Acceleration of Adjoint-Based Adaptation through Sub-Iterations

Kaihua Ding[a], Krzysztof J. Fidkowski[a,*]

[a]*Department of Aerospace Engineering, University of Michigan*
Ann Arbor, MI, USA

Abstract

In this paper, we introduce an algorithm for reducing the cost of output-based error estimation and mesh adaptation through the use of a sub-iteration algorithm. A single sub-iteration is an adaptation iteration in which the most expensive solves—the primal and fine-space adjoint—are calculated only approximately. We intersperse sub-iterations with standard full-solve iterations to obtain accurate error estimates. The use of sub-iterations in between full-solves reduces the computational cost while improving solution accuracy per full-solve iteration. We demonstrate this method for steady, compressible Euler and Navier-Stokes simulations, discretized with the discontinuous Galerkin (DG) finite-element method. Because the acceleration in the output-based method is based on sub-iterations of the adaptation cycle, regardless of the discretization, the idea is not specific to DG and can be applied to general discretizations.

Keywords: Acceleration Algorithm, Adjoint-Based Methods, Error Estimation, Discontinuous Galerkin, Mesh Adaptation

1. Introduction

Solution-adaptive techniques are becoming popular in Computational Fluid Dynamics research as a means of improving solution accuracy and reducing computational cost. They are particularly important for aerospace engineering applications, where convective transport phenomena on complex three-dimensional geometries make a priori mesh design a difficult task. One of the most rigorous solution-adaptive techniques is output-based error estimation and adaptation [1]. Compared to residual-based adaptation, i.e., using solver residuals as an

*corresponding author
Email addresses: dkaihua@umich.edu (Kaihua Ding), kfid@umich.edu (Krzysztof J. Fidkowski)

adaptive indicator, output-based methods target a scalar output of interest more efficiently for convection-dominated modeling, leading to faster convergence for the outputs of interest. Compared to feature-based adaptation, e.g., using a pressure gradient as an adaptive indicator, output based methods are less ad-hoc and require less user input.

Output-based error estimates quantify the impact of numerical discretization errors on specific scalar outputs [2, 3, 4, 5, 6, 7, 8, 9]. The resulting estimates reflect the extent to which mesh resolution and distribution affect an output of interest. Furthermore, the error estimates provide information on regions of the spatial and temporal domains that are most responsible for the output error.

However, output error estimation in its most rigorous form is not computationally economic in terms of computational (CPU) time. The CPU time inefficiency is due, in large part, to the fact that the error is typically estimated relative to a finer discretization space, $\mathcal{V}_h$, and while no primal solution is usually required on $\mathcal{V}_h$, many estimates still have to employ a fine-space *adjoint* solution in addition to a fine-space residual evaluation. These fine-space calculations generally make the cost of error estimation and adaptation burdensome for practical simulations, especially in three dimensions. The fine-space adjoint calculation, when coupled with an adaptive algorithm, produces an accurate solution using relatively few degrees of freedom (DOF)— i.e. a low adaptation DOF cost. However, calculating a fine-space adjoint at every adaptation solve corresponds to a significant increase in the CPU time for the entire simulation, and in some cases, makes the adaptive process less efficient than uniform refinement. Although the error estimates are themselves useful, this high computational cost diminishes the utility of output-based adaptation.

In this paper, we introduce and formalize an acceleration algorithm for estimating the output error and for adapting the mesh using the adjoint-weighted residual: a sub-iteration algorithm. Sub-iterations refer to re-using the fine-space adjoint for more than just one adaptation iteration. We show that this strategy reduces the computational time for all cases tested while maintaining the DOF performance of the adaptation.

The remainder of this paper is organized as follows. Section 2 presents the adjoint-weighted

residual error estimate and explains why the standard output-based method is efficient in terms of DOF while less efficient in terms of computational time, by estimating errors on a finer discretization space, $\mathcal{V}_h$. Section 3 discusses the proposed strategies for accelerating the adaptation. Section 4 presents results for several test cases, using DG for steady-state flows. Section 5 summarizes the performance of sub-iteration algorithm through results from various tests, and finally, Section 6 offers concluding remarks.

2. Output-Based Method: The Adjoint-Weighted Residual and Associated Costs

The sub-iteration algorithm accelerates output-based adaptation and is independent of the solver discretization. In this paper, we adopt the DG discretization. The target partial differential equation is a system of conservation laws,

$$\partial_t \mathbf{u} + \nabla \cdot \vec{\mathbf{F}}(\mathbf{u}) \;\; = \;\; \mathbf{0}, \tag{1}$$

where $\mathbf{u} \in \mathbb{R}^s$ is the state vector, $\vec{\mathbf{F}} \in \mathbb{R}^{d \times s}$ is the total flux, and d is the spatial dimension. We will focus on steady problems, $\partial_t \mathbf{u} = \mathbf{0}$.

2.1. The Discontinuous Galerkin Discretization

In the discontinuous Galerkin (DG) finite-element method, the state $\mathbf{u}$ is spatially approximated in functional form, using linear combinations of basis functions, usually polynomials, on each element [10, 11, 12, 13, 14, 15, 16]. No continuity constraints are imposed on the approximations across elements. Denote by $T_h = \{\Omega_e\}$ the set of N_e elements in a non-overlapping tessellation of the domain $\Omega = \bigcup_e \Omega_e$. We seek an approximate solution $\mathbf{u}_h \in \boldsymbol{\mathcal{V}}_h = [\mathcal{V}_h]^s$, where

$$\mathcal{V}_h = \left\{ u \in L_2(\Omega) : u|_{\Omega_e} \in \mathcal{P}^p \; \forall \Omega_e \in T_h \right\},$$

and $\mathcal{P}^p$ denotes polynomials of order p on each element.

We obtain a weak form of Eqn. 1 by multiplying the PDE by test functions $\mathbf{v}_h \in \boldsymbol{\mathcal{V}}_h$ and by integrating by parts to couple elements via fluxes. The convective fluxes on element faces are

handled via a traditional finite-volume (approximate) Riemann solver [17], whereas diffusive fluxes are handled using a penalty formulation, BR2 [18]. The final semilinear weak form reads

$$\mathcal{R}_h(\mathbf{u}_h, \mathbf{v}_h) = 0, \quad \forall \mathbf{v}_h \in \boldsymbol{\mathcal{V}}_h, \tag{2}$$

which, by linearity of the second argument, we can decompose into contributions from each element,

$$\mathcal{R}_h(\mathbf{u}_h, \mathbf{v}_h) = \sum_{e=1}^{N_e} \mathcal{R}_h(\mathbf{u}_h, \mathbf{v}_h|_{\Omega_e}) = 0, \quad \forall \mathbf{v}_h \in \boldsymbol{\mathcal{V}}_h. \tag{3}$$

Integrating by parts, we find that the semilinear form associated with each element is

$$\begin{aligned} \mathcal{R}_h(\mathbf{u}_h, \mathbf{v}_h|_{\Omega_e}) \;=\; & \int_{\Omega_e} \mathbf{v}_h^T \partial_t \mathbf{u}_h \, d\Omega - \int_{\Omega_e} \partial_i \mathbf{v}_h^T \mathbf{F}_i \, d\Omega \\ & + \int_{\partial\Omega_e \setminus \partial\Omega} \mathbf{v}_h^{+T} \widehat{\mathbf{F}} \, ds + \int_{\partial\Omega_e \cup \partial\Omega} \mathbf{v}_h^{+T} \widehat{\mathbf{F}}^b \, ds, \end{aligned} \tag{4}$$

where $(\cdot)^T$ denotes transpose, and on the element boundary $\partial\Omega_e$, the notations $(\cdot)^+$ and $(\cdot)^b$ respectively denote quantities taken from the element interior and boundary. On a boundary face, σ^b, the flux is typically computed directly from the boundary state, $\mathbf{u}^b$, which is a function (projection) of the interior state and the boundary-condition data, $\mathbf{u}_h^b = \mathbf{u}_h^b(\mathbf{u}_h^+, \mathrm{BC})$.

After choosing a basis for $\boldsymbol{\mathcal{V}}_h$ and using this basis for both the approximation expansion and for the test functions, we obtain a discrete system of equations (i.e. residuals),

$$\mathbf{R}(\mathbf{U}) = \mathbf{0}. \tag{5}$$

Note that both $\mathbf{U}$ and $\mathbf{R}$ lie in $\mathbb{R}^N$, where N is the total number of degrees of freedom, including equation states (s). When considering different discretization spaces, we will append a subscript h(fine space) or H(coarse space) to the variables $\mathbf{R}$, $\mathbf{U}$, and N. This system of equations can be obtained for any scheme and is the starting point for output-based error estimation

and adaptation, which is discussed in Section 2.2.

2.2. Output-Based Error Estimation

Output-based error estimates rely on the concept of an adjoint-weighted residual [19, 20, 21, 22, 23, 24, 25, 26], which is based on the definition of an output adjoint. An adjoint is the sensitivity of an output to residual perturbations. On a particular mesh, residuals are typically driven to negligible size by the solver. Subsequently, when we start varying mesh resolution, we can uncover nonzero residuals. That is, when a primal solution on a particular mesh, call it a "coarse" mesh, is transferred/injected/interpolated to a "fine" mesh, i.e., one with more degrees of freedom, residuals are generally going to be nonzero on the fine mesh. An adjoint solution on the fine space can then weight these residuals to yield an estimate of the output difference between the coarse and fine mesh solutions. This calculation is attractive because it does not require a primal solution on the fine mesh. However, it does require a fine space adjoint solution and a fine-space residual evaluation, and these are not always computationally economic in terms of CPU time. In some cases, the resultant CPU cost is comparable with standard uniform refinement to achieve the required output accuracy.

