Eighth International Conference on Computational Fluid Dynamics (ICCFD8), Chengdu, China, July 14-18, 2014

ICCFD8-2014-0249

Acceleration Techniques for Adjoint-Based Error Estimation and Mesh Adaptation

Kaihua Ding[1], Krzysztof J. Fidkowski[1], and Philip L. Roe[1]

[1]University of Michigan, Ann Arbor, MI 48105, USA
Corresponding author: kfid@umich.edu

Abstract: In this paper we introduce two ideas for reducing the cost of output-based error estimation and mesh adaptation in steady and unsteady simulations. The first of these is the use of sub-iterations during adaptations, where a sub-iteration is an adaptation iteration in which the most expensive solves, the primal and fine-space adjoint, are done only approximately. The sub-iterations are interspersed with standard full-solve iterations during which accurate error estimates are available. The use of sub-iterations reduces the computational cost without much effect on the performance per iteration. The second strategy is the use of coarser spaces in the context of an adjoint-weighted residual for creating an adaptive indicator. While the resulting error estimate is not accurate, the adaptive indicator still contains useful information, at a much-reduced computational overhead compared to standard fine-space error estimation. We demonstrate these methods for steady, compressible Euler simulations discretized with the discontinuous Galerkin (DG) finite-element method, and for unsteady scalar advection simulations discretized with the active flux method.

Keywords: Adjoint, Solution Verification, Numerical Error Estimation, Mesh Adaptation

1 Introduction

Solution-adaptive techniques are receiving increasingly more attention in Computational Fluid Dynamics research and practice as a means of improving solution accuracy and reducing computational cost. They are particularly important for aerospace engineering applications, where convection phenomena on complex three-dimensional geometries make a priori mesh design a daunting task. One of the most rigorous solution-adaptive techniques is output-based error estimation and adaptation, reviewed recently in detail in [1].

Output-based error estimation is a powerful technique for quantifying the impact of numerical discretization errors on specific scalar outputs. The resulting estimates reflect the extent to which mesh resolution and distribution affect an output of interest. Furthermore, the error estimates provide information on areas 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 cheap. 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 employ a fine-space *adjoint* solution and/or at least a fine-space residual evaluation. These fine-space calculations can make the cost of error estimation and adaptation burdensome for practical simulations.

In this paper we introduce and formalize two "shortcuts" for estimating the output error and adapting the mesh using the adjoint-weighted residual. One shortcut relies on re-using fine-space adjoint iteration for more than just one adaptation iteration. The other shortcut relies on a coarser instead of finer space for calculating the error indicator. We show that both strategies have little detrimental effect on the performance of adaptation with degrees of freedom, but that they reduce the computational time for all cases tested.

The remainder of this paper is organized as follows. In Section 2 we introduce the discretizations used in this work: discontinuous Galerkin (DG) and active flux. Section 3 presents the adjoint-weighted residual

error estimate, and Section 4 discusses the two proposed strategies for accelerating the adaptation. Section 5 presents results for several test cases, using DG for steady-state flows and the active flux method for unsteady flows, and Section 6 offers concluding remarks.

2 Discretizations

We demonstrate our acceleration techniques for error estimation and adaptation using two discretizations: the discontinuous Galerkin method and the active flux method. The target partial differential equation is a system of first-order conservation laws,

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

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. For DG, we will focus on steady problems, $\partial_t \mathbf{u} = \mathbf{0}$, whereas for active flux, we will consider unsteady problems.

2.1 Discontinuous Galerkin

DG is a finite element method in which the state $\mathbf{u}$ is spatially approximated in functional form, using linear combinations of basis functions, usually polynomials, on each element. No continuity constraints are imposed on the approximations on adjacent 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

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

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 integrating by parts to couple elements via fluxes. The convective fluxes on element faces are handled via a traditional finite-volume (approximate) Riemann solver. 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

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

where $(\cdot)^T$ denotes transpose, and on the element boundary $\partial\Omega_e$ the notations $(\cdot)^+, (\cdot)^b$ respectively denote quantities taken from the element interior and boundary. In particular, on an interior face σ^f, the convective flux $\widehat{\mathbf{F}}$ is computed using the Roe approximate Riemann solver [2], while 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 the test functions, we obtain a discrete system of equations (i.e. residuals),

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

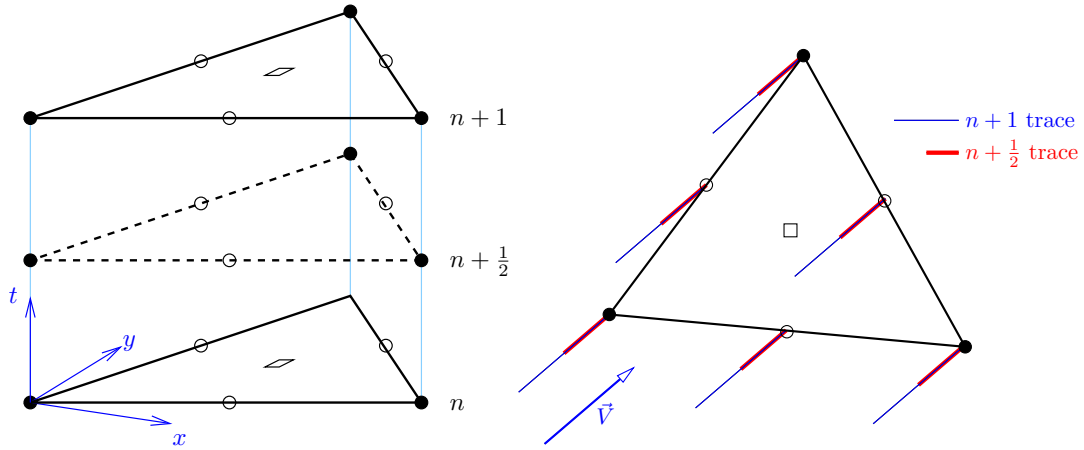
Note that both $\mathbf{U}$ and $\mathbf{R}$ lie in $\mathbb{R}^N$, where N is the total number of degrees of freedom including equation

states. 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.

2.2 Active Flux

The active flux method is a third order finite volume method developed by Eymann and Roe [3–6], building on the work of van Leer [7]. We briefly review the method for scalar advection in two dimensions, i.e. Eqn. 1 with $s = 1$ and $\vec{\mathbf{F}} = \vec{V}u$.

Active flux is an inherently unsteady discretization in which a third-order accurate solution representation at time level n is propagated to time level $n+1$ through an intermediate level $n+\frac{1}{2}$. Figure 1 illustrates the time levels and unknowns for one element of a triangular mesh. Seven unknowns pertain to each triangular



(a) Time levels

(b) Traces for vertex/edge updates

Figure 1: Illustration of the three time levels and unknown placement for one element in the active flux method. Shaded circles are vertex unknowns, open circles are edge unknowns, and the squares represent the cell average unknown.

element: one at each vertex, one at each edge midpoint, and one cell average, $\bar{u}$. At each time level, these unknowns support an augmented quadratic spatial representation of the solution. Specifically, the six vertex and edge unknowns are used as coefficients in an expansion with quadratic Lagrange basis functions. This quadratic representation is augmented by a cubic bubble function that vanishes on the element perimeter and whose magnitude is uniquely defined by the requirement that the cell average is $\bar{u}$, the seventh unknown.

With the spatial representation in hand, the update procedure for linear advection is relatively simple. It consists of three steps:

1. Determine values for the edge and vertex unknowns at time levels $n+\frac{1}{2}$ and $n+1$ by "tracing back" the solution along the velocity direction to the known augmented quadratic representation at time level n. This is illustrated schematically in Figure 1(b).

2. On each edge of the element, we now have nine solution values: three at each time level, n, $n+\frac{1}{2}$, and $n+1$. These nine values define a quadratic state, and hence flux, in space and time. Integrate this flux to obtain a third-order accurate net flux through the edge over the time step.

3. Using the integrated fluxes from all edges of the element, obtain the cell average at $n+1$ from the cell average at n via a standard finite-volume discrete conservation statement.

The active-flux method is inherently explicit, yet for the purpose of a discrete adjoint (for error estimation) we can still write the set of vertex, edge, and cell-average updates in the form of a residual system,

$$\mathbf{R}^{n+1}(\mathbf{U}^n, \mathbf{U}^{n+1}) = \mathbf{0},$$

where $\mathbf{U}^n$ is the set of all of the unknowns at time level n. Alternatively, we could use a more direct continuous adjoint formulation for our error estimates, and this is a direction we are currently pursuing.

3 The Adjoint-Weighted Residual

Output-based error estimates rely on the concept of an adjoint-weighted residual [1,8–10]. This idea is based on the definition of an output adjoint, which is a sensitivity of the output to residual perturbations. While on a particular mesh, residuals are typically driven to negligible size by the solver, 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 cheap.

Denote by $\mathbf{U}_H$, $\mathbf{U}_h$ the primal solutions on coarse, respectively fine, spaces. 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 the 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 [1, 8] reads

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

The discrete fine-space adjoint, $\boldsymbol{\Psi}_h$, is vector of the same size as the state and residual vectors that satisfies

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

$\boldsymbol{\Psi}_h$ weights the residual perturbation, $\delta\mathbf{R}_h$, to give a linearized estimate of the output difference between the coarse and fine spaces. Although in this form of the error estimate the primal state is not required, the computational and storage costs associated with Eqn. 6 are not trivial, as we can see by breaking-down each of the terms:

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

First, we need to inject the state into the fine space. One way to construct a finer space is uniform mesh refinement, which increases the degrees of freedom four-fold in two dimensions and eight-fold in three dimensions. Second, along with the increase in degrees of freedom comes computational overhead in the form of element geometry quantities, basis functions, mappings, etc., 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. In the following section, we introduce a technique for reducing the cost of this estimate.

