Efficient Output-Based Adaptation Mechanics for High-Order Computational Fluid Dynamics Methods

by

Kaihua Ding

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Aerospace Engineering)
in The University of Michigan
2018

Doctoral Committee:

Associate Professor Krzysztof J. Fidkowski, Chair
Dr. H. T. Huynh, NASA Glenn Research Center
Associate Professor Christiane Jablonowski
Professor Philip L. Roe

Kaihua Ding

dkaihua@umich.edu

ORCID iD: 0000-0003-4749-2105

ACKNOWLEDGEMENTS

I wish to express my gratitude and appreciation to my thesis supervisor, Professor Krzysztof J. Fidkowski, for giving me the opportunity to work with him and for his guidance throughout my graduate study, without whom this work wouldn't be possible. His insightful comments and enthusiasm about the project have propelled my research forward. I would like to thank my committee members: Professor Philip L. Roe, my minor advisor and long time collaborator, who has provided valuable feedback and historic perspectives during my Ph.D. career, Dr. H. T. Huynh, who I have known since my very fist conference and who graciously agreed to be on my committee even being physically far away, Professor Christiane Jablonowski, who has provided helpful ideas regarding nodal movement solution adaptation.

I would also like to thank all my fellow CFDG lab mates, whom I feel privileged to meet and befriend, Marco Ceze, Isaac Asher, Steve Kast, Johann Dahm, Devina Sanjaya, Yukiko Shimizu, Gary Collins, Kevin Doetsch, Gustavo Halila and Vivek Ojha. Special thanks to Marco, Steve and Johann for providing me with Linux / Unix, coding assistance at the beginning stage of my Ph.D.. To my friends, who have shaped my whole graduate school experience and my life for the past five years, Tim, Chris, Chuky, Doreen, Meredith, Lawren, Lindy, Soojin, Doga, Pinar, Robin, Greg, Corey, Shardul, Candice, Rohan, Daniil, Megan, Beth, Charles, Lulu, Brad, Yali, Dong, Peter and many others I have not mentioned, thank you as well. I am forever indebted to your friendship and the wonderful memory I was able to have.

Of course, I'm thankful of my parents, Yi Ding and Xiaoyan Li, for their endless support, encouragement and love in all my academic and non-academic endeavours.

Financial support from the National Aeronautics and Space Administration under the grant number, NNX12AJ70A, and the Department of Energy, both administered by Professor Krzysztof J. Fidkowski, is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF APPENDICES

Appendix

ABSTRACT

As numerical simulations are applied to more complex and large-scale problems, solution verification becomes increasingly important in ensuring accuracy of the computed results. In addition, although improvements in computer hardware have brought expensive simulations within reach, efficiency is still paramount, especially in the context of design optimization and uncertainty quantification. This thesis addresses both of these needs through contributions to solution-based adaptive algorithms, in which the discretization is modified through a feedback of solution error estimates so as to improve the accuracy. In particular, new methods are developed for two discretizations relevant to Computational Fluid Dynamics: the Active Flux method and the discontinuous Galerkin method. For the Active Flux method, which is fully-discrete third-order discretization, both the discrete and continuous adjoint methods are derived and used to drive mesh (h) refinement and dynamic node movement, also known as r adaptation. For the discontinuous Galerkin method, which is an arbitrary-order finite-element discretization, efficiency improvements are presented for computing and using error estimates derived from the discrete adjoint, and a new r-adaptation strategy is presented for unsteady problems. For both discretizations, error estimate efficacy and adaptive efficiency improvements are shown relative to other strategies.

CHAPTER I

Introduction

1.1 Background

Numerical methods lie at the heart of modern engineering design and analysis, and they have been successfully utilized in many disciplines, including structural mechanics and fluid dynamics. As these methods become more prevalent and indispensable as engineering tools, certain shortcomings become apparent regarding their accuracy and simulation times. Specifically, most production-scale numerical methods are second-order accurate, and to achieve required engineering tolerances, these methods require very fine grids, which in turn require long simulation times. The desire to achieve sufficient accuracy and to decrease computational time has lead to the development of high-order methods [1], including spectral, discontinuous Galerkin, stabilized finite element method, flux-reconstruction, and high-order finite volume methods.

Furthermore, to elicit full benefits of high-order approximation, these methods have been coupled with adaptive meshing, driven by heuristic feature-based [2,3] and more sophisticated output-based refinement [4]. Combined with solution adaptation, high-order methods can achieve their full potential on practical cases, even those with irregular solutions, such as airfoils with nonzero angle trailing edges, turbulent

boundary layers, or shocks [5]. This thesis is devoted to the development of solution adaptation methodology for high-order methods.

1.1.1 High Order Methods

The quest for accuracy in computational fluid dynamics, especially for aerospace engineering applications, has driven the development and application of high-order methods. Various high-order methods have been developed for convection-dominated flows, including high-order finite volume, streamline-upwind Petrov-Galerkin (SUPG) finite elements [6], discontinuous Galerkin (DG) finite elements [7], and hybridized [8] and discontinuous Petrov-Galerkin methods [9].

A recent workshop [1], pitted some of these methods against traditional "work-horse" second-order finite volume schemes, and arrived at the conclusion that high-order approximation is beneficial for a variety of cases relevant to aerospace engineering.

High-order finite element method (FEM) and high order finite volume method (FVM) are among the most thoroughly researched high order methods. To ensure the accuracy of our primal discretizations, high order FEM and high order FVM will be our main focus.

High Order Finite Element Method

The Galerkin finite element methods was applied to computational fluid dynamics in the 1970s because of its advantages of a compact stencil, admissibility of high solution orders and a natural treatment for diffusion problems. However, a standard Galerkin finite element method, i.e. the continuous Galerkin method, is not stable for convection-dominated flows, which are prevalent in aerospace engineering flow applications. Two approaches are taken to tackle this instability issue. The first approach

attempts to stabilize the Galerkin method by adding a stabilization term. Methods within this category include Streamline-Upwind Petrov-Galerkin (SUPG) and Galerkin least squares (GLS). The second approach attempts to view the Galerkin method from a finite volume method perspective. This approach purposely creates discontinuities among neighbouring elements and then uses inter-element fluxes as in the finite-volume method. Methods within this category include the discontinuous Galerkin (DG) method.

GLS and SUPG are similar in their stabilization formulations, with the exception that GLS includes all of the PDE terms within the stabilization, whereas SUPG only includes the convective and temporal terms. SUPG is also regarded as a Petrov-Galerkin formulation. However, accurate yet stable Petrov-Galerkin formulations are still sought by computational fluid dynamics (CFD) researchers. Presently, the understanding of low-order SUPG methods are relatively clear after four decades of research on stabilized Galerkin methods [10, 11]. Nevertheless, as a discretization, SUPG is not yet completely mature, with issues especially in high-order stabilization and high-speed flow treatment. Admittedly, there are SUPG production codes in the CFD industry. For example, the production SUPG code, AcuSolve [12], developed at Altair Engineering Inc., is still under research development for new stabilization terms suitable for high flow speeds and high-order approximation. On the other hand, discontinuous Galerkin methods have received relatively more attention and hence are are more developed. We will therefore consider DG methods as one of our main high-order discretization tools in this thesis.

High Order Finite Volume Method

The finite volume method has always been the "work horse" of CFD. The high-order finite volume discretization used in this thesis is a newly developed finite volume

method, the Active Flux method, which is a third-order finite volume method with a multi-dimensional flux and continuous state representation [13]. The Active-Flux scheme [14, 15] is able to addresses the issue of computational cost that burdens many high-order discretizations. Active Flux schemes introduce additional independent variables, the edge values/fluxes, as independent variables. In triangles, this doubles the degrees of freedom available to describe the solution on each cell, without enlarging the stencil. In particular, quadratic reconstructions in space and time are possible, yielding formally third-order accuracy. Furthermore, the system can be marched forward in time using an inexpensive explicit solution update strategy.

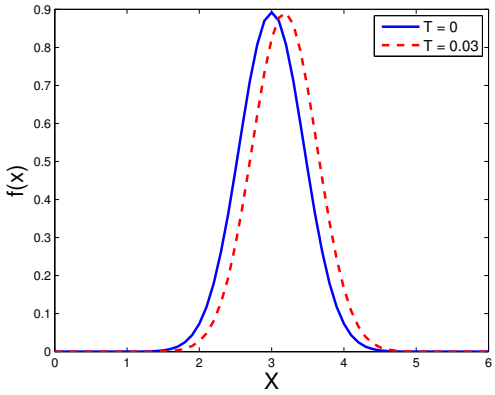
1.1.2 Solution Adaptation

Even with high order discretization tools, challenges in computational fluid dynamics remain, one of these being cost in memory and computational time. While an inexpensive high-order method is certainly desirable, such a method itself does not guarantee accuracy for a given problem. Discretization errors will still be present in the presence of non-zero mesh sizes. In this section, we therefore tackle another quest in computational fluid dynamics, that of seeking accuracy and robustness.

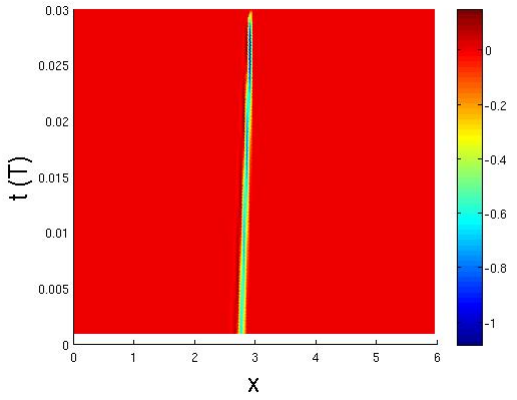
There are three main solution adaptation techniques, 1) feature-based adaptation, which utilizes a physics-related and user-defined quantity to adapt, 2) output-based adaptation, which utilizes output error sensitivity information to adapt, and 3) residual-based adaptation, which utilize residual information to adapt. Feature-based adaptation requires user knowledge about the specific physics and its definition is empirical. Thus, more often than not, feature-based adaptation's performance is case dependent without any guarantee of error reduction. On the contrary, residual-based adaptation and output-based adaptation have relatively sound theoretical support. However, residual-based adaptation is not well-suited for the convection-dominated

flow problems that pervade aerospace engineering applications.

Figure 1.1 illustrates a simple one-dimensional scalar advection problem to compare the performance of residual-based adaptation and output-based adaptation.

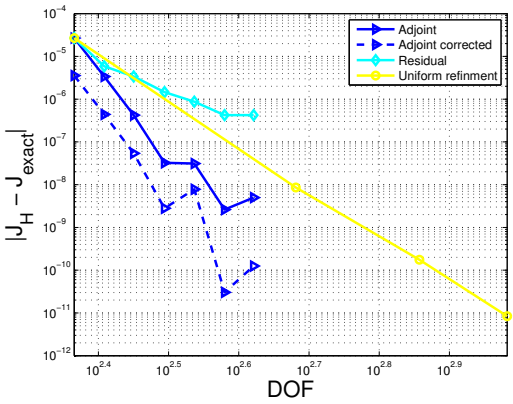


(a) Primal problem, a Gaussian profile advecting for 0.03 period



(b) Cell-avg adjoint

(c) Adapted mesh comparison



(d) Convergence curve comparison

Figure 1.1: Comparison among adjoint-based adaptation, residual-based adaptation and uniform refinement. For all adaptation mechanics, the adaptation strategy is 10% fixed fraction adaptation. Output is defined as point output at the location, $x = 2.948276$. The primal discretization is the Active Flux method.

Figure 1.1 illustrates the comparison of adjoint-based adaptation, residual-based adaptation and uniform refinement. The primal problem, Figure 1.1(a), is a Gaussian profile that advects for 0.03 of a period. According to Figure 1.1(d), the adjoint-based adaptation mechanics performs the best in dropping the absolute output error.

Figure 1.1(c) show the adapted meshes: the adjoint-based method not only refines the mesh close to the output location but also the convection trajectory. Residual-based adaptation generates a similar adapted mesh as the adjoint-based adaptation but refines the mesh in a more symmetric fashion, because the residual is large near the center of the domain for this particular Gaussian profile advection case when computing with the injected state.

The limitation of adaptation in the one-dimensional case is that for convection-dominated flows, the crucial area for accurate output easily counts for a large area of the computational domain. If the crucial area for accurate output is large, the performance will be comparable to uniform refinement. The case won't be suitable for adaptation unless the Gaussian advects for a short distance. The distance $0.03T$ is chosen in the present example. This case, Figure 1.1, is only set up to illustrate the performance differences between residual based adaptation and output based adaptation. For practical applicability, most of the research results throughout this thesis are set in two and three dimensions.

1.2 Motivation

Solution adaptation is an important scientific computing tool to ensure accuracy and to provide verification for simulation results. The solution adaptation process can be divided into two parts, 1) error estimation and localization, which has tremendously improved over the last couple of decades with the advent of adjoint-based error estimation methods; and 2) adaptation, which has been investigated on various levels, including mesh refinement, solution order-increment, and mesh-movement strategies. The second part of the solution adaptation process motivates us to create a complete picture of the adaptation process.

This thesis addresses efficient usage of adjoint-based error estimation for simulation acceleration, and adaptation, particularly via mesh motion for the Active Flux and discontinuous Galerkin methods. Three output-based adaptation algorithms are presented for the Active Flux method and the discontinuous Galerkin method: h-adaptation (mesh refinement), order-reduction-instructed h-adaptation, and r-adaptation (nodal movement adaptation). Results show h-adaptation to be more efficient than uniform mesh refinement, and in the context of h adaptation, we compare the performance and robustness of continuous and discrete adjoints. The order-reduction-instructed h-adaptation algorithm proves to be more computationally advantageous relative to regular h-adaptation. Finally, in the context of r-adaptation, we present a spring-analogy mesh motion strategy driven by an output error estimate that is able to reduce the output error by approximately a factor of two in most simulations.

Collectively, this work addressed key issues on mechanisms of output-based adaptations, particularly furthered knowledge on effective usage of adaptive indicators and output-based r-adaptation.

1.3 Thesis Overview

1.3.1 Contribution to the Active Flux Discretization

This work has three main contributions to the Active Flux method [16] [17] [18]. First, we successfully introduce a mesh motion algorithm for the Active Flux method, which is conservative and retains the scheme's third order accuracy. Second, we demonstrate that a continuous adjoint is as accurate and efficient as a discrete adjoint for adaptation purposes using h-adaptation. We show that when adjoint well-posedness and consistency are of concern, the continuous adjoint is more rigorous and robust for the Active Flux method compared to the discrete adjoint. Fi-

nally, we introduce two innovative adaptation methods: order-reduction-instructed h-adaptation and spring-analogy-based r-adaptation. Both methods are driven by adjoint solutions and are shown to be effective in reducing the output error while saving computational cost.

1.3.2 Contribution to the Discontinuous Galerkin Discretization

For the discontinuous Galerkin discretization (DG), we equip DG with adaptation acceleration techniques, an adaptive sub-iterations algorithm that leads to significant computational time savings compared to standard adaptation and commensurable computational memory savings as standard adaptation.

Output based r-adaptation is investigated for DG as well. In the r-adaptation method, we use an arbitrary Lagrangian-Eulerian framework [19–22] in order to solve a system of unsteady equations on a deforming mesh. Our mesh motion takes two forms: 1) mesh motion governed by analytical mapping functions between the reference and physical space that cause the mesh to either contract or dilate in regions where mesh resolution is needed or not, respectively; and 2) mesh motion directed by a spring analogy that indicates where resolutions is needed or not, with the motion's degrees-of-freedom equal to the number of nodes that can be moved. The finite-element discretization is applied on the static reference domain, so that the error reduction arises from the fact that the size of the mapped elements, in physical space, changes in time.

1.4 Organization

This thesis begins with the introduction of the primal and adjoint discretizations in Chapter 2. Mesh motion is introduced in Chapter 3. We show and discuss our results regarding h adaptation in Chapter 4. Then, an in-depth study on how to uti-

lize solution adaptation mechanics efficiently via adaptation acceleration techniques is presented in Chapter 5. Lastly, we present a novel output-based r-adaptation framework in Chapter 6.

CHAPTER II

Discretizations

Both the Active Flux (AF) scheme and the discontinuous Galerkin (DG) scheme are used as primary discretization tools for our output-based adaptation research. This chapter introduces both discretizations and sets the stage for the subsequent presentation of the adjoint discretization and mesh motion.

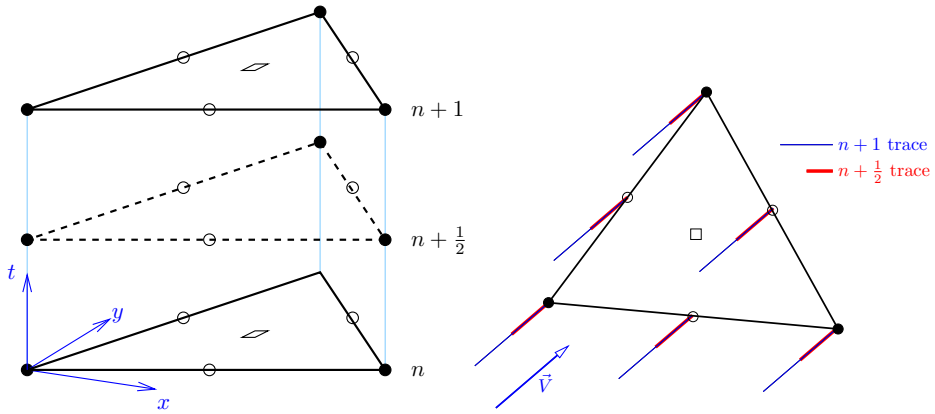
2.1 The Active Flux Method

The Active Flux method is a third-order finite volume method developed by Eymann and Roe [13–15,23,24], building on the work of van Leer [25]. The name of the active flux scheme is a direct reference to the fact that interface values are updated independently from conserved quantities [13]. In a traditional scheme, the flux at an interface is determined by the solution to a Riemann problem using reconstructions of conserved variables as the input. We refer to this type of update as a passive flux because the interface quantity is derived or interpolated from conserved quantities. An active flux is computed directly from edge values in a way that depends both on previous cell values and previous edge values.

We review the method for a first-order conservation law of the form

$$\mathbf{r}(\mathbf{u}) \equiv \frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \vec{\mathbf{F}} = \mathbf{0}, \tag{2.1}$$

where $\mathbf{u} \in \mathbb{R}^s$ is the state vector, $\vec{\mathbf{F}} \in [\mathbb{R}^s]^{\dim}$ is the flux, and $\mathbf{r}(\cdot)$ is the continuous, or strong-form residual operator. At present, we consider scalar advection, for which $s = 1$ and $\vec{\mathbf{F}} = \vec{V}\mathbf{u}$. The AF forward discretization [14,15,25] differs from traditional finite volume discretizations by defining degrees of freedom at element interfaces. The Active Flux method 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 2.1 illustrates the time levels and unknowns for one element of a triangular mesh.



(a) Time levels (b) Traces for vertex/edge updates

Figure 2.1: 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.

Seven unknowns pertain to each triangular element: one at each vertex, one at each edge midpoint, and one cell average, $\bar{u}$. The vertex and edge unknowns are not unique to the element – they are shared with the neighbors to yield a continuous solution representation. At each time level, these unknowns support an augmented quadratic spatial representation of the solution on the element. 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 2.1(b).

2. Integrate the flux on the faces. 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, on a tensor-product space-time face. We use Simpson's rule to integrate this flux, yielding 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 adjoint discretization for the Active Flux method follows the same approach but is marched backwards in time as final-time conditions are specified instead of initial conditions. In the following subsections, we present the details of the discretization in one and two spatial dimensions.

2.1.1 One-Dimensional Active Flux Discretization

In this section we present the Active Flux discretization in one dimension for the scalar advection equation:

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0,$$

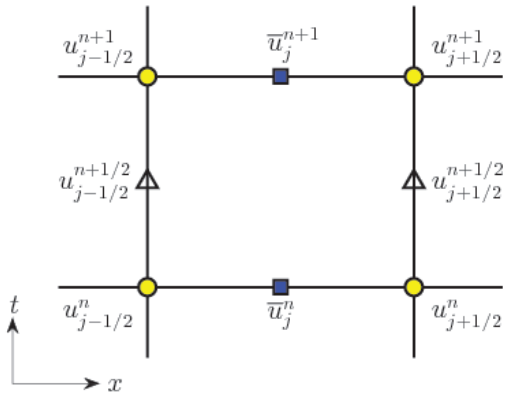


Figure 2.2: Unknowns placement in AF scheme.

where a is the advection velocity. We recall that the Active Flux discretization differs from traditional finite volume discretizations by defining degrees of freedom at element interfaces. In one spatial dimension, this just means an extra set of "vertex" unknowns at the mesh nodes, as illustrated in Figure 2.2 on a space-time diagram. These unknowns are evolved together with the cell averages through the following steps:

$$\text{Reform}: \begin{cases} u^{n+1/2}_{j+1/2} &= u^{n}_{j+1/2} \\ u^{n+1}_{j+1/2} &= u^{n}_{j+1/2} \end{cases} \tag{2.2}$$

$$\text{Vertex state pre-update:} \begin{cases} u^{n}_{j+1/2} &= S(\delta t = 0, \mathbf{c}^n_j) \\ u^{n+1/2}_{j+1/2} &= S(\delta t = 0.5\Delta t, \mathbf{c}^n_j) \\ u^{n+1}_{j+1/2} &= S(\delta t = \Delta t, \mathbf{c}^n_j) \end{cases} \tag{2.3}$$

$$\text{Flux calculation: } \overline{F}^{n+1}_{j+1/2} = \frac{1}{6}(u^{n}_{j+1/2} + 4u^{n+1/2}_{j+1/2} + u^{n+1}_{j+1/2}) \tag{2.4}$$

$$\text{Update:} \begin{cases} \overline{u}^{n+1}_{j} &= \overline{u}^{n}_{j} - \dfrac{a\Delta t}{\Delta x}(\overline{F}^{n+1}_{j+1/2} - \overline{F}^{n+1}_{j-1/2}) \\ u^{n}_{j+1/2} &= u^{n+1}_{j+1/2} \end{cases} \tag{2.5}$$

The function $S(\cdot,\cdot)$ in the pre-update step is defined as

$$S(\delta t, \mathbf{c}_j^n) = \begin{cases} R(\delta t, \mathbf{c}_j^n) & \xi_0(\delta t) \in [0,1] \\ 0 & \xi_0(\delta t) \notin [0,1] \end{cases}, \tag{2.6}$$

where $\xi_0(\delta t)$ is the reference-space location of the characteristic traced back for a time duration of δt, starting from reference coordinate ξ,

$$\xi_0(\delta t) = \xi - a\frac{\delta t}{\Delta x}. \tag{2.7}$$

The vector $\mathbf{c}_j^n$ consists of three components,

$$c_j^n : \begin{cases} c_{j,1}^n &= u_{j-1/2}^n \\ c_{j,2}^n &= \dfrac{1}{4}\left(-u_{j-1/2}^n + 6\overline{u}_j^n - u_{j+1/2}^n\right) \\ c_{j,3}^n &= u_{j+1/2}^n \end{cases}. \tag{2.8}$$

Finally, the reconstruction function $R(\cdot,\cdot)$ is given by

$$R = c_{j,1}^n\left(1-2\xi_0\right)\left(1-\xi_0\right) + 4c_{j,2}^n\xi_0\left(1-\xi_0\right) + c_{j,3}^n\xi_0\left(2\xi_0-1\right). \tag{2.9}$$

In the AF scheme, the vertex states, denoted by the vector $\mathbf{u}$ and the elemental average states, denoted by the vector $\overline{\mathbf{u}}$ are defined as two independent variables. However, the updates of $\mathbf{u}$ and $\overline{\mathbf{u}}$ are interwoven as shown above. This fact is important for the derivation of discrete adjoint formulation for the active flux method.

We assess the performance of the AF scheme by testing it with a suite of waveforms selected by Zalesak [26]: a square wave, a cosine wave, a Gaussian wave, and an elliptic wave. The Zalesak wave suite propagates across the computational domain and returns to its original position as a result of periodic boundary conditions. The solution is illustrated in Figure 2.3. The shape of the wave is qualitatively similar after one period of propagation.