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 discrete residual vectors, both 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 [19, 21, 1]

$$\underbrace{J_h\left(\mathbf{U}_h^H\right) - J_h\left(\mathbf{U}_h\right)}_{\delta J} = \mathbf{\Psi}_h^T \delta\mathbf{R}_h = -\mathbf{\Psi}_h^T \mathbf{R}_h\left(\mathbf{U}_h^H\right), \tag{6}$$

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, $\mathbf{\Psi}_h$, 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. It is a vector of the same

size as the state and residual vectors, and it satisfies the following *adjoint equation*:

$$\left(\frac{\partial \mathbf{R}_h}{\partial \mathbf{U}_h}\right)^T \mathbf{\Psi}_h + \left(\frac{\partial J_h}{\partial \mathbf{U}_h}\right)^T = \mathbf{0}. \tag{7}$$

2.3. Cost of Output-Based method

Although the fine-space primal state is not required in output-based error estimation, the computational and storage costs associated with Eqn. 6 are not trivial, as indicated below:

$$\delta J \approx \underbrace{-\mathbf{\Psi}_h^T}_{\text{fine space adjoint}} \underbrace{\mathbf{R}_h}_{\text{fine space residual operator}} \Big(\underbrace{\mathbf{U}_h^H}_{\text{injected state}} \Big). \tag{8}$$

First, we need to inject the state into the fine space [27, 28, 29, 30, 31, 32, 33, 34, 35, 36]. A common way to construct a finer space is uniform mesh refinement, which increases the spatial DOF four-fold in two dimensions and eight-fold in three dimensions, for steady problems. Second, along with the increase in DOF, computational overhead occurs 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. Third, Eqn. 8 requires the fine-space adjoint solution, and this involves either a reconstruction or a system solve on the fine space [37, 38, 39, 40, 41, 42].

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.

3. Addressing the Cost of Output-Based Adaptation: A Sub-Iteration Algorithm

As discussed in Section 2, the adjoint computation in output-based methods may be expensive, due to the need for fine-space calculations. In contrast, residual-based methods and feature-based methods are often readily computable from the primal state.

Figure 1 illustrates a standard adaptive solution scheme based on the adjoint-weighted residual. Each adaptive iteration requires the calculation of an error estimate, a critical part of which is the fine-space adjoint solve: after solving the primal problem exactly (practically, to some low residual tolerance) on the coarse space (H), we solve the coarse-space adjoint problem, inject the primal to a fine space (h), solve the fine-space adjoint corresponding to the injected primal solution, calculate the fine-space residual of the injected primal, and weight this residual by the fine-space adjoint to obtain the error estimate. A localized form of the error estimate then

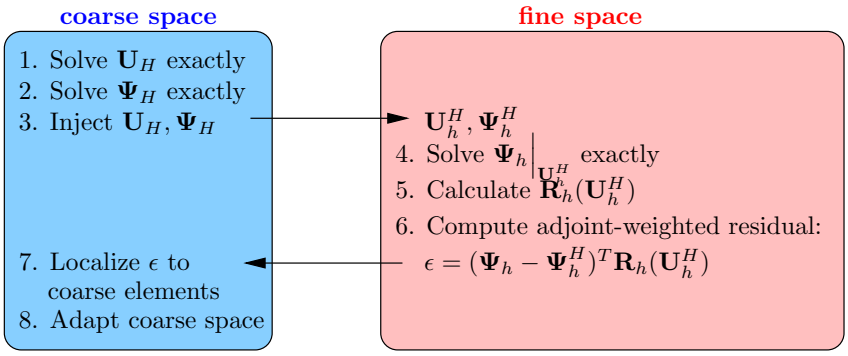


Figure 1: Schematic of a "standard" error estimation and adaptation iteration in which the fine space adjoint is solved exactly at every iteration.

drives adaptation. In a fixed-fraction setting, only the elements with the highest contribution to the error are targeted for refinement. The above-process repeats until the desired accuracy is reached.

As in the previously depicted standard adaptive scheme, we often solve the fine-space adjoint exactly in order to obtain an accurate error estimate, which we use to correct the solution and to improve the order of accuracy of the output prediction. To reduce the computational burden of this exact solve, we can approximate the fine-space adjoint, either through iterative smoothing or reconstruction [43, 44, 45, 46, 47, 48]. However, the accuracy of error estimates often suffers when using such approximations. As discussed further in Section 4, the majority of CPU time expenditure of output-based adaptation arises from the fine-space adjoint solution and error estimation.

In fixed-fraction output-based adaptation, a group of cells with the highest absolute element-wise error estimates is targeted for adaptation. Therefore, the accuracy of individual element-

wise error estimates may not affect the adaptation process, as long as the same group of cells with the highest absolute element-wise error estimates is designated as the adaptive cells. An approximate adjoint solve then possesses merit for adaptation purposes because the ranks of cell-wise error estimates determine the targeted adaptation cells, assuming that the approximate solve can yield the correct relative ranks among cells. This motivates the sub-iteration algorithm, which is based on the following two fundamental steps:

1. Calculate adaptive sub-iterations where the primal problem is not solved on every adaptive iteration so as to minimize the cost of multiple nonlinear primal solves.
2. Reuse the fine-space adjoint between adaptive iterations to avoid the cost of solving a large fine-space adjoint problem at each iteration.

Figure 2 illustrates this scheme, which consists of two types of iterations: a standard error estimation and adaptation iteration involving an exact fine-space adjoint solve, followed by one or more adaptive "sub-iterations" that further refine the mesh at a lower computational cost. Note that in these adaptive sub-iterations, neither the coarse-space primal nor the fine-space adjoint are solved exactly. However, the coarse-space adjoint is solved exactly in order to accurately quantify the coarse space error estimate and to remove the portion of error in the estimate caused by an incomplete coarse-space primal solve. This prevents the sub-iterations adaptive indicator from becoming polluted by coarse-space primal residuals that are nonzero solely because of the inexact primal solves on the sub-iterations. Instead, the sub-iteration indicator still targets errors relative to the fine space.

The effectiveness of the sub-iterations depends on, among other things, the fine-space adjoint retaining accuracy as it is transferred from one fine space (e.g. h_0) to another (e.g. h_1). In our work, we use hanging-node mesh refinement, so that this transfer is injective and results in no information loss. Of course, not losing information is itself not sufficient, and that is why we smooth the adjoint on the fine space to which it is transferred. Smoothing of the adjoint and primal solutions incorporates new characteristics of the fine space into these solutions.

Algorithm 1 presents the sub-iteration algorithm in more detail. The sub-iteration algorithm does not require changes in the primal or adjoint solvers, and it operates both on the

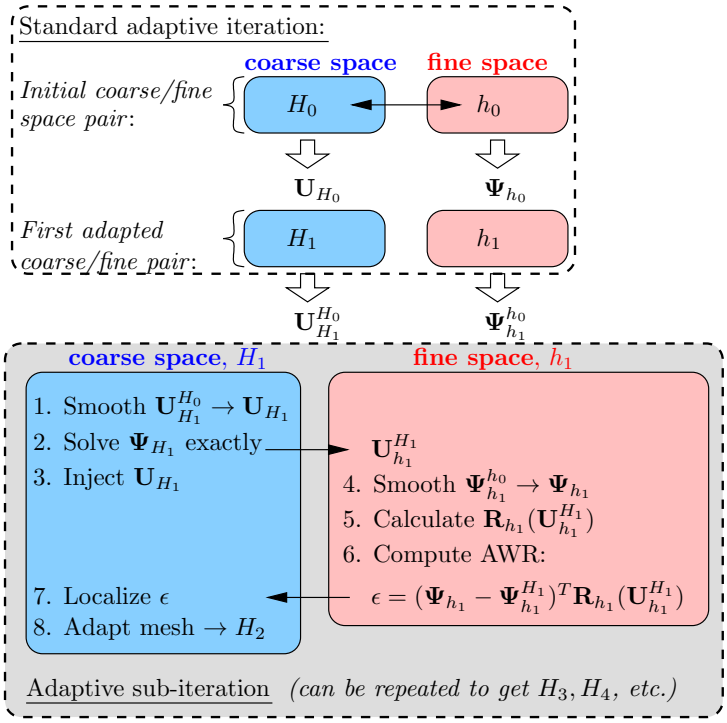


Figure 2: Schematic of the proposed error estimation and adaptation iteration in which approximate sub-iterations piggy-back on a standard adaptive iteration. The fine-space adjoint solve is reused in the sub-iterations, where it is only smoothed via an inexpensive iterative solver, 2 element block Jacobi iterations, thereby saving computational time compared to the standard approach, in which the fine-space adjoint is re-solved at every adaptive iteration.

primal solver and the adjoint solver as black-boxes / closed modules.

Algorithm 1: 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_i denotes the discretization space resulting from the i^{th} adaptation, H_{i-1} denotes the discretization space resulted from the $(i-1)^{\text{th}}$ adaptation, h_i denotes the corresponding fine space of H_i, and h_{i-1} denotes the corresponding fine space of H_{i-1}.

initialization, $\text{Cycle}_{\text{sub-iterations}}$ = `False`;

while $e_{estimate} > e_{desired}$ **do**

  if $Cycle_{sub\text{-}iterations}$ **then**

    smooth $\mathbf{U}_{H_i}^{H_{i-1}} \longrightarrow \mathbf{U}_{H_i}$;

    solve $\mathbf{\Psi}_{Hi}$ exactly;

    inject $\mathbf{U}_{Hi} \longrightarrow \mathbf{U}_{h_i}^{H_i}$;

    smooth $\mathbf{\Psi}_{h_i}^{h_{i-1}} \longrightarrow \mathbf{\Psi}_{h_i}$ calculate $\mathbf{R}_{h_i}\left(\mathbf{U}_{h_i}^{H_i}\right)$;

    compute AWR, $\epsilon = \left(\mathbf{\Psi}_{h_i} - \mathbf{\Psi}_{h_i}^{H_i}\right)^T \mathbf{R}_{h_i}\left(\mathbf{U}_{h_i}^{H_i}\right)$;

  else

    solve $\mathbf{U}_{H_i}$ exactly;

    solve $\mathbf{\Psi}_{H_i}$ exactly;

    inject $(\mathbf{U}_{H_i}, \mathbf{\Psi}_{H_i}) \longrightarrow \left(\mathbf{\Psi}_{h_i}^{H_i}, \mathbf{U}_{h_i}^{H_i}\right)$;

    solve $\mathbf{\Psi}|_{\mathbf{U}_{h_i}^{H_i}}$ exactly;

    calculate $\mathbf{R}_{h_i}\left(\mathbf{U}_{h_i}^{H_i}\right)$;

    compute AWR, $\epsilon = \left(\mathbf{\Psi}_{h_i} - \mathbf{\Psi}_{h_i}^{H_i}\right)^T \mathbf{R}_{h_i}\left(\mathbf{U}_{h_i}^{H_i}\right)$;

