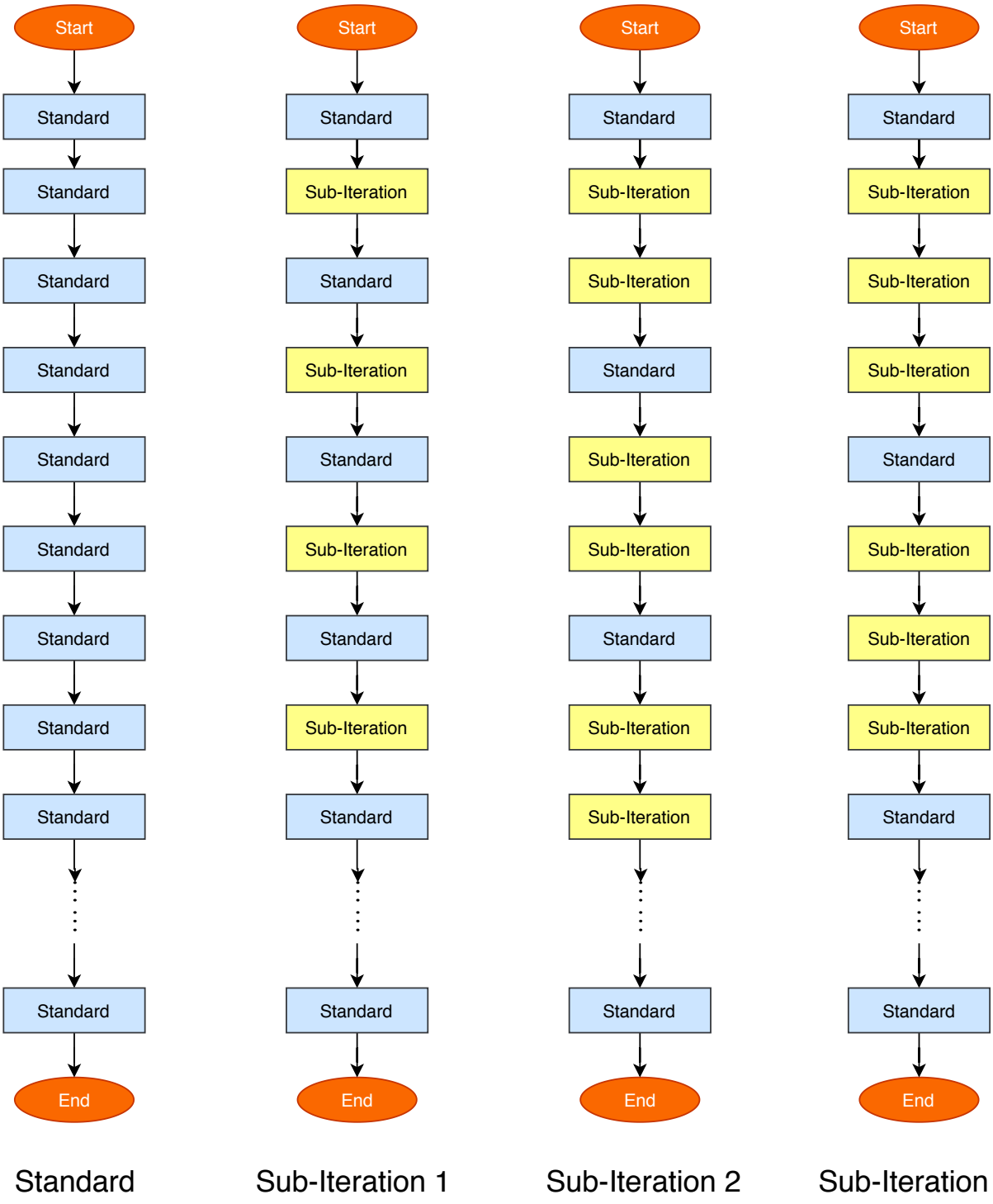
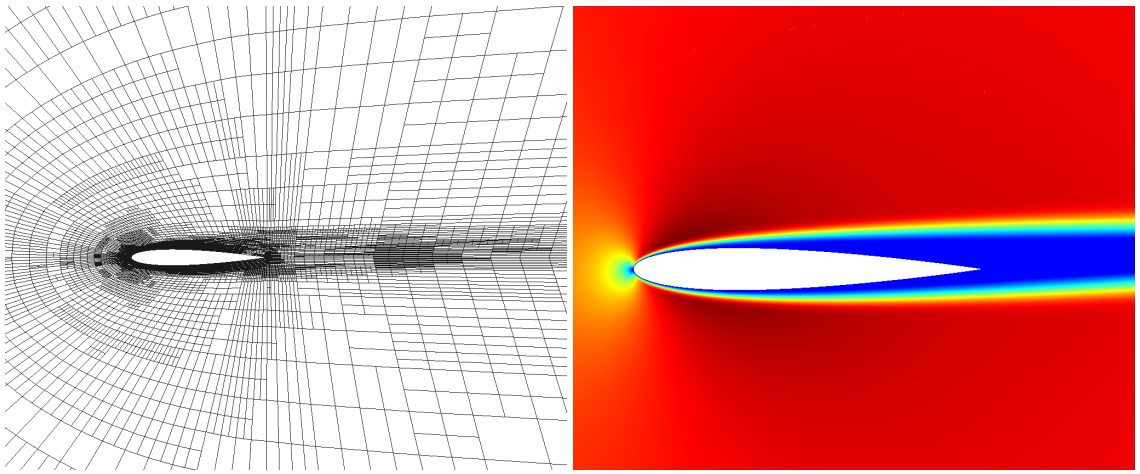
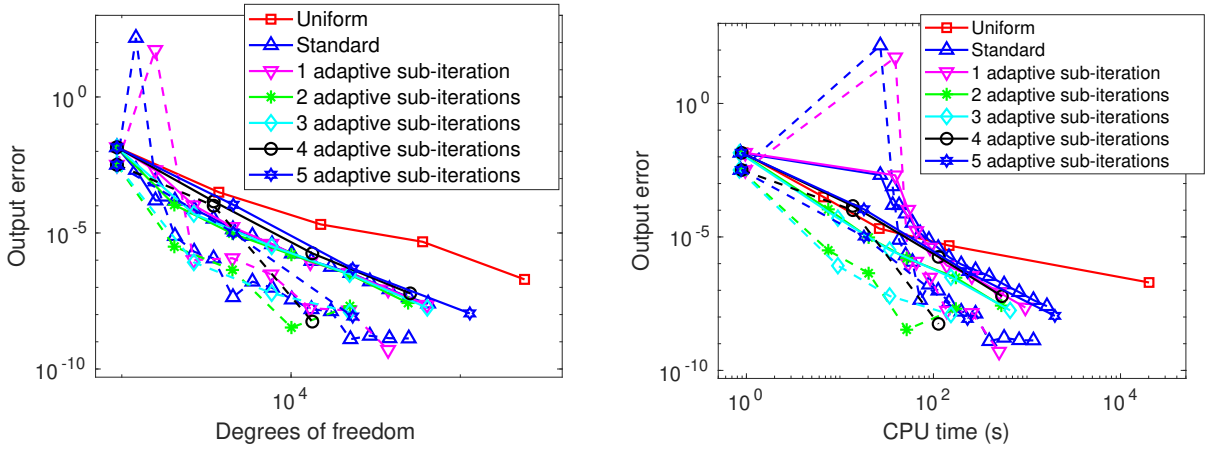


Fig. 2 Illustration of the successive adjoint based error estimation and adaptation process, for standard, sub-iteration 1, sub-iteration 2, and sub-iteration 3 algorithms.