The adjoint-weighted residual error estimate is typically paired with mesh adaptation in which the mesh is successively refined to reduce the error. Often the mesh is refined incrementally, for example when 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 the next section we thus also introduce an approach to avoid fully-repeating this calculation at each adaptation iteration.

4 Acceleration Techniques

In this section we introduce and formalize two "shortcuts" for estimating the output error using the adjoint-weighted residual, and for adapting the computational mesh. These shortcuts are

1. Adaptive "sub-iterations" in which the primal problem is not solved on every adaptive iteration so as to minimize the cost of multiple nonlinear primal solves, and in which the fine-space adjoint is used for more than one iteration.

2. Use of coarse-space indicators to drive adaptation, eliminating the need for fine-space adjoint calculations for the purpose of an adaptive indicator.

4.1 Adaptive Sub-Iterations

Figure 2 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. Specifically, after solving the primal problem exactly (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 about 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

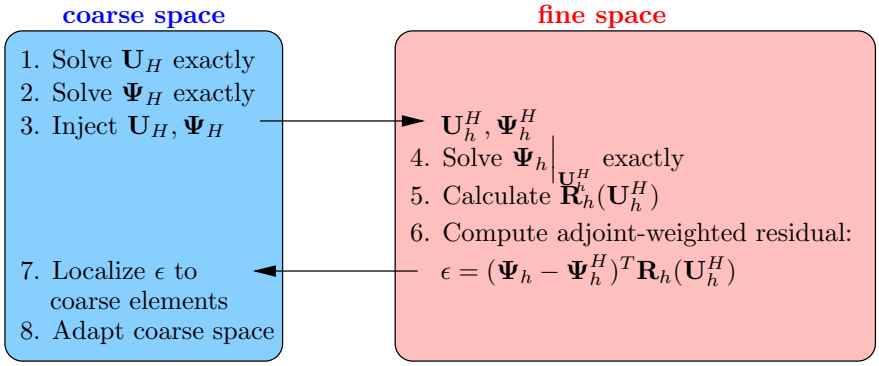


Figure 2: Schematic of a "standard" error estimation and adaptation iteration in which the fine space adjoint is solved exactly at every iteration.

estimate then drives adaptation; in a fixed-fraction setting only the elements with the highest contribution to the error are targeted for refinement. The process then repeats with another exact primal solve, exact fine-space adjoint solve etc.

In the standard adaptive scheme we solve the fine-space adjoint exactly in order to get good error estimates, which we can then use to correct the solution and to often buy ourselves an extra order of accuracy. To reduce the computational burden of this exact solve, we can approximate the fine-space adjoint, either through iterative smoothing or reconstruction [11, 12]. However, the error estimates often suffer when using such approximations.

In this work we present a more efficient adaptive solution scheme that is based on two ideas:

1. Adaptive sub-iterations in which the primal problem is not solved on every adaptive iteration so as to minimize the cost of multiple nonlinear primal solves.

2. Re-use of the fine-space adjoint between adaptive iterations, to avoid the cost of solving a large fine-space adjoint problem at each iteration.

Figure 3 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 piggy-back on this fine-space adjoint to further refine the mesh at a lower computational

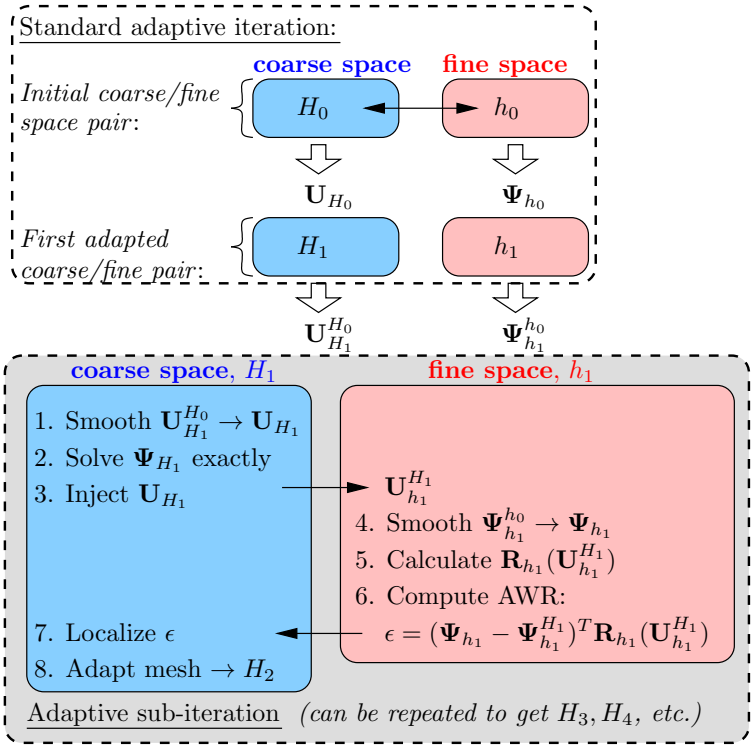


Figure 3: Schematic of the proposed error estimation and adaptation iteration in which approximate sub-iterations piggy-back on a standard adaptive iteration. In particular, the fine-space adjoint solve is reused in the sub-iterations, where it is only smoothed via an inexpensive iterative solver, thereby saving computational time compared to the standard approach in which the fine space adjoint is re-solved on every adaptive iteration.

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 and remove from the error estimate the error due to the incomplete coarse-space primal solve. This prevents the sub-iteration adaptive indicator from becoming distracted by coarse-space primal residuals that are nonzero solely because of our 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 relies in part on 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 refinements, 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.

4.2 Coarse-Space Error Estimation

Standard output error estimation relies on a fine-space adjoint solution weighting a fine-space residual. If the mesh is sufficiently resolved such that error estimates are in an asymptotic regime, then this fine-space error estimate converges to the true error at a rate that depends on certain choices in the error estimation procedure [8]. That is, the fine space introduces new information via residuals and adjoints, and hence it produces a mathematically rigorous error estimate.

However, in practice, for complex aerodynamic simulations, the meshes on which we apply output error estimation and mesh adaptation are rarely fine enough for such asymptotic results to hold. This then begs the question: to what extent is the rigorous formalism of output-based error estimation applicable to, or necessary for, practical simulations? The question is especially relevant when only computing an adaptive indicator, for which rigorous error estimates may not be necessary.

In this work we focus on one particular shortcut: instead of estimating the error between the current space, H, and a finer space, h, we propose to estimate the error between a *coarser* space, denoted by $\tilde{H}$, and the current space, H. We apply the adjoint-weighted residual formulas directly to the pair of spaces $\tilde{H}/H$, so that none of the error estimation formulas need intrinsic changes. In particular, our proposed error estimate becomes Eqn. 9,

$$\delta J \approx -\boldsymbol{\Psi}_H^T \mathbf{R}_H \left(\underbrace{\mathbf{U}_H^{\tilde{H}}}_{\boxed{\text{injected state}}} \right) \tag{9}$$

where $\mathbf{U}_H^{\tilde{H}}$ is the "coarser" solution injected into the current space. Eqn. 9 estimates the output error between the current space (H) and the coarser space ($\tilde{H}$). It does not tell us how much error is present in the current space relative to a finer space, but conceivably the localized form of Eqn. 9 could still provide useful adaptive information (albeit with a possible lag in adaptive iterations). Figure 4 illustrates schematically the use of the coarser space.

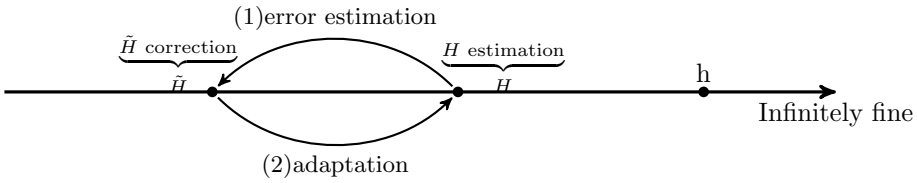


Figure 4: Illustration of spaces used for coarser-space error estimation and adaptation.

The question now is how to define the coarser space ($\tilde{H}$). We can either coarsen the mesh or decrease the scheme approximation order. Coarsening an unstructured mesh is doable but challenging. It does not generally yield a pair of *nested* spaces, in that solutions on the coarsened space will not always be representable on the current space. The nested property allows for simple injection operators and reduces

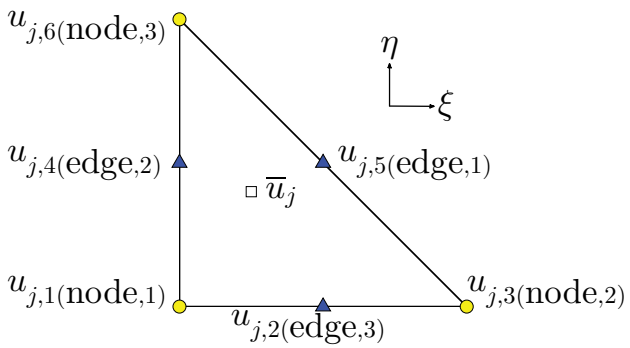


Figure 5: Unknown placement in the active flux method.

additional sources of error. Hence instead, in this work, we choose to coarsen the current space by decreasing the scheme approximation order.