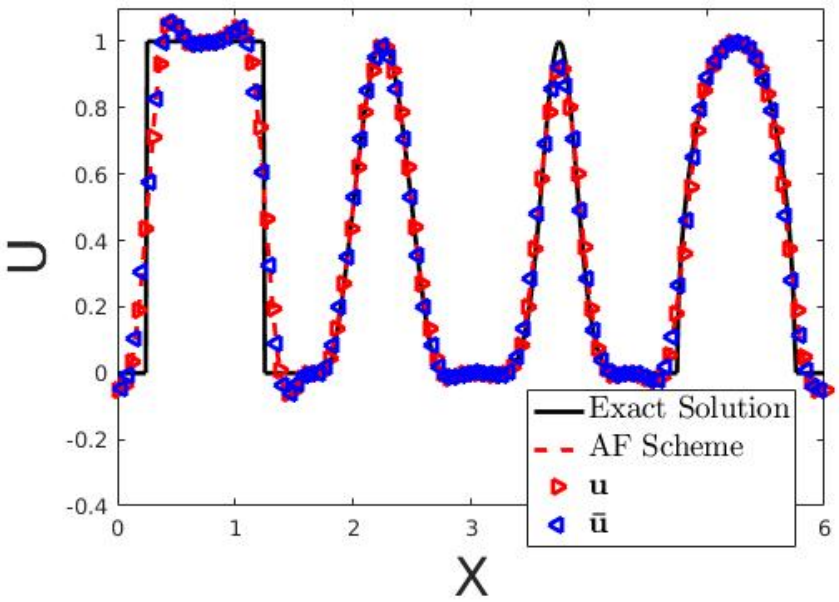


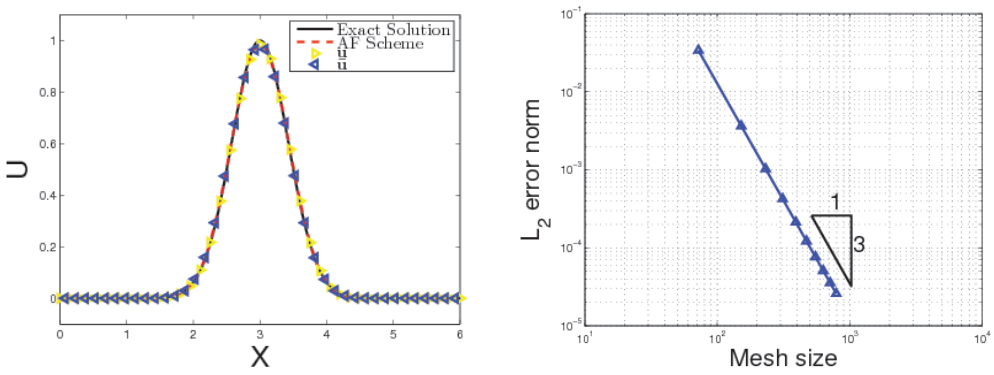
Figure 2.3: Zalesak wave suite propagation after one period with the AF scheme, with 100 cells, $CFL = 0.1$.

Next, we examine the performance of the AF scheme using a smooth initial condition: a Gaussian wave. The following L_2 solution error norm, e_{L_2}, is adopted to measure the error:

(2.10)

$$e_{L_2} = \sqrt{\frac{1}{L}\int_0^L \left[u_{\mathrm{AF}}(x) - u_{\mathrm{exact}}(x)\right]^2 dx} = \sqrt{\frac{1}{L}\sum_{j=1}^{M}\left\{\int_{(j-1)\Delta x}^{j\Delta x} \left[u_{\mathrm{AF}}(x) - u_{\mathrm{exact}}(x)\right]^2 dx\right\}}$$

Eqn. 2.10 is evaluated with a sixth-order quadrature rule. At each quadrature point, $u_{\mathrm{exact}}(x)$ is obtained by substituting x into the analytic Gaussian function and $u_{\mathrm{AF}}(x)$ is obtained by using the spatial reconstruction function in Eqn. 2.9. Figure 2.4 shows the convergence of the L_2 error norm with mesh refinement. Ignoring the pre-asymptotic first three data points in Figure 2.4(b) and applying a least squares fit to the rest of the data points with a linear function, we obtain a slope of $-2.997 \approx -3$, consistent with the third-order accuracy expected with the AF scheme.



(a) Gaussian wave test case (b) Convergence rate plot

Figure 2.4: Demonstration of third order convergence for the AF scheme in one dimension.

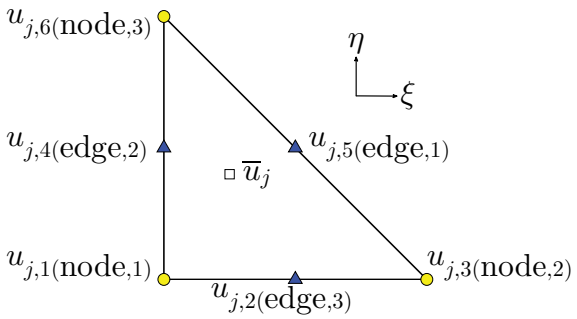


Figure 2.5: Unknown placement in the AF scheme in two-dimensions.

2.1.2 Two-Dimensional Active Flux Discretization

In two spatial dimensions, the scalar advection problem takes the form,

$$\frac{\partial u}{\partial t} + a_1 \frac{\partial u}{\partial x_1} + a_2 \frac{\partial u}{\partial x_2} = 0. \tag{2.11}$$

We consider triangular spatial elements. The unknowns now come in three varieties: node unknowns, u_{node}, edge unknowns, u_{edge}, and cell-average unknowns, $\overline{u}$. Node unknowns are placed on the mesh nodes, edge unknowns are placed at the midpoints of edges, and cell-average unknowns are associated exclusively with individual cells. Figure 2.5 illustrates this placement. The temporal evolution of states in two-dimension is similar to one dimension and is illustrated in Figure 2.1.

Edge unknown and node unknown are separated into two categories, but they

have the same evolution procedure. For this reason, we define by u_{vertex} the set of edge and node unknowns. This grouping simplifies the discrete adjoint presentation.

The two-dimensional unknowns associated with or adjacent to cell j are evolved through the following fully-discrete procedure:

$$\text{Reform}: \begin{cases} u_{j,i}^{n+1/2} &= u_{j,i}^{n} \\ u_{j,i}^{n+1} &= u_{j,i}^{n} \end{cases} \tag{2.12}$$

$$\text{Vertex state pre-update:} \begin{cases} u_{j,i}^{n} &= S(\delta t = 0, \mathbf{c}_j^n) \\ u_{j,i}^{n+1/2} &= S(\delta t = 0.5\Delta t, \mathbf{c}_j^n) \\ u_{j,i}^{n+1} &= S(\delta t = \Delta t, \mathbf{c}_j^n) \end{cases} \tag{2.13}$$

$$\text{Update:} \begin{cases} \overline{u}_j^{n+1} &= \overline{u}_j^n - \dfrac{\Delta t}{A_j} \sum_{e=1}^{3} l_e \vec{n}_e \cdot \vec{F}_e^{n+1} \\ u_{j,i}^{n} &= u_{j,i}^{n+1} \end{cases} \tag{2.14}$$

In the update equation, A_j denotes the area of cell j, and l_e is the length of edge e. The flux through a given edge e is given by an approximation to the space-time integral, using nine flux evaluations (three per edge times three time locations):

$$\begin{aligned} \text{Flux: } \vec{F}^{n+1} &= \frac{1}{9} \times \frac{1}{4} \left(\vec{f}_{\text{L}}^{n} + \vec{f}_{\text{R}}^{n} + \vec{f}_{\text{L}}^{n+1} + \vec{f}_{\text{R}}^{n+1} \right) \\ &+ \frac{1}{9} \times \left(\vec{f}_{\text{M}}^{n+1} + \vec{f}_{\text{L}}^{n+1/2} + \vec{f}_{\text{R}}^{n+1/2} + \vec{f}_{\text{M}}^{n} \right) + \frac{4}{9} \vec{f}_{\text{M}}^{n+1/2} \end{aligned} \tag{2.15}$$

The subscript, 'M', denotes the midpoint of an element edge, which is edge state ($u_2,\ u_4$ and u_5). The subscript 'L' and 'R' denote the node state to the left and to the right of the edge state ($u_1,\ u_3$ and u_6) respectively. For the advection equation, the flux at any of these points is given by $\vec{f} = \vec{a}u$.

The above update equations make use of the following definitions:

(2.16)
$$S(\delta t, \mathbf{c}_j^n) = \begin{cases} R(\delta t, \mathbf{c}_j^n) & \text{when } \{(\xi_0, \eta_0) \mid \xi_0 \in [0,1] \text{ and } \eta_0 \in [0, (1-\xi_0)]\} \\ 0 & \text{when } \{(\xi_0, \eta_0) \mid \xi_0 \notin [0,1] \text{ or } \eta_0 \notin [0, (1-\xi_0)]\} \end{cases}$$

(2.17)
$$R(\delta t, \mathbf{c}_j^n) = c_{j,1}^n \phi_{j,1}^n + c_{j,2}^n \phi_{j,2}^n + c_{j,3}^n \phi_{j,3}^n + c_{j,4}^n \phi_{j,4}^n + c_{j,5}^n \phi_{j,5}^n + c_{j,6}^n \phi_{j,6}^n + c_{j,7}^n \phi_{j,7}^n$$

(2.18)
$$\mathbf{c}_j^n : \begin{cases} c_{j,1}^n &= u_{j,1}^n \\ c_{j,2}^n &= u_{j,2}^n \\ c_{j,3}^n &= u_{j,3}^n \\ c_{j,4}^n &= u_{j,4}^n \\ c_{j,5}^n &= u_{j,5}^n \\ c_{j,6}^n &= u_{j,6}^n \\ c_{j,7}^n &= \dfrac{20}{9}\left[\overline{u}_j^n - \dfrac{1}{3}\left(u_{j,2}^n + u_{j,4}^n + u_{j,5}^n\right)\right] \end{cases}$$

(2.19)
$$\phi_{j,i}^n : \begin{cases} \phi_{j,1}^n &= 1 - 3\xi_{0,i} + 2\xi_{0,i}^2 - 3\eta_{0,i} + 4\xi_{0,i}\eta_{0,i} + 2\eta_{0,i}^2 \\ \phi_{j,2}^n &= 4\xi_{0,i} - 4\xi_{0,i}^2 - 4\xi_{0,i}\eta_{0,i} \\ \phi_{j,3}^n &= -\xi_{0,i} + 2\xi_{0,i}^2 \\ \phi_{j,4}^n &= 4\eta_{0,i} - 4\xi_{0,i}\eta_{0,i} - 4\eta_{0,i}^2 \\ \phi_{j,5}^n &= 4\eta_{0,i}\xi_{0,i} \\ \phi_{j,6}^n &= -\eta_{0,i} + 2\eta_{0,i}^2 \\ \phi_{j,7}^n &= 27\xi_{0,i}\eta_{0,i} - 27\xi_{0,i}^2\eta_{0,i} - 27\xi_{0,i}\eta_{0,i}^2 \end{cases}$$

(2.20)
$$\vec{\xi}_{0,i} = \begin{bmatrix} \xi_{0,i} \\ \eta_{0,i} \end{bmatrix} = \begin{bmatrix} \xi_i \\ \eta_i \end{bmatrix} - \underline{J}_j^{-1} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \Delta t.$$

In this equation, $[\xi_i, \eta_i]^T$ is the reference-space location of vertex i, and $\underline{J}_j$ is the reference-to-global mapping Jacobian of element j.

The L_2 error norm in two-dimensions, e_{L_2}, is

(2.21)

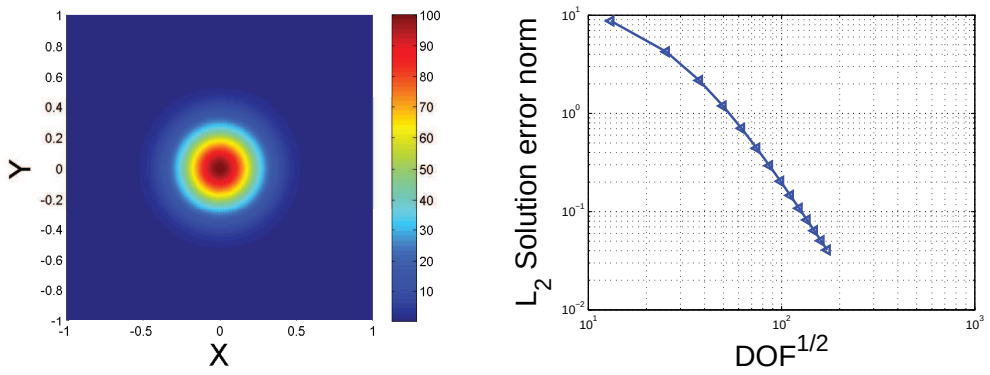
$$e_{L_2} = \sqrt{\frac{1}{A}\int_A \left[u_{\mathrm{AF}}(\mathbf{x}) - u_{\mathrm{exact}}(\mathbf{x})\right]^2 ds} = \sqrt{\frac{1}{A}\sum_{j=1}^{M}\left\{\int_{\text{Element } j} \left[u_{\mathrm{AF}}(\mathbf{x}) - u_{\mathrm{exact}}(\mathbf{x})\right]^2 ds\right\}}$$

e_{L_2} is evaluated numerically with a sixth order quadrature rule.

To assess convergence, we consider a two-dimensional Gaussian advection problem. Figure 2.6(a) shows the primal problem that we run to obtain the convergence rate curve, an isotropic Gaussian pulse advecting along the diagonal of a square domain, where periodic boundary conditions are enforced. The L_2 error norm is measured after each uniform mesh refinement, and the simulation is run for one period, with periodic boundary conditions and a velocity vector oriented at 45° relative to the (x, y) axes. The slope of the L_2 error norm convergence rate curve gradually approaches 3, as shown in Figure 2.6(b). The size of the mesh is represented by the square root of the total degrees of freedom in the mesh. Using the last two data points from Figure 2.6(b), the slope is $-2.947 \approx -3$, which is consistent with the third order accuracy expected of the active flux method.

2.2 Adjoint Discretization for the Active Flux Method

Adjoint techniques have been used in various research fields, ranging from shape optimization [27–35] to uncertainty quantification [36] and output-based error estimation [37, 38]. Adjoint methods can be classified into two formulations: discrete and continuous. In the discrete formulation, an adjoint vector is obtained by solving a linear system that results from numerically transposing the primal linearized system and driving it with the output linearization. In the continuous formulation, the



(a) Gaussian wave test case

(b) Convergence rate plot

Figure 2.6: Demonstration of third order accuracy of the Active Flux scheme in two dimensions.

adjoint is obtained by separately discretizing and solving the adjoint PDE.

The discrete adjoint approach separates the adjoint solve from the "physics" of the problem, as the adjoint system results from a purely mathematical transformation of the primal system. Furthermore, when the linearized system is already available, e.g. in an implicit primal solver, the overhead of the discrete adjoint is relatively small. However, the discrete adjoint masks some potential pitfalls, in particular in ensuring well-posedness and consistency of the adjoint. In contrast, a continuous adjoint approach explicitly addresses these points by requiring appropriate initial and boundary conditions for the adjoint.

In output-based error estimation, most approaches so far have used the discrete adjoint [32, 33, 38], though experiments with the continuous adjoint have also been attempted [39]. For light-weight solvers, such as the Active Flux method, the continuous adjoint approach may be advantageous, since linearizing and transposing the primal solver may be computationally burdensome compared to discretizing the adjoint PDE "from scratch" and applying the same light-weight solver. Nevertheless, for the purpose of comparison, in this section we present both the discrete and continuous formulations of the adjoint for the Active Flux method.

2.2.1 Discrete Adjoint for the Active Flux Discretization

A general discrete adjoint formulation of an unsteady problem reads,

$$\sum_{n=1}^{N} \left(\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^m} \right) \mathbf{\Psi}^n + \left(\frac{\partial J}{\partial \mathbf{U}^m} \right)^T = \mathbf{0}, \tag{2.22}$$

where n and m index time nodes, $\mathbf{R}^n$ is the unsteady residual at time node n, $\mathbf{\Psi}^n$ is the discrete adjoint vector at time node n, and $J(\mathbf{U}^m)$ is a scalar output. Our goal is to solve the adjoint, $\mathbf{\Psi}^n$, in Eqn. 2.22. Eqn. 2.22 is a large linear system that, when fully expanded, reads

$$\begin{bmatrix} \frac{\partial \mathbf{R}^1}{\partial \mathbf{U}^1} & \frac{\partial \mathbf{R}^1}{\partial \mathbf{U}^2} & \cdots & \frac{\partial \mathbf{R}^1}{\partial \mathbf{U}^N} \\ \frac{\partial \mathbf{R}^2}{\partial \mathbf{U}^1} & \frac{\partial \mathbf{R}^2}{\partial \mathbf{U}^2} & \cdots & \frac{\partial \mathbf{R}^2}{\partial \mathbf{U}^N} \\ \vdots & & \ddots & \vdots \\ \frac{\partial \mathbf{R}^N}{\partial \mathbf{U}^1} & \frac{\partial \mathbf{R}^N}{\partial \mathbf{U}^2} & \cdots & \frac{\partial \mathbf{R}^N}{\partial \mathbf{U}^N} \end{bmatrix}^T \begin{bmatrix} \mathbf{\Psi}^1 \\ \mathbf{\Psi}^2 \\ \vdots \\ \mathbf{\Psi}^N \end{bmatrix} + \begin{bmatrix} \left(\frac{\partial J}{\partial \mathbf{U}^1} \right)^T \\ \left(\frac{\partial J}{\partial \mathbf{U}^2} \right)^T \\ \vdots \\ \left(\frac{\partial J}{\partial \mathbf{U}^N} \right)^T \end{bmatrix} = \mathbf{0} \tag{2.23}$$

To form this system, we first need to define $\mathbf{R}$ and $\mathbf{U}$ for the AF method. In one dimension, the state vector, $\mathbf{U}$, consists of both the vertex and the cell-average unknowns,

$$\mathbf{U} = \begin{bmatrix} \mathbf{u} \\ \overline{\mathbf{u}} \end{bmatrix}. \tag{2.24}$$

In two dimensions, we group the node and edge unknowns together into one group, the vertex unknowns, $\mathbf{u}_{\text{vertex}}$, which we will also denote simply by $\mathbf{u}$. The state vector can then be written as

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_{\text{node}} \\ \mathbf{u}_{\text{edge}} \\ \overline{\mathbf{u}} \end{bmatrix} = \begin{bmatrix} \mathbf{u} \\ \overline{\mathbf{u}}. \end{bmatrix} \tag{2.25}$$

The residual, $\mathbf{R}$, is defined in a straightforward manner as

$$\mathbf{R}^{n+1} = \mathbf{U}^{n+1} - AF\left(\mathbf{U}^{n}\right), \tag{2.26}$$

where $AF(\cdot)$ is the $n \rightarrow n+1$ active flux update operator detailed in Section 2.1. Eqn. 2.26 may also be written as

$$\begin{pmatrix} \mathbf{R}^{n+1}_{\text{vertex}} \\ \mathbf{R}^{n+1}_{\text{cell avg}} \end{pmatrix} = \begin{pmatrix} \mathbf{u}^{n+1} \\ \overline{\mathbf{u}}^{n+1} \end{pmatrix} - AF \begin{pmatrix} \mathbf{u}^{n} \\ \overline{\mathbf{u}}^{n} \end{pmatrix} \tag{2.27}$$

The derivative $\partial \mathbf{R}^n / \partial \mathbf{U}^m$ yields an unsteady Jacobian matrix,

$$\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^m} = \begin{bmatrix} \dfrac{\partial \mathbf{R}^n_{\text{vertex}}}{\partial \mathbf{u}^m} & \dfrac{\partial \mathbf{R}^n_{\text{vertex}}}{\partial \overline{\mathbf{u}}^m} \\ \dfrac{\partial \mathbf{R}^n_{\text{cell avg}}}{\partial \mathbf{u}^m} & \dfrac{\partial \mathbf{R}^n_{\text{cell avg}}}{\partial \overline{\mathbf{u}}^m} \end{bmatrix}. \tag{2.28}$$

All four block components of $\partial \mathbf{R}^n / \partial \mathbf{U}^m$ can be calculated by applying the chain rule to the AF scheme update equations.

While the general unsteady Jacobian in Eqn. 2.23 appears daunting to calculate and use, most blocks are zeros. Since the definition of $\mathbf{R}^{n+1}$ only involves states at two time steps, n and $(n+1)$, the $(n+1)^{\text{th}}$ block row of $\partial \mathbf{R}^n / \partial \mathbf{U}^m$, consists of only two non-zero terms: $\partial \mathbf{R}^{n+1} / \partial \mathbf{U}^n$ and $\partial \mathbf{R}^{n+1} / \partial \mathbf{U}^{n+1}$. Thus, the adjoint equation becomes

$$\begin{bmatrix} \dfrac{\partial \mathbf{R}^1}{\partial \mathbf{U}^1} & & & \\ \dfrac{\partial \mathbf{R}^2}{\partial \mathbf{U}^1} & \dfrac{\partial \mathbf{R}^2}{\partial \mathbf{U}^2} & & \\ & & \ddots & \\ & & \dfrac{\partial \mathbf{R}^N}{\partial \mathbf{U}^{N-1}} & \dfrac{\partial \mathbf{R}^N}{\partial \mathbf{U}^N} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\Psi}^1 \\ \boldsymbol{\Psi}^2 \\ \vdots \\ \boldsymbol{\Psi}^N \end{bmatrix} + \begin{bmatrix} \left(\dfrac{\partial J}{\partial \mathbf{U}^1}\right)^T \\ \left(\dfrac{\partial J}{\partial \mathbf{U}^2}\right)^T \\ \vdots \\ \left(\dfrac{\partial J}{\partial \mathbf{U}^N}\right)^T \end{bmatrix} = \mathbf{0}. \tag{2.29}$$

Our problem narrows down to calculating the derivatives, $\partial \mathbf{R}^{n+1} / \partial \mathbf{U}^{n+1}$ and $\partial \mathbf{R}^{n+1} / \partial \mathbf{U}^{n}$. Considering first the latter,

$$\frac{\partial \mathbf{R}^{n+1}}{\partial \mathbf{U}^n} = \begin{bmatrix} \dfrac{\partial \mathbf{R}^{n+1}_{\text{vertex}}}{\partial \mathbf{u}^n} & \dfrac{\partial \mathbf{R}^{n+1}_{\text{vertex}}}{\partial \overline{\mathbf{u}}^n} \\ \dfrac{\partial \mathbf{R}^{n+1}_{\text{cell avg}}}{\partial \mathbf{u}^n} & \dfrac{\partial \mathbf{R}^{n+1}_{\text{cell avg}}}{\partial \overline{\mathbf{u}}^n} \end{bmatrix} \tag{2.30}$$

The above block sub-matrices need to be calculated individually. We show how to determine $\partial \mathbf{R}_{\text{vertex}}^{n+1}/\partial \mathbf{u}^n$ in the case of one and two spatial dimensions; other derivative matrices can be determined similarly.