(a) Final drag-adapted mesh

(b) Mach contours (0 to 0.58)



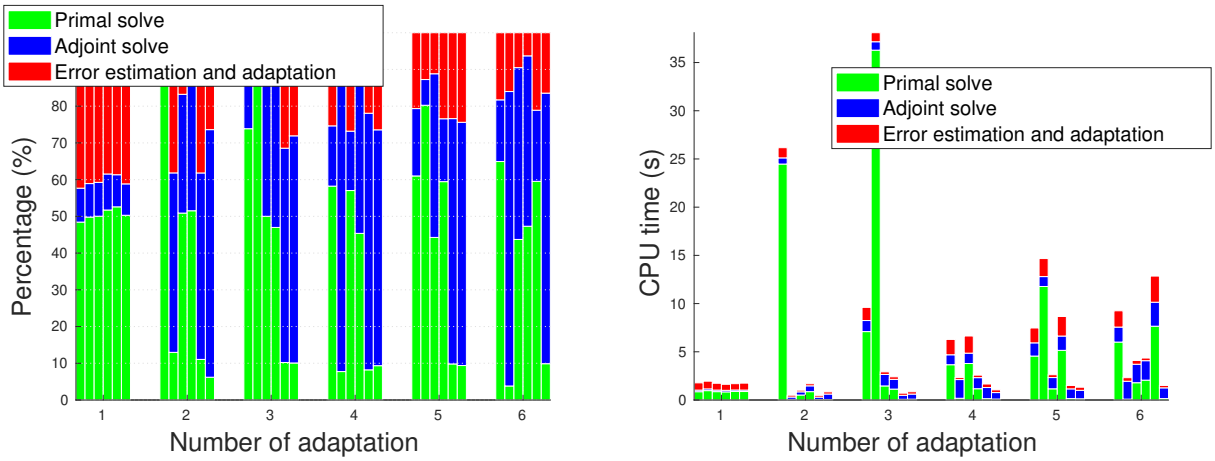
(c) Drag convergence with DOF

(d) Drag convergence with CPU time

Fig. 3 NACA 0012, $M = 0.5$, $\alpha = 2^\circ$ and Re $= 5000$: effect of sub-iterations on drag convergence. In all 5 cases employing sub-iterations, the fine-space adjoint is reused on the sub-iterations with only one element block-Jacobi smoothing iteration as the extra solve. The current space primal is only block-Jacobi smoothed on the current space, but the linear coarse-space adjoint problem is solved exactly for all iterations. Dashed lines indicate the remaining error after correction with the estimate. Note, for the output-based method, the error estimate corrected convergence curves, the dashed lines, are the final post-processed convergence curves.

Nevertheless, in Fig. 3(d), the higher frequency sub-iteration methods still outperform the standard adaptation.

Figure 4 shows the CPU-time breakdown comparison among standard and sub-iterative adaptations. The first 3 iterations of the CPU-time breakdown data contain noise – this is the spike of absolute CPU time for the first bar (standard adaptation) in the 2nd iteration and the spike of the second bar in the third iteration (1 sub-iteration). Both CPU time spikes correspond exactly to each spike point in Figure 3(c) and 3(d). The zig-zag spikes likely indicate a lack of resolution on the coarser mesh in the earlier stage of simulation. Thus, it is not meaningful to look at the first 3 iterations to study the behavior of the 1 sub-iteration method, whose error estimate is likely not yet accurate. During the sub-iterations, the CPU time spent on the primal solve and the error estimation decreases relative to the standard adaptation, but the CPU time spent on the current-space adjoint solves remains similar. As a result, whenever the adaptation mechanics enters a sub-iteration cycle, the total time drops (Figure 4(b)) while the percentage of the time taken by the adjoint solve increases (Figure 4(a)).



(a) Percentage CPU time breakdown

(b) Raw CPU time breakdown

Fig. 4 NACA 0012, $M = 0.5$, $\alpha = 2^\circ$ and Re $= 5000$: CPU time breakdown results. For the first 6 adaptive iterations, we show 2 group-bar plots. One group-bar plot illustrates percentage CPU time expenditure and the other one illustrates absolute CPU time expenditure. In each group-bar plot, a grouped column consists of 6 bars: standard adaptation, and adaptation with 1-5 sub-iterations. Each of these bars is divided vertically into three parts, which indicate the CPU time contribution of the primal solve, performed on H space (green), the adjoint solve, performed on H space (blue), and the error estimation and adaptation, performed on h space (red). Note that the latter includes any and all fine-space solves.

II. Sub-Iteration Algorithm for Unsteady Problems

The sub-iteration idea of reusing primal states and fine-space adjoints information extends naturally to unsteady problems. However, because of the extra dimension in time. Both space and time reconstruction are required for unsteady sub-iterations. That is, we reconstruct $\mathbf{U}^{j'}_{H_{k-1}}$ from spatial discretization H_{k-1} at j'^{th} time index, to obtain $\mathbf{U}^{j\,H_{k-1}}_{H_k}$, where $j \in \left[1, \ldots, N_t^{k^{\text{th}}\text{ sub-iterations}}\right]$ and $j' \in \left[1, \ldots, N_t^{(k-1)^{\text{th}}\text{ sub-iterations}}\right]$, in Fig. 5(b). Nevertheless, the idea of saving computational cost by reusing primal and fine-space adjoints still holds.

The unsteady adjoint equation reads,

$$\sum_{i=1}^{N_t} \left(\frac{\partial \mathbf{R}_h^i}{\partial \mathbf{U}_h^j}\right)^T \boldsymbol{\Psi}_h^i + \left(\frac{\partial J_h}{\partial \mathbf{U}_h^j}\right)^T = \mathbf{0}, \tag{4}$$

where $i, j \in [1 \dots N_t]$ are time indices in the unsteady discretization (explicit or implicit). Unsteady adjoint-based error estimation involves adjoints and primal state residuals from the entire simulation history. The unsteady adjoint error estimate formula reads,

$$\delta J = -\sum_{i=1}^{Nt} {\Psi^i}_h^T \mathbf{R}_h^i \left({\mathbf{U}^j}_h^H \right). \tag{5}$$

The unsteady adjoint-based error estimation and adaptation process can be described schematically, as in Fig. 5(a), which is different from the steady version in Fig. 1(a).

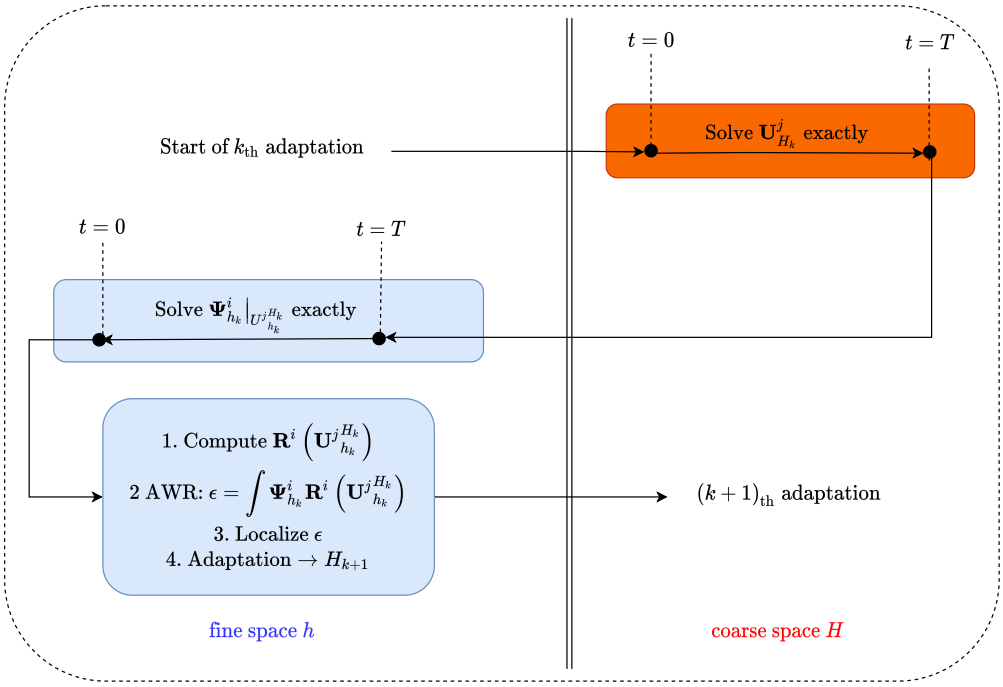
Unsteady adjoint-based adaptation has more adaptation strategy choices compared to steady simulations. Steady simulations can only support static adaptation, i.e., adapting the spatial mesh in between adaptive iterations. However, unsteady simulation can use both static adaptation, i.e., adapting the spatial and temporal meshes at the end of the primal solve with temporally and spatially marginalized error indicators; and dynamic adaptation, i.e., adapting the mesh before the end of primal solve, $1 \leq i < N_t$, both temporally and spatially. For this paper, we only focus on static adaptation for unsteady simulations. Namely, we adapt the temporal and spatial resolution in between adaptive iterations, using marginalized error indicators. Regardless, the schematic that we show, Fig. 5(a), can be used for all kinds of adaptation strategies.