    $e_{\text{estimate}} = \epsilon$;

  end

  localize ϵ to coarse elements;

  adapt coarse space;

  if $(\mod [Cycle_i, \beta] = 0)$ **then**
    $\text{Cycle}_{\text{sub-iterations}}$ = `True`

  else
    $\text{Cycle}_{\text{sub-iterations}}$ = `False`

  end

end

post-processing;

4. Results

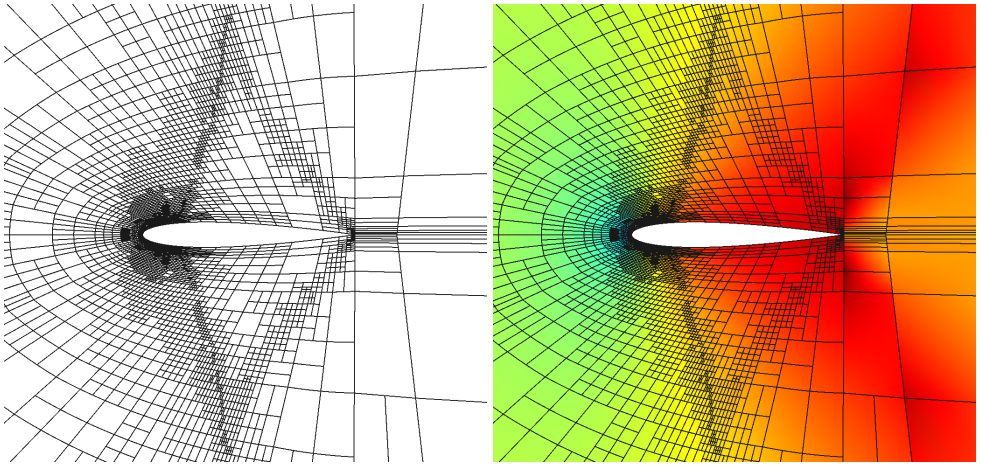
In this section, we present results of our error estimation and adaptation acceleration algorithm. We demonstrate the adaptive sub-iteration algorithm for the discontinuous Galerkin discretization of the steady-state Euler and Navier-Stokes simulations. We choose a p enriched fine spaces when constructing adjoint based error estimate formulation, which is more computationally economic than using a h enriched space.

4.1. Transonic airfoil with a fishtail shock

We first consider a NACA 0012 airfoil in inviscid $M = 0.95$ flow at $\alpha = 0$. The flow is transonic and we use element-wise artificial viscosity, discretized using BR2 [18], to stabilize the solution. We consider drag coefficient prediction using an approximation order of $p = 1$ and an adaptive fixed fraction of $f = 0.1$ for adaptive (sub-)iterations. The initial mesh consists of 234 quadrilaterals, curved with a quartic geometry representation.

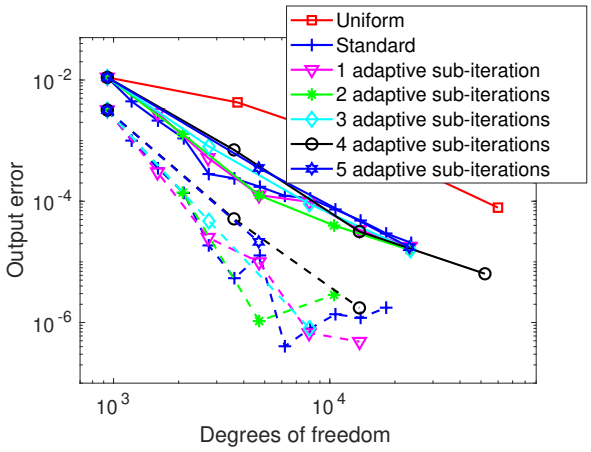
Figure 3 shows a comparison of several adaptive techniques for this case. These include simple uniform refinement, a standard method without sub-iterations, and five methods with sub-iterations. As shown in Figure 3, the adaptation targets a "lambda"-shaped structure in the transonic flow region and leaves the trailing edge fishtail shock virtually untouched. Both the standard adaptive approach and ones that use adaptive sub-iterations yield nearly the same adapted meshes. They also yield nearly the same output convergence with respect to DOF (the solid color lines in Figure 3(c)), meaning our approximations in the sub-iterations do not have a strong effect on the adaptive indicator that dictates the elements chosen for refinement. Furthermore, the outputs corrected by the error estimates (dashed line) are also similar to the output-based approaches, which is not overly surprising because during adaptive sub-iterations we only report the error estimates when we carry out an exact adjoint solve.

The more interesting plot, however, is the one in Figure 3(d), which shows the convergence of the drag output against computational time. Both the uncorrected and corrected outputs now converge faster for the runs with adaptive sub-iterations. It follows that convergence is faster because each sub-iteration is cheaper than a regular adaptive iteration due to smoothing of the coarse primal and the fine adjoint. The bottoming-out of the corrected output in this case is

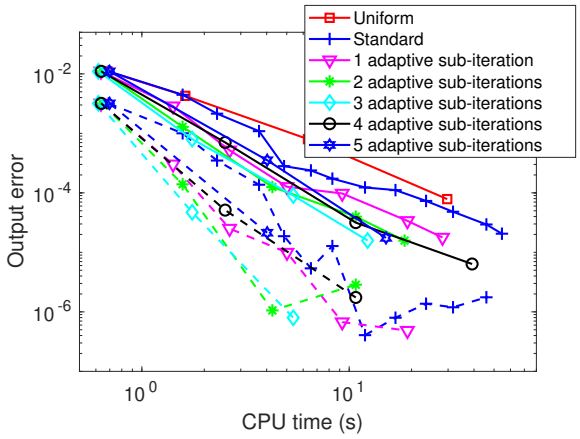


(a) Final drag-adapted mesh

(b) Mach contours (0 to 1.5)



(c) Drag convergence with DOF



(d) Drag convergence with CPU time

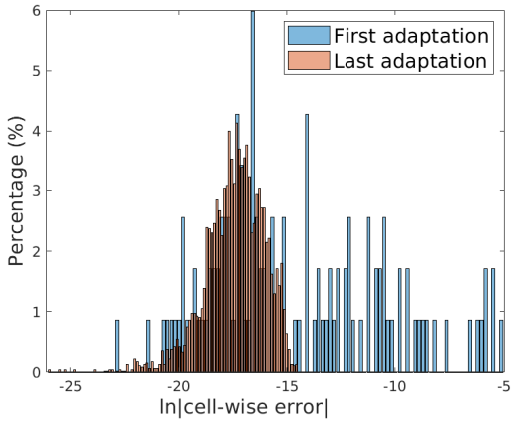
Figure 3: NACA 0012, $M = 0.95$, $\alpha = 0°$: effect of sub-iterations on drag convergence. Output of interest is drag coefficient. In all 5 cases employing sub-iterations, the fine-space adjoint is reused on the sub-iterations with only one element block-Jacobi smoothing iteration as the extra solve. The current-space primal is also 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 corrects the convergence curve, and the result is shown by the dashed line.

likely due to a relatively loose residual convergence tolerance of 10^{-8} used in the calculations. The fastest convergence curve seems to be the dashed green star line, 2 sub-iterations adaptive refinement. Noted, from 3 sub-iterations to 5 sub-iterations, corrected outputs converge progressively slower as the number of sub-iterations increase as shown in Figure 3(d).

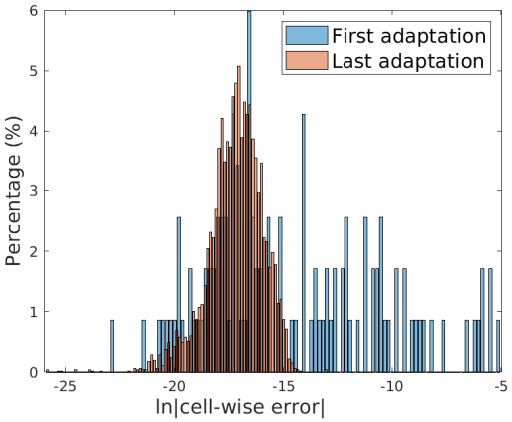
Figure 4 shows histograms of the elemental error indicator (obtained from localizing the error estimate) for the first and last adaptive iterations of all adaptive methods. We see that, after adaptations, all of the methods yield an error histogram shifted to the left – meaning that elements with high errors were targeted for refinement. Moreover, the histograms are similar among the methods, which indicates that they are performing comparably. Figure 5(a) shows, for the standard adjoint-weighted residual method, how the error equi-distributes over the elements with adaptive refinement. While, initially fewer than 20% of the elements accounted for 99% of the error, by the final adaptive iteration, 99% of the error is distributed among a much larger portion of the elements, totalling 85%. Figure 5(b) further illustrates this point: both the mean and standard deviation of the error indicator drop with each adaptive iteration.

Figure 6 shows the CPU time breakdown for the different adaptation strategies. Figure 6(a) shows the CPU time percentage breakdown and Figure 6(b) shows the actual CPU time breakdown[1]. Figure 6(b) reveals the benefits of sub-iterations. In both Figure 6(a) and Figure 6(b), each number of adaptation (depicted as a grouped bar) comprises six adaptation strategies. We take the first bar in each group of six as the benchmark, since this represents the standard adaptive mesh refinement. Looking at the second bar, we see that the total height of this bar is similar to the first bar for every even iteration, and noticeably lower for every odd iteration. This is due to the 1 sub-iteration, which occurs at every even total iteration number: on these iterations, the primal and fine-space adjoints are only smoothed (green and red lines are much shorter). Note that the coarse-space adjoint is still solved exactly, so that the blue lines are always of similar size. Looking at the third bar in each group, the case of 2 sub-iterations, we see a similar trend but now with the bar height similar to the standard one only every three iterations (since the other two are the quick sub-iterations). Switching attention to the fourth bar, 3 sub-iterations,

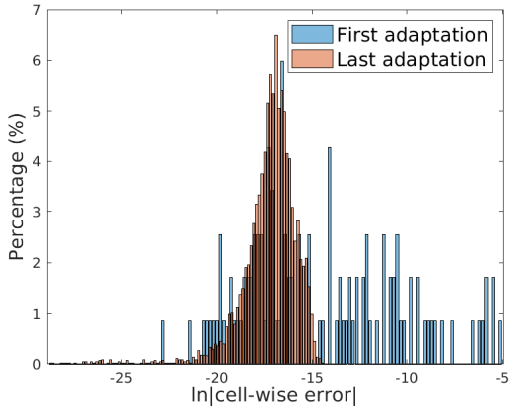
[1]Timings were performed on a workstation with dual socket Intel Xeon 2.1 GHz 8-core processors and 96GB total RAM.