One Dimensional Discrete Adjoint Formulation

For a scalar problem in one spatial dimension, the expression for $\mathbf{R}_{\text{vertex}}^{n+1}$ reads

$$\mathbf{R}_{\text{vertex}}^{n+1} = \begin{bmatrix} R_{1/2} \\ R_{1+1/2} \\ \vdots \\ R_{M+1/2} \end{bmatrix}^{n+1} = \begin{bmatrix} u_{1/2} \\ u_{1+1/2} \\ \vdots \\ u_{M+1/2} \end{bmatrix}^{n+1} - AF_{\text{vertex}}\left(\mathbf{U}^n\right), \tag{2.31}$$

where M is the number of elements. To simplify notation, we name the vertex state update process of the AF scheme as a function: $AF_{\text{vertex}}(\cdot)$. In addition, we index the vertex residuals and states by half indices. From Eqn. 2.31, the derivative $\partial \mathbf{R}_{\text{vertex}}^{n+1}/\partial \mathbf{u}^n$ is an $(M+1)\times(M+1)$ matrix,

$$\frac{\partial \mathbf{R}_{\text{vertex}}^{n+1}}{\partial \mathbf{u}^n} = \begin{bmatrix} \dfrac{\partial R_{1/2}^{n+1}}{\partial u_{1/2}^n} & \dfrac{\partial R_{1/2}^{n+1}}{\partial u_{1+1/2}^n} & \cdots & \dfrac{\partial R_{1/2}^{n+1}}{\partial u_{M+1/2}^n} \\ \dfrac{\partial R_{1+1/2}^{n+1}}{\partial u_{1/2}^n} & \dfrac{\partial R_{1+1/2}^{n+1}}{\partial u_{1+1/2}^n} & \cdots & \dfrac{\partial R_{1+1/2}^{n+1}}{\partial u_{M+1/2}^n} \\ \vdots & & \ddots & \vdots \\ \dfrac{\partial R_{M+1/2}^{n+1}}{\partial u_{1/2}^n} & \dfrac{\partial R_{M+1/2}^{n+1}}{\partial u_{1+1/2}^n} & \cdots & \dfrac{\partial R_{M+1/2}^{n+1}}{\partial u_{M+1/2}^n} \end{bmatrix} \tag{2.32}$$

Each row in Eqn. 2.32 corresponds to a derivative of $R_{j+1/2}^{n+1}$ with respect to $\mathbf{u}^n$. Recalling the discretization of the Active Flux method in Section 2.1, the $(j+1)^{\text{th}}$ row of the matrix in Eqn. 2.32 is

$$\frac{\partial R_{j+1/2}^{n+1}}{\partial \mathbf{u}^n} = \frac{\partial R_{j+1/2}^{n+1}}{\partial AF_{\text{vertex}}(\mathbf{U})^n} \frac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} \left[\frac{\partial S^n}{\partial c_{j,1}^n} \frac{\partial c_{j,1}^n}{\partial \mathbf{u}^n} + \frac{\partial S^n}{\partial c_{j,2}^n} \frac{\partial c_{j,2}^n}{\partial \mathbf{u}^n} + \frac{\partial S^n}{\partial c_{j,3}^n} \frac{\partial c_{j,3}^n}{\partial \mathbf{u}^n} \right]. \tag{2.33}$$

In the above equation, $n \in [0, 1, \cdots, N]$, $j \in [0, 1, \cdots, M]$, and

$$
\begin{aligned}
\frac{\partial R^{n+1}_{j+1/2}}{\partial A F_{\text{vertex}}(\mathbf{U}^n)} &= -1 \\
\frac{\partial A F_{\text{vertex}}(\mathbf{U}^n)}{\partial S^n} &= 1 \\
\frac{\partial S^n}{\partial c^n_{j,1}} &= (1 - 2\xi_{0,j+1/2})(1 - \xi_{0,j+1/2}) \\
\frac{\partial S^n}{\partial c^n_{j,2}} &= 4\xi_{0,j+1/2}(1 - \xi_{0,j+1/2}) \\
\frac{\partial S^n}{\partial c^n_{j,3}} &= \xi_{0,j+1/2}(2\xi_{0,j+1/2} - 1) \\
\frac{\partial c^n_{j,1}}{\partial \mathbf{u}^n} &= \left[0, \cdots, 1_{j-1/2}, \cdots, 0\right] \\
\frac{\partial c^n_{j,2}}{\partial \mathbf{u}^n} &= \left[0, \cdots, -\frac{1}{4}_{j-1/2}, -\frac{1}{4}_{j-1/2}, \cdots, 0\right] \\
\frac{\partial c^n_{j,3}}{\partial \mathbf{u}^n} &= \left[0, \cdots, 1_{j+1/2}, \cdots, 0\right]
\end{aligned}
\tag{2.34}
$$

Using a similar procedure, we obtain the rest of the components of the derivative matrices $\partial \mathbf{R}^{n+1}/\partial \mathbf{U}^n$ and $\partial \mathbf{R}^{n+1}/\partial \mathbf{U}^{n+1}$. Both $\partial \mathbf{R}^{n+1}/\partial \mathbf{U}^n$ and $\partial \mathbf{R}^{n+1}/\partial \mathbf{U}^{n+1}$ turn out to be invariant in time. Moreover, $\partial \mathbf{R}^{n+1}/\partial \mathbf{U}^{n+1}$ is an identity matrix. Let $\mathbf{A} \equiv \partial \mathbf{R}^{n+1}/\partial \mathbf{U}^n$; the unsteady discrete adjoint formulation for the active flux method then becomes

$$
\begin{bmatrix}
\mathbf{I} & \mathbf{A}^T & & & \\
 & \mathbf{I} & \mathbf{A}^T & & \\
 & & \ddots & \ddots & \\
 & & & \mathbf{I} & \mathbf{A}^T \\
 & & & & \mathbf{I}
\end{bmatrix}
\begin{bmatrix}
\boldsymbol{\Psi}^1 \\ \boldsymbol{\Psi}^2 \\ \vdots \\ \boldsymbol{\Psi}^N
\end{bmatrix}
+
\begin{bmatrix}
\left(\frac{\partial J}{\partial \mathbf{U}^1}\right)^T \\
\left(\frac{\partial J}{\partial \mathbf{U}^2}\right)^T \\
\vdots \\
\left(\frac{\partial J}{\partial \mathbf{U}^N}\right)^T
\end{bmatrix}
= \mathbf{0}.
\tag{2.35}
$$

We are especially interested in the structure of the $\mathbf{A}$ matrix, which is sparse. For the case of an $M = 20$ element mesh, the $\mathbf{A}$ matrix structure is illustrated in Figure 2.7.

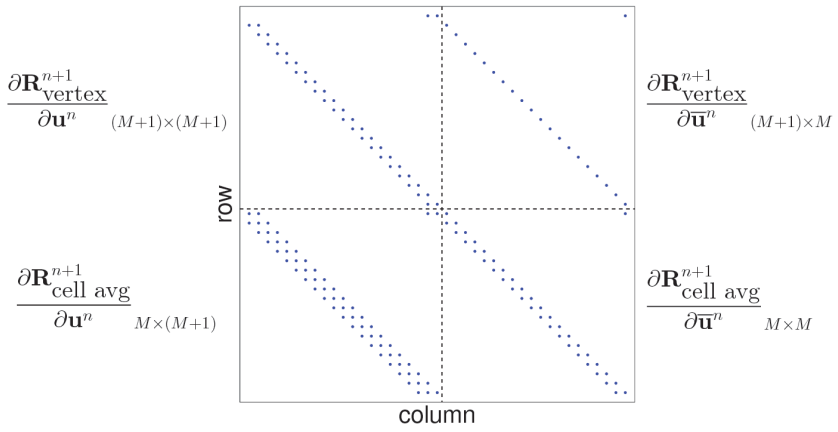


Figure 2.7: **A** matrix structure in one dimension, cell number $M = 20$ cells.

Two Dimensional Discrete Adjoint Formulation

In two spatial dimensions, let M denote the number of elements and K the number of vertices, including nodes and edges, in the computational mesh. The expression for $\mathbf{R}^{n+1}_{\text{vertex}}$ reads,

$$\mathbf{R}^{n+1}_{\text{vertex}} = \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_K \end{bmatrix}^{n+1} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_K \end{bmatrix}^{n+1} - AF_{\text{vertex}}\left(\mathbf{U}^n\right). \tag{2.36}$$

The derivative $\partial \mathbf{R}^{n+1}_{\text{vertex}}/\partial \mathbf{u}^n$ is a $K \times K$ matrix,

$$\frac{\partial \mathbf{R}^{n+1}_{\text{vertex}}}{\partial \mathbf{u}^n} = \begin{bmatrix} \frac{\partial R_1^{n+1}}{\partial u_1^n} & \frac{\partial R_1^{n+1}}{\partial u_2^n} & \cdots & \frac{\partial R_1^{n+1}}{\partial u_K^n} \\ \frac{\partial R_2^{n+1}}{\partial u_1^n} & \frac{\partial R_2^{n+1}}{\partial u_2^n} & \cdots & \frac{\partial R_2^{n+1}}{\partial u_K^n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial R_K^{n+1}}{\partial u_1^n} & \frac{\partial R_K^{n+1}}{\partial u_2^n} & \cdots & \frac{\partial R_K^{n+1}}{\partial u_K^n} \end{bmatrix}. \tag{2.37}$$

Each row in Eqn. 2.37 corresponds to a derivative of R_i^{n+1} with respect to $\mathbf{u}^n$. Recalling the discretization of the active flux method in Section 2.1, the i^{th} row of Eqn. 2.37

is the residual of the i^{th} vertex state with respect to all other vertex states. Suppose that when the i^{th} vertex state is updated, the signal, in the form of traced-back characteristics, comes from the j^{th} element. In this case, there would be at most 6 nonzero terms in the vector, $\partial {R_i}^{n+1}/\partial \mathbf{u}^n$,

(2.38)
$$\frac{\partial R_i^{n+1}}{\partial \mathbf{u}^n} = \begin{bmatrix} \cdots & \dfrac{\partial R_i^{n+1}}{\partial u_{j,1}^n} & \cdots & \dfrac{\partial R_i^{n+1}}{\partial u_{j,2}^n} & \cdots \dfrac{\partial R_i^{n+1}}{\partial u_{j,3}^n} & \cdots \dfrac{\partial R_i^{n+1}}{\partial u_{j,4}^n} & \cdots \dfrac{\partial R_i^{n+1}}{\partial u_{j,5}^n} & \cdots \dfrac{\partial R_i^{n+1}}{\partial u_{j,6}^n} & \cdots \end{bmatrix},$$

where $\cdot$ denotes zeros. By the chain rule, these derivatives evaluate to

(2.39)
$$\frac{\partial R_i^{n+1}}{\partial \mathbf{u}^n} : \begin{cases} \dfrac{\partial R_i^{n+1}}{\partial u_{j,1}^n} &= \dfrac{\partial R_i^{n+1}}{\partial AF_{\text{vertex}}(\mathbf{U})^n} \dfrac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} \dfrac{\partial S^n}{\partial c_{j,1}} \dfrac{\partial c_{j,1}^n}{\partial u_{j,1}^n} \\ \dfrac{\partial R_i^{n+1}}{\partial u_{j,2}^n} &= \dfrac{\partial R_i^{n+1}}{\partial AF_{\text{vertex}}(\mathbf{U})^n} \dfrac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} \left(\dfrac{\partial S^n}{\partial c_{j,2}} \dfrac{\partial c_{j,2}^n}{\partial u_{j,2}^n} + \dfrac{\partial S^n}{\partial c_{j,7}} \dfrac{\partial c_{j,7}^n}{\partial u_{j,2}^n} \right) \\ \dfrac{\partial R_i^{n+1}}{\partial u_{j,3}^n} &= \dfrac{\partial R_i^{n+1}}{\partial AF_{\text{vertex}}(\mathbf{U})^n} \dfrac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} \dfrac{\partial S^n}{\partial c_{j,3}} \dfrac{\partial c_{j,3}^n}{\partial u_{j,3}^n} \\ \dfrac{\partial R_i^{n+1}}{\partial u_{j,4}^n} &= \dfrac{\partial R_i^{n+1}}{\partial AF_{\text{vertex}}(\mathbf{U})^n} \dfrac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} \left(\dfrac{\partial S^n}{\partial c_{j,4}} \dfrac{\partial c_{j,4}^n}{\partial u_{j,4}^n} + \dfrac{\partial S^n}{\partial c_{j,7}} \dfrac{\partial c_{j,7}^n}{\partial u_{j,4}^n} \right) \\ \dfrac{\partial R_i^{n+1}}{\partial u_{j,5}^n} &= \dfrac{\partial R_i^{n+1}}{\partial AF_{\text{vertex}}(\mathbf{U})^n} \dfrac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} \left(\dfrac{\partial S^n}{\partial c_{j,5}} \dfrac{\partial c_{j,5}^n}{\partial u_{j,5}^n} + \dfrac{\partial S^n}{\partial c_{j,7}} \dfrac{\partial c_{j,7}^n}{\partial u_{j,5}^n} \right) \\ \dfrac{\partial R_i^{n+1}}{\partial u_{j,6}^n} &= \dfrac{\partial R_i^{n+1}}{\partial AF_{\text{vertex}}(\mathbf{U})^n} \dfrac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} \dfrac{\partial S^n}{\partial c_{j,6}} \dfrac{\partial c_{j,6}^n}{\partial u_{j,6}^n} \end{cases}$$

In the above equation,

$$
\begin{aligned}
\frac{\partial R_i^{n+1}}{\partial AF_{\text{vertex}}} &= -1 & \frac{\partial c_{j,1}}{\partial u_{j,1}} &= 1 \\
\frac{\partial AF_{\text{vertex}}(\mathbf{U})^n}{\partial S^n} &= 1 & \frac{\partial c_{j,2}}{\partial u_{j,2}} &= 1 \\
\frac{\partial S^n}{\partial c_{j,1}} &= \phi_{j,1}^n & \frac{\partial c_{j,3}}{\partial u_{j,3}} &= 1 \\
\frac{\partial S^n}{\partial c_{j,2}} &= \phi_{j,2}^n & \frac{\partial c_{j,4}}{\partial u_{j,4}} &= 1 \\
\frac{\partial S^n}{\partial c_{j,3}} &= \phi_{j,3}^n & \frac{\partial c_{j,5}}{\partial u_{j,5}} &= 1 \\
\frac{\partial S^n}{\partial c_{j,4}} &= \phi_{j,4}^n & \frac{\partial c_{j,6}}{\partial u_{j,6}} &= 1 \\
\frac{\partial S^n}{\partial c_{j,5}} &= \phi_{j,5}^n & \frac{\partial c_{j,7}}{\partial u_{j,2}} &= -\frac{20}{27} \\
\frac{\partial S^n}{\partial c_{j,6}} &= \phi_{j,6}^n & \frac{\partial c_{j,7}}{\partial u_{j,4}} &= -\frac{20}{27} \\
\frac{\partial S^n}{\partial c_{j,7}} &= \phi_{j,7}^n & \frac{\partial c_{j,7}}{\partial u_{j,5}} &= -\frac{20}{27}
\end{aligned}
\tag{2.40}
$$

Following a similar procedure, we obtain the rest of the components of the derivative $\partial \mathbf{R}^{n+1}/\partial \mathbf{U}^n$ and $\partial \mathbf{R}^{n+1}/\partial \mathbf{U}^{n+1}$. We again define $\mathbf{A} \equiv \partial \mathbf{R}^{n+1}/\partial \mathbf{U}^n$. Figure 2.8 shows the structure of the $\mathbf{A}$ matrix for one particular unstructured mesh with 48 elements and 113 nodes. Although the fill pattern is unstructured, the matrix remains sparse.

Discrete Adjoint Implementation

To solve Eqn. 2.35, we note that the coefficient matrix in Eqn. 2.35 is upper triangular, and hence, backward substitution may be applied,

$$
\begin{aligned}
\boldsymbol{\Psi}^N &= -\left(\frac{\partial J}{\partial \mathbf{U}^N}\right)^T \\
\boldsymbol{\Psi}^{N-1} + \mathbf{A}^T \boldsymbol{\Psi}^N &= -\left(\frac{\partial J}{\partial \mathbf{U}^{N-1}}\right)^T \\
&\vdots \\
\boldsymbol{\Psi}^1 + \mathbf{A}^T \boldsymbol{\Psi}^2 &= -\left(\frac{\partial J}{\partial \mathbf{U}^1}\right)^T
\end{aligned}
\tag{2.41}
$$

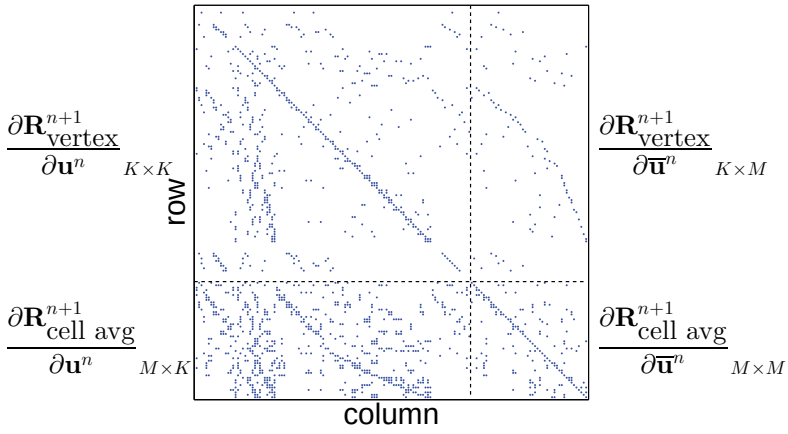


Figure 2.8: Nonzero fill pattern of the $\mathbf{A}$ matrix in two dimensions, for a mesh with $M = 48$ cells and K nodes.

Accordingly, our adjoint code employs "reverse" time marching. Since our primal problem of interest is linear, the adjoint solution does not depend on the primal solution. However, the adjoint solve is still performed after the primal solve in anticipation of nonlinear problems.

Discrete Adjoint Verification

As a verification test of the discrete adjoint implementation, we consider the one dimensional periodic transport of the initial profile shown in Figure 2.9. This linear "hat" profile was chosen for the sake of error estimation: it is exactly representable on both the coarse and the fine spaces, so that the error estimates are not polluted by varying initial conditions (which, for example, would occur for any initial condition that was not at most a piecewise quadratic).

The output J is defined as the eleventh node value ($X = 3$) at final time step. We first take a qualitative look at the adjoint solution. Figure 2.10 shows snapshots of the adjoint propagation at the first time step, i.e. $\mathbf{\Psi}^1$, the sixtieth time step, i.e. $\mathbf{\Psi}^{60}$, the one hundred and fiftieth time step, i.e. $\mathbf{\Psi}^{150}$ and the adjoint at the final

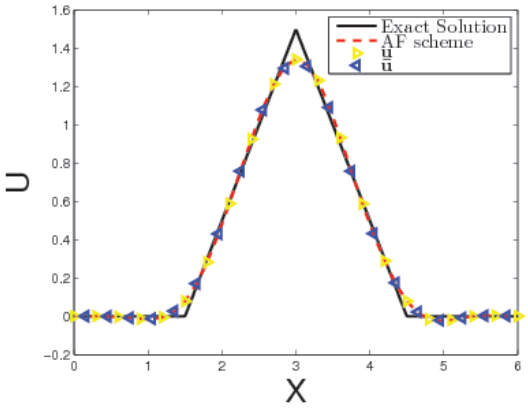
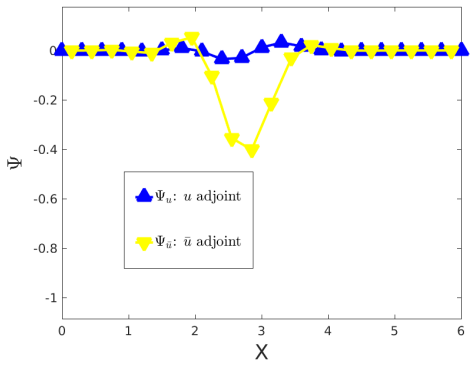
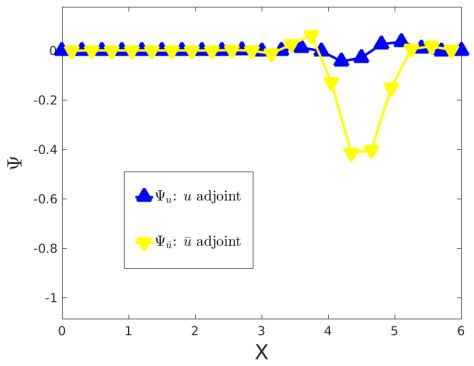


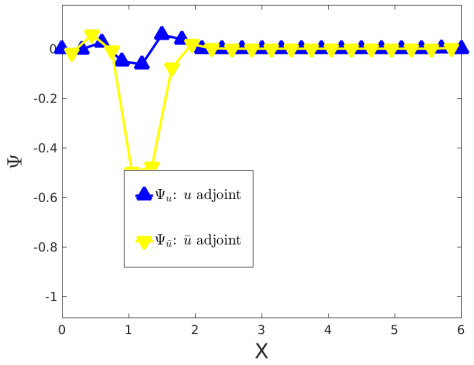
Figure 2.9: Linear hat wave function propagated for one period.



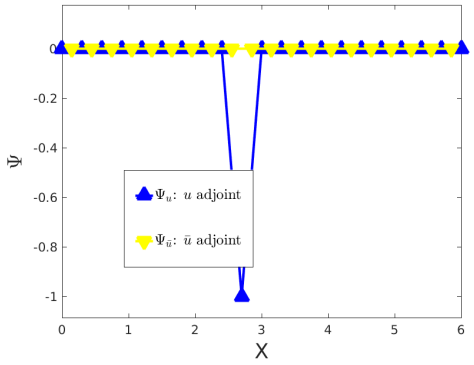
(a) Adjoint distribution at the first time step



(b) Adjoint distribution at the sixtieth time step



(c) Adjoint distribution at the one hundred and fiftieth time step



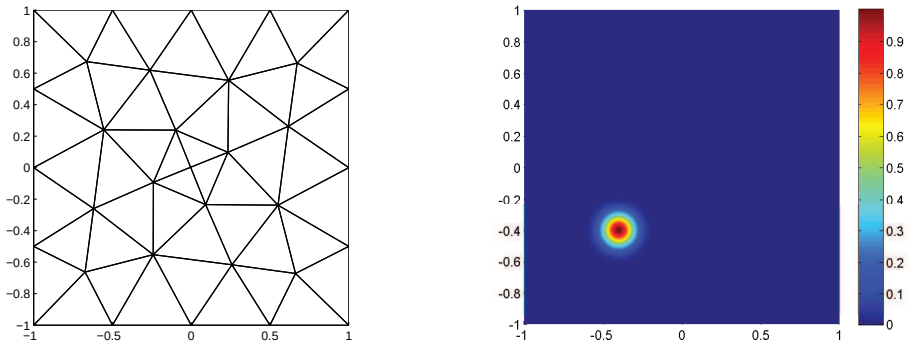
(d) Adjoint distribution at the final time step

Figure 2.10: Snapshots of adjoint propagation on our computational domain. CFL = 0.1

step, i.e. $\mathbf{\Psi}^N$.

The region of nonzero adjoint is wider at the initial time compared to the final time. In fact, at the final time, there is only one nonzero point, as expected since no other states can impact the output. At the initial time step, a residual perturbation in any point on the computational domain has a better chance of affecting the output as the state advects across the computational domain. So, the output is sensitive to a relatively larger area at the initial time step.

In two dimensions, considering the test case of a advecting Gaussian wave as shown in Figure 2.11. Figure 2.11(a) shows the initial unstructured mesh that we used. Figure 2.11(b) shows our primal problem. Here, we enforced inflow boundary condition on the left boundary and lower boundary of the computational domain. A Gaussian pulse originally centered at coordinate, $X = [-0.4,\ -0.4]$ advects along the diagonal of the square until it arrives the ending center point $X = [0.4,\ 0.4]$.



(a) Initial mesh

(b) Primal problem illustration

Figure 2.11: Initial mesh and primal problem illustration

Three types of adjoint distribution results, point output adjoint, domain integral output adjoint and line integral output adjoint are presented in Figure 2.12. Figure 2.12 is generated in the immediate fine space of the mesh shown in Figure 2.11(a).

The point output is defined at $[0.4, 0.4]$. The domain integral output is defined as

the state integral over the whole computational domain. The line integral output is the integral over upper and right boundaries. Recall the definition of the adjoint as the sensitivity of an output to residual source perturbations. For the point output, the 'sensitive area', which has a higher adjoint value, mainly concentrate around the diagonal of our computational domain as shown in Figure 2.12(a). For the domain integral output, its 'sensitive area' is the area swept over by the convective flow, which is in a rectangle shape for our case as is shown in Figure 2.12(c). For the line integral output, its 'sensitive area' is in a corner shape, which eventually develops into the corner formed by the outflow boundary, upper and right boundaries, of our computational domain, as is shown in Figure 2.12(e). Figure 2.12(a), (c) and (e) demonstrate that our adjoint implementation is correct in a qualitative sense.

For a quantitative verification of the adjoint, we consider sensitivity analysis and error estimation.