The sub-iteration algorithm schematic for unsteady simulation follows the same idea as the sub-iteration algorithm for steady simulation, Fig. 1(b). The proposed schematic of an unsteady sub-iteration algorithm is given in Fig 5(b). In contrast to standard method (Fig. 5(a)), the primal state $\mathbf{U}^j_{H_k}$ in sub-iteration algorithm (Fig. 5(b)) is not acquired thorough an exact solve. For the k^{th} sub-iteration adaptation cycle, solution from the $(k-1)^{\text{th}}$ adaptation ($\mathbf{U}^{j'}_{H_{k-1}}$), is injected into H_k, a spatially and temporally adapted discretization space, then smoothed to acquire $\mathbf{U}^j_{H_k}$. Noted that we used j' and j, two different temporal indices. Because unsteady adaptations allow for temporal adaptation, the adaptated time history from the $(k-1)^{\text{th}}$ adaptation is different from the k^{th} adaptation. For example, the j'^{th} time node from the $(k-1)^{\text{th}}$ adaptation may not exist in the k^{th} adaptation. Fig. 6 plots the standard adaptation's time history of the first and the second adaptations of the Section III.A case. Fig. 6 shows that the second adaptation contains more time nodes (j) than the first adaptation (j'). What's more, j time nodes and j' time nodes appear to be mismatched.

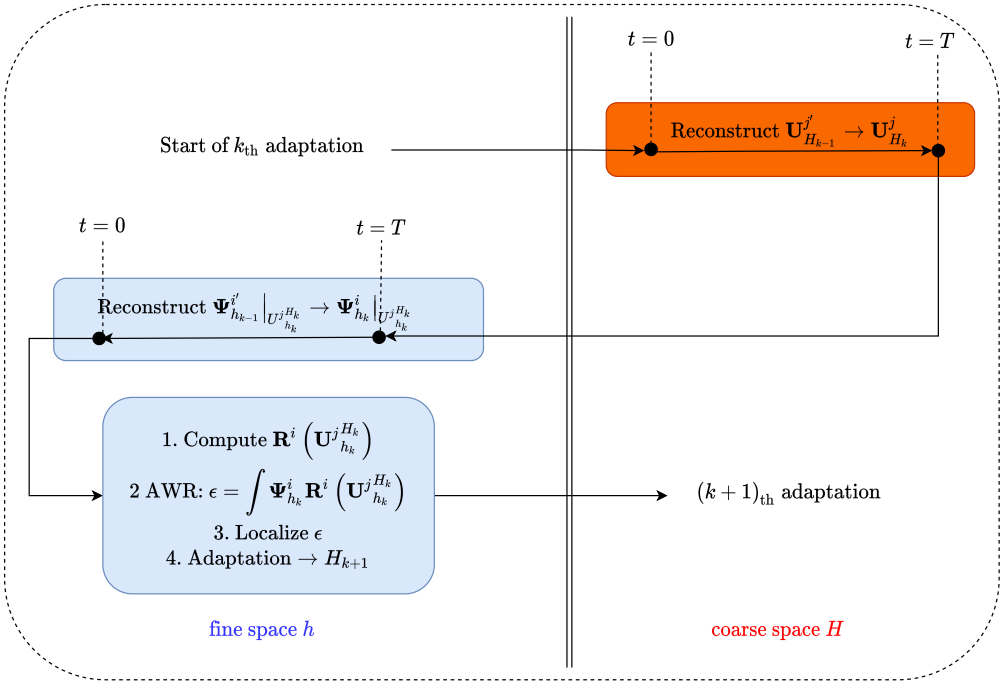
To obtain all the time nodes from the k^{th} adaptation (j instead of j'), other than spatial injections, we also need to reconstruct $\mathbf{U}^{j'}_{H_{k-1}}$ in time to compute $\mathbf{U}^j_{H_k}$ [22]. We adopt the polynomial-based temporal reconstruction mentioned in [22]. For the linear case, the combination of spatial and polynomial-based temporal reconstruction [22] is sufficient to produce a relatively accurate $\mathbf{U}^j_{H_k}$. However, polynomial-based temporal reconstruction [22] cannot accurately capture highly non-linear features, for instance, vortex shedding phenomena. As a result, for highly non-linear physical features, the simulation often runs into non-physical errors. Thus, we post-process the reconstructed primal states by solving the reconstructed primal states to a low residual tolerance, e.g., to 10^{-2} to avoid the non-physicality. The effect of this low tolerance primal post-processing on overall adaptation CPU time cost is small. Unlike steady adjoint-based adaptation, in which the primal solve often accounts for 30% to 50% of the overall CPU time (the first and the second case of [2]), the primal solve in unsteady adjoint-based adaptation simulations often accounts for less than 20% of the overall CPU time (Section III). Consequently, the cost introduced by post-processing temporally reconstructed primal states is small.

This phenomenon of reduced unsteady primal CPU time percentage is due to the increased fine-space computation costs in unsteady simulations. In 2D, a steady simulation's fine-space cost in DOF is a factor of 4, whereas an unsteady simulation's fine-space cost rises to a factor of 8 in DOF. In 3D, these factors become 8 and 16, respectively. A similar percentage reduction of primal CPU time cost, due to difference in fine-space computations, was also observed in steady simulation [2], 2D versus 3D, i.e., for 2D cases, the primal CPU time often accounts for more than 50% of the overall CPU cost (the first and second case in [2]), while, for 3D, the primal CPU time often accounts for less than 20% of the overall CPU cost (the third case in [2]).

The standard unsteady sub-iteration adjoint solve does not usually compute the coarse space, H, adjoint. Unlike the steady case, unsteady adjoints take the form of a series of time-dependent quantities. For steady simulations, computing coarse-space H, adjoints first, then injecting the coarse-space adjoints into a fine space, h, and applying a smoothing process not only provides a coarse-space adjoint correction but also carries a similar cost compared to solving adjoints on the fine space directly. For unsteady adjoints, repeatedly injecting the entire unsteady adjoint series into the fine space, then time stepping or smoothing adjoints is more expensive than constructing the fine-space terminal adjoint condition once and performing a fine-space linear solve. Consequently, we generally do not store coarse H space



(a) Standard adaptation



(b) Unsteady sub-iteration adaptation

Fig. 5 Schematic of an unsteady adjoint-based error estimation and adaptation iteration, in which the fine-space adjoint is solved exactly at every iteration.

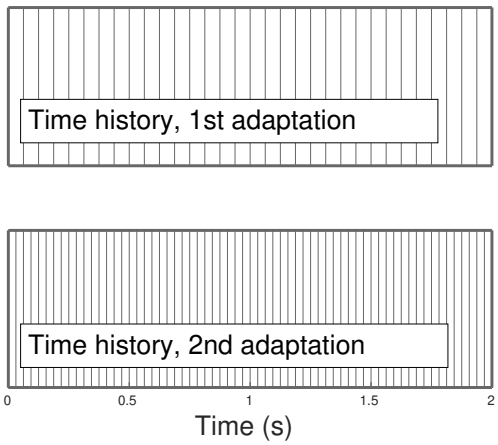


Fig. 6 Time history difference between 2 adaptation iterations.

adjoints in unsteady simulations. Therefore, we cannot copy the steady sub-iteration algorithm's adjoint approximation strategy (Fig. 1(b)) directly. Inspired by [22], we choose to reconstruct adjoints using polynomial-based reconstruction, by reusing fine-space adjoints information, without any post-processing. Through this approach, the unsteady adjoints computation time is greatly reduced. Owing to the fact that the unsteady adjoint solution accounts for the majority of the CPU time expenditure in unsteady simulations (higher space-time dimension than steady problems), the overall CPU time costs are also reduced.

In summary, for the steady sub-iteration algorithm, spatially injected primal states and adjoints can achieve decent accuracy after a small number of smoothing iterations, [2]. However, for unsteady sub-iterations, both spatial injections and temporal reconstruction are needed. For unsteady simulations, a few smoothing iterations are often not sufficient to obtain a physically sound solution, as spatial injections are limited by the discritization scheme's order, and temporal reconstruction is limited by the power of reconstruction polynomials. As a result, non-physical states are likely to appear without post-processing the reconstructed primal states. To overcome this issue, we choose to solve our primal state to a low tolerance, ϵ, to avoid non-physical states. The value of ϵ does not strongly affect the overall CPU time expenditure, as the primal state computational time cost is much smaller than the adjoint time for unsteady simulations (Fig. 8(c) and Fig. 11(c)). The adjoint represents sensitivity information and does not suffer from the same physicality constraint as the primal states.