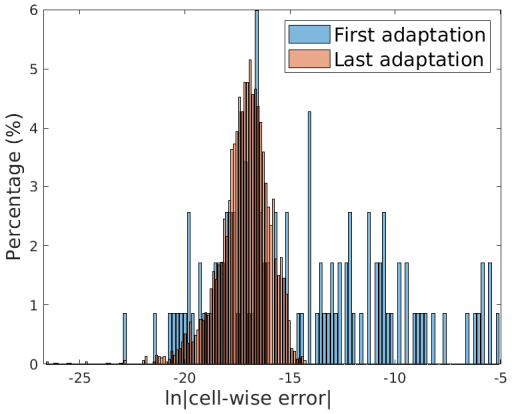
(a) Standard AWR adaptation



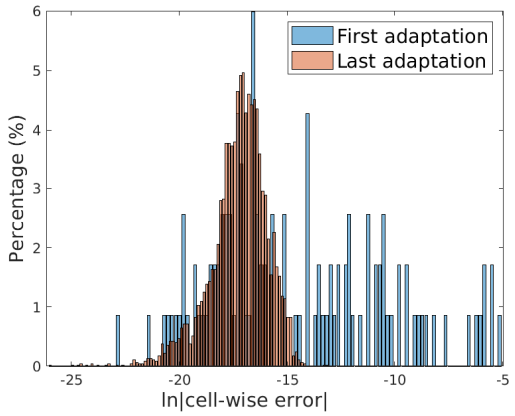
(b) One sub-iteration



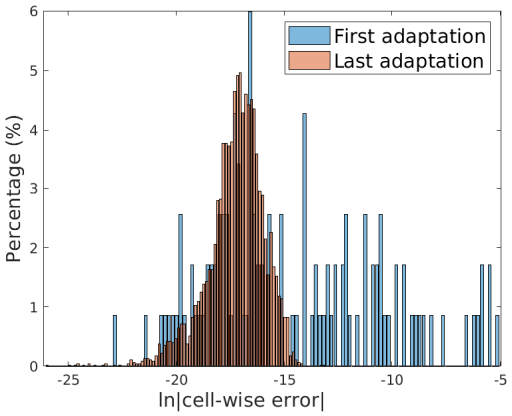
(c) Two sub-iterations



(d) Three sub-iterations

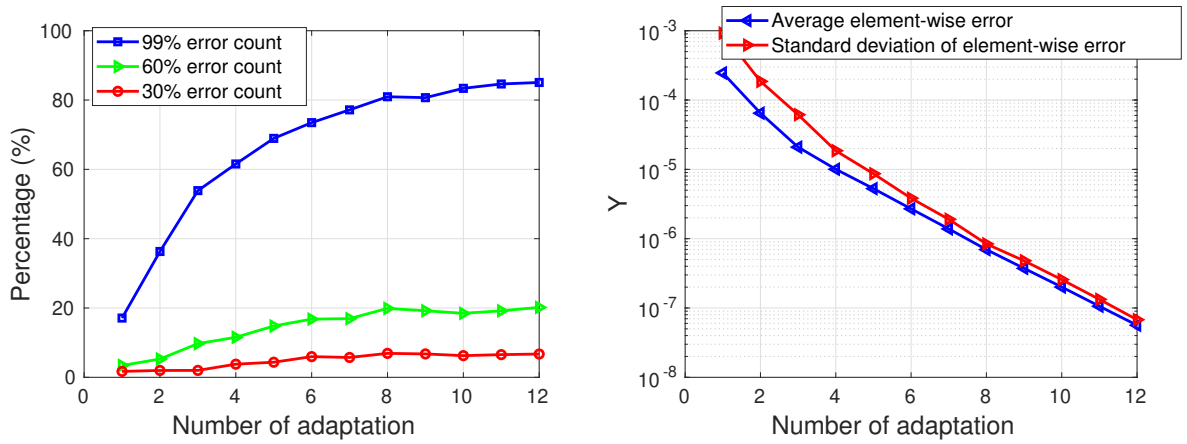


(e) Four sub-iterations



(f) Five sub-iterations

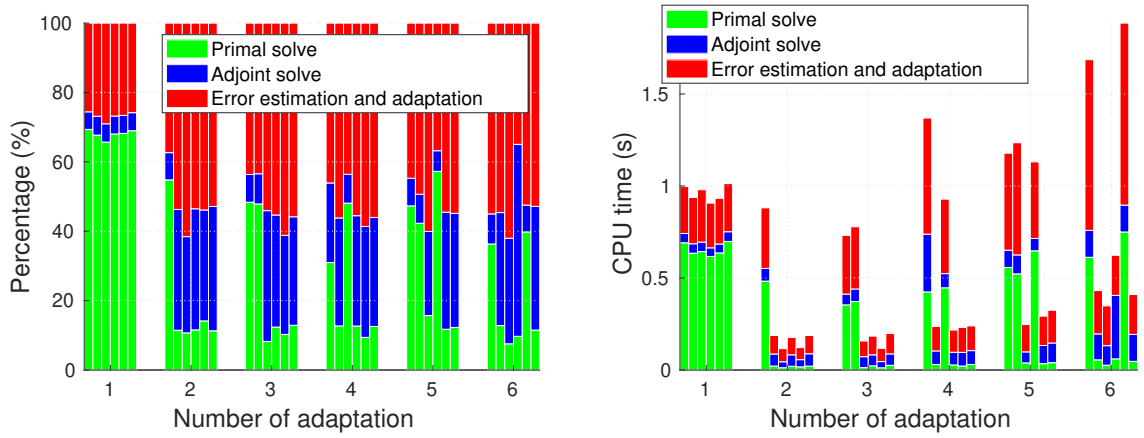
Figure 4: NACA 0012, $M = 0.95$, $\alpha = 0°$: comparison of error indicator distributions.



(a) % of elements holding the top 30/60/99% of the error

(b) The mean and standard deviation of the error indicator

Figure 5: NACA 0012, $M = 0.95$, $\alpha = 0°$: sample statics of the standard adjoint-weighted residual adaptive refinement, element wise error indicator mean, standard deviation and error count.



(a) Percentage CPU time breakdown

(b) Raw CPU time breakdown

Figure 6: NACA 0012, $M = 0.95$, $\alpha = 0°$: CPU time breakdown results. For the first 6 adaptive iterations, we show 2 group-bar plots. One group-bar plot illustrates the percentage CPU time expenditure and the other one illustrates absolute CPU time expenditure. In each group-bar plot, a group column consists of 6 bars: standard adaptation, and adaptation with 1-5 sub-iterations. Each individual bar is divided vertically into three parts, which indicate the CPU time contribution of the primal solve, performed in H space (green), the adjoint solve, performed in H space (blue), and the error estimation and adaptation, performed in h space (red). Note that the latter includes any fine-space solves.

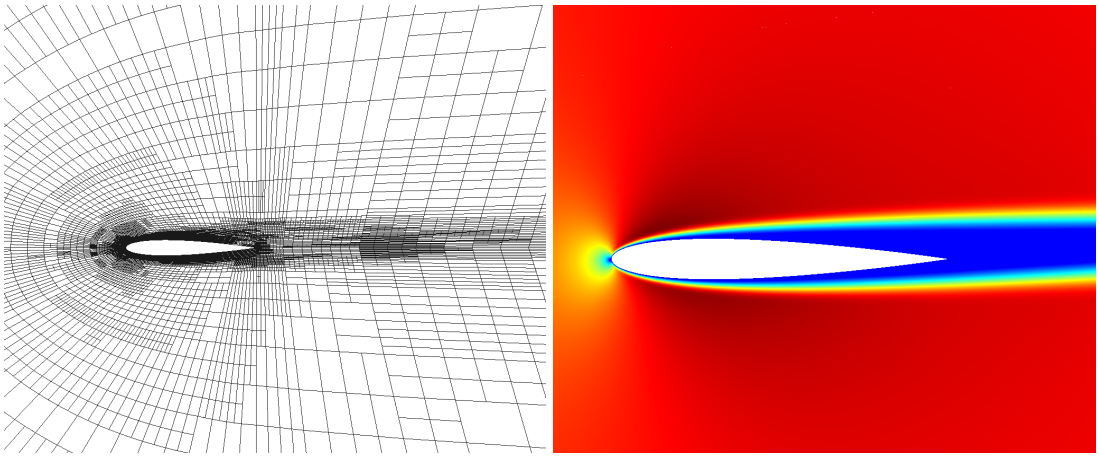
every fourth iteration is around the similar height (the other 3 are quick sub-iterations) with the standard adaptation. For the the fifth bar, 4 sub-iterations, every fifth iteration is around the similar height with the standard adaptation (the other 4 are quick sub-iterations). Lastly, when we observe the sixth bar of each group, 5 sub-iteration, every sixth iteration is of similar height as the standard adaptation (the other 5 are quick sub-iterations). Figure 6(a) confirms this trend, showing that during sub-iterations, the adjoint solve time, which is similar for all methods, eventually consumes the largest percentage of the CPU time. Since we saw in Figure 3 that the standard and sub-iteration methods perform similarly in terms of DOF, sub-iteration method have a CPU time advantage for a given level of accuracy.

The transonic fishtail case demonstrates the benefit of sub-iterations in reducing computational time while achieving a similar level of accuracy compared to standard output-based methods with given DOF. To further test the capability of sub-iterations, we next present test cases for a subsonic airfoil in viscous flow and a three-dimensional wing in invisid flow.