In DG, reducing the approximation order is simple, since DG discretizations can be defined for an arbitrary order p. Therefore, here we will target the active flux scheme, for which we have already explored standard adaptation techniques in previous work [13]. As described in Section 2.2, the active flux method is third-order in space and time. In order to implement our idea, we need to come up with some strategies to create a fictional reduced-order space ($\tilde{H}$). The active flux method uses three independent types of states: edge states, node states, and cell average states, as illustrated in Figure 5. To create discretization errors due to the drop of state approximation error, we have many options. Below we outline five strategies.

- **Strategy 1**

  The first strategy is to eliminate three out of seven independent states on each cell: the three edge states. This strategy is illustrated in Figure 6(a). We then treat the remaining three node states as basis coefficients in a linear approximation of the state, as in the $p = 1$ discontinuous Galerkin (DG) finite element method. New edge states are calculated by interpolation with this linear approximation. As a result, the new edge states are no longer independent from the node states. We lose degrees of freedom, and we expect this coarser space to be second-order accurate. We carry out this calculation on each element separately, but because edge states are uniquely defined in the active flux scheme, we use the average edge states whenever we have two different edge states at the same location.

- **Strategy 2**

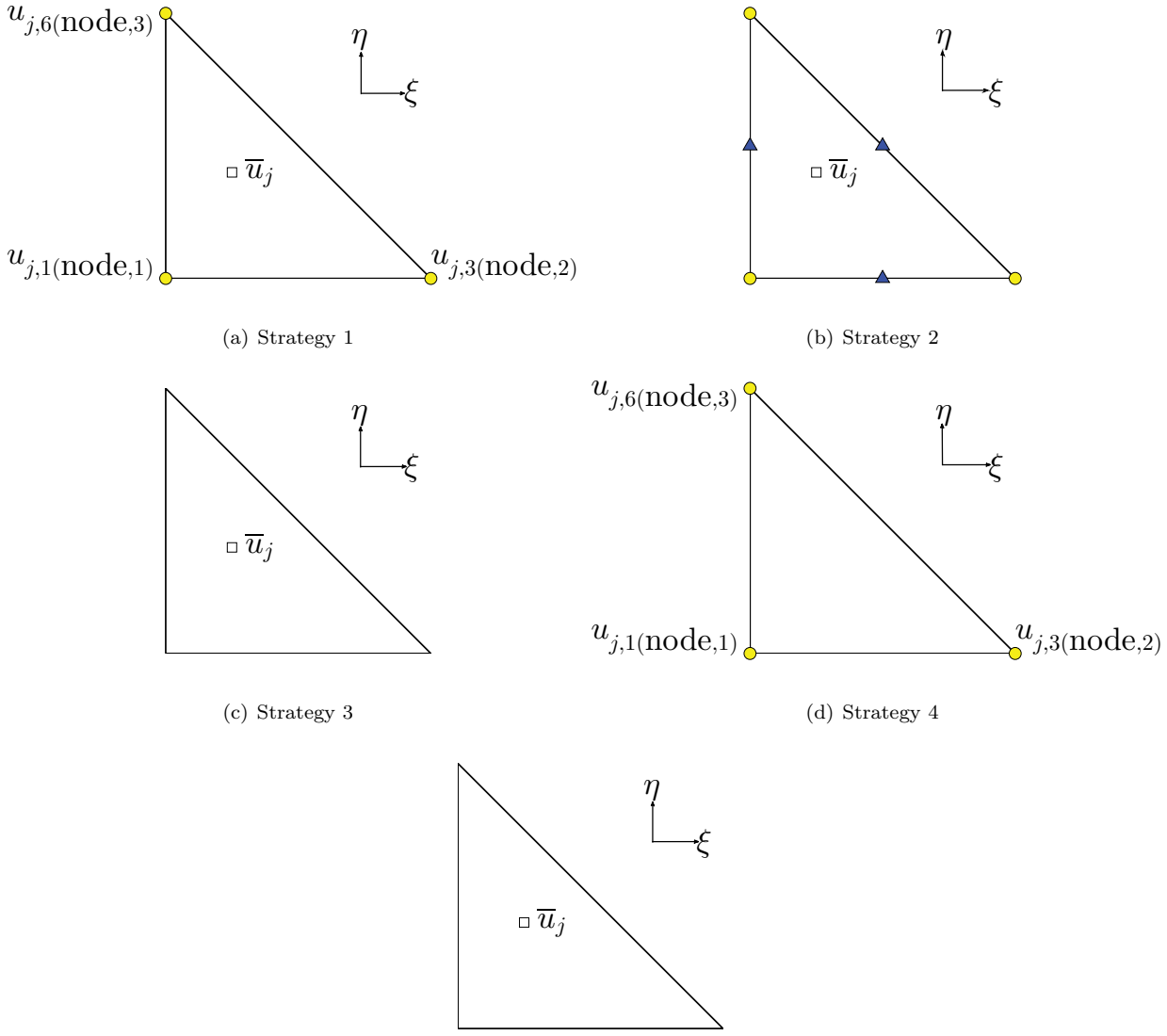
  In this strategy, we only keep the cell average state, as shown in Figure 6(b). Looping over elements in some lexicographical order, whichever nodes and edges are hit first become the new node and edge states. As a result, the approximation basis functions are rather ad hoc and we lose accuracy of our approximation. Specifically, we expect first-order accuracy.

- **Strategy 3**

  In this strategy, we also only keep the cell average state, as shown in Figure 6(c). In contrast to Strategy 2, though we still loop over elements to see whichever nodes and edges we hit first, we do not assign the previous node or edge states to be the new node and edge states. Instead, we assign the new node and edge states using the cell average states.

- **Strategy 4**

  In this strategy we drop the order of the numerical approximation via a least-squares projection to a linear basis in each element, as shown in Figure 6(d). Hence, seven degrees of freedom on space H become three degrees of freedom on space $\tilde{H}$. Edge and cell average states on $\tilde{H}$ are interpolated from this linear approximation. These interpolated states are no longer independent from the node states, and thus the scheme approximation order drops. Because interface unknowns are shared among



(a) Strategy 1

(b) Strategy 2

(c) Strategy 3

(d) Strategy 4

(e) Strategy 5

Figure 6: Illustration of proposed five strategies.

neighboring elements in the active flux method, whenever we have two unknowns at the same location, we use their average values.

- **Strategy 5**

  We reduced the approximation order by one to create strategy 4, but we can also go further, e.g. a reduction by two, as illustrated in Figure 6(e). For the case of the active flux method, which is third order, we then reduce to first order – i.e. one degree of freedom per cell. The coarser space $\tilde{H}$ discretization then becomes a standard first-order finite volume method. We still use least-squares projection to solve for the single unknown per element. On each element, edge and node states are assigned the same value as the cell average states. These values are then averaged to produce unique node and edge states whenever we have multiple unknowns at the same location.

5 Results

In this section we present results of our error estimation and adaptation acceleration strategies applied to the discontinuous Galerkin and the active flux methods.

5.1 Steady-State DG Simulations

We demonstrate the adaptive sub-iteration technique for the discontinuous Galerkin discretization of the steady-state Euler equations in two spatial dimensions.

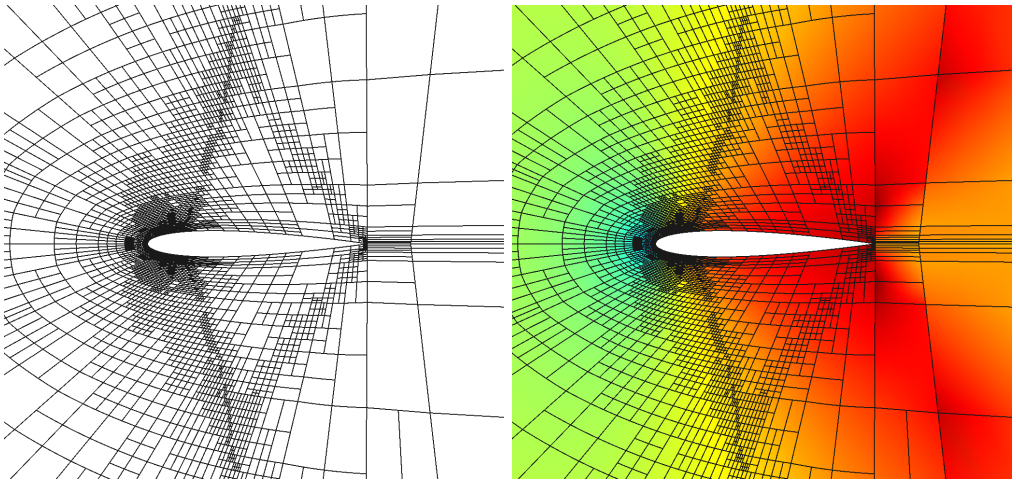
5.1.1 Transonic airfoil with a fishtail shock

We first consider a NACA 0012 airfoil in inviscid (Euler) flow at Mach number $M = 0.95$ and angle of attack $\alpha = 0°$. The flow is transonic and we use element-wise artificial viscosity [14], discretized using the second-form of Bassi and Rebay [15], to stabilize the solution. We consider drag prediction using an approximation order of $p = 1$ and a fixed fraction of $f = 0.1$ for adaptive (sub-)iterations. The initial mesh consists of 234 quadrilaterals, curved with a quartic geometry representation.