Steady sub-iteration (Fig 1(b)) and unsteady sub-iteration (Fig 5(b)), share the same overall sub-iteration formulations (Fig 2). Following the same naming convention that we used for steady sub-iterations (Section I.B), for the remaining of the paper, we address a single unsteady standard adaptation (Fig. 1(a)) as a standard block and a single unsteady sub-iteration adaptation (Fig. 5(b)) as a sub-iteration block.

We describe the unsteady sub-iteration algorithm in more details in Algorithm 1.

Algorithm 1: Unsteady sub-iteration algorithm: e_{estimate} is the error estimate, e_{desired} is the desired error, $\text{Cycle}_{\text{sub-iterations}}$ is a Boolean flag indicating whether or not to perform sub-iterations, β denotes the frequency of sub-iterations, H_k denotes the discretization space resulting from the k^{th} adaptation, H_{k-1} denotes the discretization space resulting from the $(k-1)^{\text{th}}$ adaptation, h_k denotes the corresponding fine space of H_k, and h_{k-1} denotes the corresponding fine space of H_{k-1}. The symbols H and h refer to both spatial and temporal discretization. k indexes adaptation iteration. i, i', j, j' = time indices in the unsteady discretization (explicit or implicit).

initialization, $\text{Cycle}_{\text{sub-iterations}}$ = `False`;
while $e_{estimate} > e_{desired}$ **do**
  if $Cycle_{sub\text{-}iterations}$ **then**
    reconstruct and smooth $\mathbf{U}^{j'}{}_{H_{k-1}} \longrightarrow \mathbf{U}^{j}{}_{H_k}$;
    inject $\mathbf{U}^{j}_{H_k} \longrightarrow \mathbf{U}^{j}{}^{H_k}_{h_k}$;
    reconstruct $\mathbf{\Psi}^{i'}{}_{h_{k-1}} \longrightarrow \mathbf{\Psi}^{i}{}_{h_k}$ calculate $\mathbf{R}^{i}_{h_k}\left(\mathbf{U}^{j}{}^{H_k}_{h_k}\right)$;
    compute AWR, $\epsilon = -\sum_{i=1}^{Nt} \mathbf{\Psi}^{i}{}^{T}_{h} \mathbf{R}^{i}_{h}\left(\mathbf{U}^{j}{}^{H}_{h}\right)$;
  else
    solve $\mathbf{U}^{j}_{H_k}$ exactly;
    inject $\mathbf{U}^{j}_{H_k} \longrightarrow \mathbf{U}^{j}{}^{H_k}_{h_k}$;
    solve $\mathbf{\Psi}^{i}_{h_k}\Big|_{\mathbf{U}^{j}{}^{H_k}_{h_k}}$ exactly;
    calculate $\mathbf{R}^{i}_{h_k}\left(\mathbf{U}^{j}{}^{H_k}_{h_k}\right)$;
    compute AWR, $\epsilon = -\sum_{i=1}^{Nt} \mathbf{\Psi}^{i}{}^{T}_{h} \mathbf{R}^{i}_{h}\left(\mathbf{U}^{j}{}^{H}_{h}\right)$;
    $e_{\text{estimate}} = \epsilon$;
  end
  localize ϵ to coarse elements;
  adapt coarse space;
  k ++ ;
  if (mod $[k, \beta] = 0$) **then**
    $\text{Cycle}_{\text{sub-iterations}}$ = `True`
  else
    $\text{Cycle}_{\text{sub-iterations}}$ = `False`
  end
end
post-processing;

Our research work is based on xflow, a high-order finite element CFD in-house library ([23]). We present our results in Section III.

III. Results

In this section, we demonstrate the unsteady sub-iteration algorithm – Algorithm 1, with a linear case and a non-linear case. To have a readily available benchmark, we opt for the same physical setups used in [22]. Unlike [22], instead of applying the MOESS method to these setups, we apply unsteady sub-iteration algorithms. Adjoints used for both Section III.A and Section III.B results are continuous in time.

A. Linear Case – Scalar Advection and Diffusion

We present a 2D advection-diffusion case. The advection-diffusion governing equation is

$$\frac{\partial u}{\partial t} + \nabla \cdot \left(\vec{V} u\right) - \nabla \cdot (\nu \nabla u) = 0, \tag{6}$$

where u is the primal state, $\vec{V}$ is the advection velocity and $\nu = 0.01$ is the viscosity. The initial condition is a Gaussian distribution centered at $(2.1, 0.9)$, with an amplitude of 1 and variance of 0.02, depicted in Fig. 7(a). The velocity field is induced by an irrotational counterclockwise vortex flow centered at $(-1, -1)$, with a vortex strength of $\Gamma = 3$. The Péclét number based on the vortex strength is $Pe \equiv \Gamma/\nu = 300$. We study this linear case on a square mesh of 3 by 3, $p = 1$, with 576 quadrilateral elements, as shown in Fig. 9(a).

Our output of interest is a space time weighted integral of u,

$$\bar{J} = \int_0^T \int_\Omega w_t\,(t)\, w_\Omega\,(t)\, u\,(x,t)\, d\Omega dt, \tag{7}$$

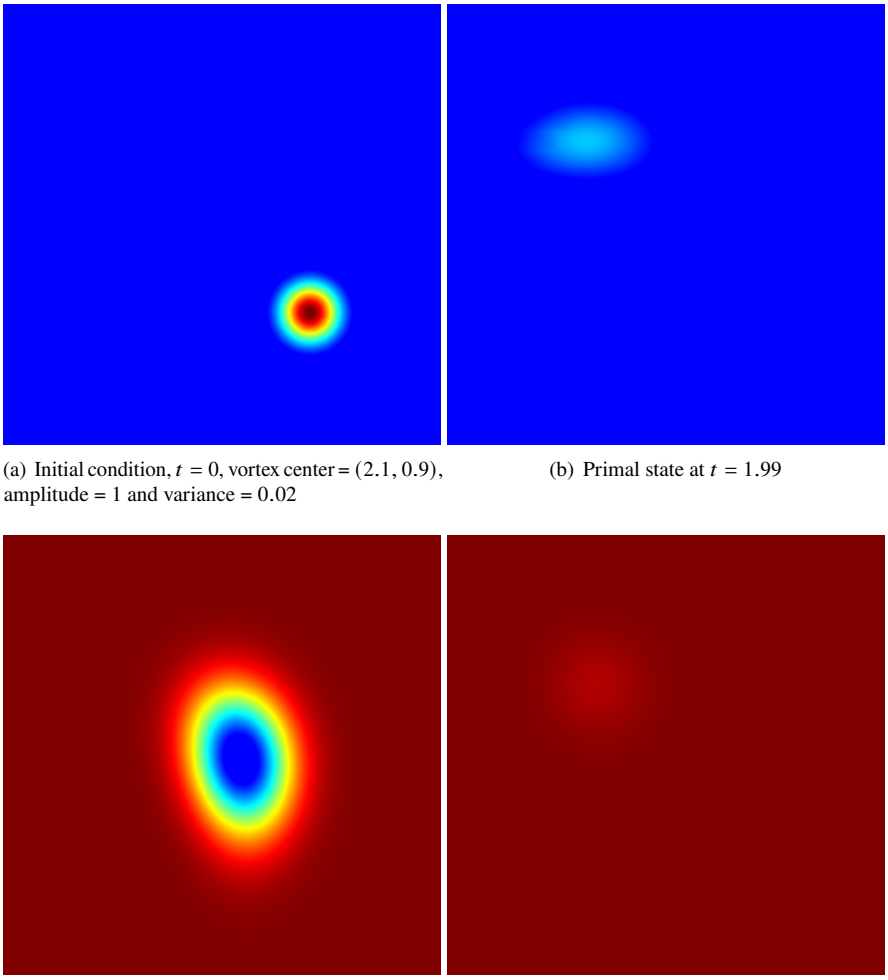
where the spatial weight function is $w_\Omega = \exp\left[-10(x-1)^2 - 10(y-2)^2\right]$, and the temporal weight is $w_t = \exp\left[-30(t-1)^2\right]$. Including the spatial and temporal weight functions in our output of interest, ensures that the adjoint is a smooth function in space and time.