4.2. An airfoil in viscous subsonic flow

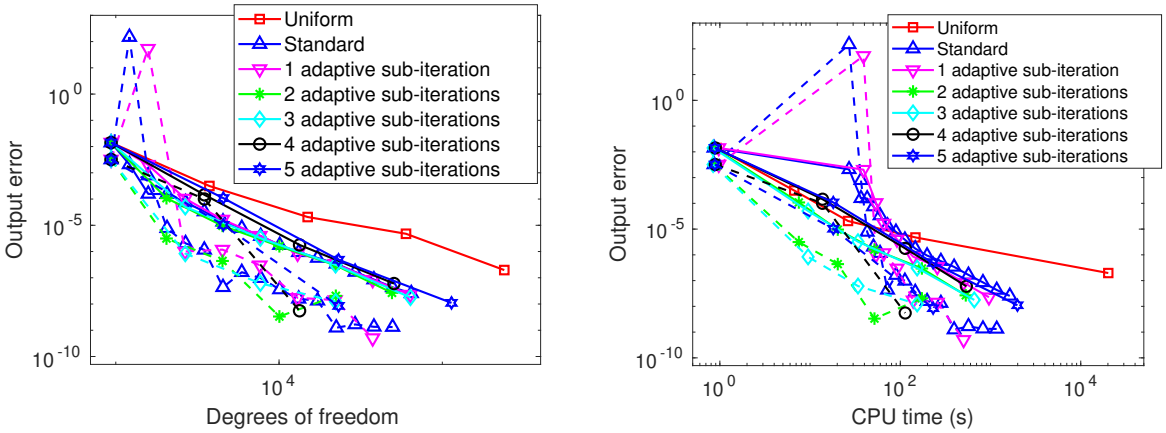
The second test case is a NACA 0012 airfoil at a free-stream Mach number of 0.5, angle of attack of 2°, Prandtl number of 0.71, and Reynolds number of 5000. Our output of interest for this case is drag coefficient. As in the previous case, the initial mesh consists of 234 quadrilaterals, curved with a quartic geometry representation. The fixed fraction for adaptation is also the same, $f = 0.1$, and the approximation order is $p = 2$.

Figure 7 shows a comparison of the various adaptive strategies for this case. Figure 7(c) shows that the methods with sub-iterations exhibit a similar output error convergence behavior with DOF compared to standard adaptation. However, as shown in Figure 7(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 increase, from 3 to 5, the error-estimate-corrected curves converge progressively slower. This progressively slower convergence behavior as sub-iteration frequency increases is consistent with our observation in Section 4.1, Figure 3(d). As the number of sub-iterations rises, the accuracy of adaptive indicators gradually deteriorates and sub-iteration based adaptations becomes less effective. Nevertheless,



(a) Final drag-adapted mesh

(b) Mach contours (0 to 0.58)



(c) Drag convergence with DOF

(d) Drag convergence with CPU time

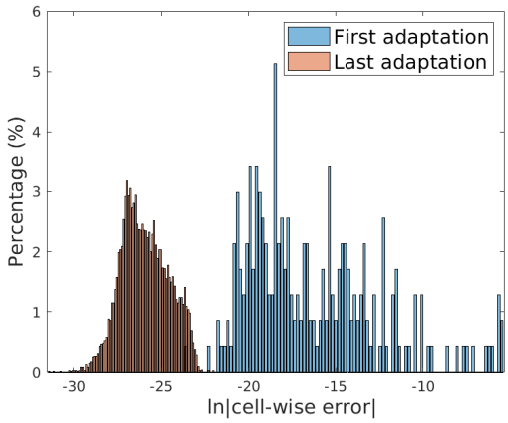
Figure 7: NACA 0012, $M = 0.5$, $\alpha = 2°$ and Re $= 5000$: effect of sub-iterations on drag convergence. In all 5 cases employing sub-iterations, the fine-space adjoint is reused on the sub-iterations with only one element block-Jacobi smoothing iteration as the extra solve. The current space primal is also 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.

in Figure 7(d), the higher frequency sub-iteration methods still outperform standard adaptation.

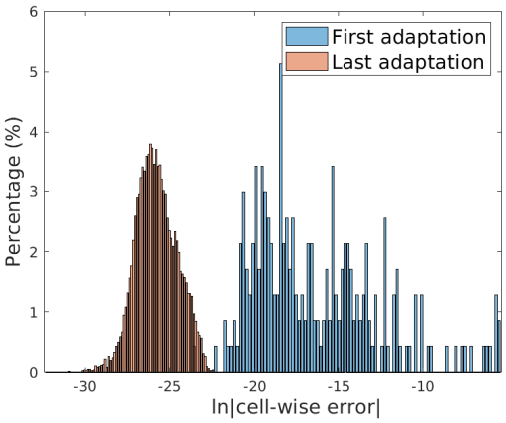
Figure 8 shows histograms of the error indicator distribution for the different adaptation strategies. Again, we see a similar trend for all six methods: the error distribution tightens and shifts to the left from the first to the last adaptive iteration. Figure 9(a) shows, for the standard adjoint-weighted residual method, another look at how the error equidistributes over the elements with adaptive refinement. While, initially, only about 8% of the elements account for 99% of the error, by the final adaptive iteration, 99% of the error is distributed among a much larger portion of the elements, totalling 80%. The same error equidistribution trend is observed for the 30% error count and 60% error count curves.

Figure 9(b) shows the mean and standard deviation of the localized error for the standard adaptation method. The spikes at the 2nd adaptation for both the mean and standard deviation curves, is likely due to the coarser mesh in the early stage of the simulation, on which neither primal solution nor error indicator are particularly well resolved. After the 2nd adaptation, both curves drop monotonically with each adaptation iteration. After the first couple of adaptation iterations, the methods employing sub-iterations show a similar trend.

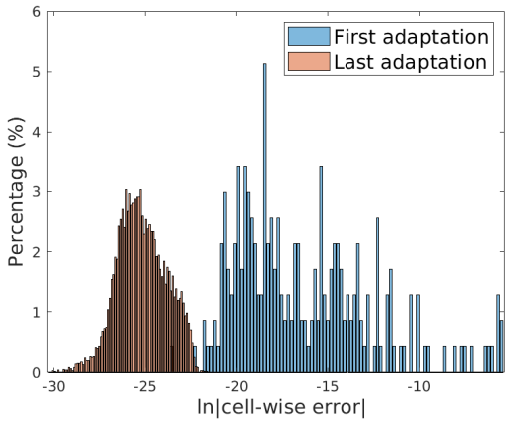
Figure 10 shows the CPU-time breakdown comparison among standard and sub-iterative adaptation. The first 3 iterations of the CPU-time breakdown data contain noise – this is the spike of absolute CPU time for the first bar (standard adaptation) in the 2nd iteration and the spike of the second bar in the third iteration (1 sub-iteration). These two CPU time spikes corresponds exactly to the 2 spike points in Figure 7(c) and 7(d). These zig-zag spikes likely indicate 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. For the remaining sub-iteration methods, the CPU time breakdown results are similar to the fishtail case in the previous section. 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 10(b)) while the percentage of the time taken by the adjoint solve



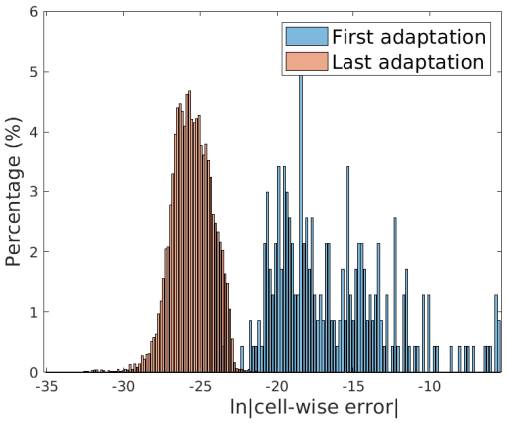
(a) Standard AWR adaptation



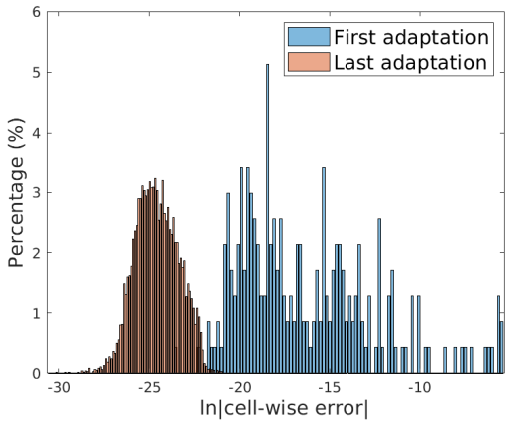
(b) One sub-iteration



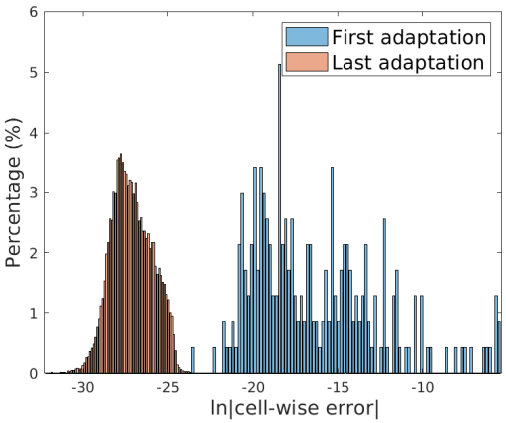
(c) Two sub-iterations



(d) Three sub-iterations

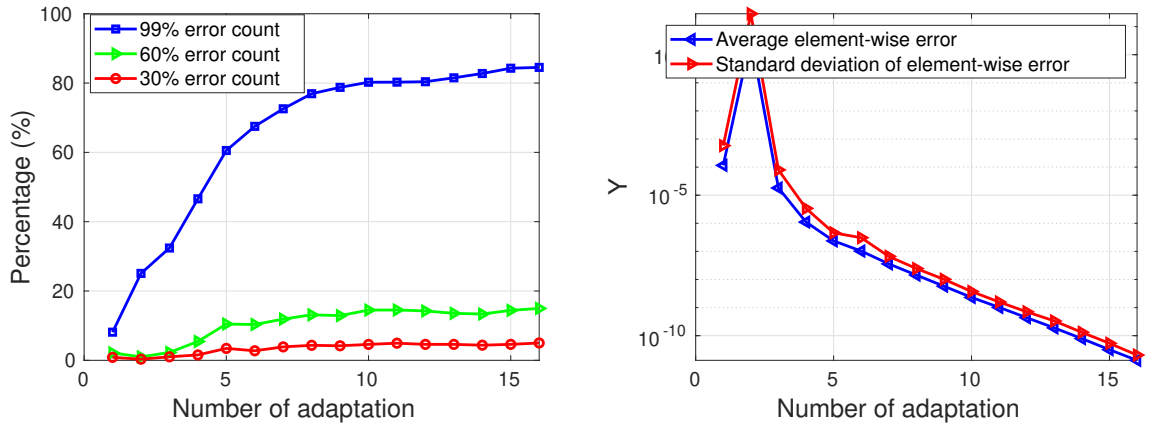


(e) Four sub-iterations



(f) Five sub-iterations

Figure 8: NACA 0012, $M = 0.5$, $\alpha = 2^\circ$ and Re $= 5000$: comparison of error indicator distributions.