Discrete Sensitivity Analysis

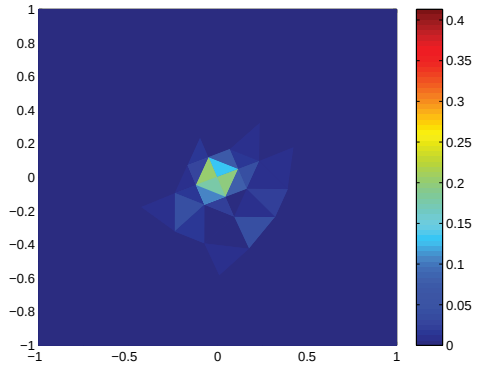
Given an unsteady adjoint solution, a parameter sensitivity is calculated as

$$\frac{dJ}{d\mu} = \sum_{n=1}^{N} \mathbf{\Psi}^n \left(\frac{\partial \mathbf{R}^n}{\partial \mu} \right) \tag{2.42}$$

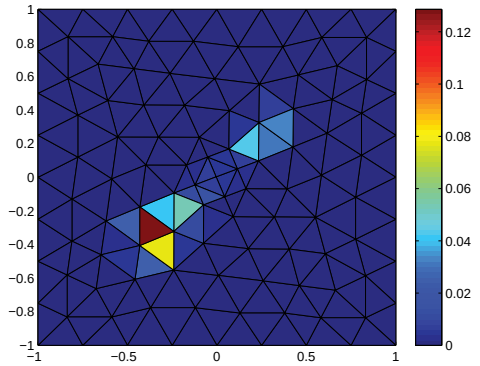
In this case, the parameter μ is chosen to be the amplitude of a spatially-sinusoidal perturbation to the initial condition. Theoretically, with the sensitivity information, we can predict the change of the output J along a change in the parameter, $\delta\mu$, without running the forward solver. This output perturbation is given by

$$\delta J = \left(\frac{dJ}{d\mu} \right) \delta\mu \tag{2.43}$$

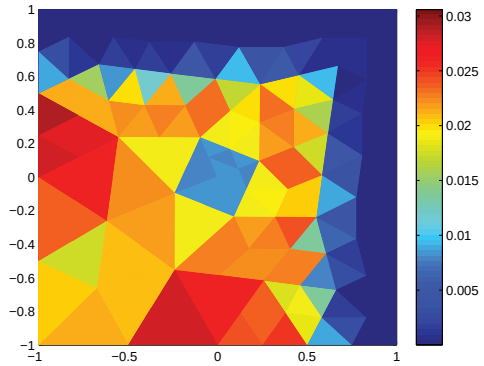
We choose to perturb state, $\mathbf{U}$, to measure the accuracy of our adjoint cases. Therefore, calculating $dJ/d\mu$ and $dJ/d\mathbf{U}$ are essentially the same. The calculation



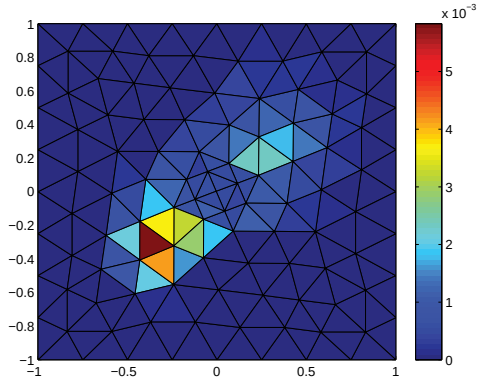
(a) Adjoint distribution of point output



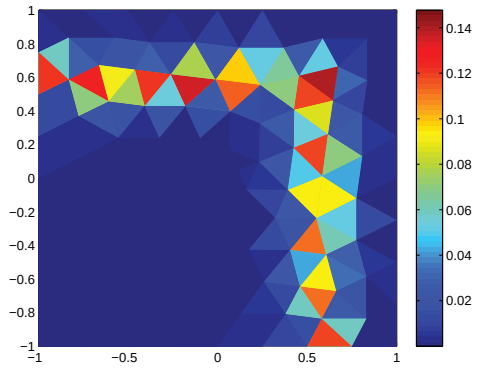
(b) Error indicator(adjointed weighted residual) of point output



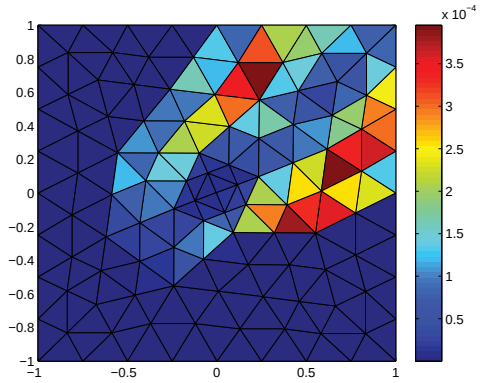
(c) Adjoint distribution of domain integral output



(d) Error indicator(adjointed weighted residual) of domain integral output



(e) Adjoint distribution of line integral output



(f) Error indicator(adjointed weighted residual) of line integral output

Figure 2.12: 2D adjoint and error indicator distribution at time step $N_h/2$

of the derivative, $dJ/d\mathbf{U}$ is done utilizing the quadratic spatial formulation implied by the active flux method. For one-dimensional case, the derivative calculation is simple. For two-dimensional case, we briefly show how to calculate derivatives of three types of output, point output, line integral output and domain integral output. Quantitative verification of sensitivity tests are also shown in this paper.

1. $dJ/d\mathbf{U}$ for Point Output

   In order to calculate derivative for point output, we need to first locate the element j, that contains the desired point output. Recall the active flux discretization from Section 2.1,

$$\frac{dJ}{d\mathbf{U}} = \Bigg[\underbrace{\cdots \phi_{1,j} \cdots \phi_{3,j} \cdots \phi_{6,j} \cdots}_{\text{node states}} \underbrace{\cdots \frac{27\phi_{2,j} - 20\phi_{7,j}}{27} \cdots \frac{27\phi_{4,j} - 20\phi_{7,j}}{27} \cdots \frac{27\phi_{5,j} - 20\phi_{7,j}}{27} \cdots}_{\text{edge states}} \underbrace{\cdots \frac{20\phi_{7,j}}{9} \cdots}_{\text{cell average states}} \Bigg] \tag{2.44}$$

2. $dJ/d\mathbf{U}$ for Line Integral Output

   Our line integral output is defined as the integral over the outflow boundary,

$$J_{\text{line integral}} = \sum_{j=1}^{\text{BCline}} J_j \tag{2.45}$$

   J_j is line integral at the boundary edge of element j, which locates on the outflow boundary. We uses 1D quadrature rule to integrate J_j numerically. Thus,

$$\frac{dJ_{\text{line integral}}}{d\mathbf{U}} = \sum_{j=1}^{\text{BCline}} \frac{dJ_j}{d\mathbf{U}} \tag{2.46}$$

and

$$\frac{dJ_j}{d\mathbf{U}} = \det(J) \sum_{q=1}^{Q} \frac{dJ_q}{d\mathbf{U}} w_q \tag{2.47}$$

w_q is the weight of 1D quadrature point evaluated at point output J_q, Q is the total number of quadrature points.

3. $dJ/d\mathbf{U}$ for Domain Integral Output

When the output is defined as the integral over the computational domain,

$$J_{\text{domain integral}} = \sum_{j=1}^{M} J_j \tag{2.48}$$

J_j is area integral over element j. We uses 2D quadrature rule to calculate it numerically.

$$\frac{dJ_{\text{domain integral}}}{d\mathbf{U}} = \sum_{j=1}^{M} \frac{dJ_j}{d\mathbf{U}} \tag{2.49}$$

and

$$\frac{dJ_j}{d\mathbf{U}} = \sum_{q=1}^{Q} \det(J) \frac{dJ_q}{d\mathbf{U}} w_q \tag{2.50}$$

w_q is the weight of the q^{th} 2D quadrature point, Q is the total number of quadrature points.

To test the adjoint-based sensitivity, we run the active flux solver twice: first without a parameter perturbation, i.e. $\mu = 0$, and second with a parameter perturbation,

Table 2.1: Initial-condition sensitivity test comparing an actual output perturbation with an adjoint-based sensitivity calculation. For the linear problem and output under consideration, the adjoint result is exact.

Output types	Actual perturbation	Predicted perturbation
1D point output	0.039062093143630	0.039062093143630
2D point output	-11.828244710966711	-11.828244710966711
2D line integral	2.243744802369696	2.243744802369697
2D domain integral	-21.666898164171524	-21.666898164171496

$\mu = \delta\mu$. The output perturbation should be predicted with the adjoint sensitivity. A comparison of the actual perturbation and the predicted perturbation for the test case under consideration is given in Table 2.1

As Table 2.1 shows, the output perturbation is predicted exactly. We can draw two conclusions here: (1) our theory for the active-flux adjoint implementation is working; (2) because the current problem is linear, the prediction is exact, but this will not be the case for general nonlinear problems.

Discrete Adjoint Error Estimation

The adjoint can be used to estimate the numerical error in an output computed using two discretization spaces. Denote by $\mathbf{U}_H$, $\mathbf{U}_h$ the primal solutions on coarse, respectively fine, spaces. These could be a mesh and a uniform refinement of it. 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 discrete fine-space adjoint, $\mathbf{\Psi}_h$, is vector of the same size as the fine-space state and residual vectors that satisfies

$$\left(\frac{\partial \mathbf{R}_h}{\partial \mathbf{U}_h}\right)^T \mathbf{\Psi}_h + \left(\frac{\partial J_h}{\partial \mathbf{U}_h}\right)^T = \mathbf{0}. \tag{2.51}$$

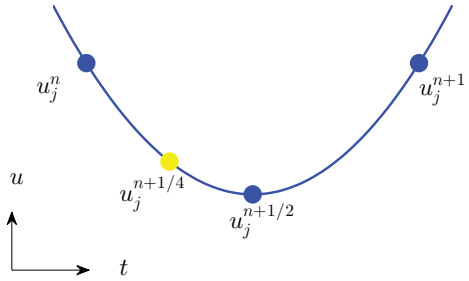


Figure 2.13: Temporal state injection within a time step.

The standard adjoint-weighted residual error estimate [32, 38] reads

$$\underbrace{J_h\left(\mathbf{U}_h^H\right) - J_h\left(\mathbf{U}_h\right)}_{\delta J} \approx \mathbf{\Psi}_h^T \delta \mathbf{R}_h = -\mathbf{\Psi}_h^T \mathbf{R}_h\left(\mathbf{U}_h^H\right). \tag{2.52}$$

In this work, we obtain the fine space (h) by subdividing both the spatial cells and the time steps of the coarse (H) discretization. That is, in one spatial dimension, the fine mesh has $M_h = 2M_H$ cells and in two spatial dimensions, the fine mesh has $M_h = 4M_H$. In both cases, the number of time steps doubles, $N_h = 2N_H$. We assess the accuracy of the error estimate through the definition of an error effectivity,

$$\eta_h = \frac{-\mathbf{\Psi}_h^T \mathbf{R}(\mathbf{U}_h^H)}{J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h)}, \tag{2.53}$$

where $\mathbf{U}_h^H$ indicates the coarse solution injected into the fine space. This injection is performed by evaluating the quadratic spatial and temporal reconstructions implied by the AF discretization. For spatial reconstruction, we can use the existing active flux basis functions. For temporal reconstruction, we first evaluate the coefficients of the temporal reconstruction, assuming the intermediate state ($u_j^{n+1/4}$) lies on a quadratic curve connecting states from adjacent time steps (u_j^n and u_j^{n+1}) through $u_j^{n+1/2}$, as illustrated in Figure 2.13.

The subscript on η_h in Eqn. 2.53 indicates that this is an effectivity measured

Table 2.2: Error estimation test

Output types	Actual error	Estimated error
1D point output	-0.062837526306201	-0.062837526306203
2D point output	-0.168407731651310	-0.168407731651310
2D line integral	-0.023801197849757	-0.023801197849757
2D domain integral	$3.226858860222309 \times 10^{-4}$	$3.226858860222372 \times 10^{-4}$

relative to the fine space. We will denote the effectivity relative to the exact solution by η, and this is calculated by replacing $J_h(\mathbf{U}_h)$ in Eqn. 2.53 with the exact output.

We now apply the adjoint to estimate the error in the output relative to a finer discretization, using the adjoint-weighted residual in Eqn. 2.52. The error is compared with actually solving the problem on the finer discretization. The comparison is presented in Table 2.2. As Table 2.2 shows, the error estimate is accurate up to machine precision. The associated effectivity is $\eta_h \approx 1$, also up to machine precision.

For one-dimensional case, we further verified the convergence rate of the absolute value of the estimated error($|\epsilon_{jH}|$) and the actual error($|J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h)|$) on a smooth problem, advection of a Gaussian wave. Figure 2.14 presents 10 pairs of data points for $|\epsilon_{jH}|$ and $|J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h)|$. Applying a least square fit to the data points in Figure 2.14, and ignoring the first 4 data points, the slope of the $|J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h)|$ data set is $-2.970697749024125 \approx -3$ and the slope of the $|\epsilon_{jH}|$ data set is $-2.96935936216698 \approx -3$, which is again consistent with the expectation of third-order accuracy for AF schemes. We note that the error estimate for an arbitrary initial condition will generally not be accurate up to machine precision according to our discussion at the beginning of Section 6.2.4 (we need exact representation on the course space).

We then consider the error indicator for adaptivity by plotting the temporally-marginalized (summed over time steps) error indicator, ϵ_{jH} from Eqn. 4.5, versus cell number in Figure 2.15. From Figure 2.15 we see that the output error indicator is

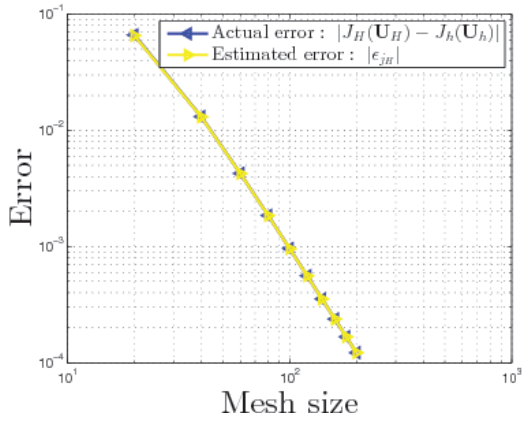


Figure 2.14: Convergence rate of $|J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h)|$ and $|\epsilon_{jH}|$ for a Gaussian wave advection problem.

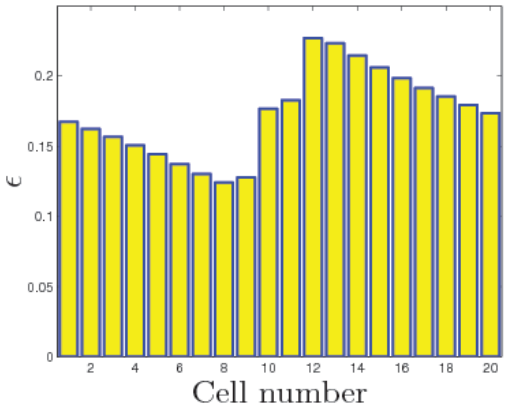


Figure 2.15: Temporally-marginalized adaptive indicator, ϵ_{jH} from Eqn. 4.5.

fairly evenly distributed over the domain, with slightly larger values near the output measurement location due to the high sensitivity of that area near later times. The uniformity of the error indicator for this simple problem is expected as in a periodic wave propagation on a static mesh, the entire domain requires resolution. Adaptive refinement is therefore not necessary in such a case, but we expect it to be important for more complex, higher-dimensional problems, i.e, two dimensions.

We presented the error indicators of two-dimensional cases in Figure 2.12(b), (d) and (f), which are calculated with the error localization strategy we came up with in Eqn. 4.6. Comparing the left hand side of Figure 2.12(Figure 2.12 (a), (c), (e)), which quantify the sensitivty informaiton, adjoint, and the right hand side of Figure 2.12(Figure 2.12(b), (d), (f)), which quantify the localized discretization error information, adjoint weighted residual, we roughly have an idea about how different the sensitivity and localized error estimation information is.

We noticed, in Figure 2.12(b), the point output error indicator distribution is rather uneven. Few elements have high error indicator values, but most elements have almost zero error indicator value. In terms of this particular output, it means some parts of our mesh have especially high discretization error and some parts of the mesh don't have much discretization error. In other words, some parts of the mesh need to be refined to reduce the discretization error, while some parts of the mesh don't need to be refined at all. We can expect adaptive mesh refinement technique to be more advantageous over uniform mesh refinement in this case. Thus, we evaluate the performance of our theoretical work using point output. As for the line integral over the outflow boundary and domain integral output, their values basically depend on the information of the whole computational domain or almost the whole computational domain. It's not necessary to use adaptive techniques for

these two cases.

2.2.2 Continuous Adjoint for the Active Flux Discretization

The continuous adjoint equations are derived from the primal problem, Eqn. 2.1 using an augmented Lagrangian, which for an unsteady simulation requires integration over the entire space-time domain,

$$\mathcal{L} \equiv J(\mathbf{u}) - \int_T \int_\Omega (\boldsymbol{\psi})^T \, \mathbf{r}(\mathbf{u}) \, d\Omega \, dt. \tag{2.54}$$

In this expression T denotes the temporal domain, Ω is the spatial domain, and $\boldsymbol{\psi} \in \mathbb{R}^s$ is the adjoint. Linearizing the Lagrangian and requiring that it is stationary with respect to permissible state variations, $\delta\mathbf{u} \in \mathcal{V}^{\text{permissible}}$, we have

$$J'[\mathbf{u}](\delta\mathbf{u}) - \int_T \int_\Omega (\boldsymbol{\psi})^T \, \mathbf{r}'[\mathbf{u}](\delta\mathbf{u}) \, d\Omega \, dt = 0 \qquad \forall \; \delta\mathbf{u} \in \mathcal{V}^{\text{permissible}}, \tag{2.55}$$

where the primes denote Fréchét linearization with respect to the arguments in square brackets. The second term can be integrated by parts to yield a partial differential equation for the adjoint equation; boundary terms from this integral then provide boundary conditions for the adjoint.

Using the case of scalar advection to showcase the derivation of the continuous adjoint, the adjoint PDE is identical to the primal one. We can see this by carrying

out the integration by parts in Eqn. 2.55,

$$\begin{aligned} J'[u](\delta u) &= \int_T \int_\Omega \psi \left[\frac{\partial(\delta u)}{\partial t} + \nabla \cdot (\vec{V}\delta u) \right] d\Omega\, dt \\ &= \int_T \int_\Omega \psi \frac{\partial(\delta u)}{\partial t}\, d\Omega\, dt + \int_T \int_\Omega \psi \nabla \cdot (\vec{V}\delta u)\, d\Omega\, dt \\ &= -\int_T \int_\Omega \frac{\partial \psi}{\partial t} \delta u\, d\Omega\, dt + \left[\int_\Omega \psi \delta u\, d\Omega \right]_0^T \\ &\quad -\int_T \int_\Omega \vec{V} \cdot \nabla \psi\, \delta u\, d\Omega\, dt + \int_T \int_{\partial\Omega} \psi \delta u \vec{V} \cdot \vec{n}\, ds\, dt \\ &= -\int_T \int_\Omega \left[\frac{\partial \psi}{\partial t} + \vec{V} \cdot \nabla \psi \right] \delta u\, d\Omega\, dt + \left[\int_\Omega \psi \delta u\, d\Omega \right]_0^T \\ &\quad + \int_T \int_{\partial\Omega} \psi \delta u \vec{V} \cdot \vec{n}\, ds\, dt \end{aligned} \tag{2.56}$$

The first term in the final right-hand side is the domain-interior adjoint equation; it may have additional source terms if the output J depends on the space-time interior state. For the sake of simplicity, we assume that the output J only depends on the state at the final time $t = T$, via

$$J(u) = \int_\Omega j(u)\, d\Omega \Big|_{t=T}.$$

Linearizing this equation and matching terms in Eqn. 2.56, we obtain the adjoint PDE,

$$\frac{\partial \psi}{\partial t} + \vec{V} \cdot \nabla \psi = 0, \tag{2.57}$$

subject to *terminal* conditions,

$$\begin{aligned} J'[u](\delta u) &= \int_\Omega j'[u](\delta u)\, d\Omega \Big|_{t=T} = \int_\Omega \psi \delta u\, d\Omega \Big|_{t=T} \\ &\Rightarrow \quad \psi(T) = j'[u]. \end{aligned} \tag{2.58}$$

We test the continuous adjoint using a scalar advection problem with periodic boundaries. Note, periodic boundaries eliminate the last term on the right-hand side

in Eqn. 2.56. To test the adjoint, we consider perturbing the initial condition, $u(0)$, by δu_0. Assuming that our adjoint ψ satisfies Eqn. 2.57 and Eqn. 2.58, Eqn. 2.56 reduces to the following expression for the expected output perturbation:

$$\delta J \equiv J'[u](\delta u_0) = -\int_\Omega \psi \, \delta u_0 \, d\Omega \Big|_{t=0}. \tag{2.59}$$

Given an approximation of the adjoint, ψ_H (what we solve for), in a finite discretization space denoted by H, the perturbation estimate becomes

$$\delta J \approx -\int_\Omega \psi_H \, \delta u_0 \, d\Omega \Big|_{t=0}. \tag{2.60}$$

Sensitivity Test in One Dimension

For the one-dimensional sensitivity test, the primal initial condition, output weight function, and state perturbation are chosen as shown in Table 2.3.

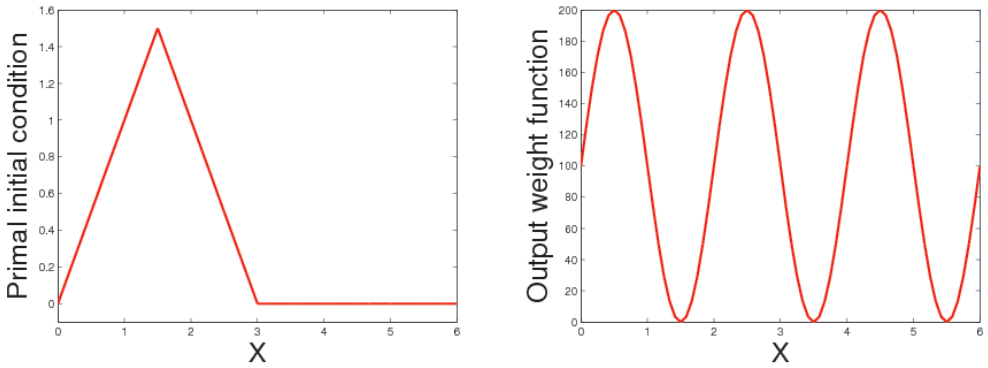
Table 2.3: Adjoint sensitivity test setup for a 1D scalar advection problem.

Primal initial condition	Output weight function	State perturbation
$u_0 = \begin{cases} x & x \in [0, 1.5) \\ 3 - x & x \in [1.5, 3) \\ 0 & x \in [3, 6] \end{cases}$	$j' = 100\,(\sin(\pi x) + 1)$	$\delta u_0 = 0.5\,(\sin(\pi x) + 1)$

Figure 2.16 plots these functions and shows the adjoint space-time field. Note that this is just the propagation of the output weight-function backwards in time from $t = T$ to $t = 0$. Computing the adjoint-based sensitivity in Eqn. 2.59, we obtain $\delta J = 445.56604\ldots$, which agrees with the actual output perturbation as the space-time mesh is refined.

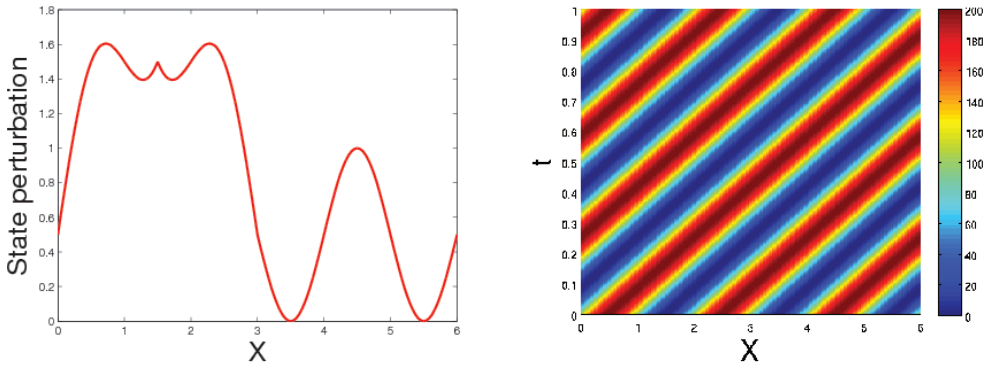
Sensitivity Test in Two Dimensions

For the sensitivity test in two dimensions, the primal problem is the same as illustrated in Figure 2.4(a). Figure 2.17 shows the primal perturbation used to calculate sensitivities. The simulation time is one period at a CFL number of 0.2.



(a) Primal initial condition

(b) Output weight function



(c) Perturbed primal initial condition

(d) Adjoint propagation in space-time

Figure 2.16: Primal and adjoint initial/terminal conditions and the adjoint space-time field for a 1D scalar advection simulation.