Figure 7 shows a comparison of several adaptive techniques for this case. These include simple uniform refinement, a standard method without sub-iterations, and two methods with sub-iterations. As shown in Figure 7, 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 degrees of freedom, which means that our approximations in the sub-iterations do not have a strong effect on the adaptive indicator that dictates which elements are chosen for refinement. Furthermore, the outputs corrected by the error estimates (dashed lines) are also similar for 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.

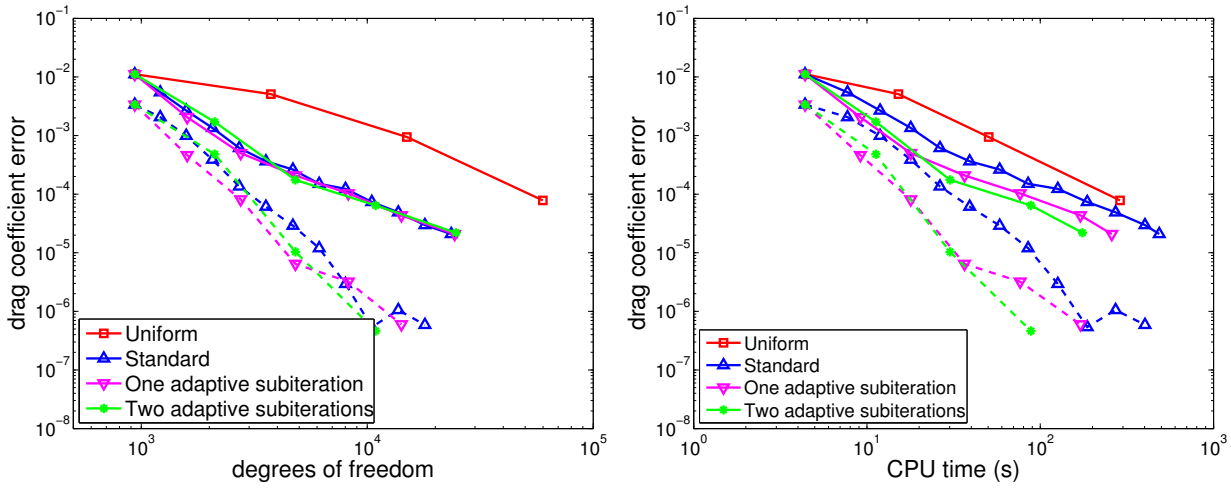
The more interesting plot, however, is the one in Figure 7(d), which shows the convergence of the drag output against computational time. Both the uncorrected and corrected outputs now converge faster for the runs with adaptive sub-iterations. This makes sense because each sub-iteration is cheaper than a regular adaptive iteration due to smoothing of the coarse primal and the fine adjoint. For the later adaptations, the benefit of sub-iterations is at times as much as an order of magnitude error reduction for a given computational time.

Figure 8(a) shows normalized histograms of the elemental error indicator (obtained from localizing the error estimate) for the first and last adaptive iterations of three of the methods. We see that after adapting, all of the methods yield a normalized error histogram shifted to the left – meaning that elements with high errors were targeted for refinement. Moreover, the normalized histograms are similar among the methods, which indicates that they are performing comparably. Figure 8(d) shows, for the standard adjoint-weighted residual method, how the error equidistributes 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 85% of the elements. Figure 9 further illustrates this point: both the mean and standard deviation of the error indicator drop with each adaptive iteration.



(a) Final drag-adapted mesh

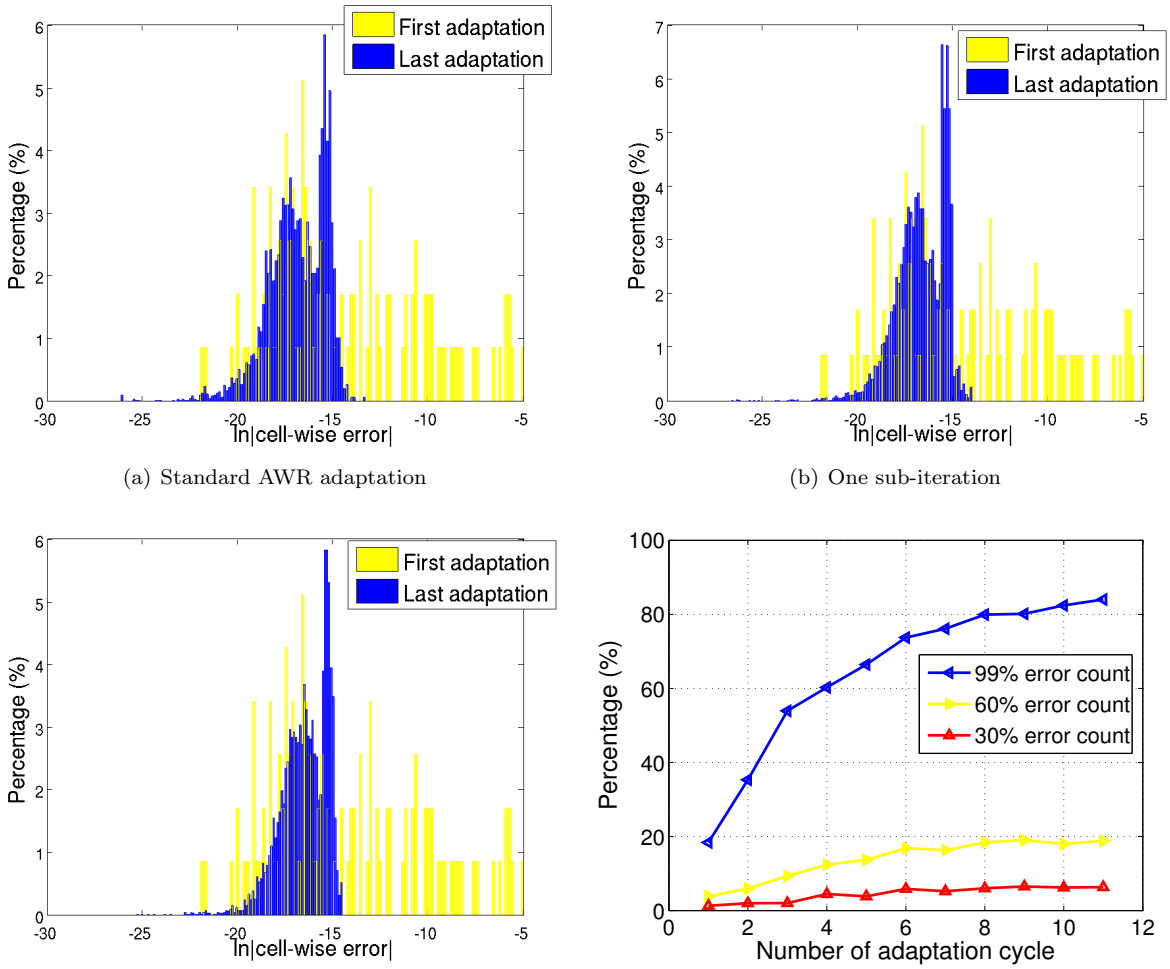
(b) Mach contours (0 to 1.5)



(c) Drag convergence with DOF

(d) Drag convergence with CPU time

Figure 7: NACA 0012, $M = 0.95$, $\alpha = 0°$: effect of sub-iterations on drag convergence. In both of the cases employing sub-iterations, the fine-space adjoint was reused on the sub-iterations with only one element block-Jacobi smoothing iteration as the extra solve. The current-space primal was also only block-Jacobi smoothed on the current space, but the linear coarse-space adjoint problem was solved exactly for all iterations. Dashed lines indicate the remaining error after correction with the estimate.



(a) Standard AWR adaptation

(b) One sub-iteration

(c) Two sub-iterations

(d) % of elements holding the top 30/60/90% of the error

Figure 8: NACA 0012, $M = 0.95$, $\alpha = 0°$: comparison of error indicator distributions.

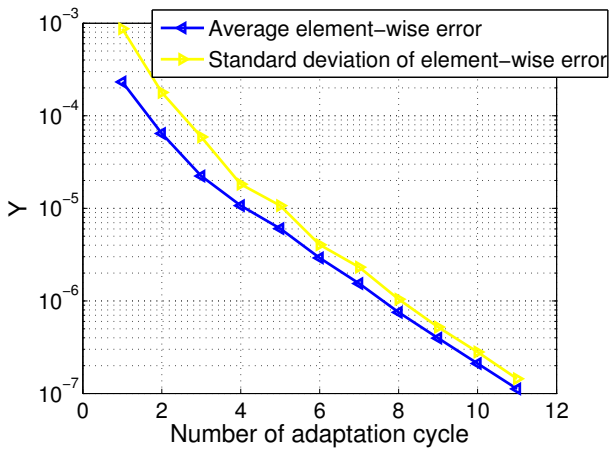
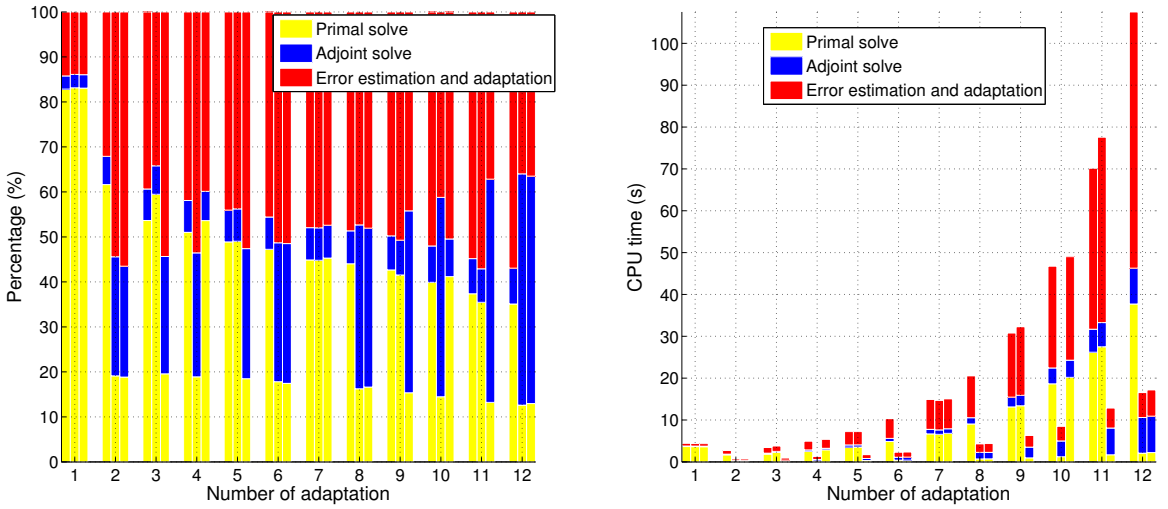


Figure 9: NACA 0012, $M = 0.95$, $\alpha = 0°$: convergence of the mean and standard deviation of the error indicator with adaptive mesh refinement for the standard adjoint-weighted residual method.