We apply, standard adaptation, uniform h refinement, uniform p refinement, sub-iteration 1 adaptation, sub-iteration 2 adaptation and sub-iteration 3 adaptation strategies, to the primal problem described in Fig 7. For uniform h and uniform p refinement, time-step subdivision is employed. For standard adaptation, sub-iteration 1, sub-iteration 2, and sub-iteration 3, both spatial (p) and temporal (number of time-steps) degrees of freedom are adapted. In Fig. 8(a), "Degrees of freedom" (DOF) = "spatial degrees of freedom" × "temporal degrees of freeedom". Using the DOF cost measurement, all of the sub-iteration curves, sub-iteration 1, sub-iteration 2 and sub-iteration 3 (green stars, cyan diamonds, and black circles), coincide with the standard adaptation convergence curve (blue triangles), which means that the sub-iteration and standard adaptation converge at a similar rate. As shown in Fig. 8(a), all sub-iteration and standard adaptation strategies are below the uniform p and uniform h curves (red squares and blue crosses), which shows that all sub-iteration and standard adaptation strategies converge faster than uniform refinement, h and p.

In the CPU plot in Fig. 8(b), all sub-iteration curves are below standard uniform h and uniform p convergence curves, which means that the sub-iterations converge faster, with sub-iteration 3 being the fastest. The accelerated convergence rate of sub-iterations can be best explained thorough Fig. 8(c) and Fig. 8(d). In Fig. 8(c) and Fig. 8(d), we present 5 bar-groups corresponding to 5 adaptation iterations (horizontal axis). Each bar-group contains 4 bars. From left to right, they are: standard, sub-iteration 1, sub-iteration 2 and sub-iteration 3. First, we focus on Fig. 8(c). During the first adaptation, all adaptation strategies perform similarly. During the second adaptation, the 1$^{\text{st}}$ bar in the 2$^{\text{nd}}$ bar are taller than the rest. Recalling from Fig. 2, at the second adaptation, sub-iteration 1, sub-iteration 2, and sub-iteration 3 all enter the sub-iteration block (Fig. 5(b)), while the leftmost bar – the standard method, is the only method that enters a standard block (Fig. 5(a)). Hence, the leftmost bar (standard) at the second adaptation (Fig 8(c) and Fig. 8(d)) is the tallest. For the same reason, during the third adaptation, the 1$^{\text{st}}$ (standard) bar and the 2$^{\text{nd}}$ (sub-iteration 1) bar are the tallest because they enter the standard block (Fig. 5(a)), while the 3$^{\text{rd}}$ (sub-iteration 2) bar and 4$^{\text{th}}$ (sub-iteration 3) bar enter the sub-iteration block (Fig. 5(b)). The height difference in the fourth and the fifth bar-groups can be explained the same way, by referring to Fig. 2.

All sub-iteration methods and standard method have similar performances in terms of DOF (Fig. 8(a)), which means that their adaptation choices are similar. However, as Fig. 8(c) shows, because sub-iteration 1, sub-iteration 2 and sub-iteration 3 have a smaller CPU time costs during their sub-iteration block, compared to the standard method, the overall cumulative CPU time costs of sub-iteration 1, sub-iteration 2, and sub-iteration 3 end up being faster than the standard method. Fig. 8(d) shows that when a method enters a sub-iteration, it has a smaller percentage of the blue bar, which indicate a less-expensive adjoint solve.

In Fig. 8(a) and Fig. 8(b), the dashed lines are error estimate corrected output error convergence curves. The initial and the last p adapted domain are plotted side-by-side in Fig. 9. All order-adapted domain plots look similar, (standard, sub-iteration 1, sub-iteration 2 and sub-iteration 3). We choose to only plot sub-iteration 3's order adapted solution domain, shown in Fig. 9.



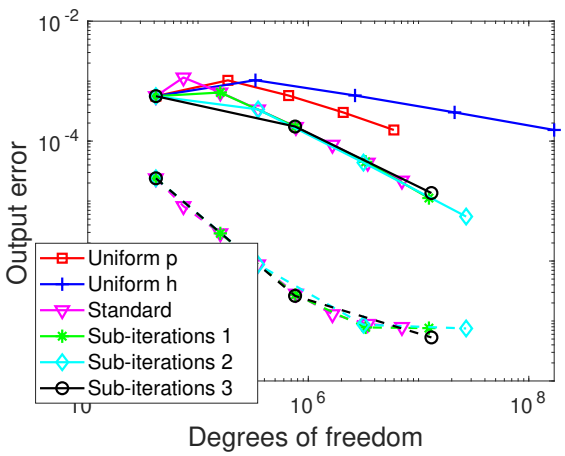
(a) Initial condition, $t = 0$, vortex center = $(2.1, 0.9)$, amplitude = 1 and variance = 0.02

(b) Primal state at $t = 1.99$

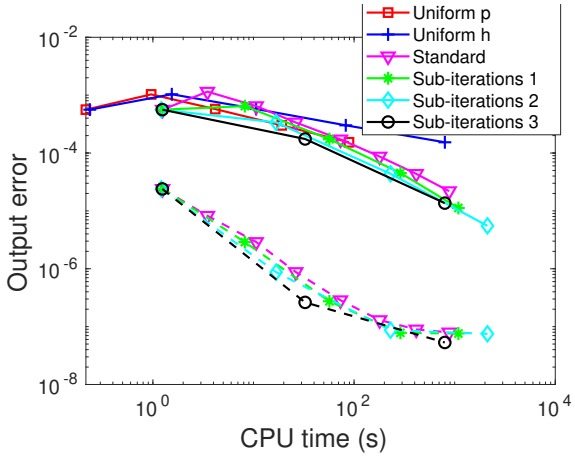
(c) Adjoint at $t = 0$

(d) Adjoint at $t = 1.24$

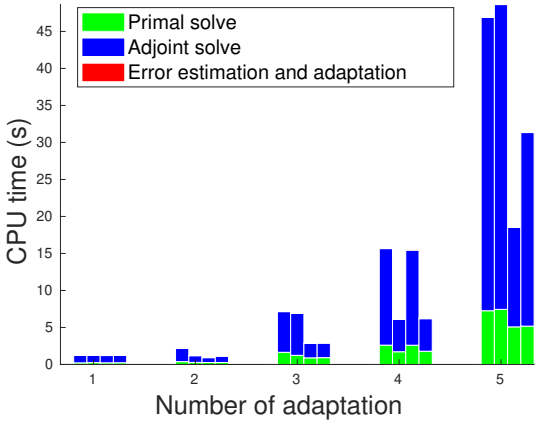
Fig. 7 The primal problem setup for the scalar advection-diffusion problem, where advection is induced by an irrotational counter-clockwise vortex. The diffusivity is $\nu = 0.01$. The irrotational counter-clockwise vortex is located at $(-1, -1)$, vortex strength $\Gamma = 3$. The Péclét number is $Pe = 300$. The adjoint solution plotted in Fig. 7(c) and 7(d) corresponds to the output in Eqn. 7.



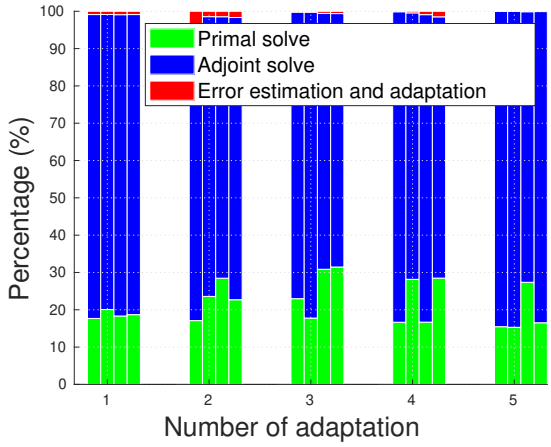
(a) Output error versus DOF convergence study