Note, the Courant Friedrichs Lewy (CFL) number is defined as $|\vec{V}|\Delta t/h$, where h is a measure of the smallest cell size in the mesh. For the adjoint, the output is an area integral at the final time,

$$J = \int_{\Omega} u\, w(x,y) d\Omega, \tag{2.61}$$

where the smooth weight, $w(x,y)$, which becomes the adjoint terminal condition, is

$$w(x,y) = \sin\left((x+1)\,\pi\right)\sin\left((y+1)\,\pi\right) \qquad x \in [0\ 1],\ y \in [0\ 1]. \tag{2.62}$$

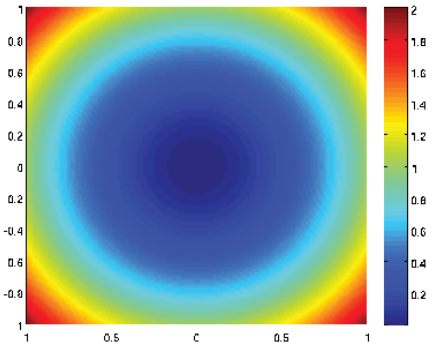


Figure 2.17: Primal perturbation used in the two-dimensional sensitivity study: $\delta u_0(x,y) = x^2 + y^2$.

Table 2.4 shows the sensitivity test result for the continuous adjoint. The error in the computed sensitivities decreases as the number of cells, i.e. resolution, increases.

	$J_H - J_{\text{perturb}}$	adjoint prediction
$N_{\text{cell}} = 450$	-7.58936×10^{-1}	-9.33519×10^{-1}
$N_{\text{cell}} = 800$	-6.73992×10^{-1}	-7.15801×10^{-1}
$N_{\text{cell}} = 1250$	-6.53913×10^{-1}	-6.62186×10^{-1}

Table 2.4: Results of the two-dimensional sensitivity test of the continuous adjoint discretization .

Error Estimation

Just like the discrete adjoint, the continuous adjoint can also be used for error estimation via an adjoint-weighted residual. Consider a "coarse" discretization level

denoted by the subscript H. Denote a space-time interpolation of the coarse-space solution by $u_H(x,t)$. The strong-form residual of this coarse-space solution will generally not be zero. Assuming linear equations, we can write,

$$r(u_H) = r(u + \delta u) = r'[u](\delta u) \neq 0, \tag{2.63}$$

where u is the exact primal solution, and $\delta u \equiv u_H - u$. From the linearized Lagrangian in Eqn. 2.55, we can compute the error in the output as a result of using u_H instead of u:

$$\delta J = \int_T \int_\Omega \psi \, r(u_H) \, d\Omega \, dt. \tag{2.64}$$

This expression requires the exact adjoint, ψ, which for general problems will be unavailable. Instead, we approximate the exact adjoint by solving it on a *fine space*, denoted by the subscript h. In the Active Flux method, this fine space is obtained by uniformly refining the space-time mesh. Substituting the fine-space adjoint into Eqn. 2.64 gives,

$$\delta J \approx \int_T \int_\Omega \psi_h \, r(u_H) \, d\Omega \, dt \tag{2.65}$$

Writing $r(u)$ in compact space-time form as $r(u) = \overline{\nabla} \cdot (\overline{V} u)$, where the bar indicates a combined space-time derivative or velocity, we can integrate the above error estimation formula by parts over the entire space-time domain, denoted by $\Omega\!\!I$, to obtain,

$$\begin{aligned} \delta J &\approx \int_{\Omega\!\!I} \psi_h \, r(u_H) \, d\Omega\!\!I \\ &= \underbrace{-\int_{\Omega\!\!I} \overline{\nabla} \psi_h \cdot \overline{V} u_H \, d\Omega\!\!I}_{\text{0 by the adjoint eqn.}} + \int_{\partial\Omega\!\!I} \psi_h (\overline{V} u_H) \cdot \overline{n} \, dS. \end{aligned} \tag{2.66}$$

This surface-integral error estimate is applied on each space-time cell of the fine-space mesh. Since continuous adjoint sensitivities will not exactly match the true

sensitivities, we verify the continuous adjoint through error estimates. In particular we perform an error estimate convergence study to verify our continuous adjoint in Chapter 3.

2.3 Discontinuous Galerkin Discretization

In this section, we review the discontinuous Galerkin method for conservation laws. Additional details can be found in previous works [40, 41].

2.3.1 Conservation Equations

Consider a conservation law given by the partial differential equation (PDE)

$$\partial_t \mathbf{u} + \partial_i \mathbf{H}_i(\mathbf{u}, \nabla \mathbf{u}) = \mathbf{0}, \tag{2.67}$$

where $\mathbf{u} \in \mathbb{R}^s$ is the state vector, $\mathbf{H}_i \in \mathbb{R}^s$ is the i^{th} component of the total flux, $1 \leq i \leq d$ indexes the spatial dimension d, and summation is implied on the repeated index i. We decompose the total flux into convective and diffusive parts,

$$\mathbf{H}_i = \mathbf{F}_i(\mathbf{u}) + \mathbf{G}_i(\mathbf{u}, \nabla \mathbf{u}), \tag{2.68}$$

$$\mathbf{G}_i(\mathbf{u}, \nabla \mathbf{u}) = -\mathbf{K}_{ij}(\mathbf{u})\, \partial_j \mathbf{u}, \tag{2.69}$$

where $\mathbf{F}_i \in \mathbb{R}^s$ is the i^{th} component of the inviscid/convective flux, $\mathbf{G}_i \in \mathbb{R}^s$ is the i^{th} component of the viscous flux, and $\mathbf{K}_{ij} \in \mathbb{R}^{s \times s}$ is the (i, j) component of the viscous diffusivity tensor. For steady problems, the temporal derivative is zero, $\partial_t \mathbf{u} = \mathbf{0}$.

2.3.2 Solution Approximation

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 the set of N_e elements in a non-overlapping tessellation

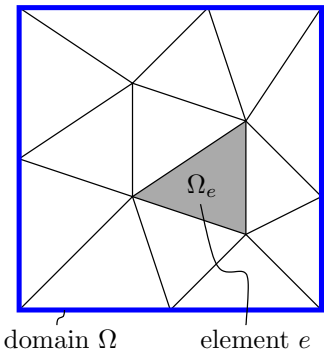


Figure 2.18: Partition of a square domain into 14 triangular elements.

of the domain Ω, as illustrated in Figure 2.18.

$$\mathbf{u}_h(\vec{x}) \approx \sum_{e=1}^{N_e} \sum_{n=1}^{N_p} \mathbf{U}_{en} \phi_{en}(\vec{x}), \tag{2.70}$$

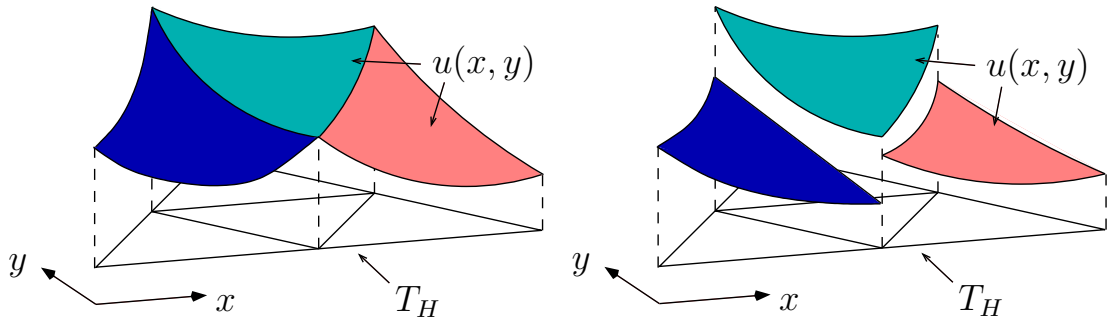
where

$$\begin{aligned}
N_p &= \text{number of basis functions needed for an order } p \text{ approximation} \\
\phi_{en}(\vec{x}) &= n^{\text{th}} \text{ order } p \text{ basis function on element } e \text{ (zero on all other elements)} \\
p &= \text{order of spatial basis functions on each element} \\
N_e &= \text{number of elements} \\
\mathbf{U}_{en} &= \text{vector of } s \text{ coefficients on } n^{\text{th}} \text{ basis function on element } e
\end{aligned}$$

Formally, we can write that $\mathbf{u}_h \in \boldsymbol{\mathcal{V}}_h = [\mathcal{V}_h]^s$, 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 element e. The lack of continuity in the solution approximation differentiates DG from continuous finite element methods, as illustrated in Figure 2.19. It adds computational expense, as we do not expect to approximate solutions that are discontinuous at every element interface, but it also provides convective stability and simplifies hanging-node mesh refinement and local order enrichment.



(a) Continuous Galerkin (b) Discontinuous Galerkin

Figure 2.19: Solution approximation using continuous and discontinuous basis functions. Though the solution is discontinuous in DG methods, the inter-element flux is single valued, as in finite volume methods.

2.3.3 Weak Form

We obtain a weak form of Eqn. 2.67 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, but the diffusive treatment is trickier and requires stabilization. Multiple diffusion formulations exist [42], and we employ the second form of Bassi and Rebay (BR2) [43].

We can write the final semilinear weak form as

$$\mathcal{R}_h(\mathbf{u}_h, \mathbf{v}_h) = 0, \quad \forall \mathbf{v}_h \in \boldsymbol{\mathcal{V}}_h, \tag{2.71}$$

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

$$\mathcal{R}_h(\mathbf{u}_h, \mathbf{v}_h) = \sum_{e=1}^{N_e} \mathcal{R}_h(\mathbf{u}_h, \mathbf{v}_h|_{\Omega_e}) = 0, \quad \forall \mathbf{v}_h \in \boldsymbol{\mathcal{V}}_h. \tag{2.72}$$

Integrating by parts and applying the BR2 diffusion treatment, we find that the

semilinear form associated with each element is

$$\begin{aligned}\mathcal{R}_h(\mathbf{u}_h, \mathbf{v}_h|_{\Omega_e}) &= \int_{\Omega_e} \mathbf{v}_h^T \partial_t \mathbf{u}_h \, d\Omega - \int_{\Omega_e} \partial_i \mathbf{v}_h^T \mathbf{H}_i \, d\Omega \\ &+ \int_{\partial\Omega_e \setminus \partial\Omega} \mathbf{v}_h^{+T} \left(\widehat{\mathbf{F}} + \widehat{\mathbf{G}} \right) \, ds + \int_{\partial\Omega_e \cup \partial\Omega} \mathbf{v}_h^{+T} \left(\widehat{\mathbf{F}}^b + \widehat{\mathbf{G}}^b \right) \, ds \\ &- \int_{\partial\Omega_e \setminus \partial\Omega} \partial_i \mathbf{v}_h^{+T} \mathbf{K}_{ij}^+ \left(\mathbf{u}_h^+ - \widehat{\mathbf{u}}_h \right) n_j \, ds \\ &- \int_{\partial\Omega_e \cup \partial\Omega} \partial_i \mathbf{v}_h^{+T} \mathbf{K}_{ij}^b \left(\mathbf{u}_h^+ - \mathbf{u}_h^b \right) n_j \, ds \end{aligned} \tag{2.73}$$

where $(\cdot)^T$ denotes transpose, and on the element boundary $\partial\Omega_e$ the notations $(\cdot)^+, (\cdot)^-, (\cdot)^b$ respectively denote quantities taken from the element interior, neighbor element, and boundary. The last two terms symmetrize the semilinear form for adjoint consistency. The key fluxes are defined as,

$$\begin{aligned}\widehat{\mathbf{u}}_h &= (\mathbf{u}_h^+ + \mathbf{u}_h^-)/2 \\ \widehat{\mathbf{F}} &= \widehat{\mathbf{F}}(\mathbf{u}_h^+, \mathbf{u}_h^-, \vec{n}) \\ \widehat{\mathbf{G}} &= \widehat{\mathbf{G}}(\mathbf{u}_h^+, \mathbf{u}_h^-, \nabla\mathbf{u}_h^+, \nabla\mathbf{u}_h^-, \vec{n}) \\ \widehat{\mathbf{F}}^b &= \widehat{\mathbf{F}}^b(\mathbf{u}_h^b, \mathrm{BC}, \vec{n}) \\ \widehat{\mathbf{G}}^b &= \widehat{\mathbf{G}}^b(\mathbf{u}_h^b, \nabla\mathbf{u}_h^+, \mathrm{BC}, \vec{n})\end{aligned}$$

In particular, on an interior face σ^f, the convective flux $\widehat{\mathbf{F}}$ is computed using the Roe approximate Riemann solver [24], and the BR2-stabilized viscous flux is

$$\widehat{\mathbf{G}} = \frac{1}{2} \left(\mathbf{G}_i^+ + \mathbf{G}_i^- \right) n_i^+ + \eta \frac{1}{2} \left(\boldsymbol{\delta}_i^+ + \boldsymbol{\delta}_i^- \right) n_i^+, \tag{2.74}$$

where the auxiliary variables $\boldsymbol{\delta}_i^+, \boldsymbol{\delta}_i^- \in [\boldsymbol{\mathcal{V}}_h]^d$ have support on the two elements adjacent to the interior face σ^f and are obtained by solving $\forall \boldsymbol{\tau}_{hi} \in \boldsymbol{\mathcal{V}}_h$

$$\int_{\Omega_e^+} \boldsymbol{\tau}_{hi}^T \boldsymbol{\delta}_i^+ \, d\Omega + \int_{\Omega_e^-} \boldsymbol{\tau}_{hi}^T \boldsymbol{\delta}_i^- \, d\Omega = \int_{\sigma^f} \frac{1}{2} \left(\boldsymbol{\tau}_{hi}^{+T} \boldsymbol{K}_{ij}^+ + \boldsymbol{\tau}_{hi}^{-T} \boldsymbol{K}_{ij}^- \right) (\mathbf{u}_h^+ - \mathbf{u}_h^-) n_j^+ \, ds. \tag{2.75}$$

η is a stabilization factor that should not be less than the number of faces per element, and in our work is taken to be (at least) twice the maximum number of faces on adjacent elements.

On a boundary face σ^b, fluxes are typically computed directly from the boundary state, $\mathbf{u}^b$, which is a function (projection) of the interior state and the boundary-condition data, $\mathbf{u}_h^b = \mathbf{u}_h^b(\mathbf{u}_h^+, \mathrm{BC})$. One exception is the convective flux for a boundary condition in which a complete exterior state is specified: in such a case, an approximate-Riemann solver is used to compute the boundary convective flux, $\widehat{\mathbf{F}}^b$. The BR2-stabilized boundary viscous flux is

$$\widehat{\mathbf{G}}^b = \Pi_G^{\mathrm{BC}} \left[\mathbf{G}_i(\mathbf{u}_h^b, \nabla \mathbf{u}_h^+) n_i + \eta \boldsymbol{\delta}_i n_i \right], \tag{2.76}$$

where the auxiliary variable $\vec{\boldsymbol{\delta}} \in [\boldsymbol{\mathcal{V}}_h]^d$ has support on the element adjacent to the boundary face σ^b and is obtained by solving $\forall \boldsymbol{\tau}_{hi} \in \boldsymbol{\mathcal{V}}_h$

$$\int_{\Omega_e} \boldsymbol{\tau}_{hi}^T \boldsymbol{\delta}_i \, d\Omega \;=\; \int_{\sigma^b} \boldsymbol{\tau}_{hi}^{+T} \boldsymbol{K}_{ij}^b (\mathbf{u}_h^+ - \mathbf{u}_h^b) n_j^+ \, ds. \tag{2.77}$$

The projection Π_G^{BC} in Eqn. 2.76 incorporates boundary conditions on the viscous flux, such as a prescribed heat flux in the compressible Navier-Stokes equations.

2.3.4 Discrete System

We choose as test functions the trial basis functions introduced in Eqn. 2.70, ϕ_{en}, where

$$\mathrm{span}\,\{\phi_{en}\} = \mathcal{V}_h. \tag{2.78}$$

Recall that e is the element number and n indexes the polynomial basis functions inside element e. Each state equation is tested with the same set of functions, which means that we can define a size-s residual vector for the n^{th} test function in element

e by

$$\mathbf{R}_{en} \equiv \{\mathcal{R}_h(\mathbf{u}_h, \phi_{en}\mathbf{e}_r)\}_{r=1\ldots s} \in \mathbb{R}^s, \tag{2.79}$$

where $\mathbf{e}_r \in \mathbb{R}^s$, $r = 1 \ldots s$, is a vector of all zeros except a 1 in position r. We now use the convention that dropping a subscript means considering the set of values over the entire range of that subscript. So $\mathbf{R}_e$ is the set of residuals over all states and basis functions inside element e, and $\mathbf{R}$ is the set of all residuals for all elements in the domain. Using the same convention for the state, $\mathbf{U}$, we can write the discrete system of equations (i.e. residuals) compactly as

$$\mathbf{R}(\mathbf{U}) = \mathbf{0}. \tag{2.80}$$

Note that both $\mathbf{U}$ and $\mathbf{R}$ lie in $\mathbb{R}^N$, where the total number of degrees of freedom including equation states is $N = N_e N_p s$. When considering different discretization spaces, we will append a subscript h or H to the variables $\mathbf{R}$, $\mathbf{U}$, and N.

Finally, the number of basis functions per element depends on the approximation space and order p. We consider approximation spaces that are given by the span of monomials in reference space coordinates ξ_i, $\Pi_{i=1}^d \xi_i^{p_i}$: for a full-order space, $\sum_{i=1}^d p_i \leq p$, and for a tensor-product space, $\forall i, p_i \leq p$. The resulting dimensions of these spaces are

$$\text{full-order basis set: } N_p = \binom{p}{d} = \frac{p!}{d!(p-d)!} \qquad \text{tensor-product basis set: } N_p = (p+1)^d.$$

2.3.5 Nonlinear Solver

To solve Eqn. 2.80, we use a preconditioned Newton-Krylov method augmented with pseudo-transient continuation. Starting from an initial guess, usually the free-stream condition, and a conservative time step set using a Courant-Friedrichs-Lewy

(CFL) number of approximately unity, the Newton solver iterates until steady state. The linear system at each Newton iteration is solved using an element-line preconditioned General Minimal Residual Krylov subspace method [44, 45]. As successful Newton updates are taken, the time step is increased according to an exponential growth formula for the CFL number. Details on the solver, including the incorporation of physical constraints on some of the variables, can be found in [46]. For unsteady problems and implicit time marching, the same procedure is applied at every time step or stage.

2.3.6 Discrete Adjoint

Suppose that we are interested in a scalar output computed from the solution to our PDE. Using the discrete state vector, we write

$$\text{output } J \;=\; J(\mathbf{U}). \tag{2.81}$$

The discrete adjoint, $\mathbf{\Psi} \in \mathbb{R}^N$, is a vector of sensitivities of the output to the N residuals. That is, each entry of the adjoint tells us the effect that a perturbation in the same entry in the residual vector would have on the output J. One common source of residual perturbations is changes in input parameters for a problem, and so we ground the adjoint presentation in the context of a local sensitivity analysis.

2.3.7 Local Sensitivity Analysis

Consider a situation in which N_μ parameters, $\boldsymbol{\mu} \in \mathbb{R}^{N_\mu}$, affect the PDE in Eqn. 2.80. We can then write the following chain of dependence,

$$\underbrace{\boldsymbol{\mu}}_{\text{inputs} \in \mathbb{R}^{N_\mu}} \rightarrow \underbrace{\mathbf{R}(\mathbf{U}, \boldsymbol{\mu}) = 0}_{N \text{ equations}} \rightarrow \underbrace{\mathbf{U}}_{\text{state} \in \mathbb{R}^N} \rightarrow \underbrace{J(\mathbf{U})}_{\text{output (scalar)}} . \tag{2.82}$$

We are interested in how J changes (locally for nonlinear problems) with $\boldsymbol{\mu}$,

$$\frac{dJ}{d\boldsymbol{\mu}} \in \mathbb{R}^{1 \times N_\mu} = N_\mu \text{ sensitivities.} \tag{2.83}$$

Note, if J depends directly on $\boldsymbol{\mu}$ we would add $\frac{\partial J}{\partial \boldsymbol{\mu}}$ to the above sensitivity, but for clarity of presentation we consider only the case when $J = J(\mathbf{U})$. Several options exist for computing these sensitivities. Two direct ones are finite differencing, in which the input parameters are perturbed one at a time, and forward linearization, in which the sequence of operations in Eqn. 2.82 is linearized. Both of these become expensive when N_μ is moderate or large, because of the need to re-solve the system $\mathbf{R}(\mathbf{U}, \boldsymbol{\mu}) = 0$ for each parameter. A third choice is the adjoint approach, which requires an inexpensive residual perturbation calculation followed by an adjoint weighting to compute the effect on the output. That is, we write

$$\frac{dJ}{d\boldsymbol{\mu}} = \boldsymbol{\Psi}^T \frac{\partial \mathbf{R}}{\partial \boldsymbol{\mu}}. \tag{2.84}$$

This approach is efficient for computing a large number of sensitivities for one output, as the cost is one residual perturbation calculation and one vector product per sensitivity.

The central idea in the adjoint approach is that we do not need to solve the forward problem each time we want a sensitivity. The adjoint method precomputes the effect of $\mathbf{R}$ on J, which is the expensive step. The resulting N sensitivities are stored in the vector $\boldsymbol{\Psi}$.

2.3.8 The Adjoint System

To derive an equation for the adjoint, we consider the chain of operations we would take in computing the sensitivities via a direct approach. In the following steps, we assume small perturbations.

1. Input $\quad \boldsymbol{\mu} \to \boldsymbol{\mu} + \delta\boldsymbol{\mu}$
2. Residual $\quad \mathbf{R}(\mathbf{U}, \boldsymbol{\mu} + \delta\boldsymbol{\mu}) = \delta\mathbf{R} \neq 0 \quad \to \quad \mathbf{R}(\mathbf{U}, \boldsymbol{\mu}) + \frac{\partial \mathbf{R}}{\partial \boldsymbol{\mu}}\Big|_{\mathbf{U},\boldsymbol{\mu}} \delta\boldsymbol{\mu} = \delta\mathbf{R}$
3. State $\quad \mathbf{R}(\mathbf{U} + \delta\mathbf{U}, \boldsymbol{\mu} + \delta\boldsymbol{\mu}) = 0 \quad \to \quad \mathbf{R}(\mathbf{U}, \boldsymbol{\mu}) + \frac{\partial \mathbf{R}}{\partial \boldsymbol{\mu}}\Big|_{\mathbf{U},\boldsymbol{\mu}} \delta\boldsymbol{\mu} + \frac{\partial \mathbf{R}}{\partial \mathbf{U}}\Big|_{\mathbf{U},\boldsymbol{\mu}} \delta\mathbf{U} = 0$
4. Output $\quad J(\mathbf{U} + \delta\mathbf{U}) = J(\mathbf{U}) + \delta J \quad \to \quad \delta J = \frac{\partial J}{\partial \mathbf{U}} \delta\mathbf{U}$

Subtracting step 2 from step 3, we obtain

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}}\Big|_{\mathbf{U},\boldsymbol{\mu}} \delta\mathbf{U} = -\delta\mathbf{R} \quad \Rightarrow \quad \delta\mathbf{U} = -\left[\frac{\partial \mathbf{R}}{\partial \mathbf{U}}\right]^{-1} \delta\mathbf{R}. \tag{2.85}$$

Combining this result with the output linearization in step 4 gives the output perturbation in terms of the residual perturbation,

$$\delta J = \frac{\partial J}{\partial \mathbf{U}} \delta\mathbf{U} = \underbrace{-\frac{\partial J}{\partial \mathbf{U}} \left[\frac{\partial \mathbf{R}}{\partial \mathbf{U}}\right]^{-1}}_{\boldsymbol{\Psi}^T \in \mathbb{R}^N} \delta\mathbf{R}. \tag{2.86}$$

Taking the transpose of the equation $\boldsymbol{\Psi}^T = -\frac{\partial J}{\partial \mathbf{U}} \left[\frac{\partial \mathbf{R}}{\partial \mathbf{U}}\right]^{-1}$ and moving everything to the left-hand side gives the *adjoint equation*,

$$\left(\frac{\partial \mathbf{R}}{\partial \mathbf{U}}\right)^T \boldsymbol{\Psi} + \left(\frac{\partial J}{\partial \mathbf{U}}\right)^T = \mathbf{0}. \tag{2.87}$$

The n^{th} component of $\boldsymbol{\Psi}$ is the sensitivity of J to changes in the n^{th} residual.