Figure 10 shows the CPU time breakdown for the different adaptation strategies. Figure 10(a) shows the CPU time percentage breakdown and Figure 10(b) shows the actual CPU time breakdown[1].



(a) Percentage CPU time breakdown

(b) Raw CPU time breakdown

Figure 10: NACA 0012, $M = 0.95$, $\alpha = 0°$: CPU time breakdown results. At each of the 12 adaptive iterations, we show three bar plots, which are, from left to right: standard adaptation, adaptation with one sub-iteration, and adaptation with two sub-iterations. Each of these bars is divided vertically into three parts, which indicate the CPU time contribution of the primal solve (yellow), the adjoint solve (blue), and the error estimation and adaptation (red). Note that the latter includes the fine-space solves.

Figure 10(b) reveals the benefits of sub-iterations. We take the first bar in each group of three 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 one sub-iteration, which occurs at every even total iteration number: on these iterations, the primal and fine-space adjoints are only smoothed (yellow 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 two 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). Figure 10(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 7 that the standard, one sub-iteration, and two sub-iteration methods perform similarly in degrees of freedom, the methods with sub-iterations have a CPU time advantage for a given level of accuracy.

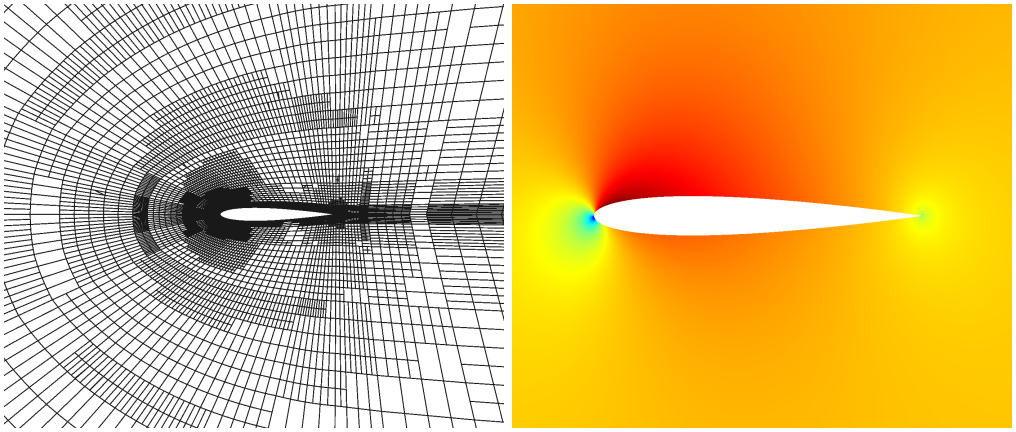
The transonic fishtail case demonstrates the strength of sub-iterations. To further test the capability of the proposed technique, we next present test cases for a subsonic airfoil and a three dimensional wing.

5.1.2 An airfoil in subsonic flow

The second test case is a NACA 0012 airfoil at a free-stream Mach number of 0.5 and angle of attack of 2°. 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 $p = 2$.

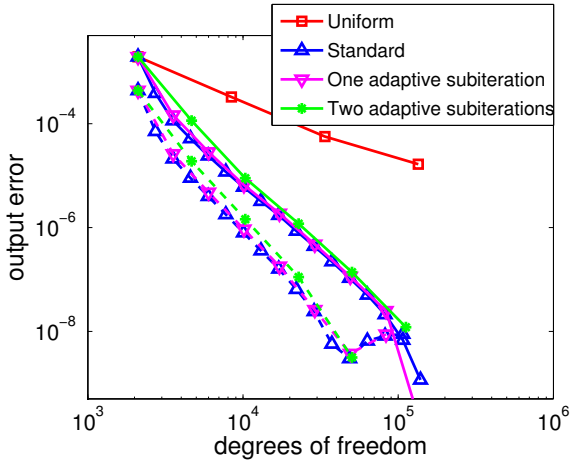
Figure 11 shows a comparison of the various adaptive strategies for this case. Figure 11(c) shows that the methods with sub-iterations exhibit a similar output error convergence behavior with degrees of freedom

[1]Timings were performed on the University of Michigan Flux cluster, on nodes that each had two six-core 2.67 GHz Intel Xeon X5650 processors and 48GB RAM.

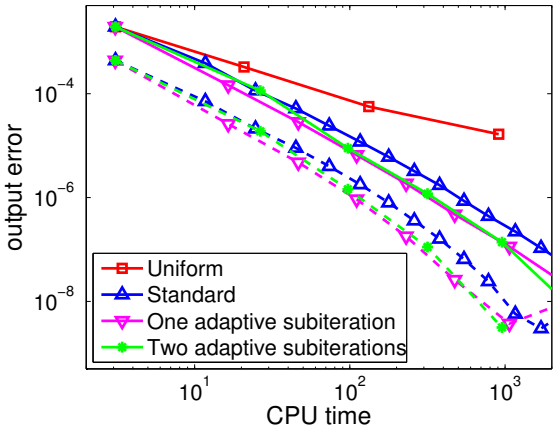


(a) Final drag-adapted mesh

(b) Mach contours (0 to 0.72)



(c) Drag convergence with DOF

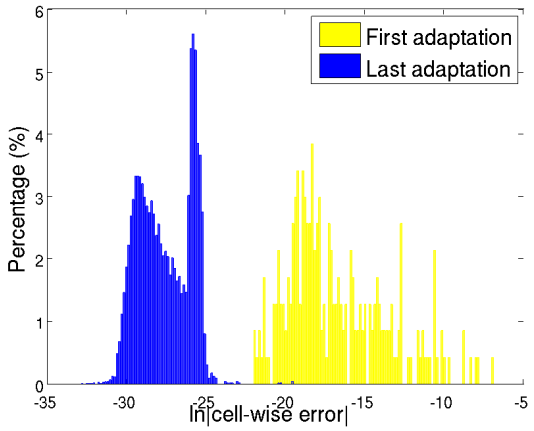


(d) Drag convergence with CPU time

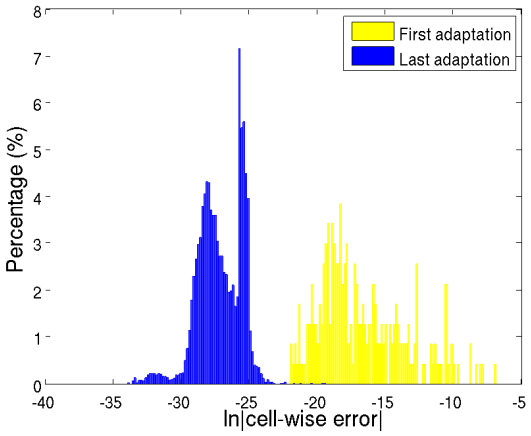
Figure 11: NACA 0012, $M = 0.5$, $\alpha = 2°$: effect of sub-iterations on drag convergence. In both of the cases employing sub-iterations, the fine-space adjoint was reused on the sub-iterations with only one element block-Jacobi smoothing iteration as the extra solve. The current-space primal was also only block-Jacobi smoothed on the current space, but the linear coarse-space adjoint problem was solved exactly for all iterations. Dashed lines indicate the remaining error after correction with the estimate.

compared to standard adaptation. However, as shown in Figure 11(d), sub-iterations show an advantage in CPU time over standard adaptation. The bottoming-out of the corrected output in this case is likely due to a relatively loose residual convergence tolerance of 10^{-8} used in the calculations.