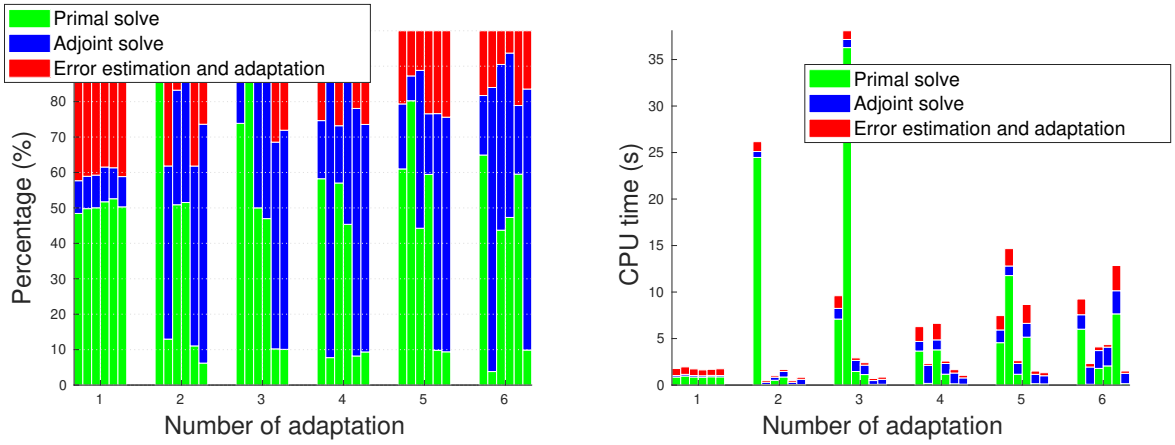


(a) % of elements holding the top 30/60/99% of the error

(b) The mean and standard deviation of the error indicator

Figure 9: NACA 0012, $M = 0.5$, $\alpha = 2°$ and Re $= 5000$: sample statics of the standard adjoint-weighted residual adaptive refinement, element wise error indicator mean, standard deviation and error count.

increases (Figure 10(a)).



(a) Percentage CPU time breakdown

(b) Raw CPU time breakdown

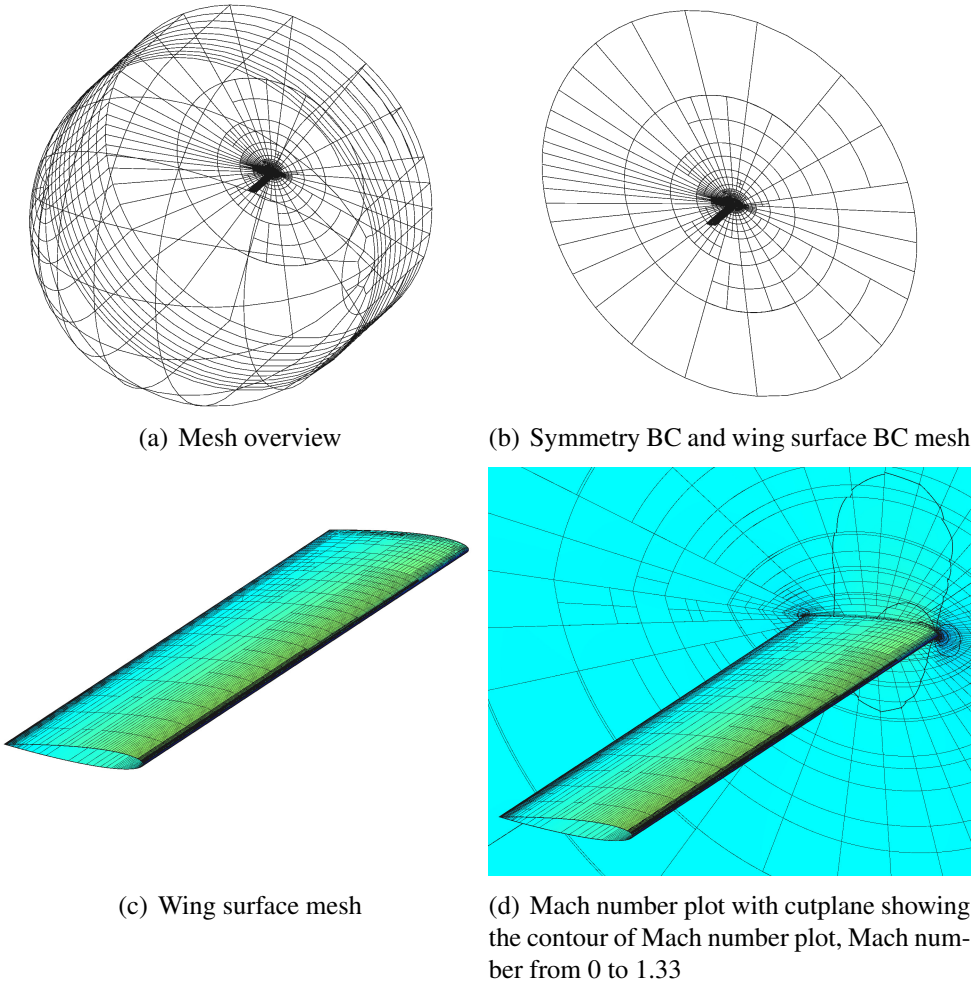
Figure 10: NACA 0012, $M = 0.5$, $\alpha = 2°$ and Re $= 5000$: CPU time breakdown results. For the first 6 adaptive iterations, we show 2 group-bar plots. One group-bar plot illustrates percentage CPU time expenditure and the other one illustrates absolute CPU time expenditure. In each group-bar plot, a group 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.

4.3. A three-dimensional wing

In this section, we demonstrate the performance of sub-iterations for adaptive simulations of a three-dimensional wing. This wing is untapered, untwisted, of aspect ratio 10, and with a

NACA 0012 airfoil cross-section, rounded via a 180° revolution at the wing tip. The wing is flying in inviscid flow at $M = 0.4$ and $\alpha = 3^\circ$. Artificial viscosity shock capturing is used in this case to enable convergence in the presence of the singular trailing vortex cores. The initial mesh for this case contains 4608 hexahedral elements curved to cubic geometry representation. Drag is again the output of interest, the approximation order is $p = 1$, and the fixed fraction for adaptation is $f = 0.04$.

Figure 11 shows the final mesh obtained from adaptation using the standard adjoint-weighted residual. We see that the leading edge, trailing edge, and parts of the wake are targeted for refinement. Figure 12 shows the output error convergence for the various methods versus DOF



(a) Mesh overview

(b) Symmetry BC and wing surface BC mesh

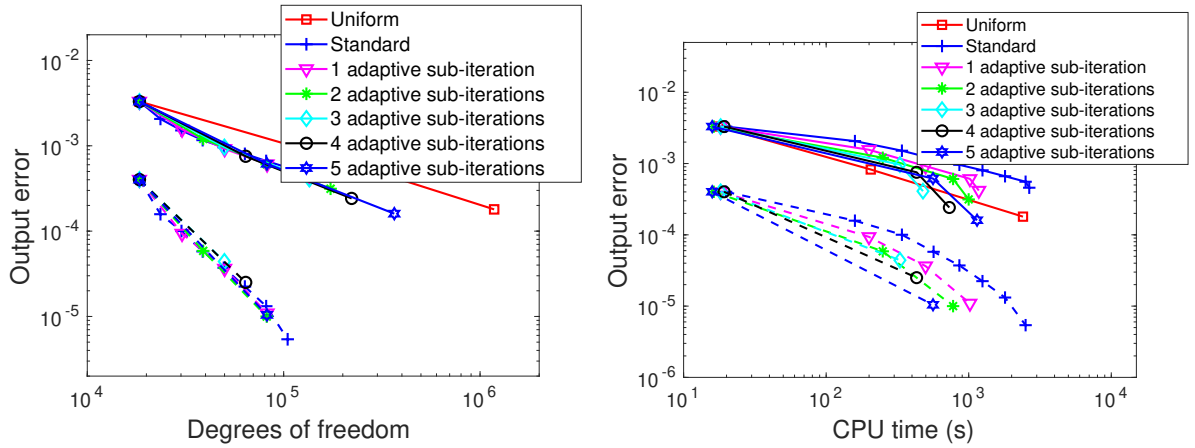
(c) Wing surface mesh

(d) Mach number plot with cutplane showing the contour of Mach number plot, Mach number from 0 to 1.33

Figure 11: NACA 0012 wing, $M = 0.4$, $\alpha = 3^\circ$: adapted mesh and surface Mach contours.

and CPU time. As expected, the standard and sub-iteration methods have similar performance with DOF, and the methods with sub-iterations perform better with CPU time. For the latter

adaptations in 12(b), the benefit of sub-iterations is at times as much as an order of magnitude error reduction for a given computational time. In Figure 12(b), among all sub-iterations methods, it appears that the greater the number of sub-iterations, the faster the error estimate corrected output converges. The 4 sub-iteration and 5 sub-iteration runs only contain two data points, as the greater the number of sub-iterations, the longer the error estimate reporting intervals. As the number of sub-iterations increases, we expect deterioration of error estimates and adaptive indicators, which would lead to less effective adaptation. For this reason, the number of sub-iterations should not be too large – 2 or 3 appears to be a reasonable general choice. In the present case, as shown in Figure 12(b), all sub-iteration methods converge faster than standard adaptation in CPU time measurement.