Since $\mathbf{R}(\mathbf{U}, \boldsymbol{\mu}) = 0$, from step 2 above we have $\delta\mathbf{R} = \frac{\partial \mathbf{R}}{\partial \boldsymbol{\mu}}\Big|_{\mathbf{U},\boldsymbol{\mu}} \delta\boldsymbol{\mu}$ and Eqn. 2.86 becomes

$$\delta J = \boldsymbol{\Psi}^T \frac{\partial \mathbf{R}}{\partial \boldsymbol{\mu}}\Big|_{\mathbf{U},\boldsymbol{\mu}} \delta\boldsymbol{\mu} \quad \Rightarrow \quad \frac{dJ}{d\boldsymbol{\mu}} = \boldsymbol{\Psi}^T \frac{\partial \mathbf{R}}{\partial \boldsymbol{\mu}}\Big|_{\mathbf{U},\boldsymbol{\mu}}. \tag{2.88}$$

Therefore, once we have $\boldsymbol{\Psi}$, no more solves are required for new sensitivities for the same output. Note that the calculation of $\frac{\partial \mathbf{R}}{\partial \boldsymbol{\mu}}$ is typically very cheap compared to a forward solve.

Although we have presented the adjoint in the context of a parameter sensitivity analysis, we will that residual perturbations also arise when discrete solutions are viewed from an enriched space. This will be the motivation for using adjoint solutions in output error estimation.

2.3.9 Adjoint Consistency

The solution to Eqn. 2.87 is a *discrete* adjoint, $\mathbf{\Psi}$, which at the simplest level we can think of as a vector of N numbers. However, the adjoint also has a continuous counterpart, call it $\boldsymbol{\psi}(\vec{x})$, and we can think of the N numbers in $\mathbf{\Psi}$ as expansion coefficients in an approximation of $\boldsymbol{\psi}$ using the same basis functions used for the primal problem. The accuracy of this approximation is of interest for various reasons, including error estimation.

Suppose that the exact primal solution, $\mathbf{u} \in \boldsymbol{\mathcal{V}}$, satisfies

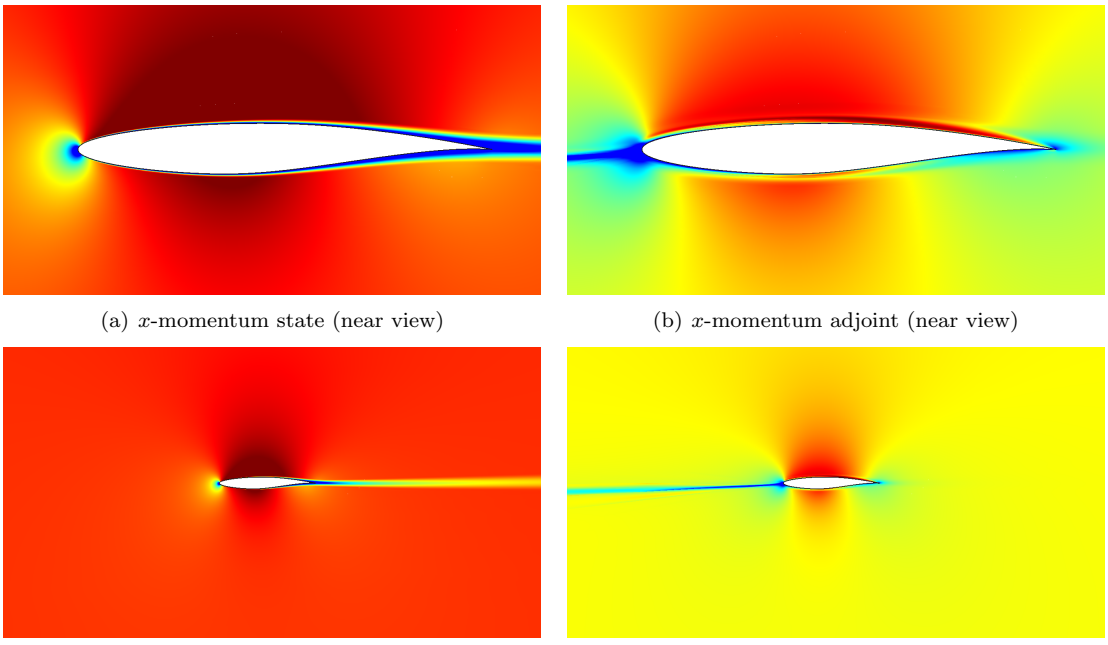
$$\mathcal{R}(\mathbf{u}, \mathbf{v}) = 0, \qquad \forall \mathbf{v} \in \boldsymbol{\mathcal{V}}, \tag{2.89}$$

for an appropriately defined space $\boldsymbol{\mathcal{V}}$. The exact adjoint $\boldsymbol{\psi} \in \boldsymbol{\mathcal{V}}$ then satisfies

$$\mathcal{R}'[\mathbf{u}](\mathbf{v}, \boldsymbol{\psi}) + \mathcal{J}'[\mathbf{u}](\mathbf{v}) = 0, \qquad \forall \mathbf{v} \in \boldsymbol{\mathcal{V}}, \tag{2.90}$$

where the primes denote Fréchét linearization about the arguments in square brackets, and $\mathcal{R}$ and $\mathcal{J}$ are the continuous versions of the semilinear form and output functional, respectively. While we have assumed that both $\mathbf{u}$ and $\boldsymbol{\psi}$ lie in $\boldsymbol{\mathcal{V}}$, this may not always be the case [47].

The exact adjoint can be regarded as a Green's function that relates source perturbations in the original partial differential equation to perturbations in the output [48,49]. A sample adjoint solution is shown in Figure 2.20 for Reynolds-averaged compressible flow over an airfoil. While the adjoint solution often shares qualitative



(a) x-momentum state (near view)

(b) x-momentum adjoint (near view)

(c) x-momentum state (far view)

(d) x-momentum adjoint (far view)

Figure 2.20: Comparison of the primal solution (x-momentum component) and the adjoint solution (conservation of x-momentum equation component) for a drag output in Reynolds-averaged turbulent flow over an RAE 2822 airfoil. The color scales are clipped to show the interesting features of each quantity – in the adjoint plots, yellow is near zero.

characteristics similar to the primal, such as the presence of a boundary layer in a high-Reynolds number flow, it also shows marked differences, such as the "wake reversal" seen in the far-field view in Figure 2.20. In this case, the output is drag, and upstream of the airfoil, residual perturbations on/near the stagnation streamline (the flow that is going to hit or come closest to the airfoil) will have a larger magnitude impact on the drag than residual perturbations elsewhere; hence we see an adjoint "reversed" wake telling us that there are large sensitivities to perturbations in front of the airfoil. The figure shown tells us that this is the case for residual perturbations in the conservation of x-momentum equations, but plots of the other adjoint components show similar behavior.

The adjoint field depicted in Figure 2.20 is the discrete adjoint solution on a fine mesh. It can only be regarded as a faithful representation of the exact adjoint if the discretization is in some manner consistent with the exact adjoint problem. *Primal consistency* in the variational problem requires that the exact solution $\mathbf{u}$ satisfy the discrete variational statement,

$$\mathcal{R}_h(\mathbf{u}, \mathbf{v}) = 0, \qquad \forall \mathbf{v} \in \boldsymbol{\mathcal{W}}_h, \tag{2.91}$$

where $\boldsymbol{\mathcal{W}}_h = \boldsymbol{\mathcal{V}}_h + \boldsymbol{\mathcal{V}} = \{\mathbf{h} = \mathbf{f} + \mathbf{g} : \mathbf{f} \in \boldsymbol{\mathcal{V}}_h, \mathbf{g} \in \boldsymbol{\mathcal{V}}\}$. Similarly, the combination of the discrete semi-linear form $\mathcal{R}_h$ and the functional $\mathcal{J}_h$ is said to be *adjoint consistent* if [47, 50, 51]

$$\mathcal{R}'_h[\mathbf{u}](\mathbf{v}, \boldsymbol{\psi}) + \mathcal{J}'_h[\mathbf{u}](\mathbf{v}) = 0, \qquad \forall \mathbf{v} \in \boldsymbol{\mathcal{W}}_h. \tag{2.92}$$

Discretizations that are not adjoint consistent may still be *asymptotically adjoint consistent* if Eq. 2.92 holds in the limit $h \to 0$, by which we mean the limit of uniformly increasing resolution, over suitably normalized $\mathbf{v} \in \boldsymbol{\mathcal{W}}_h$. For non-variational discretizations, the definition of consistency must involve an approximation operator

to map exact solutions into discrete spaces [52].

Adjoint consistency has an impact on the convergence of not only the adjoint approximation but also the primal approximation [42, 47]. In error estimation, an adjoint-inconsistent discretization can lead to irregular or oscillatory adjoint solutions that pollute the error estimate with noise and lead to adaptation in incorrect areas [47]. Enforcing adjoint consistency imposes restrictions on the output definition and on the interior and boundary discretizations that enter into the semi-linear form. These restrictions have been studied by several authors in the context of the discontinuous Galerkin method [42, 47]. In general, discretizations that are found to be adjoint inconsistent can often be made adjoint consistent by adding terms to either the semi-linear form or the output functional.

2.4 Summary

The research contribution from this chapter include,

- Built both the discrete and continuous adjoint system for the adtive flux method.
- Equipped the active flux method with error estimation capability. Theoretical work was tested on linear problems.
- Discovered the necessary condition for discontinuous Galerkin method to super-converge for acoustic cases, detailed in Appendix A. (*Acoustics code for DG was created to assist the research of Dr. Duoming Fan. The written code has been throughly examined in her thesis [53]. However this is not the main focus of my thesis. Thus, details of this discovery is documented in Appendix instead.*)

CHAPTER III

Mesh Motion

In this chapter, we present mesh motion algorithms for both the Active Flux (AF) method as well as the discontinuous Galerkin (DG) method. We then use these algorithms to perform adaptation using mesh motion, i.e. r-adaptation. There are two popular approaches in mesh motion algorithm development, 1) space-time discretizations that explicitly account for control volume changes over time, and 2) discretizations based on the arbitrary Lagrangian-Eulerian (ALE) framework.

As the Active Flux method is explicit and fully-discrete, a space-time approach is the most natural. This approach requires integrating over a general space-time domain and invoking the divergence theorem. The resulting motion-enabled AF method turns out to be fully conservative.

On the other hand, in the present work DG is used in semi-discrete form, for which the ALE framework is more suitable. This framework already exists in the DG code chosen for this work, which is Xflow. As the ALE method results in a discretization that is not exactly conservative, the development of the DG r-adaptation indicator requires careful manipulation in order for effective error reduction. Chapter 6 contains the details of this development.

3.1 Mesh Motion Algorithm for the Active Flux Discretization

3.1.1 One Spatial Dimension

In one dimension, the scalar advection equation with flow speed a reads,

$$\frac{\partial (au)}{\partial x} + \frac{\partial u}{\partial t} = 0, \tag{3.1}$$

and this can be re-written in compact, space-time, form as,

$$\nabla \cdot (\mathbf{V}u) \quad = \quad 0, \tag{3.2}$$

where $\mathbf{V} = [a\ 1]$, $\mathbf{X} = [x\ t]$, and $\nabla = [\frac{\partial}{\partial x}, \frac{\partial}{\partial t}]$. Consider now a cell undergoing mesh motion, as illustrated in Figure 3.1.

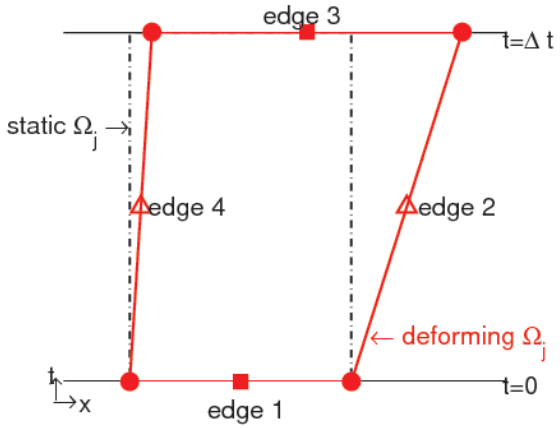


Figure 3.1: One dimensional element undergoing mesh motion: the node positions do not stay fixed as time progresses, so that the element is no longer rectangular in space-time.

Integrating Eqn. 3.2 over the non-rectangular space-time volume, Ω_j, yields,

$$\int_{\Omega_j} \nabla \cdot (\mathbf{V}u) = 0 \ \Rightarrow \ \int_{\partial\Omega_j} \mathbf{n} \cdot (\mathbf{V}u) = 0. \tag{3.3}$$

In the present case, with four space-time edges, the above space-time surface integral leads to a sum of integrals over the four edges,

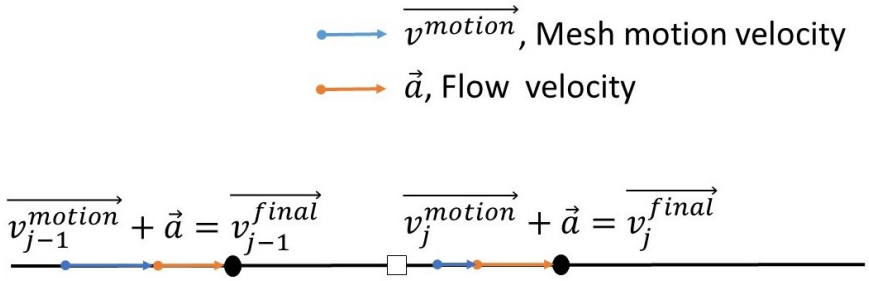


Figure 3.2: One dimensional characteristics tracing illustration. With motion present, the characteristic speed, $\vec{v}^{\text{final}}$, is determined by both the flow speed, $\vec{a}$, and, nodal movement velocity, $\vec{v}^{\text{motion}}$. Here, $\vec{v}^{\text{final}} = \vec{a} + \vec{v}^{\text{motion}}$. The new CFL condition needs to take this modified characterstic velocity into consideration.

$$
\begin{aligned}
I_1 &= \text{Integral on edge 1} \\
&= \int_{x_L(t=0)}^{x_R(t=0)} \begin{bmatrix} n_x & n_t \end{bmatrix} \cdot \begin{bmatrix} a & 1 \end{bmatrix} u ds \qquad (n_x, n_t) = [0, -1] \qquad (3.4) \\
I_2 &= \text{Integral on edge 2} \\
&= \int_0^{\Delta t} \begin{bmatrix} n_x & n_t \end{bmatrix} \cdot \begin{bmatrix} a & 1 \end{bmatrix} u \left| \frac{ds}{dt} \right| ds \qquad (3.5) \\
I_3 &= \text{Integral on edge 3} \\
&= \int_{x_L(t=\Delta t)}^{x_R(t=\Delta t)} \begin{bmatrix} n_x & n_t \end{bmatrix} \cdot \begin{bmatrix} a & 1 \end{bmatrix} u ds \qquad (n_x, n_t) = [0, 1] \qquad (3.6) \\
I_4 &= \text{Integral on edge 4} \\
&= \int_0^{\Delta t} \begin{bmatrix} n_x & n_t \end{bmatrix} \cdot \begin{bmatrix} a & 1 \end{bmatrix} u \left| \frac{ds}{dt} \right| ds \qquad (3.7)
\end{aligned}
$$

Summing these integrals and setting the result to zero lets us solve for the cell average at $t = \Delta t$,

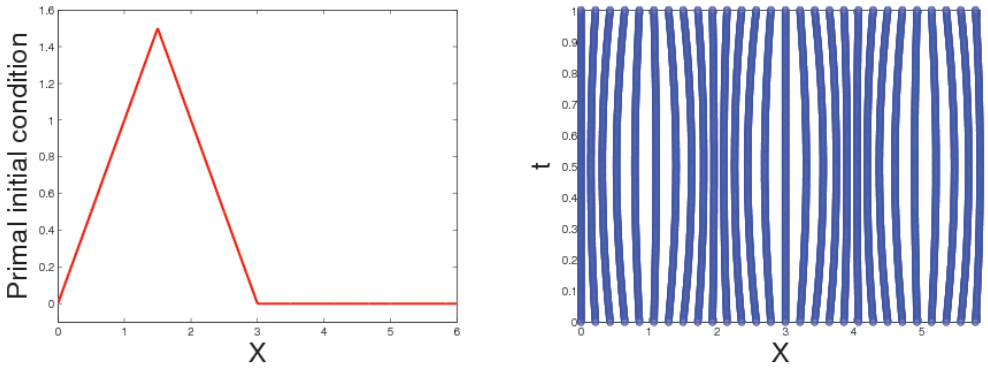
$$
\begin{aligned}
I_3 &= -I_1 - I_2 - I_4 \\
\implies \left[(x_R - x_L)\, \bar{u} \right]_{t=\Delta t} &= -I_1 - I_2 - I_4 \qquad (3.8) \\
\implies \bar{u}_{t=\Delta t} &= \frac{-I_1 - I_2 - I_4}{(x_R - x_L)_{t=\Delta t}}
\end{aligned}
$$

Equation 3.8 essentially equates to a modified characteristic tracing for the Active Flux method with mesh motion, Figure 3.2. Figure 3.2 also indicates a modified CFL condition due to the modified characteristic tracing.

To test the mesh-motion formulation of the Active Flux method with mesh motion, we consider a problem with a prescribed motion of the form,

$$x(t) = \sin(x_0 \pi)(t - 0.5), \tag{3.9}$$

where x_0 is the original, time $t = 0$, coordinate. Figure 3.3(b) shows the resulting motion of 30 initially equally-spaced nodes under this mapping. On this deforming mesh, with periodic boundaries, we consider a primal problem in which a linear hat primal state, shown in Figure 3.3(a), advects with constant velocity for one period.



(a) Primal problem initial condition

(b) Nodes subject to prescribed motion

Figure 3.3: Test problem for the Active Flux method with mesh motion in one spatial dimension.

At the end of one period, we measure the L_2 error norm of the solution error (the exact solution is just the initial condition),

$$e_{L_2} = \sqrt{\frac{1}{L}\int_0^L \left[u_{\text{AF}}(x) - u_{\text{exact}}(x)\right]^2 dx} = \sqrt{\frac{1}{L}\sum_{j=1}^{n_{cell}} \left\{ \int_{(j-1)\Delta x}^{j\Delta x} \left[u_{\text{AF}}(x) - u_{\text{exact}}(x)\right]^2 dx \right\}} \tag{3.10}$$

In Eqn. 3.10, n_{cell} is the the total number of cells, and order 7 Gauss-Legendre quadrature is used to evaluate the integrals. Figure 3.4 shows the behavior of the

error with mesh refinement. We observe that the L_2 error norm converges at a rate of approximately 3, consistent with our expectations, as the Active Flux method is third-order accurate.

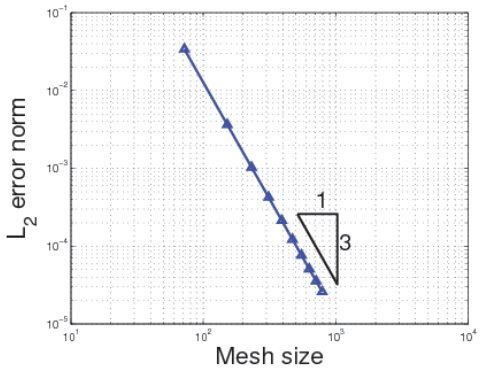


Figure 3.4: Convergence study for the active flux method on a moving mesh in one spatial dimension, with CFL=0.1.

3.1.2 Two Spatial Dimensions

In two spatial dimensions, the scalar advection equation reads

$$\frac{\partial(a_1 u)}{\partial x_1} + \frac{\partial(a_2 u)}{\partial x_2} + \frac{\partial u}{\partial t} = 0, \tag{3.11}$$

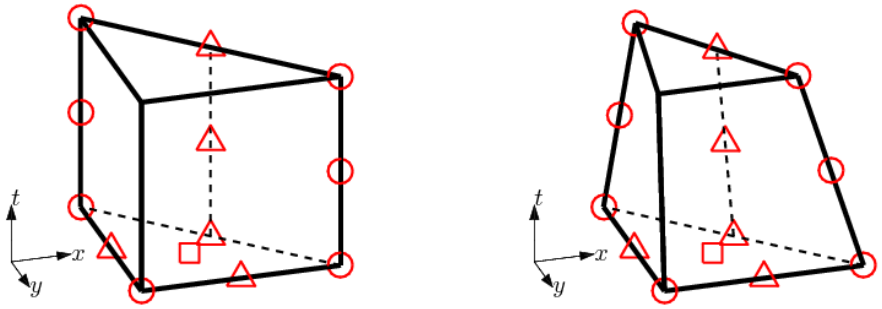
which can be re-written as,

$$\nabla \cdot (\mathbf{V}u) = 0. \tag{3.12}$$

Here, $\mathbf{V} = \begin{bmatrix} a_1 & a_2 & 1 \end{bmatrix}$ and the space-time coordinate is $\mathbf{X} = \begin{bmatrix} x_1 & x_2 & t \end{bmatrix}$. Integrating Eqn. 3.12 in a space-time control volume Ω_j, we obtain,

$$\int_{\Omega_j} \nabla \cdot (\mathbf{V}u)\, d\Omega \equiv \oint_{\partial\Omega_j} [(u\mathbf{V}) \cdot \mathbf{n}]\, dS, \tag{3.13}$$

where, $\mathbf{n}$ is the outward pointing unit space-time normal on each face of control volume Ω_j. As Figure 3.5 shows, solving the governing equations becomes a problem of evaluating five surface integrals.



(a) Space-time control volume without mesh motion, a right triangular prism.

(b) Space-time control volume with mesh motion, a deformed prism.

Figure 3.5: Space-time control volume in two dimensions

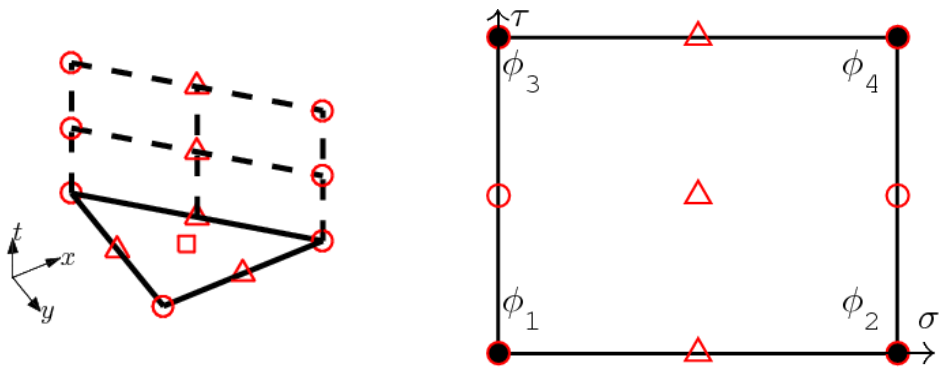
With or without motion, the top and bottom surfaces of the space-time control volume, Figure 3.5(a) and Figure 3.5(b), are flat because we require that the unknowns reside at the same time point after each time step. Integrals on these faces can be obtained directly by using quadrature, a 6th order rule in our work.

$$I_{\text{top}} = \int_{\text{top surface}} \mathbf{V} u_H \cdot \mathbf{n}_{\text{top}} dS, \qquad \mathbf{n}_{\text{top}} = [0 \;\; 0 \;\; 1], \tag{3.14}$$

$$I_{\text{bottom}} = \int_{\text{bottom surface}} \mathbf{V} u_H \cdot \mathbf{n}_{\text{bottom}} dS, \qquad \mathbf{n}_{\text{bottom}} = [0 \;\; 0 \;\; -1]. \tag{3.15}$$

On the other hand, the remaining three surfaces of the prism in Figure 3.5(b), will generally not be flat in the presence of mesh motion. To integrate on these surfaces, we use a bilinear map to transform the face to a 2D reference square, and we then apply Simpson's rule in reference space. The mapping between the physical face and reference square is illustrated in Figure 3.6.