(b) Output error versus CPU time convergence study

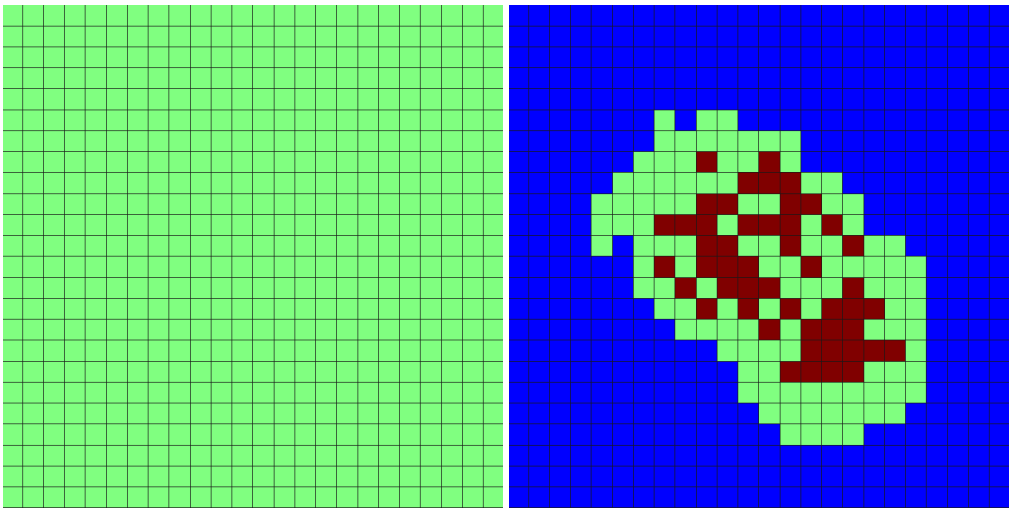


(c) CPU time breakdown



(d) Percentage CPU time breakdown

Fig. 8 Convergence studies for unsteady sub-iteration algorithm (Fig. 8(a) and Fig. 8(a)) and adaptation CPU time cost analysis (Fig. 8(c) and Fig. 8(d)). In Fig. 8(a) and Fig. 8(b), the dashed lines are error estimate corrected convergence curves. In Fig. 8(c) and Fig. 8(d), we present 5 bar-groups corresponding to 5 adaptation iterations (horizontal axis), each bar-group contains 4 bars, from left to right, they are standard, sub-iteration 1, sub-iteration 2 and sub-iteration 3.



(a) Initial condition, $p = 1$ (b) Last adaptation iteration $p \in (1, 3)$

Fig. 9 The order approximations used by solver. Initially, the approximation order for the entire computational domain is, $p = 1$, Fig. 9(a). Fig. 9(b) shows the order approximations of the last adaptation iteration, for sub-iteration 3.

As Fig. 9(b) shows, the area swept by the Gaussian vortex, near the center of the domain, is important to resolve the vortex advection and diffusion problem. Qualitatively, standard and sub-iteration algorithm produce similar adapted meshes.

B. Non-linear Case – Airfoil Gust Encounter

This case describes a NACA 0012 airfoil in a vertical gust encounter. In the beginning of the simulation, shown in Fig. 10(a), a vertical gust is placed in front of the airfoil.

To describe such physical phenomena, the initial condition for the airfoil is computed by converging a steady state, with $Re = 1000,\ M_\infty = 0.4, \alpha = 2^\circ$ and with adiabatic wall boundaries. We assume constant viscosity, and $Pr = 0.72$. After converging the described steady-state flow, we superimpose a vertical gust by perturbing the vertical velocity (perpendicular to free-stream) with an amplitude of U_∞ and width of 1 chord length, centered 5 chords upstream of the leading edge. We offset the perturbation to reach zero 1 chord length upstream of the leading edge, so that the nearby region of the airfoil stays unperturbed. Fig. 10(a) shows the relative position of the vertical gust and the airfoil, at $t = 0$. We run the simulation from $t = 0$ to $t = 15$. In addition, the chord length $c = 1$, $\rho_\infty = 1$, and $|U_\infty| = 0$. As a result, the free-stream conservative state vector becomes

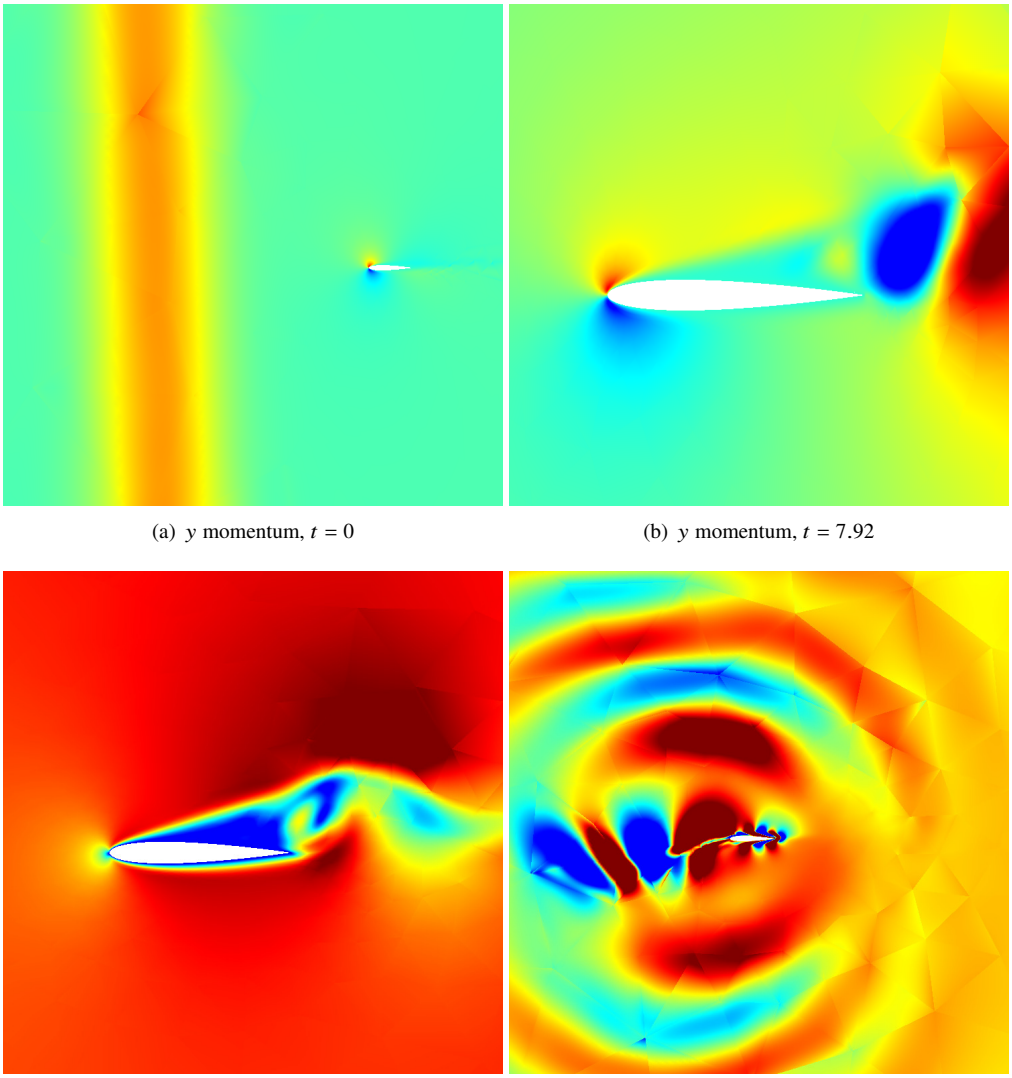
$$[\rho,\ \rho u,\ \rho v, \rho E] = \left[1,\ 1,\ 0,\ 0.5 + 1/\left[M_\infty^2 \gamma(\gamma - 1)\right]\right]. \tag{8}$$

Here, 1 time unit is the chord length divided by the free-stream speed. Fig. 10(c) shows the solution midway thorough the simulation at $t = 7.64$. Fig. 10(c) shows that the gust encounter induces flow separation and vortex shedding. Fig. 10(b) illustrates the contour of y-momentum at $t = 7.92$, in which the vortex shedding is clearly visible behind the airfoil. The computational domain is a square that extends 100 chord lengths away from the airfoil.

Our scalar output of interest is an integral in time,

$$\bar{J} = \int_0^T w\,(t)\,L\,(t)\,dt. \tag{9}$$

Here, $w\,(t) = \exp\left(-0.5\,(t - 7.5)^2\right)$ is a temporal weight function and $L\,(t)$ is the instantaneous lift force on the airfoil. We use a temporally continuous adjoint, and a time-weighted output of interest to ensure, that the adjoint is smooth in



(a) y momentum, $t = 0$

(b) y momentum, $t = 7.92$

(c) Mach number at $t = 7.64$ (0 to 0.5)

(d) y-momentum adjoint, at $t = 3.26$

Fig. 10 NACA 0012 gust encounter simulation.

time. We show a snapshot of the corresponding adjoint at $t = 3.26$ in Fig. 10(d). In Fig. 10(d), adjoint contours show our output of interest is sensitive to both acoustic waves (the rings), as well as to the upstream incoming gust (the blue circles in front of the airfoil).