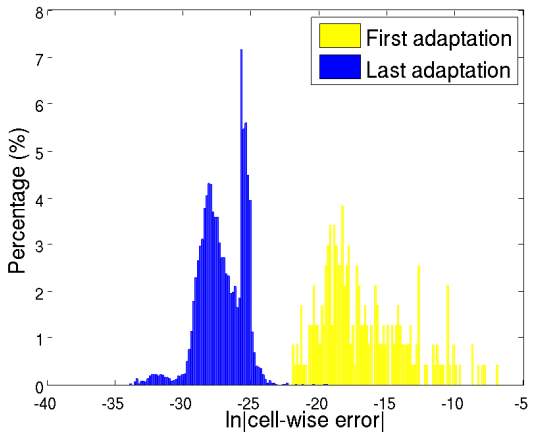
Figure 12 shows normalized histograms of the error indicator distribution for the different adaptation strategies. Again, we see a similar trend for all three methods: the error distribution tightens and shifts to the left from the first to the last adaptive iteration. There are some differences in the normalized histogram for the lowest errors, but these are least important to adaptation: at the larger error values, the normalized histograms appear very similar. Figure 12(d) 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 15% of the elements accounted for 99% of the error, by the final adaptive iteration, 99% of the error is distributed among a much larger 80% of the elements. We see an interesting trend in the 30% and 60% curves, which dip in the later adaptation interations. This indicates that eventually, there is small number of "troublesome" elements that contribute a big fraction to the error – likely near the trailing edge. Hanging-node (bisection) refinement does not decrease the size of these elements fast enough at each fixed-fraction adaptive iteration, so that their lower convergence rate eventually shows through.



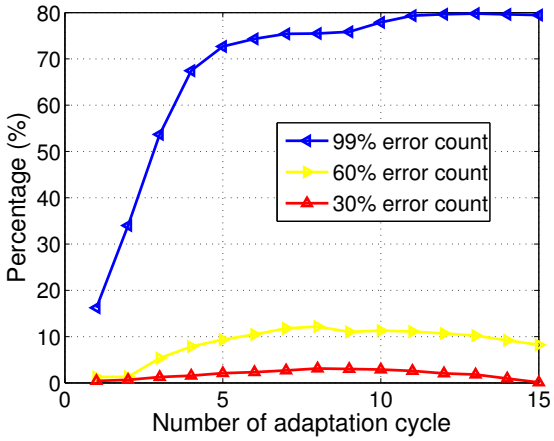
(a) Standard AWR adaptation



(b) One sub-iteration



(c) Two sub-iterations



(d) % of elements holding the top 30/60/90% of the error

Figure 12: NACA 0012, $M = 0.5$, $\alpha = 2°$: comparison of error indicator distributions.

Figure 13 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.

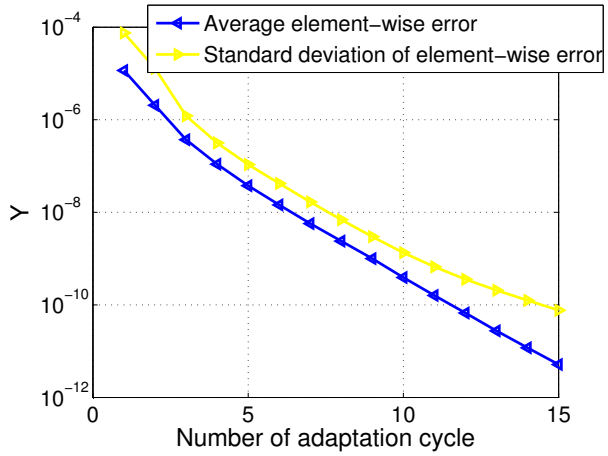
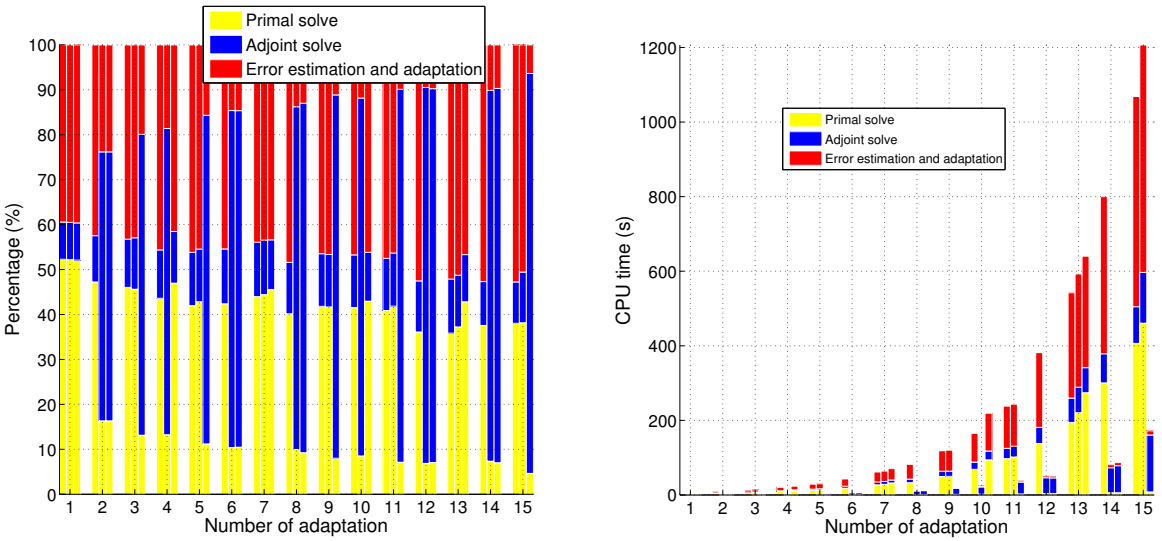


Figure 13: NACA 0012, $M = 0.5$, $\alpha = 2°$: convergence of the mean and standard deviation of the error indicator with adaptive mesh refinement for the standard adjoint-weighted residual method.

Figure 14 shows the CPU-time breakdown comparison among standard adaptation and versions with sub-iterations. The 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 are similar. As result, whenever the adaptation mechanics enters a sub-iteration cycle, the total time drops (Figure 14(b)) while the percent of the time taken by the adjoint solve increases (Figure 14(a)).



(a) Percentage CPU time breakdown

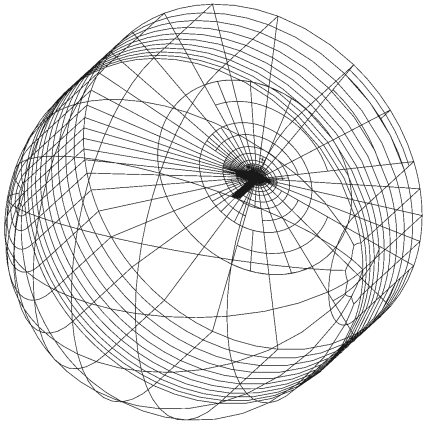
(b) Raw CPU time breakdown

Figure 14: NACA 0012, $M = 0.5$, $\alpha = 2°$: CPU time breakdown results. At each of the 15 adaptive iterations, we show three bar plots, which are, from left to right: standard adaptation, adaptation with one sub-iteration, and adaptation with two sub-iterations. Each of these bars is divided vertically into three parts, which indicate the CPU time contribution of the primal solve (yellow), the adjoint solve (blue), and the error estimation and adaptation (red). Note that the latter includes the fine-space solves.

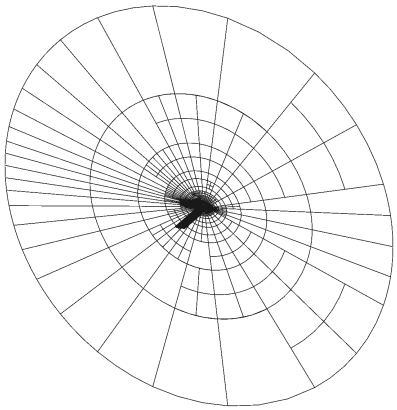
5.1.3 3D Wing case

In this section we demonstrate the performance of sub-iteration adaptation for 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 at $M = 0.4$ and $\alpha = 3°$. 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 is $f = 0.1$.

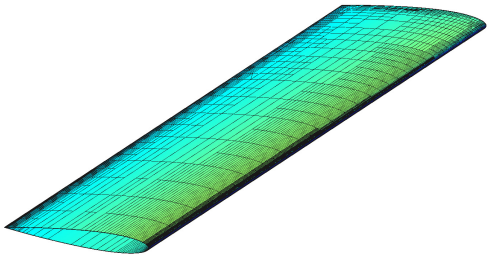
Figure 15 shows the final mesh obtained from adaptation using the standard adjoint-weighted residual. We see that the leading edge, trailing edge, and parts of the wake are targeted for refinement. Figure 16 shows