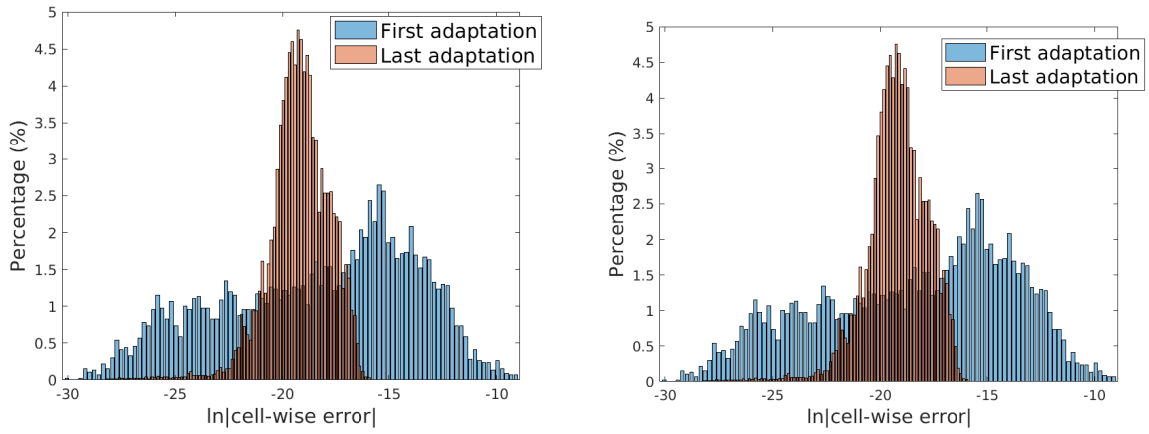
(a) Convergence with degrees of freedom

(b) Convergence with CPU time

Figure 12: NACA 0012 wing, $M = 0.4$, $\alpha = 3°$: effect of sub-iterations on drag convergence. Output of interest is drag coefficient. 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 also 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 curve, the dashed lines, are the final post-processed convergence curves.

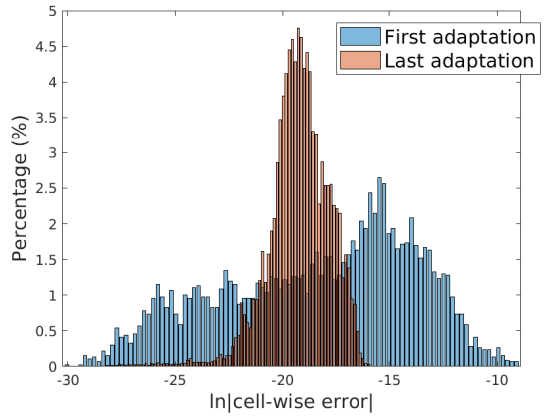
Figure 13 shows the error histograms for all adaptive methods. These error histogram distributions again appear similar among all methods, each showing a decreasing-error trend from the first to the last adaptation iteration. Figure 14(a) shows that the number of elements responsible for 99% of the error rises from 40% to just over 80% in the course of adaptation. The number of elements responsible for 30% and 60% of the error also increases but then

stagnates, likely due to large error contributions from elements in singular areas of the flow, as observed in the previous section.

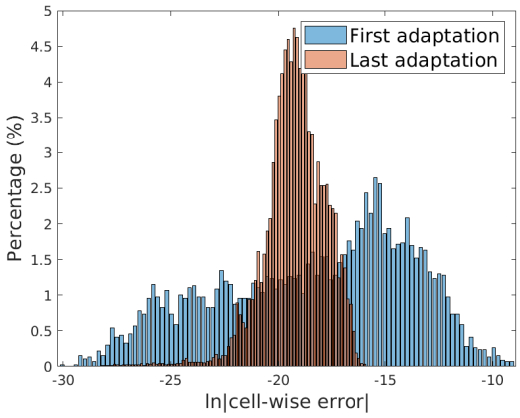


(a) Standard AWR adaptation

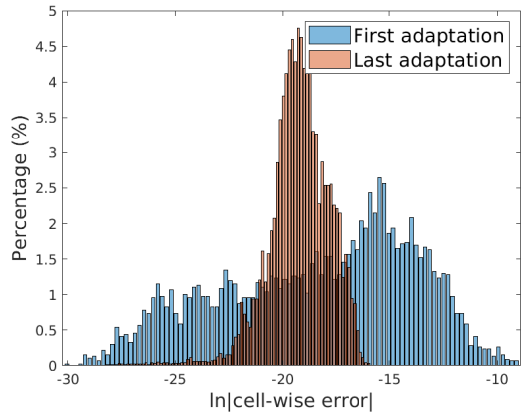
(b) One sub-iteration



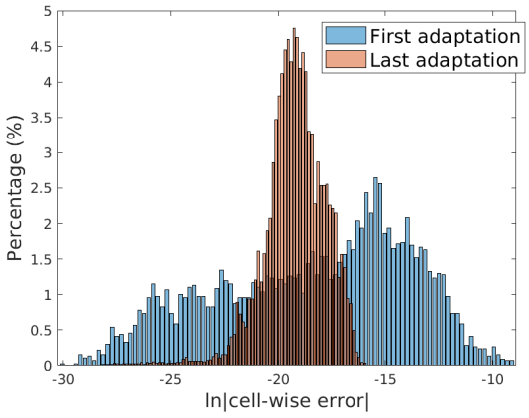
(c) Two sub-iterations



(d) Three sub-iterations



(e) Four sub-iterations

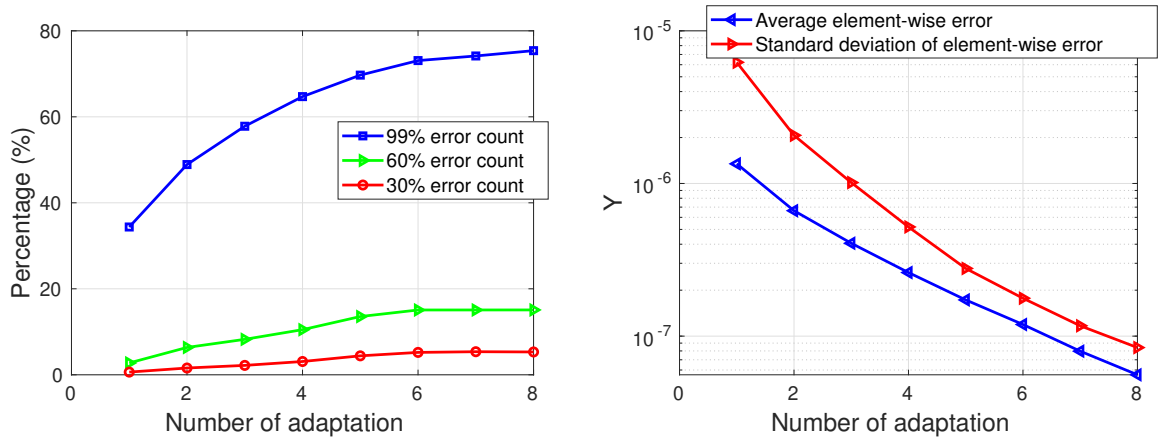


(f) Five sub-iterations

Figure 13: NACA 0012 wing, $M = 0.4$, $\alpha = 3°$: comparison of error indicator distributions.

Figure 14(b) shows the mean and standard deviation of the localized error for the standard

adaptation method. Both of these drop monotonically with each adaptation iteration. The methods employing sub-iterations show a nearly identical trend.



(a) % of elements holding the top 30/60/99% of the error (b) The mean and standard deviation

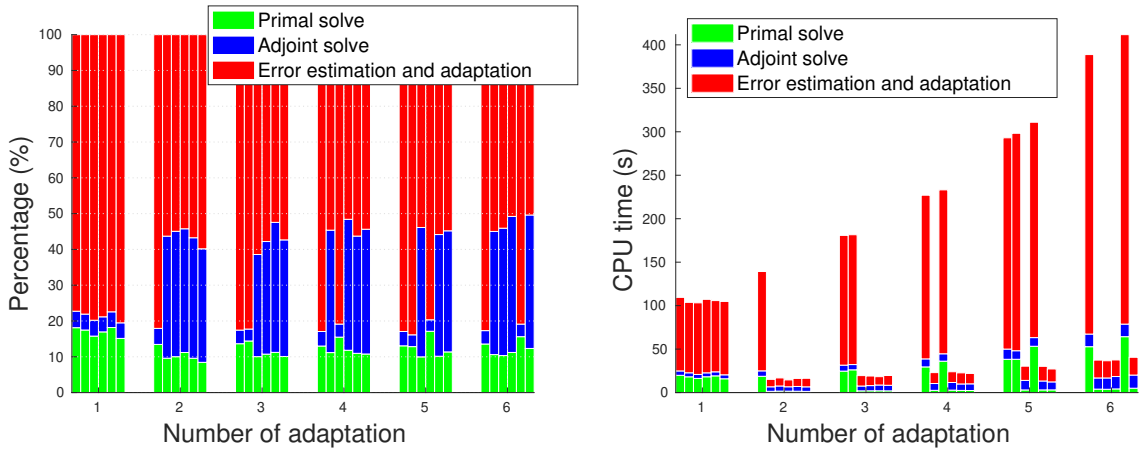
Figure 14: NACA 0012 wing, $M = 0.4$, $\alpha = 3°$: sample statics of the standard adjoint-weighted residual adaptive refinement, element wise error indicator mean, standard deviation and error count.

The CPU time breakdown is shown in Figure 15. We again observe a sharp drop in computational time during the sub-iterations in Figure 15(b), and an enlargement of the blue section of the columns, the relative adjoint solve time, in Figure 15(a). We note that in this three-dimensional case, the fine-space adjoint solve at $p = 2$ is significantly more expensive than a solve at $p = 1$, which accounts for the large contribution of the error estimation and adaptation (red portion) to the total CPU time. The fine space CPU time cost is significantly reduced among all methods employing sub-iterations (shortened red portion, Figure 15(b)), which, in turn, reduced overall CPU time cost for all sub-iteration methods (shortened overall bar height, Figure 15(b)).

5. Summary

We summarize the results of the sub-iteration algorithm from Section 4 with the following observations:

- When measuring CPU time, sub-iterations offer noticeable savings, as shown in Figures 3(d), 7(d) and 12(b), because during sub-iterations, the primal and fine-space adjoint are only smoothed, not solved exactly.



(a) Percentage CPU time breakdown

(b) Raw CPU time breakdown

Figure 15: NACA 0012 wing, $M = 0.4$, $\alpha = 3°$: 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 group 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 fine-space solves.

- In terms of DOF, the use of sub-iterations maintains the performance of error estimation and adaptation, as shown in DOF convergence curves in Figures 3(c), 7(c) and 12(a).

- The sub-iteration algorithm achieves output error reduction (i.e., element-wise error equi-distribution) in the same fashion as standard adaptation, shown in the error equi-distribution analysis histogram in Figures 4, 8 and 13.