We adopt static p adaptation, which means that both the mesh and the time steps are adapted once at the end of the primal solution. "Total degrees of freedom" (DOF) = "spatial degrees of freedom" × "number of time-steps". Adaptation growth factor for all adaptation strategies is 1.05. Fig. 11(a) shows the convergence of output errors versus DOF, for 4 adaptation strategies: standard, sub-iteration 1, sub-iteration 2, and sub-iteration 3, and uniform p refinement. Uniform p refinement is performed in order to provide a benchmark output error convergence curve for our convergence studies. In terms of space-time DOF measurement, all adaptation strategies (blue crosses, magenta triangles, green stars, and cyan diamonds), converge faster than the uniform p refinement (red squares). Because this airfoil gust encounter simulation is highly non-linear, the output error versus DOF convergence curve tends to be oscillatory. All adaptation curves oscillate while trending downward. The more interesting plot is perhaps Fig. 11(b). Unlike in Fig. 11(a), the uniform p refinement curve in Fig, 11(b), is not far above all adaptation convergence curves any more. For simulations that do not contain any singularities, e.g., shock waves, uniform p adaptation converges relatively fast. Let's focus on the dashed lines in Fig. 11(b) – estimate-corrected output error. All of the estimate-corrected adaptation curves are below uniform p refinement. This means that all adaptation strategies converge faster than uniform p refinement, in both CPU time and DOF. Moreover, among all of the "zig-zag" dashed lines, the dashed magenta triangles (sub-iteration 1), the dashed green stars (sub-iteration 2), and the dashed cyan diamonds (sub-iteration 3) are all below the dashed blue crosses (the standard adaptation), which means all sub-iteration strategies converge faster than the standard strategy as well.

The reason for the sub-iteration methods converging faster than the standard adaptation can be explained by Fig. 11(c) and Fig. 11(d). In Fig. 11(c), for the first adaptation iteration, all adaptation strategy have roughly the same heights. For the second adaptation iteration, the 1^{st} bar (standard) is significantly taller than all of the rest, because all sub-iteration methods (1, 2 and 3) enter the sub-iteration block during the second adaptation iteration (Fig. 2). For the third adaptation iteration, the 1^{st} bar (standard) and the 2^{nd} bar (sub-iteration 1) enter the standard block, while the 3^{rd} and 4^{th} bar enter sub-iteration block. The fourth and fifth bar-groups' individual bar height differences can be explained in like manner. Within a bar-group (4 individual stacked bars), the taller bars (higher CPU time cost) are associated with a standard block (Fig. 5(a)) and the shorter bars (lower CPU time cost) are associated with a sub-iteration block (Fig. 5(b)). What's more, comparing Fig. 11(d) and Fig. 11(c) side by side, it is clear that during a sub-iteration block, the fine-space adjoint solution time is greatly reduced, during which the solver spends most of the computational time on primal state reconstruction instead to ensure physicality (Fig. 11(d)).

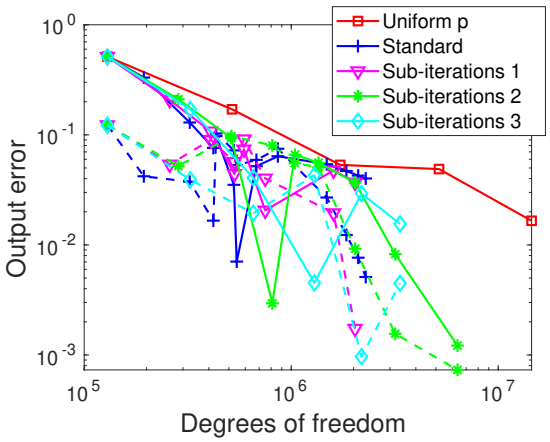
Qualitatively, all adaptation strategies – standard, sub-iteration 1, sub-iteration 2, and sub-iteration 3, produce similar adapted meshes. We choose to only plot the p-adapted domain by sub-iteration 3 at its last adaptation iteration (the 25^{th}), shown in Fig. 12(b). All adaptation strategies start out with $p = 1$, shown in Fig. 12(a), and end with an adapted domain similar to Fig. 12(b).

IV. Summary

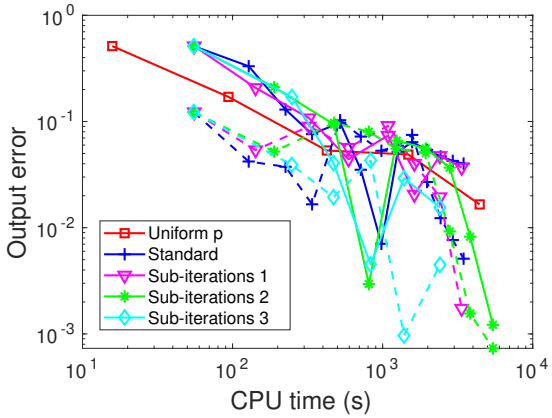
In this paper, we formulated the unsteady sub-iteration algorithm and presented results using p adaptation. The unsteady sub-iteration algorithm's computational steps are similar to steady sub-iterations [2]. Due to the added temporal dimension, the computational cost of unsteady adjoint-based adaption is considerably higher than its steady counterpart. Because of this increased CPU time in unsteady simulations, unsteady simulations can particularly benefit from the CPU time cost reduction.

As we presented in Section III, with DOF measurements, the sub-iteration algorithm's performance is similar to the standard algorithm. With CPU time cost measurement, the sub-iteration algorithm tends to outperform the standard method. We summarize our findings regarding the unsteady sub-iteration algorithm as the following,

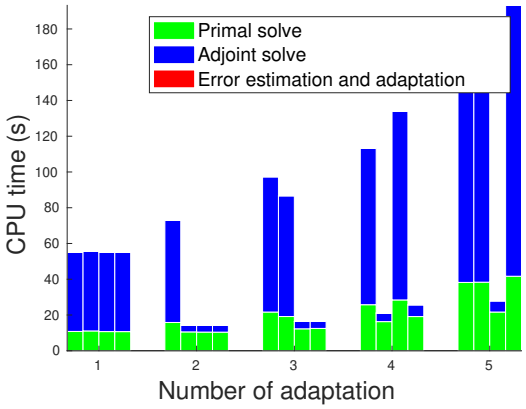
- Reconstruction instead of solving the fine-space adjoint on each adaptive iteration, is the main contributor to unsteady sub-iteration CPU time saving.
- For linear problems, reconstructing primal states suffices for the unsteady sub-iteration algorithm. However, for non-linear problems, reconstructing primal sates without post-processing can yield non-physical states.
- We recommend the unsteady sub-iteration approach of reconstructing the fine-space adjoints and reconstructing & post-processing the primal states, of the previous iterations. This approach tends to be widely applicable for both linear and non-linear cases studied.