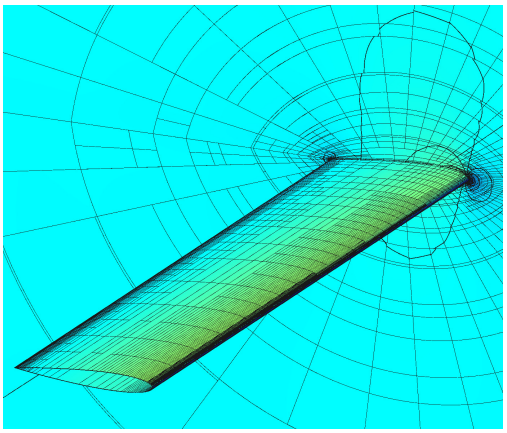
(a) Mesh overview



(b) Symmetry BC and wing surface BC mesh



(c) Wing surface mesh

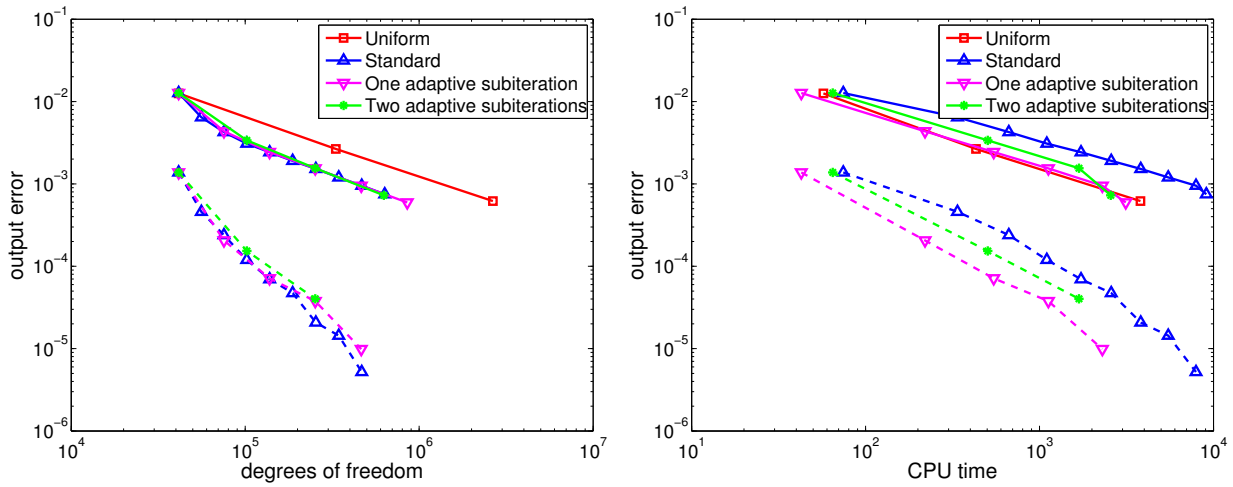


(d) Mach number plot with cutplane showing the contour of Mach number plot, Mach number from 0 to 1.33

Figure 15: NACA 0012 wing, $M = 0.4$, $\alpha = 3°$: adapted mesh and surface Mach contours.

the output error convergence for the various methods versus degrees of freedom and CPU time. As expected, the standard and sub-iteration methods have similar performance with degrees of freedom, and the methods with sub-iterations perform better with CPU time. Figure 17 shows the normalized error histograms for the various methods. These again appear similar among the methods, each showing a decreasing-error trend from the first to the last adaptation iteration. Figure 17(d) 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.

The CPU time breakdown is shown in Figure 18. We again observe a sharp drop in computational time



(a) Convergence with degrees of freedom

(b) Convergence with CPU time

Figure 16: NACA 0012 wing, $M = 0.4$, $\alpha = 3°$: effect of sub-iterations on drag convergence. In both of the cases employing sub-iterations, the fine-space adjoint was reused on the sub-iterations with only one element block-Jacobi smoothing iteration as the extra solve. The current-space primal was also only block-Jacobi smoothed on the current space, but the linear coarse-space adjoint problem was solved exactly for all iterations. Dashed lines indicate the remaining error after correction with the estimate.

during the sub-iterations in Figure 18(b), and an enlargement of the blue section of the columns, the relative adjoint solve time, in Figure 18(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.

In summary, at least for the cases tested, the use of sub-iterations has a minimal effect on the performance of error estimation and adaptation when measured with degrees of freedom. However, when measuring CPU time, sub-iterations offer a noticeable savings because during sub-iterations, the primal problem and fine-space adjoint problems are only smoothed, not solved exactly.

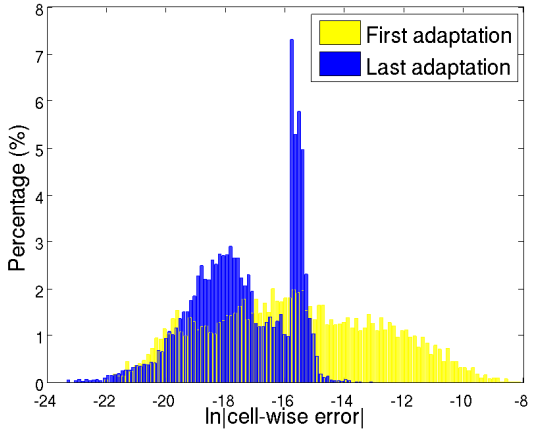
5.2 Unsteady Active-Flux Simulations

In this section, we present results showing how the use of a coarser space fares in driving mesh adaptation in the active-flux method. The current mesh adaptation mechanics is static, meaning that the mesh does not change in time. According to the discussion in Section 4.2, from an implementation point of view, there is essentially no difference between conventional error estimation and coarse-space error estimation, aside from the choice of spaces. We expect the error estimates themselves to be less useful compared to those obtained from fine-space error estimation, although they may not be completely without value – this is something we are currently investigating. At present, we test the coarse-space selection strategies outlined in the previous section in their abilities to drive adaptation; that is, in identifying elements that need to be adapted.

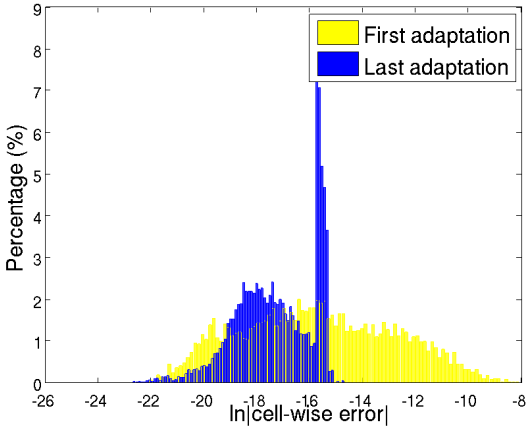
In two dimensions, we consider the test case of an advecting Gaussian wave as shown in Figure 19. Figure 19(a) shows the initial unstructured mesh, and Figure 19(b) shows the primal solution. Here, an inflow boundary condition is enforced on the left and lower boundaries of the square domain. A Gaussian pulse originally centered at coordinate, $\vec{x} = (-0.4,\ -0.4)$ advects diagonally, until it arrives at the point $\vec{x} = (0.4,\ 0.4)$.

The output is defined as a point value at the end of simulation at coordinate $\vec{x} = (0.4,\ 0.4)$. For such a localized output, the whole mesh does not need refinement, since the output only depends on the information from a small part of the computational domain. Hence this is a reasonable test case for adaptation.

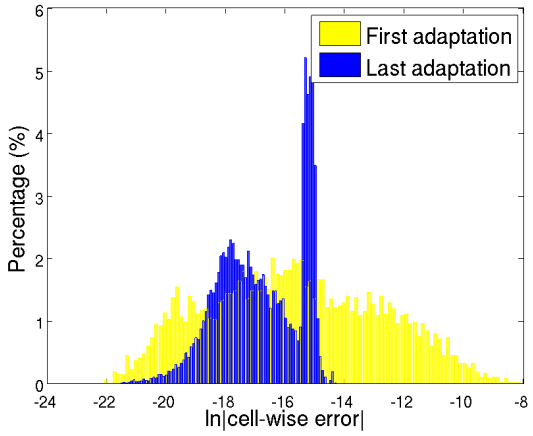
Figure 20 compares the absolute error convergence performance of conventional mesh adaptation, all five of our "adaptation acceleration" strategies, and uniform refinement. In our implementation, conventional