- The sub-iteration algorithm achieves CPU time reduction because it reduces the $\mathcal{V}_h$ adjoint solve cost —the $\mathcal{V}_h$ adjoint solve makes up a large fraction of the total CPU cost in output-based adaptive simulations. As shown in Figures 6, 10 and 15, the sub-iteration algorithm reduces the red columns' length where such columns represent the CPU time expenditure on $\mathcal{V}_h$.

- For all tested cases, the optimal sub-iteration frequency was found to be around two.

6. Conclusions

In this paper, we have presented a general algorithm for accelerating output-based error estimation and mesh adaptation by minimizing the fine space, h, operations in output-based methods. The algorithm consists of sub-iterations during adaptation, where at each sub-iteration the primal and fine-space adjoint solves are performed only approximately. These are usually the most expensive solves of every adaptive iteration and hence the cheaper approximate solves (block-Jacobi smoothing) yield noticeable computational time savings. DOF performance of the sub-iterations adaptations is on a par with the standard method, as long as the current-space adjoint is solved exactly to remove errors arising from the approximate primal solves. We demonstrated examples of using sub-iterations for the DG finite-element method for steady-state Euler and Navier-Stokes simulations.

References

[1] K. J. Fidkowski, D. L. Darmofal, Review of output-based error estimation and mesh adaptation in computational fluid dynamics, American Institute of Aeronautics and Astronautics Journal 49 (4) (2011) 673–694.

[2] M. A. Park, Adjoint-based, three-dimensional error prediction and grid adaptation, AIAA Paper 2002-3286 (2002).

[3] M. A. Park, Three–dimensional turbulent RANS adjoint–based error correction, AIAA Paper 2003-3849 (2003).

[4] E. M. Lee-Rausch, M. A. Park, W. T. Jones, D. P. Hammond, E. J. Nielsen, Application of parallel adjoint-based error estimation and anisotropic grid adaptation for three-dimensional aerospace configurations, AIAA Paper 2005-4842 (2005).

[5] R. Hartmann, J. Held, T. Leicht, Adjoint-based error estimation and adaptive mesh refinement for the RANS and $k - \omega$ turbulence model equations, Journal of Computational Physics 230 (11) (2011) 4268 – 4284. doi:10.1016/j.jcp.2010.10.026.

[6] E. J. Nielsen, J. Lu, M. A. Park, D. L. Darmofal, An implicit, exact dual adjoint solution method for turbulent flows on unstructured grids, Computers and Fluids 33 (2004) 1131–1155.

[7] M. B. Giles, M. C. Duta, J.-D. Müller, N. A. Pierce, Algorithm developments for discrete adjoint methods, AIAA Journal 41 (2) (2003) 198–205.

[8] J. Müller, P. Cusdin, On the performance of discrete adjoint CFD codes using automatic differentiation, International Journal for Numerical Methods in Fluids 47 (8–9) (2005) 939–945.

[9] T. Barth, M. Larson, A posteriori error estimates for higher order Godunov finite volume methods on unstructured meshes, in: R. Herban, D. Kröner (Eds.), Finite Volumes for Complex Applications III, Hermes Penton, London, 2002, pp. 41–63.

[10] T. Richter, Discontinuous Galerkin as time-stepping scheme for the Navier-Stokes equations, in: Fourth International Conference on High Performance Scientific Computing Modeling, Simulation and Optimization of Complex Processes, Hanoi, Vietnam, 2009.

[11] H. Huynh, A flux reconstruction approach to high-order schemes including discontinuous Galerkin methods, AIAA Paper 2007-4079 (2007).

[12] A. Burbeau, P. Sagaut, C. Bruneau, A problem-independent limiter for high–order Runge-Kutta discontinuous Galerkin methods, Journal of Computational Physics 169 (2001) 111–150.

[13] F. Bassi, S. Rebay, High–order accurate discontinuous finite element solution of the 2-d euler equations, Journal of Computational Physics 138 (1997) 251–285.

[14] F. Bassi, S. Rebay, An implicit high–order discontinuous galerkin method for the steady state compressible navier-stokes equations, Computational Fluid Dynamics 98, Proceedings from the Fourth European Computational Fluid Dynamics Conference 2 (1998) 1227–1233.

[15] B. Cockburn, C.-W. Shu, Runge-kutta discontinuous galerkin methods for convection-dominated problems, Journal of Scientific Computing 16 (3) (2001) 173–261.

[16] P. Solín, K. Segeth, I. D. Zel, Higher–Order Finite Element Methods, Chapman and Hall, 2003.

[17] P. L. Roe, Approximate Riemann solvers, parameter vectors, and difference schemes, Journal of Computational Physics 43 (1981) 357–372.

[18] F. Bassi, S. Rebay, GMRES discontinuous Galerkin solution of the compressible Navier-Stokes equations, in: K. Cockburn, Shu (Eds.), Discontinuous Galerkin Methods: Theory, Computation and Applications, Springer, Berlin, 2000, pp. 197–208.

[19] R. Becker, R. Rannacher, An optimal control approach to a posteriori error estimation in finite element methods, in: A. Iserles (Ed.), Acta Numerica, Cambridge University Press, 2001, pp. 1–102.

[20] R. Hartmann, P. Houston, Adaptive discontinuous Galerkin finite element methods for the compressible Euler equations, Journal of Computational Physics 183 (2) (2002) 508–532.

[21] D. A. Venditti, D. L. Darmofal, Anisotropic grid adaptation for functional outputs: application to two-dimensional viscous flows, Journal of Computational Physics 187 (1) (2003) 22–46.

[22] T. Apel, S. Grosman, P. K. Jimack, A. Meyer, A new methodology for anisotropic mesh refinement based upon error gradients, Applied Numerical Mathematics 50 (2004) 329–341.

[23] G. Kunert, A posteriori H1 error estimation for a singularly perturbed reaction diffusion problem on anisotropic meshes, Journal of Numerical Mathematics 25 (2005) 408–428.

[24] T. Leicht, R. Hartmann, Multitarget error estimation and adaptivity in aerodynamic flow simulations, Inter-

national Journal for Numerical Methods in Fluids 56 (2008) 2111–2138.

[25] D. Pagnutti, C. Ollivier-Gooch, A generalized framework for high order anisotropic mesh adaptation, Computers and Structures 87 (11-12) (2009) 670 – 679.

[26] T. Richter, A posteriori error estimation and anisotropy detection with the dual-weighted residual method, International Journal for Numerical Methods in Fluids 62 (2010) 90–118.

[27] K. Ding, K. J. Fidkowski, P. L. Roe, Adjoint-based error estimation and mesh adaptation for the active flux method, AIAA Paper 2013–2942 (2013).

[28] K. Ding, K. J. Fidkowski, P. L. Roe, Acceleration techniques for adjoint-based error estimation and mesh adaptation, Eighth International Conference on Computational Fluid Dynamics ICCFD8-0249 (2014).

[29] K. Ding, K. J. Fidkowski, P. L. Roe, Continuous adjoint based error estimation and r-refinement for the active flux method, AIAA Paper 2016-0832 (2016).

[30] K. Ding, K. J. Fidkowski, Output error control using r-adaptation, AIAA Paper 2017-0832 (2017).

[31] J. Peraire, J. Peiró, K. Morgan, Adaptive remeshing for three-dimensional compressible flow computations, Journal of Computational Physics 103 (1992) 269–285.

[32] N. Nguyen, J. Peraire, An interpolation method for the reconstruction and recognition of face images, The 2nd International Conference on Computer Vision Theory and Applications (VISAPP), Barcelona Spain.

[33] N. C. Nguyen, A. T. Patera, J. Peraire, A 'best points' interpolation method for efficient approximation of parmetrized functions, International Journal for Numerical Methods in Engineering DOI: 10.1002/nme.2086.

[34] D. Kraaijpoel, Seismic ray fields and ray field maps: Theory and algorithms, Ph.D. thesis, Utrecht University, The Netherlands (2003).

[35] M. J. Castro-Diaz, F. Hecht, B. Mohammadi, O. Pironneau, Anisotropic unstructured mesh adaptation for flow simulations, International Journal for Numerical Methods in Fluids 25 (1997) 475–491.

[36] W. Rachowicz, D. Pardo, L. Demkowicz, Fully automatic hp-adaptivity in three dimensions, Tech. Rep. 04-22, ICES (2004).

[37] G. F. Naterer, D. Rinn, Towards entropy detection of anomalous mass and momentum exchange in incompressible fluid flow, International Journal for Numerical Methods in Fluids 39 (11) (2002) 1013–1036.

[38] A. Jameson, Aerodynamic design via control theory, Journal of Scientific Computing 3 (1988) 233–260.

[39] F. Beux, A. Dervieux, Exact-gradient shape optimization of a 2-D Euler flow, Finite Elements in Analysis and Design 12 (1992) 281–302.

[40] W. K. Anderson, V. Venkatakrishnan, Aerodynamic design optimization on unstructured grids with a continuous adjoint formulation, Tech. Rep. Report 97-9, ICASE (1997).

[41] J. Elliott, J. Peraire, Practical three-dimensional aerodynamic design and optimization using unstructured meshes, AIAA Journal 35 (9) (1997) 1479–1485.

[42] E. Nielsen, W. Anderson, Aerodynamic design optimization on unstructured meshes using the Navier-Stokes equations, AIAA Journal 37 (11) (1999) 1411–1419.

[43] K. J. Fidkowski, Output error estimation strategies for discontinuous Galerkin discretizations of unsteady convection-dominated flows, International Journal for Numerical Methods in Engineering 88 (12) (2011) 1297–1322. doi:10.1002/nme.3224.

[44] S. M. Kast, K. J. Fidkowski, Output-based mesh adaptation for high order Navier-Stokes simulations on deformable domains, Journal of Computational Physics 252 (1) (2013) 468–494. doi:10.1016/j.jcp.2013.06.007.

[45] R. E. Bank, R. K. Smith, Mesh smoothing using a posteriori error estimates, SIAM Journal on Numerical Analysis 34 (3) (1997) 979–997.

[46] W. Heinrichs, Line relaxation for spectral multigrid methods, Journal of Computational Physics 77 (1988) 166–182.

[47] I. Thomas, T. Sonar, On a second order residual estimator for numerical schemes for nonlinear hyperbolic conservation laws, Journal of Computational Physics 171 (1) (2001) 227–242.

[48] D. De Zeeuw, K. G. Powell, An adaptively refined Cartesian mesh solver for the Euler equations, Journal of Computational Physics 104 (1993) 56–68.