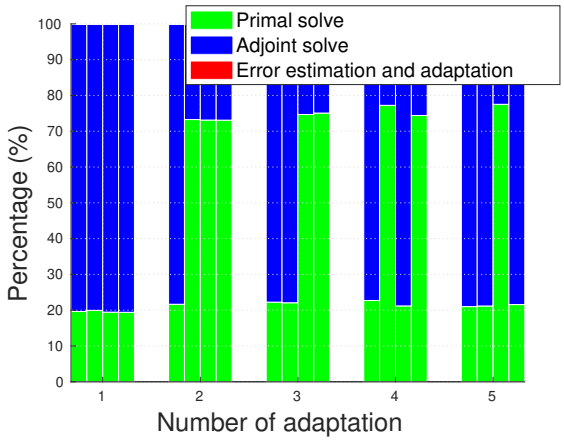
(a) Percentage CPU time breakdown



(b) Percentage CPU time breakdown

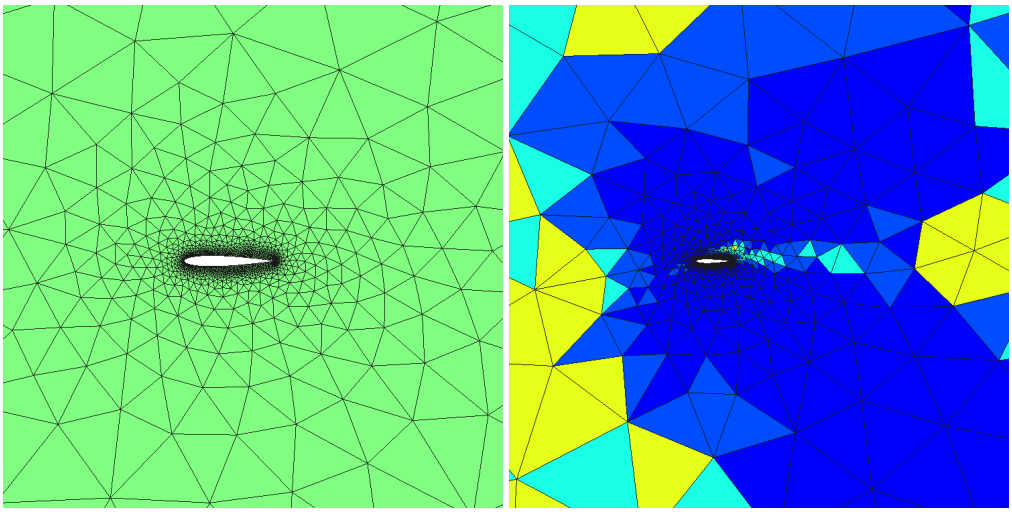


(c) Percentage CPU time breakdown



(d) Percentage CPU time breakdown

Fig. 11 Convergence studies for unsteady sub-iteration algorithm (Fig. 11(a) and Fig. 11(b)) and adaptation CPU time cost analysis (Fig. 11(c) and Fig. 11(d)). In Fig. 11(a) and Fig. 11(b), the dashed lines are error estimate corrected convergence curves. In Fig. 11(c) and Fig. 11(d), we present 5 bar-groups corresponding to 5 adaptation iterations (horizontal axis). Each bar-group contains 4 bars. From left to right, they are: standard, sub-iteration 1, sub-iteration 2 and sub-iteration 3.



(a) Initial condition, $p = 1$ (b) Last adaptation iteration $p \in (1, 6)$

Fig. 12 The order approximations used by the solver. Initially, the approximation order for the entire computational domain is, $p = 1$, Fig. 12(a). Fig. 12(b) shows the order approximations of the last adaptation iteration, for sub-iteration 3.

- It is necessary to intersperse standard blocks with sub-iteration blocks, in order to obtain accurate error estimates. Without error estimate corrections, adapted solutions may be even slower than uniform p refinement (Fig. 11(b)).
- If we view time as an extra dimension in simulations, 1D simulations include 1D steady simulations; 2D simulations include 2D steady simulation and 1D unsteady simulations; 3D simulations include 3D steady simulation and 2D unsteady simulation; lastly, 4D simulation include 3D unsteady simulations. The higher the dimension, the more sensitive the "CPU versus error" convergence curves are to adjoint CPU time expenditure (the third case in the result section of [2], the case of Section III.B, and the case of Section III.A), which makes adjoint computation time acceleration the main contributor to overall simulation time acceleration. The lower the dimension, the more sensitive the "CPU versus error" convergence curves are to primal CPU time expenditure (the first and the second case in the result section of [2]), which makes primal computation time acceleration the main contributor to overall simulation time acceleration. The overall CPU time expenditure is not sensitive to error estimation and adaptation time expenditure: i.e., for h-adaptation, the adaptation time accounts for a small percentage of the overall CPU time expenditure [2], and for p-adaptation, adaptation time accounts for less than 1% of the overall CPU time expenditure (Section III).

References

[1] Fidkowski, K. J., and Darmofal, D. L., "Output-Based Error Estimation and Mesh Adaptation: Overview and Recent Results," AIAA Paper 2009-1303, 2009.

[2] Ding, K., and Fidkowski, K. J., "Acceleration of adjoint-based adaptation through sub-iterations," *Computers and Fluids*, Vol. 202, 2020. https://doi.org/10.1016/j.compfluid.2020.104491.

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

[4] Venditti, D. A., and Darmofal, D. L., "Anisotropic grid adaptation for functional outputs: application to two-dimensional viscous flows," *Journal of Computational Physics*, Vol. 187, No. 1, 2003, pp. 22–46.

[5] Fidkowski, K. J., and Darmofal, D. L., "Review of Output-Based Error Estimation and Mesh Adaptation in Computational Fluid Dynamics," *American Institute of Aeronautics and Astronautics Journal*, Vol. 49, No. 4, 2011, pp. 673–694.

[6] Ding, K., Fidkowski, K. J., and Roe, P. L., "Adjoint-Based Error Estimation and Mesh Adaptation for the Active Flux Method," AIAA Paper 2013–2942, 2013.

[7] Ding, K., Fidkowski, K. J., and Roe, P. L., "Acceleration Techniques for Adjoint-Based Error Estimation and Mesh Adaptation," Eighth International Conference on Computational Fluid Dynamics ICCFD8-0249, 2014.

[8] Ding, K., Fidkowski, K. J., and Roe, P. L., "Continuous Adjoint Based Error Estimation and r-Refinement for the Active Flux Method," AIAA Paper 2016-0832, 2016.

[9] Ding, K., and Fidkowski, K. J., "Output Error Control Using r-Adaptation," AIAA Paper 2017-0832, 2017.

[10] Peraire, J., Peiró, J., and Morgan, K., "Adaptive Remeshing for Three-Dimensional Compressible Flow Computations," *Journal of Computational Physics*, Vol. 103, 1992, pp. 269–285.

[11] Nguyen, N., and Peraire, J., "An Interpolation Method for the Reconstruction and Recognition of Face Images," *The 2nd International Conference on Computer Vision Theory and Applications (VISAPP), Barcelona Spain*, 2007.

[12] Nguyen, N. C., Patera, A. T., and Peraire, J., "A 'Best Points' Interpolation Method for Efficient Approximation of Parmetrized Functions," *International Journal for Numerical Methods in Engineering*, Vol. DOI: 10.1002/nme.2086, 2007.

[13] Kraaijpoel, D., "Seismic Ray Fields and Ray Field Maps: Theory and Algorithms," Ph.D. thesis, Utrecht University, The Netherlands, 2003.

[14] Castro-Diaz, M. J., Hecht, F., Mohammadi, B., and Pironneau, O., "Anisotropic unstructured mesh adaptation for flow simulations," *International Journal for Numerical Methods in Fluids*, Vol. 25, 1997, pp. 475–491.

[15] Rachowicz, W., Pardo, D., and Demkowicz, L., "Fully Automatic hp-Adaptivity in Three Dimensions," Tech. Rep. 04-22, ICES, 2004.

[16] Naterer, G. F., and Rinn, D., "Towards entropy detection of anomalous mass and momentum exchange in incompressible fluid flow," *International Journal for Numerical Methods in Fluids*, Vol. 39, No. 11, 2002, pp. 1013–1036.

[17] Jameson, A., "Aerodynamic Design via Control Theory," *Journal of Scientific Computing*, Vol. 3, 1988, pp. 233–260.

[18] Beux, F., and Dervieux, A., "Exact-gradient shape optimization of a 2-D Euler flow," *Finite Elements in Analysis and Design*, Vol. 12, 1992, pp. 281–302.

[19] Anderson, W. K., and Venkatakrishnan, V., "Aerodynamic Design Optimization on Unstructured Grids with a Continuous Adjoint Formulation," Tech. Rep. Report 97-9, ICASE, 1997.

[20] Elliott, J., and Peraire, J., "Practical Three-Dimensional Aerodynamic Design and Optimization Using Unstructured Meshes," *AIAA Journal*, Vol. 35, No. 9, 1997, pp. 1479–1485.

[21] Nielsen, E., and Anderson, W., "Aerodynamic Design Optimization on Unstructured Meshes Using the Navier-Stokes Equations," *AIAA Journal*, Vol. 37, No. 11, 1999, pp. 1411–1419.

[22] Fidkowski, K. J., "Output-based space-time mesh optimization for unsteady flows using continuous-in-time adjoints," *Journal of Computational Physics*, Vol. 341(15):258–277, 2017. https://doi.org/https://doi.org/10.1016/j.jcp.2017.04.005.

[23] Ceze, M. A., and Fidkowski, K. J., "Drag Prediction Using Adaptive Discontinuous Finite Elements," AIAA Paper 2013-0051, 2013.