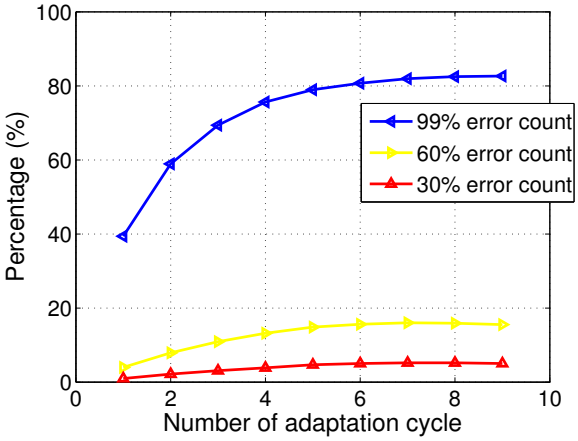
(a) Standard AWR adaptation



(b) One sub-iteration

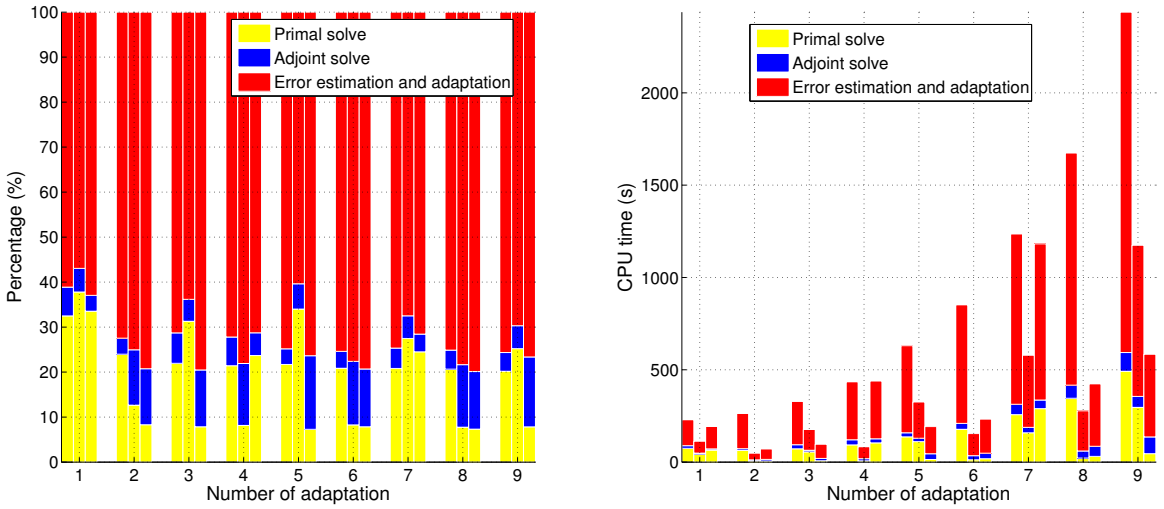


(c) Two sub-iterations



(d) % of elements holding the top 30/60/90% of the error

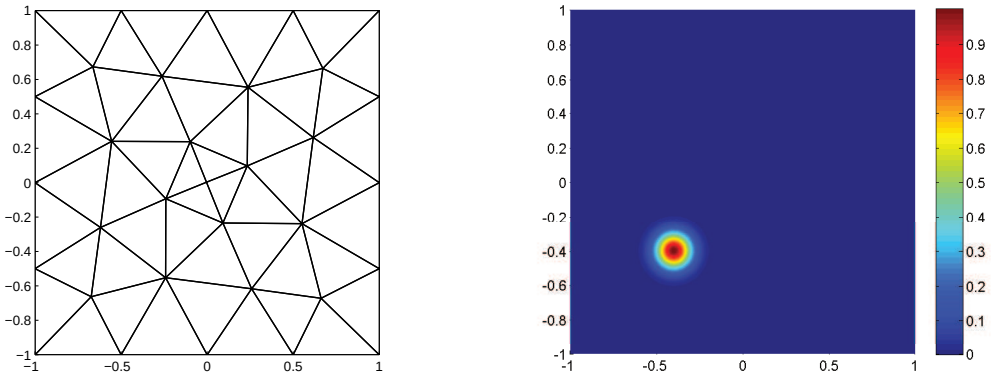
Figure 17: NACA 0012 wing, $M = 0.4$, $\alpha = 3°$: comparison of error indicator distributions.



(a) Percentage CPU time breakdown

(b) Raw CPU time breakdown

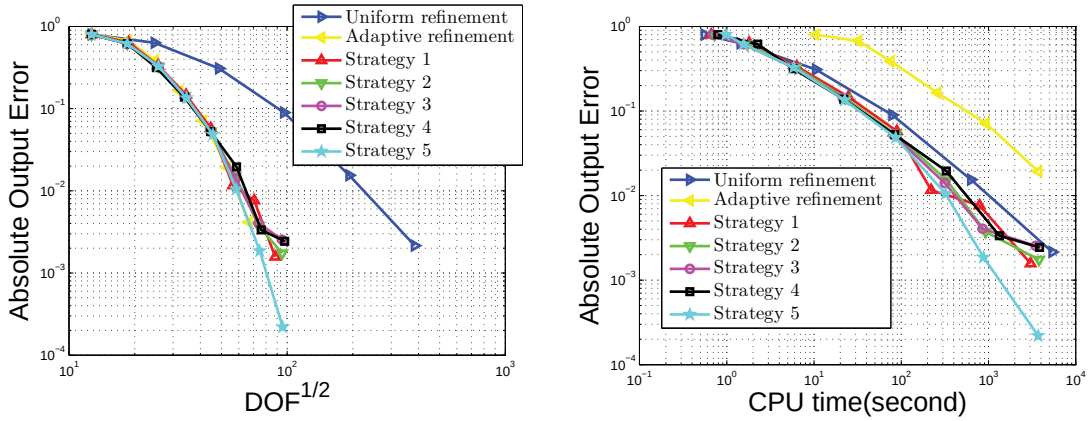
Figure 18: NACA 0012 wing, $M = 0.4$, $\alpha = 3°$: CPU time breakdown results. At each of the 9 adaptive iterations, we show three bar plots, which are, from left to right: standard adaptation, adaptation with one sub-iteration, and adaptation with two sub-iterations. Each of these bars is divided vertically into three parts, which indicate the CPU time contribution of the primal solve (yellow), the adjoint solve (blue), and the error estimation and adaptation (red). Note that the latter includes the fine-space solves.



(a) Initial mesh

(b) Primal solution

Figure 19: Initial mesh and primal solution for a scalar advection problem. The advection velocity is up and to the right.



(a) Error convergence comparison in terms of spatial degrees of freedom

(b) Error convergence comparison in terms of computational time

Figure 20: Scalar advection with active flux: error convergence comparison.

output-based adaptation is only able to outperform uniform mesh refinement when using spatial degrees of freedom to measure cost. With this measurement, we found all five curves of "adaptation acceleration" strategies in Figure 20(a) have very similar behaviours to the conventional fine-space output based mesh adaptation strategy. However, using CPU time as the cost measurement, conventional output-based mesh adaptation lags behind all of our strategies and behind uniform mesh refinement due to the simplicity of the problem, as illustrated in Figure 20(b). On the other hand, the adaptive strategies that employ coarse-space error estimates do perform better than uniform mesh refinement in CPU time. This means that we can expect much more benefit out of the adaptation mechanics developed from coarse space error estimation. Taking a look at Figure 20(a), the "adaptation acceleration" strategies almost follow the same path as the conventional output based adaptation. However, they are much cheaper to evaluate, and this could be a significant savings for complex three-dimensional problems.

Although the adaptive indicators from our "adaptation acceleration" strategies show good performance (for one problem), a natural objection to the use of a coarse space is that associated error estimates do not give us a useful measure of the actual error on the current mesh. The idea of using a fine space is that this space is closer to the infinite-dimensional space on which the exact solution typically lives. In this sense, the use of a coarse space is akin to a "retrospective" strategy, where all we see is the history of how the error evolved with adaptation and which areas were refined. While we have no *direct* foresight into how much error is left and exactly where to go next, we are currently looking at how to make use of the information in the error history to predict the remaining error on the current mesh. Specifically, we are considering a priori error estimates and extrapolation techniques, both of which we expect to be valid in the asymptotic regime when these error estimates would be necessary.

6 Conclusions

In this paper we have presented two general techniques for accelerating output-based error estimation and mesh adaptation. The first of these is the idea of sub-iterations during adaptation, where at each sub-iteration the primal and fine-space adjoint solves are done only approximately. These are usually the most expensive parts of every adaptive iteration and hence the cheaper approximate solves (block-Jacobi smoothing) yield noticeable cost savings. Performance of the adaptations relative to the standard method does not suffer as long as the current-space adjoint is still solved exactly to remove errors arising from the approximate primal solves. We demonstrated examples of using sub-iterations with the discontinuous Galerkin finite-element method for steady-state Euler simulations.

The second acceleration technique is the use of coarse spaces for creating the adaptive indicator. This retrospective error estimation strategy does not directly yield accurate error estimates for the current-space

solution, but it does provide useful information for adaptation. Using degrees of freedom as a metric, the coarse-space strategies perform similarly to the standard adjoint-weighted residual method. However, when using CPU time as the metric, the coarse-space strategies show a significant benefit. In particular, for the unsteady scalar advection case tested with the active flux method, standard adaptation could not beat uniform refinement in CPU time, whereas the coarse-space methods did.

Acknowledgments

The authors acknowledge support from the University of Michigan and the the National Aeronautics and Space Administration under grant number NNX12AJ70A.

References

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

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

[3] Timothy A. Eymann and Philip L. Roe. Active flux schemes. 49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition 2011–382, 2011.

[4] Timothy A. Eymann and Philip L. Roe. Active flux schemes for systems. 20th AIAA Computational Fluid Dynamics Conference 2011–3840, 2011.

[5] Timothy A. Eymann. *Active Flux Schemes*. PhD thesis, The University of Michigan, Ann Arbor, 2013.

[6] Timothy A. Eymann and Philip L. Roe. Multidimensional active flux schemes. 21st AIAA Computational Fluid Dynamics Conference 2011–3840, 2013-2940.

[7] Bram Van Leer. Towards the ultimate conservative difference scheme IV. a new approach to numerical convection. *Journal of Computational Physics*, 23:276–299, 1977.

[8] R. Becker and R. Rannacher. An optimal control approach to a posteriori error estimation in finite element methods. In A. Iserles, editor, *Acta Numerica*, pages 1–102. Cambridge University Press, 2001.

[9] Ralf Hartmann and Paul Houston. Adaptive discontinuous Galerkin finite element methods for the compressible Euler equations. *Journal of Computational Physics*, 183(2):508–532, 2002.

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

[11] Krzysztof J. Fidkowski. Output error estimation strategies for discontinuous Galerkin discretizations of unsteady convection-dominated flows. *International Journal for Numerical Methods in Engineering*, 88(12):1297–1322, 2011.

[12] Steven M. Kast and Krzysztof J. Fidkowski. Output-based mesh adaptation for high order Navier-Stokes simulations on deformable domains. *Journal of Computational Physics*, 252(1):468–494, 2013.

[13] Kaihua Ding, Krzysztof J. Fidkowski, and Philip L. Roe. Adjoint-based error estimation and mesh adaptation for the active flux method. AIAA Paper 2013–2942, 2013.

[14] P.-O. Persson and J. Peraire. Sub-cell shock capturing for discontinuous Galerkin methods. AIAA Paper 2006-112, 2006.

[15] F. Bassi and S. Rebay. GMRES discontinuous Galerkin solution of the compressible Navier-Stokes equations. In Karniadakis Cockburn and Shu, editors, *Discontinuous Galerkin Methods: Theory, Computation and Applications*, pages 197–208. Springer, Berlin, 2000.