In Figure 3.6(b), the black dots represent nodes of bilinear basis functions in



(a) Surface unknowns (b) Reference space surface

Figure 3.6: Mapping between a 3D space-time face in physical space and a reference square.

reference $[\sigma, \tau]$ space, namely,

$$\begin{aligned} \phi_1 &= (1-\sigma)(1-\tau) \\ \phi_2 &= \sigma(1-\tau) \\ \phi_3 &= (1-\sigma)\tau \\ \phi_4 &= \sigma\tau \ . \end{aligned} \tag{3.16}$$

The mapping from reference ($\boldsymbol{\xi} = [\sigma, \tau]$) to physical ($\mathbf{X}_{\text{face}}$) space then reads,

$$\mathbf{X}_{\text{face}} = \sum_{i=1}^{4} \mathbf{X}_i \phi_i(\boldsymbol{\xi}), \tag{3.17}$$

where $\mathbf{X}_i$ are the space-time coordinates of the four face corners. The integral on the 3D surface then transforms into an integral on the reference square, Γ, via

(3.18)

$$\int_{\text{surface}_j} \left[(\mathbf{V}u_H) \cdot \mathbf{n}_{\text{surface}_j}\right] dS = \int_{\Gamma} \left[(\mathbf{V}u_H) \cdot \mathbf{n}_{\text{surface}_j} J_{\text{surface}_j}\right] d\Gamma, \qquad j = 1, 2, 3$$

where

$$\mathbf{n}J = \frac{\partial \mathbf{X}_{\text{face}}}{\partial \sigma} \times \frac{\partial \mathbf{X}_{\text{face}}}{\partial \tau},$$

and J is the determinant of the Jacobian matrix,

$$\underline{J} = \frac{\partial \mathbf{X}_{\text{face}}}{\partial \boldsymbol{\xi}}. \tag{3.19}$$

The scheme remains fully conservative because the fluxes between space-time elements remain uniquely defined.

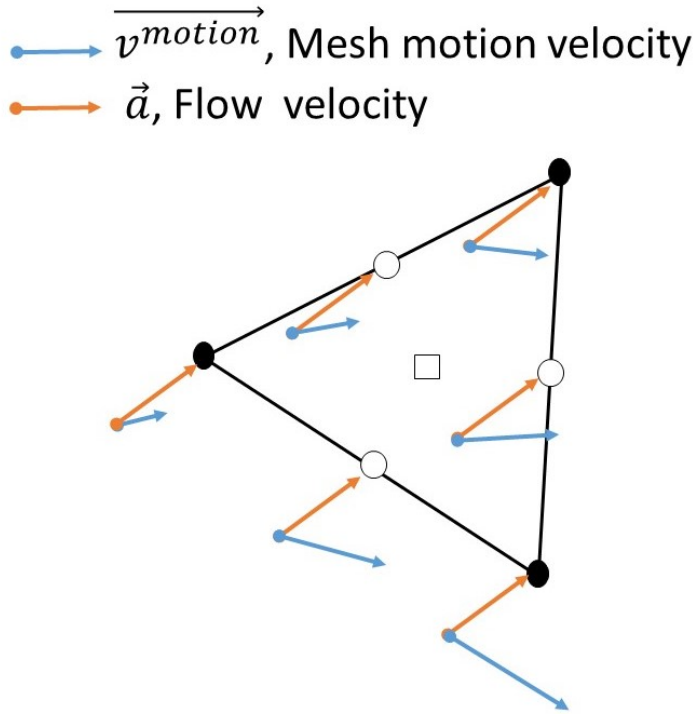


Figure 3.7: Two dimensional characteristics tracing illustration. With motion present, the characteristic speed, $\vec{v}^{\text{final}}$, is determined by both the flow speed, $\vec{a}$, and, nodal movement velocity, $\vec{v}^{\text{motion}}$. Here, $\vec{v}^{\text{final}} = \vec{a} + \vec{v}^{\text{motion}}$. The new CFL condition needs to take this modified characterstic velocity into consideration.

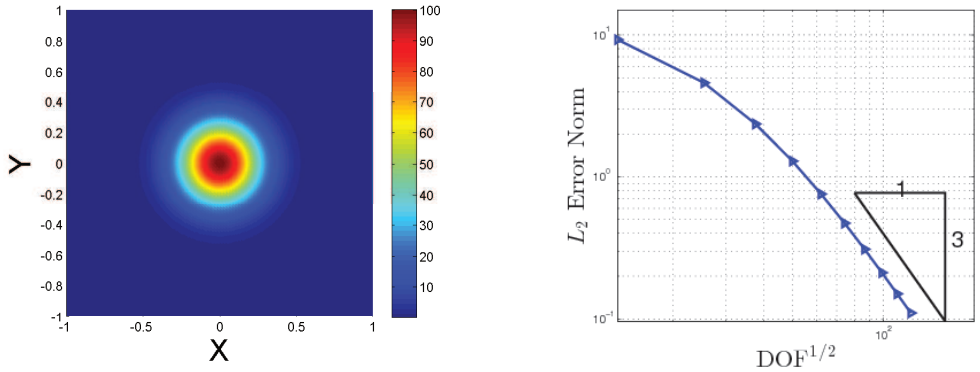
In turn, just like in one dimension, the modified flux leads to modified CFL condition, illustrated in Figure 3.7. The new characteristics velocity that determines the time step, is the vector sum between local nodal mesh motion velocity and flow velocity.

As in the case of one spatial dimension, we perform a convergence study for the Active Flux method with mesh motion. The problem of interest is two-dimensional advection with periodic boundaries, illustrated in Figure 3.8(a). The initial condition is a Gaussian pulse at the domain center, and the advection velocity is $[a_1, a_2] =$

$[2,\ 2]$. The output of interest is the L_2 error norm,

$$e_{L_2} = \sqrt{\frac{1}{\text{domain area}} \int_\Omega \left[u_{\text{AF}} - u_{\text{exact}}\right]^2 d\Omega}, \tag{3.20}$$

where a 6th order Gauss-Legendre quadrature rule is used for numerical integration.



(a) Primal problem initial condition

(b) Scheme convergence study

Figure 3.8: Convergence study for the active flux method with motion in two dimensions, periodic BCs, $a = [2\ 2]^T$, simulation time of one period, and CFL = 0.1.

The motion prescribed for the convergence study is a modified version of that presented by Persson *et al* [19],

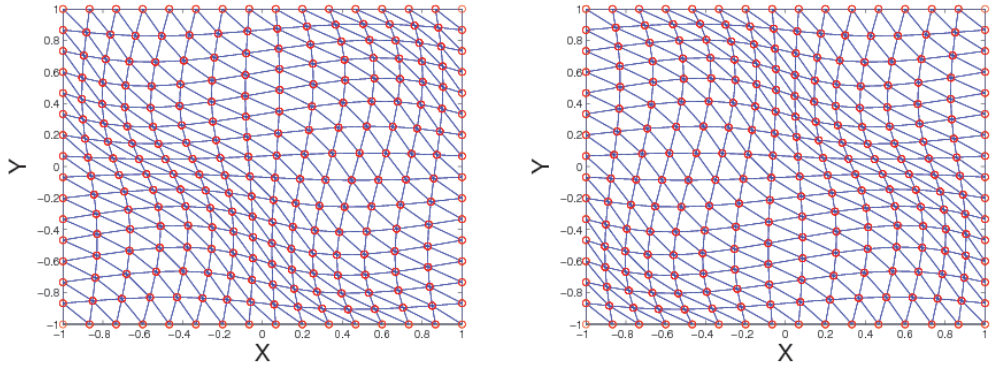
$$\begin{aligned} x_1\left(X_1, X_2, t\right) &= X_1 + 0.1 \sin\left(\pi X_1\right) \sin(\pi X_2) \sin(2\pi t/t_0), \\ x_2\left(X_1, X_2, t\right) &= X_2 + 0.1 \sin\left(\pi X_2\right) \sin(\pi X_1) \sin(2\pi t/t_0), \end{aligned} \tag{3.21}$$

where X_1 and X_2 are nodal coordinates on the initial-time mesh, $t_0 = 1.0$ period and t is arbitrary time point between 0 and t_0.

Figure 3.8(b) shows the result of the convergence study. The error appears to converge at approximately third order, which again agrees with our expectations, as the Active Flux method is third-order accurate.

3.1.3 Elaboration on the Modified CFL Condition

With mesh motion present, the CFL condition for the Active Flux discretization changes. This section elaborates on how the new CFL condition is measured for the Active Flux discretization with mesh motion.



(a) $t = 0.3t_0$ (b) $t = 0.7t_0$

Figure 3.9: Snapshots of the mapping used in the two-dimensional convergence-rate verification study.

A necessary condition for the convergence of a finite difference method for a hyperbolic PDE is that the numerical domain of dependence contains the physical domain of dependence. This requirement is known as the Courant-Friedrichs-Levy or CFL condition, named after the authors who first described this requirement.

The CFL condition states

$$\text{CFL} \equiv \frac{|s_i|\,\Delta t}{d_i}, \tag{3.22}$$

where, $|s_i|$ is the maximum wave speed within the control volume i. d_i is the diameter of cell i, in our case the hydraulic diameter,

$$d_i = \frac{2 \times \text{area of the control volume triangle}}{\text{perimeter of the triangle}}$$

This is the diameter of the largest circle that can be inscribed in the triangle. d_i is measured at every current time point. Therefore, the Active Flux method with mesh motion can still achieve a maximum CFL number of 1.

3.2 Adjoint Discretization and Verification for the Active Flux Method with Mesh Motion

3.2.1 Adjoint Discretization

For problems involving mesh motion, we consider the continuous adjoint formulation of the Active Flux method, as described in Section 2.2.2. Since the continuous adjoint equation has the same structure as the primal equation, it is solved in an analogous manner. The primary difference is that the computation starts at the final time and proceeds backwards to the initial time. This requires tracing characteristics in the opposite direction from the primal problem. The same modifications that are made for incorporating mesh motion into the primal problem are also made for the adjoint problem.

3.2.2 Sensitivity Test with Motion

When mesh motion is present, the sensitivity formulation remains exactly the same as described in Eqn. 2.60. We are interested in sensitivities to initial conditions, and since these are prescribed at time $t = 0$, when there is no mesh deformation, the mesh motion changes do not affect the formulation of the sensitivity test. The only effect of mesh motion is on the computation of the adjoint $(\psi_H)_{t=0}$, which is time-marched backwards on the deformed mesh.

With the motion illustrated in Figure 3.3(b), a domain weighted integral output is studied. Initial condition and perturbation for the sensitivity test is tabulated in Table 2.3. The sensitivity test passes to machine-precision. That is, the output perturbations predicted by the adjoint-based formulation agree exactly with the output perturbations obtained by re-running the code with perturbed initial conditions. This exact agreement only occurs when the adjoint terminal condition, i.e. the output weight, is both continuous and slope continuous throughout the periodic

computational domain.

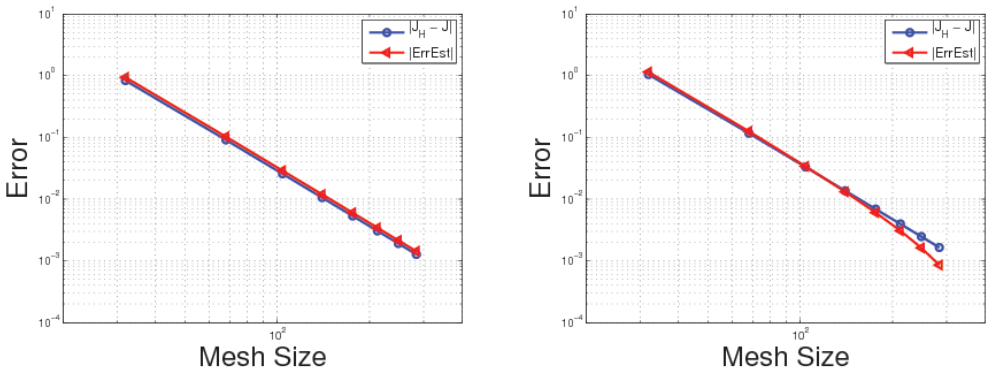
3.2.3 Error Estimation Study

Output-based error estimation for moving mesh problems is similar to static mesh problems. We solve the continuous adjoint problem on a finer space, consisting of uniform refinement in both space and time. At each time step in this adjoint solution procedure, we compute the adjoint-weighted residual on the sub-elements arising from the uniform refinement. Summed together, these adjoint-weighted residuals give the estimate of the total space-time numerical error in the output. Taken individually, the weighted residuals provide an indicator for adaptive refinement.

One Spatial Dimension

Consider the one-dimensional advection problem described in Section 3.1.1 and illustrated in Figure 2.16. In this case, the error estimate obtained from an adjoint on a uniformly-refined fine space gives $\delta J \approx 0.6673$, whereas the true output error is 0.5931, a difference of about 13%. We now show how this error converges with uniform mesh refinement.

We consider the primal initial condition shown in Figure 2.16(a). Using the same output weighting function as in the sensitivity study, we compute the adjoint-based error estimate at various levels of mesh refinement. Figure 3.10 shows the convergence of the error estimate, with mesh motion off (a) and on (b).



(a) No Movement

(b) With prescribed mesh motion

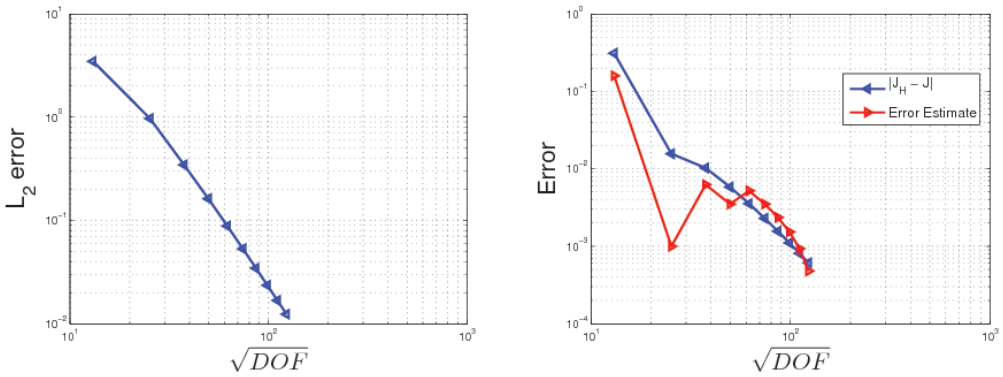
Figure 3.10: Error convergence study for a one-dimensional scalar advection problem. The mesh motion prescribed for Figure 3.10(b) is the same as in Figure 3.4(b). The slopes of all lines are approximately 3.

We note that with mesh motion present, both the output of interest and the error estimate exhibit the expected third order convergence rate.

Two Spatial Dimensions

To verify the error estimate in two dimensions, we consider the same moving-mesh problem as in the sensitivity study. The output is given in Eqn. 2.61, as a weighted integral of the primal state at the final time. The mesh motion also remains the same, as given in Eqn. 3.21. Figure 3.11 shows the results of the error convergence study.

Figure 3.11(a) shows convergence of the primal L_2 error norm to confirm that the case is converging correctly at CFL = 0.7, (CFL = 0.7 is the largest CFL number our solver can run for the chosen mapping.) Figure 3.11(b) shows the convergence of the continuous adjoint error estimate, which is approximately third order.



(a) L2 convergence test, CFL = 0.7

(b) Error estimate convergence study

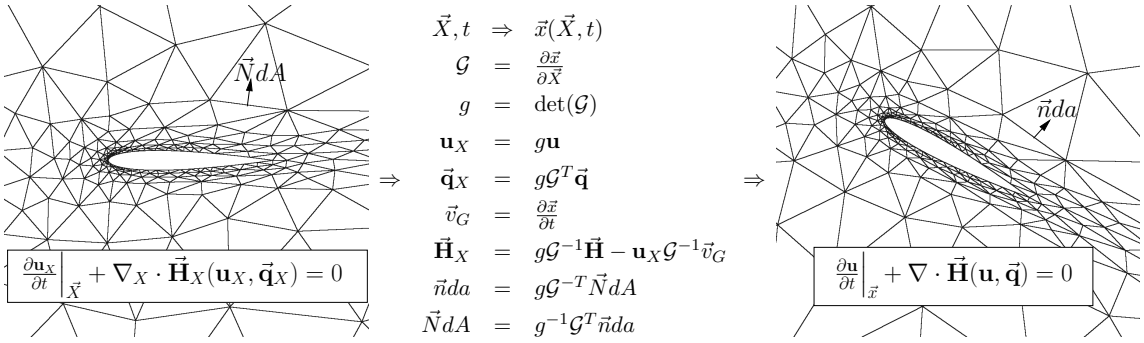
Figure 3.11: Two-dimensional scalar advection: error estimate convergence study at CFL = 0.7 and a simulation time of one period. The slope of all lines is approximately 3.

3.3 Discontinuous Galerkin Discretization

In an arbitrary Lagrangian-Eulerian (ALE) method, the mesh can move at a velocity different from that of the flow, which is useful for modeling problems in which objects move or deform. In the present case, we are not specifically concerned with deforming domains, and instead use the motion of the mesh to redistribute resolution in the physical domain.

3.3.1 The Arbitrary Lagrangian-Eulerian Mapping

The idea of ALE is to map the original PDE on a deforming physical domain to a modified PDE on a static reference domain. Figure 3.12 illustrates this mapping. Table 3.1 defines the relevant quantities in this mapping. Note that bold indicates a state vector, an arrow indicates a spatial vector, and an underline indicates a spatial matrix. The expressions for the transformations of the normals are obtained using $dv = gdV$ for infinitesimal volumes and $d\vec{l} = \mathcal{G}d\vec{L}$ for infinitesimal vectors, as derived in Persson *et al* [19].



(a) Reference domain: $\vec{X}, \mathbf{u}_X, \vec{\mathbf{F}}_X$ (b) Mapping (c) Physical domain: $\vec{x}, \mathbf{u}, \vec{\mathbf{F}}$

Figure 3.12: ALE mapping between the reference and physical domains.

Table 3.1: Key quantities used in the ALE mapping.

$\vec{X}$	=	reference-domain coordinates	da	=	differential area on physical domain
$\vec{x}$	=	physical-domain coordinates	dA	=	differential area on reference domain
$\underline{G}$	=	mapping Jacobian matrix	$\vec{v}_G$	=	grid velocity, $\partial\vec{x}/\partial t$
g	=	determinant of Jacobian matrix	$\mathbf{u}$	=	physical state
$\vec{n}$	=	normal vector on physical domain	$\mathbf{u}_X$	=	state approximated on reference domain
$\vec{N}$	=	normal vector on reference domain	$\vec{\mathbf{F}}$	=	flux vector on physical domain
$v(t)$	=	physical domain (dynamic)	$\vec{\mathbf{F}}_X$	=	flux vector on reference domain
V	=	reference domain (static)			

The partial differential equation on the physical domain is

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \vec{\mathbf{F}}(\mathbf{u}, \nabla\mathbf{u}) = \mathbf{0}, \qquad \vec{\mathbf{F}} = \vec{\mathbf{F}}^i(\mathbf{u}) - \vec{\mathbf{F}}^v(\mathbf{u}, \nabla\mathbf{u}), \tag{3.23}$$

where both inviscid and viscous fluxes are included. Integrating over a time-varying volume $v(t)$ yields,

$$\int_{v(t)} \frac{\partial \mathbf{u}}{\partial t} dv + \int_{\partial v(t)} \vec{\mathbf{F}} \cdot \vec{n}\, da = 0, \qquad \vec{n} \text{ is outward-pointing on } \partial v(t). \tag{3.24}$$

We now transform to the reference domain, V. The boundary integral of the flux is

$$\int_{\partial v(t)} \vec{\mathbf{F}} \cdot \vec{n}\, da = \int_{\partial V} \vec{\mathbf{F}} \cdot (g\underline{G}^{-T}\vec{N})\, dA = \int_{\partial V} (g\underline{G}^{-1}\vec{\mathbf{F}}) \cdot \vec{N}\, dA. \tag{3.25}$$

The first integral in Eqn. 3.23 transforms using Leibniz's rule (a.k.a Reynolds' trans-

port theorem in this case),

$$\begin{aligned}\int_{v(t)} \frac{\partial \mathbf{u}}{\partial t} &= \frac{d}{dt}\int_{v(t)} \mathbf{u}\, dv - \int_{\partial v(t)} (\mathbf{u}\vec{v}_G)\cdot \vec{n}\, da \\ &= \frac{d}{dt}\int_V \mathbf{u} g\, dV - \int_{\partial V} (\mathbf{u}\vec{v}_G)\cdot (g\underline{G}^{-T}\vec{N})\, dA \\ &= \int_V \frac{\partial (g\mathbf{u})}{\partial t}\, dV - \int_{\partial V} (g\mathbf{u}\underline{G}^{-1}\vec{v}_G)\cdot \vec{N}\, dA. \end{aligned} \tag{3.26}$$

Substituting Equations 3.25 and 3.26 into Eqn. 3.24 and applying the divergence theorem gives the PDE on the reference domain,

$$\begin{aligned}\frac{\partial \mathbf{u}_X}{\partial t}\Big|_X + \nabla_X \cdot \vec{\mathbf{F}}_X(\mathbf{u}_X, \nabla_X \mathbf{u}_X) &= 0, \\ \text{where} \qquad \mathbf{u}_X &= g\mathbf{u}, \\ \vec{\mathbf{F}}_X &= g\mathcal{G}^{-1}\vec{\mathbf{F}} - \mathbf{u}_X \mathcal{G}^{-1}\vec{v}_G. \end{aligned} \tag{3.27}$$

∇_X denotes the gradient with respect to the reference coordinates. We break up the transformed flux, $\vec{\mathbf{F}}_X$, into inviscid and viscous fluxes by lumping the grid-velocity term into the inviscid flux,

$$\vec{\mathbf{F}}_X = \vec{\mathbf{F}}^i_X - \vec{\mathbf{F}}^v_X, \qquad \vec{\mathbf{F}}^i_X = g\mathcal{G}^{-1}\vec{\mathbf{F}}^i - \mathbf{u}_X\mathcal{G}^{-1}\vec{v}_G, \qquad \vec{\mathbf{F}}^v_X = g\mathcal{G}^{-1}\vec{\mathbf{F}}^v.$$

The gradient of the state transforms via the chain an product rules. Using implied summation,

$$\begin{aligned}\nabla \mathbf{u} = \frac{\partial \mathbf{u}}{\partial x_j} = \frac{\partial (g^{-1}\mathbf{u}_X)}{\partial X_d}\frac{\partial X_d}{\partial x_j} &= \left(g^{-1}\frac{\partial \mathbf{u}_X}{\partial X_d} - g^{-2}\frac{\partial g}{\partial X_d}\mathbf{u}_X\right) G^{-1}_{dj} \\ &= g^{-1}\left(\frac{\partial \mathbf{u}_X}{\partial X_d} - g^{-1}\frac{\partial g}{\partial X_d}\mathbf{u}_X\right) G^{-1}_{dj}, \end{aligned} \tag{3.28}$$

where d and j index the reference and physical coordinates, respectively. We also have,

$$\mathcal{G} = G_{jd} = \frac{\partial x_j}{\partial X_d}, \qquad \delta_{ji} = \frac{\partial x_j}{\partial x_i} = \frac{\partial x_j}{\partial X_d}\frac{\partial X_d}{\partial x_i} = G_{jd}G^{-1}_{di}, \qquad \Rightarrow \quad \mathcal{G}^{-1} = G^{-1}_{di} = \frac{\partial X_d}{\partial x_i}.$$

3.3.2 Discretization

In a DG setting, discretization of the new reference-domain equation requires modifications to the numerical flux function on inter-element faces, to the boundary conditions, to the face normal vectors, and to the quadrature integration weights. These modifications are based on the reference-to-global mapping and its derivatives.

The weighted residual statement of Eqn. 3.27 on the reference domain is obtained from the PDE by multiplying by test functions (defined in the reference domain) and integrating over reference-domain elements. The resulting terms in the semilinear form for one reference-domain element, κ, are as follows:

$$\text{(total)} \quad \mathcal{R}_X(\mathbf{u}_X, \mathbf{v}) = \mathcal{R}^u_X(\mathbf{u}_X, \mathbf{v}) + \mathcal{R}^i_X(\mathbf{u}_X, \mathbf{v}) + \mathcal{R}^v_X(\mathbf{u}_X, \mathbf{v})$$

$$\begin{aligned}
\text{(unsteady)} \quad \mathcal{R}^u_X(\mathbf{u}_X, \mathbf{v}) &= \int_\kappa \frac{\partial u_{X,k}}{\partial t} v_k dV \\
\text{(inviscid)} \quad \mathcal{R}^i_X(\mathbf{u}_X, \mathbf{v}) &= -\int_\kappa \partial_{X_d} v_k F^i_{X,dk} dV + \int_{\partial\kappa} v_k^+ \widehat{F^i_{X,dk} N_d}\, dA \\
\text{(viscous)} \quad \mathcal{R}^v_X(\mathbf{u}_X, \mathbf{v}) &= \int_\kappa \partial_{X_d} v_k F^v_{X,dk} dV - \int_{\partial\kappa} v_k^+ \widehat{F^v_{X,dk} N_d}\, dA
\end{aligned}$$

where $\mathbf{v}$ is a test function in the reference domain, d indexes the reference domain spatial coordinates, and k indexes the state vector. The hats indicate numerical fluxes on element interfaces or domain boundaries, and the $^+$ superscript indicates quantities taken from the element interior.

The discretization would be straightforward were it not for the fact that fluxes and boundary conditions are specified on the physical domain. A natural approach that minimizes intrusion into the code is to express the reference-space fluxes and boundary conditions in terms of the physical fluxes and boundary conditions.

Inviscid flux

$$\vec{\mathbf{F}}^i_X = g\mathcal{G}^{-1}\vec{\mathbf{F}}^i - \mathbf{u}_X\mathcal{G}^{-1}\vec{v}_G = g\mathcal{G}^{-1}\left(\vec{\mathbf{F}}^i - \mathbf{u}\vec{v}_G\right). \tag{3.29}$$

The inviscid flux includes the standard Galilean transformation expected from changing reference frames and also a multiplication by $g\mathcal{G}^{-1}$, which is done by post-processing the equation-set specific flux.

On element interfaces, evaluation of the numerical flux $\widehat{F^i_{X,dk}N_d}$ also requires changes. Using

$$\vec{N}dA = g^{-1}\mathcal{G}^T\vec{n}da \quad \Rightarrow \quad N_d dA = g^{-1}(\mathcal{G}^T)_{dj}n_j da, \quad \text{and} \quad n_j da = g(\mathcal{G}^{-T})_{jd}N_d dA,$$

where $(\mathcal{G}^T)_{dj} = G_{jd}$ and $(\mathcal{G}^{-T})_{jd} = G^{-1}_{dj}$, we obtain

$$\begin{aligned} F^i_{X,dk}N_d\,dA &= gG^{-1}_{dj}\left(F^i_{jk} - u_k v_{G,j}\right)N_d\,dA \\ &= \left(F^i_{jk} - u_k v_{G,j}\right)gG^{-1}_{dj}N_d\,dA \\ &= \left(F^i_{jk} - u_k v_{G,j}\right)n_j\,da \end{aligned}$$

Without mesh motion the numerical flux calculation returns $\widehat{F^i_{jk}n_j}$. With mesh motion present, the flux has to be modified to operate on $\left(F^i_{jk} - u_k v_{G,j}\right)$ instead of F^i_{jk}. This is a simple but intrusive change because we need to modify equation-set specific functions (i.e. the Riemann solvers) to take as input a grid velocity, $v_{G,j}$.

For example, given two states $\mathbf{u}^L$ and $\mathbf{u}^R$, the Roe flux without mesh motion reads

$$\left[F^i_{jk}n_j\right]^{\text{Roe}} = \frac{1}{2}\left(F^L_{jk}n_j + F^R_{jk}n_j\right) - \frac{1}{2}\left|A_{kl}(\mathbf{u}^{\text{Roe}})\right|(u^R_l - u^L_l).$$

With mesh motion present, the Roe flux becomes

$$\begin{aligned} \left[(F^i_{jk} - u_k v_{G,j})n_j\right]^{\text{Roe}} = {} & \frac{1}{2}\left(F^L_{jk}n_j + F^R_{jk}n_j\right) \\ & -\frac{1}{2}\left(u^L_k + u^R_k\right)u_G - \frac{1}{2}\left|A_{kl}(\mathbf{u}^{\text{Roe}}) - \delta_{kl}u_G\right|(u^R_l - u^L_l), \end{aligned}$$

where $u_G = v_{G,j} n_j$ is the component of the grid velocity in the direction of the physical normal $\vec{n}$. The new terms consist of an addition to the flux of the average state multiplied by the mesh velocity, and a shift of the eigenvalues of the linearization about the Roe-average state, $\mathbf{u}^{\text{Roe}}$.

Viscous flux The reference-domain viscous flux is related to the physical viscous flux through

$$F^v_{X,dk} = g G^{-1}_{di} F^v_{ik}. \tag{3.30}$$

If the physical viscous flux is calculated using a diffusion matrix and the physical state gradient, $F^v_{ik} = A_{ijkl} \partial_{x_j} u_l$, then, using Eqn. 3.28 for the physical gradient, the reference-domain viscous flux is, using implied summation,

$$\begin{aligned} F^v_{X,dk} &= g G^{-1}_{di} A_{ijkl} \partial_{x_j} u_l \\ &= g G^{-1}_{di} A_{ijkl} g^{-1} \left(\partial_{X_c} u_{X,l} - u_{X,l} g^{-1} \partial_{X_c} g \right) G^{-1}_{cj} \\ &= \underbrace{G^{-1}_{di} A_{ijkl} G^{-1}_{cj}}_{A_{X,dckl}} \left(\partial_{X_c} u_{X,l} - u_{X,l} g^{-1} \partial_{X_c} g \right), \end{aligned} \tag{3.31}$$

where c, d index the reference domain coordinates. $A_{X,dckl}$ represents the diffusion matrix on the reference domain. It can be re-written in a more symmetrical form as

$$A_{X,dckl} = G^{-1}_{di} A_{ijkl} G^{-1}_{cj} = G^{-1}_{di} A_{ijkl} G^{-T}_{jc}.$$

3.3.3 Boundary Conditions

The physical convective boundary flux, $\vec{\mathbf{F}}^{ib}$, is modified to account for mesh motion as given in Eqn. 3.29,

$$\vec{\mathbf{F}}^{ib}_X = g \mathcal{G}^{-1} \left(\vec{\mathbf{F}}^{ib} - \mathbf{u}^b \vec{v}_G \right).$$

We note that the physical boundary flux must be aware of motion on the boundary, $\vec{v}_G$. For example, on a moving wall, the flow tangency boundary condition states

that the normal component of the fluid velocity is equal to the normal component of the boundary motion velocity (which would be zero without mesh motion). This physical consideration is separate from the subtraction of $\mathbf{u}^b\vec{v}_G$ above – both must be included.

Calculation of the viscous contribution on a boundary requires not only the boundary state, $\mathbf{u}^b$, but also the boundary flux. For pure Dirichlet boundary conditions, the state gradient information is taken from the interior. In other cases, the physical viscous flux is prescribed on the boundary (e.g. zero heat flux for an adiabatic wall), and in these cases, the viscous flux contribution is added directly to the residual. Let's call Q_k^b the prescribed boundary viscous flux dotted with the physical normal. Then in our residual contribution, we will be integrating,

$$Q_k^b da = F_{ik}^v n_i da = g^{-1} G_{id} F_{X,dk}^v \, g G_{ci}^{-1} N_c dA = F_{X,dk}^v N_d dA.$$

This means that the prescribed boundary viscous flux is the same in both the physical and the reference domains. That is, no transformation needs to be applied to Q_k^b when adding the viscous flux contribution to the residual.

3.3.4 Analytical Mesh Motions

General mesh mapping strategies using blending functions are given in Persson *et al* [19]. The requirement is to prescribe a reference-to-physical mapping at every point. This can be achieved by *blending* a rigid-body motion, such as pitching and plunging for an airfoil, to zero at the farfield. Smooth mappings are desirable, as highly-nonlinear mappings will stress integration requirements for the residual contributions (very high quadrature rules will be required). In addition severely-distorted elements are more likely to be generated for highly-nonlinear mappings, limiting the allowable time step for explicit methods and increasing the possibility

of negative element Jacobians.

3.3.5 The Geometric Conservation Law (GCL)

Due to the nonlinear and non-polynomial nature of general mappings, a constant state in the physical domain ($\mathbf{u} = \bar{\mathbf{u}} = \text{const.}$) will generally not be representable using a standard polynomial basis in the reference domain ($\mathbf{u}_X = g\bar{\mathbf{u}}$ will not be a polynomial in X). This means that in an unsteady free-stream test, the free-stream will generally not be preserved exactly. Persson *et al* [19] describe one technique, a geometric conservation law (GCL), for addressing this problem. This technique relies on approximating (in reference space) $\mathbf{u}_{\bar{X}} = \bar{g}\mathbf{u} = \bar{g}g^{-1}\mathbf{u}_X$ instead of $\mathbf{u}_X$, where $\bar{g}$ is a separate variable approximated using the same basis and marched using the same unsteady solver as the state to solve the following equation:

$$\frac{\partial \bar{g}}{\partial t} - \nabla_X \cdot (g\underline{G}^{-1}\vec{v}_G) = 0.$$

Note that now a constant physical state ($\mathbf{u}_X = g\bar{\mathbf{u}}$) is representable, since $\mathbf{u}_X = \bar{g}\bar{\mathbf{u}}$ is a polynomial in the discrete approximation space. In the present work, however, we do not use a geometric conservation law, as experiments in previous work show that the impact of the lack of geometric conservation diminishes with increased resolution, e.g. via a higher-order discretization in DG [20].

3.3.6 Arbitrary Lagrangian-Eulerian Framework

In an arbitrary Lagrangian-Eulerian (ALE) method, the mesh can move at a velocity different from that of the flow, which is useful for modeling problems in which objects move or deform. In the present case, we are not specifically concerned with deforming domains, and instead use the motion of the mesh to redistribute resolution in the physical domain.

The ALE method uses a map between the static reference domain and the deforming physical domain and solves transformed equations on the reference domain [19]. This transformation is illustrated graphically in Figure 3.13, and Table 3.2 defines key quantities.

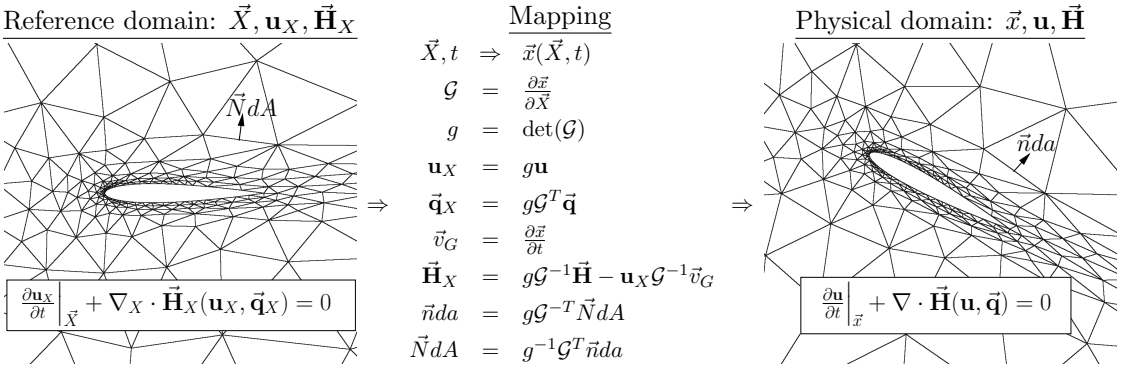


Figure 3.13: Summary of the mapping between reference and physical domains. The equations are solved on the reference domain, which remains fixed for all time. When denoting reference-domain quantities, we use a subscript X.

Table 3.2: Definitions of variables used in the ALE mapping. Bold indicates a state vector and an arrow indicates a spatial vector.

$\vec{X}$	=	reference-domain coordinates	$\vec{x}$	=	physical-domain coordinates
$\mathbf{u}_X$	=	state on reference domain	$\mathbf{u}$	=	physical state
$\vec{\mathbf{q}}_X$	=	gradient variable on reference domain	$\vec{\mathbf{q}}$	=	physical gradient variable
$\vec{\mathbf{H}}_X$	=	flux vector on reference domain	$\vec{\mathbf{H}}$	=	flux vector on physical domain
dA	=	differential area on reference domain	da	=	differential area on physical domain
$\vec{N}$	=	normal vector on reference domain	$\vec{n}$	=	normal vector on physical domain
V	=	reference domain (static)	$v(t)$	=	physical domain (dynamic)
$\mathcal{G}$	=	mapping Jacobian matrix	$\vec{v}_G$	=	grid velocity, $\partial\vec{x}/\partial t$
g	=	determinant of Jacobian matrix			

Using a general mapping, $\vec{x} = \vec{x}(\vec{X}, t)$, the PDE on the reference domain becomes

$$\left.\frac{\partial \mathbf{u}_X}{\partial t}\right|_{\vec{X}} + \nabla_X \cdot \vec{\mathbf{H}}_X(\mathbf{u}_X, \nabla_X \mathbf{u}_X) = \mathbf{0}, \tag{3.32}$$

where $\mathbf{u}_X = g\mathbf{u}$, $\vec{\mathbf{H}}_X = g\mathcal{G}^{-1}\vec{\mathbf{H}} - \mathbf{u}_X\mathcal{G}^{-1}\vec{v}_G$, and ∇_X denotes the gradient with respect to the reference coordinates. The transformed flux, $\vec{\mathbf{H}}_X$, separates into inviscid and viscous contributions,

$$\vec{\mathbf{H}}_X = \vec{\mathbf{F}}_X + \vec{\mathbf{G}}_X, \qquad \vec{\mathbf{F}}_X = g\mathcal{G}^{-1}\vec{\mathbf{F}} - \mathbf{u}_X\mathcal{G}^{-1}\vec{v}_G, \qquad \vec{\mathbf{G}}_X = g\mathcal{G}^{-1}\vec{\mathbf{G}}. \tag{3.33}$$

Eqn. 3.32 is the form to which the DG method is applied. The mapping Jacobian determinant, g, might not be a polynomial in $\vec{X}$, which leads to slight conservation errors that can be mitigated with a geometric conservation law (GCL) [19, 20]. Such a GCL is not used in this work as the conservation errors decrease with higher-order approximation and adaptation [20].

3.3.7 Discrete Adjoint for Discontinuous Galerkin Discretization

Discrete adjoint is used for error estimation computation along with mesh motion for discontinuous Galerkin discretization to perform r-adaptation, which had been successfully implemented for our in house code, Xflow, more details could be found in [20, 54].

3.4 Summary

The research contribution from this chapter include,

- Developed the mesh motion algorithm for the Active Flux method, which turns out to be fully conservative.
- Verified our theoretical work on adjoint-based error estimation with mesh motion for the Active Flux method.

CHAPTER IV

h-Adaptation

4.1 Introduction

This chapter presents algorithms for adapting computational meshes using output-based techniques, primarily for the Active Flux discretization, but also for the discontinuous Galerkin method. The adaptation mechanics considered in this chapter is referred to as h-adaptation: refinement or coarsening of the computational mesh.

h-Adaptation can be broadly classified into two categories: local h-adaptation and global re-meshing. Local h-adaptation changes the mesh resolution by applying local mesh operations, such as cell subdivision, node insertions/deletions, edge collapse, etc. On the other hand, global re-meshing refers to the generation of a new mesh for the whole computational domain, often employing a Riemannian metric field to encode the desired mesh size and orientation [55, 56].

Output-based h-adaptation has been studied for various discretizations, including the finite volume method and the discontinuous Galerkin method [38]. This chapter will focus primarily on the study of h-adaptation for the newly-developed Active Flux scheme. The h-adaptation strategies developed for the Active Flux method are then applied to several representative problems. The mechanics consist of local adaptation: uniform refinement of cells, followed by connection of newly-added edge

midpoint nodes to neighboring vertices, to avoid hanging nodes.

Iinitially we will work with aggregate spatial adaptation indicators obtained by summing over all time steps, i.e. marginalizing in time. In this setting, the error associated with a coarse cell j_H is

$$\epsilon_{j_H} = \sum_{n_h=1}^{N_h} \sum_{j_h=1}^{M_h} |\varepsilon_{j_h}^{n_h}|. \tag{4.1}$$

Local h-adaptation could be done uniformly or anisotropically. Currently, all of our h-adaptation is uniform, although we did explore at a preliminary level the possibilities of conducting anisotropic h-adaptation for the Active Flux method, as described in Appendix B.

4.2 h-Adaptation for the Active Flux Method

Figure 4.1 shows a flow chart that explains the organization of the Active Flux h-adaptation mechanics. Note that the adjoint solution is required for output-based adaptation, which targets individual outputs. Following the flow and adjoint solutions, the application of h-adaptation requires error localization and mesh adaptation.

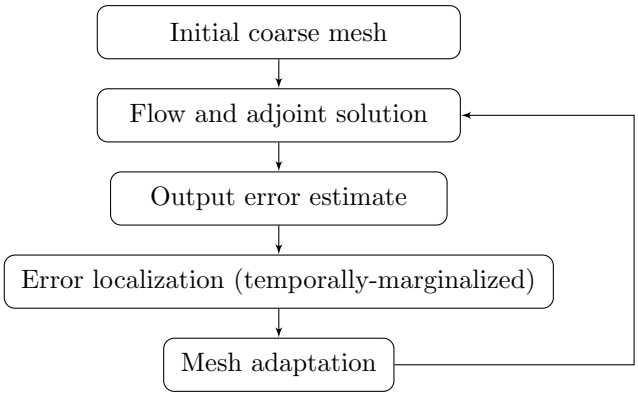


Figure 4.1: Output-based adaptation flowchart.

4.2.1 Error Localization

To obtain an adaptive indicator for the Active Flux method, we localize the error estimate in Eqn. 2.65 to space-time elements of the fine-space discretization:

$$\delta J = -\int_T \int_\Omega \psi_h \, r(u_H) \, d\Omega \, dt = -\sum_{k=1}^{N_t} \sum_{e=1}^{N_{\text{cell}}} \int_{t^k}^{t^{k+1}} \int_{\Omega_e} \psi_h \, r(u_H) \, d\Omega \, dt, \tag{4.2}$$

where N_t is the number of time steps and N_{cell} is the number of elements. Although this formula is based on the continuous adjoint, it has a parallel in the discrete adjoint approach.

Due to the fully-explicitly nature of the Active Flux method, spatial and temporal errors are tightly coupled and cannot be easily separated. Instead of separating these two types of error, we localize total error estimate values to individual element contributions. The total error estimate is

$$\text{Error}_{\text{total}} = \sum_{k=1}^{N_t} \sum_{e=1}^{N_{\text{cell}}} \text{Error}_{k,e} \tag{4.3}$$

The contribution of space-time element k, e is, with an absolute value,

$$\text{Localized Error} = |\text{Error}_{k,e}| \tag{4.4}$$

This indicator then drives space-time adaptation, both in the discrete and continuous adjoint approaches.

When using the discrete adjoint, we must decide how to map the fine-space residual, $\mathbf{R}(\mathbf{U}_h^H)$, back to the coarse space, both spatially and temporally. In time, we associate coarse time step $n_H, 1 \leq n_H \leq N_H$, to the residuals of the two fine-space updates contained within n_H, namely $n_h = 2n_H - 1$ and $n_h = 2n_H$. Regarding the spatial cells, we associate cell-average residuals directly to their host cells, and we split vertex residuals evenly between the adjacent cells. In one dimension, the

resulting error contribution of cell j at time step n is

$$\varepsilon_j^n = \underbrace{\frac{1}{2}\mathbf{\Psi}^n_{\text{vertex},j}\mathbf{R}^n_{\text{vertex},j} + \frac{1}{2}\mathbf{\Psi}^n_{\text{vertex},j+1}\mathbf{R}^n_{\text{vertex},j+1}}_{\text{vertex residual contribution}} + \underbrace{\mathbf{\Psi}^n_{\text{cell avg},j}\mathbf{R}^n_{\text{cell avg},j}}_{\text{cell average residual contribution}} \tag{4.5}$$

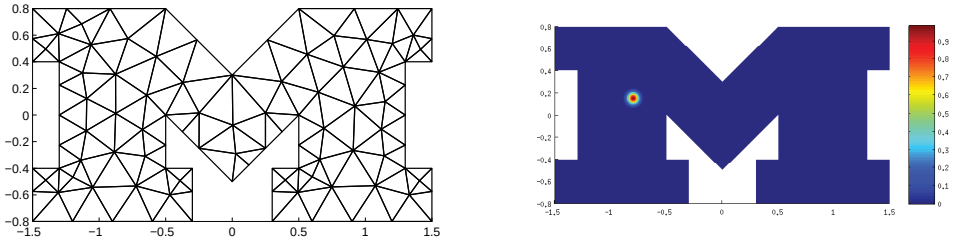
In two dimensions, we split node and edge errors to adjacent elements. Suppose that each node in our mesh is adjacent to s_n elements, and each edge is adjacent to $s_e = 2$ elements. The resulting error contribution of cell j at time step n is

$$\varepsilon_j^n = \underbrace{\sum_{i=1}^{3}\frac{1}{s_n}\mathbf{\Psi}^n_{\text{node},i}\mathbf{R}^n_{\text{node},i}}_{\text{node residual contribution}} + \underbrace{\sum_{e=1}^{3}\frac{1}{s_e}\mathbf{\Psi}^n_{\text{edge},e}\mathbf{R}^n_{\text{edge},e}}_{\text{edge residual contribution}} + \underbrace{\mathbf{\Psi}^n_{\text{cell avg},j}\mathbf{R}^n_{\text{cell avg},j}}_{\text{cell average residual contribution}} \tag{4.6}$$

The error indicator for a coarse cell/time-step is then taken as the absolute value of the sum of ε_j^n over the children cells/time-steps.

4.2.2 Continuous versus Discrete Adjoint Based h-Adaptation

This section characterizes the difference in performance of discrete versus continuous adjoint-based h-adaptation through a detailed numerical study. The initial problem setup is described in Figure 4.2(a) and Figure 4.2(b), a Gaussian pulse advecting through a 24-sided polygon.



(a) Initial mesh

(b) Primal problem illustration, $t = 0$

Figure 4.2: Output error convergence for various refinement strategies. Application of theoretical work on Michigan 'M' mesh.

The Michigan 'M' mesh is representative of a complex domain. Inflow and outflow boundary conditions are enforced on the border of the 'M' mesh depending on the bulk flow velocity. The initial mesh is shown in Figure 4.2(a). The flow advection velocity is $[2,\ 0.1]$. A Gaussian pulse originally centered at $[-0.8,\ 0.15]$, as shown in Figure 4.2(b), advects with the flow until its center arrives at the coordinate $[0.8,\ 0.23]$. The output is a domain weighted integral computed at the final time,

$$J = \int_{\Omega} w\,(x, y)\, u\,(x, y)\, d\Omega, \qquad t = t_{\text{end}} \tag{4.7}$$

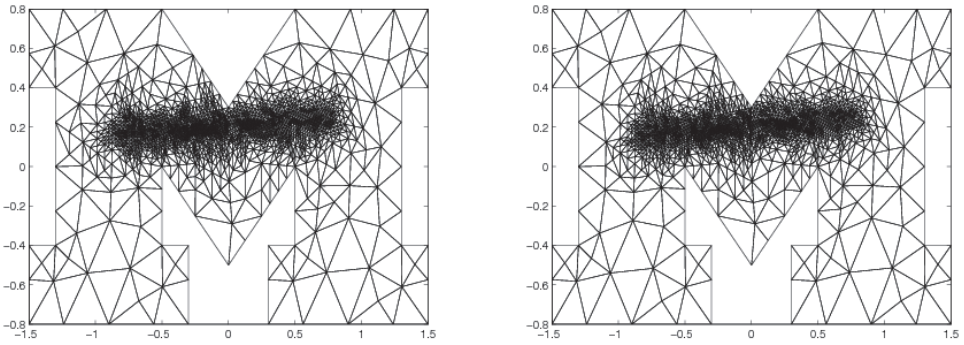
$$w = c \sin(x\pi) \sin(y\pi), \qquad c = 100.00 \tag{4.8}$$

The domain integral output in Equation 4.7 is formulated with a smooth weight so that the adjoint terminal condition is also smooth.

The discrete and continuous adjoint derivations for weighted domain integral outputs, using the Active Flux method, can be found in previous works [17, 18]. Both adjoints are tested and pass sensitivity and error estimation tests [16–18].

We conduct h-adaptation using the discrete and continuous adjoint, and we then compare the adaptation results by plotting the adapted meshes. Figure 4.3 shows

the final adapted meshes, and Figure 4.4 shows the convergence of the output error versus degrees of freedom.



(a) Driven by discrete adjoint, 8723 cells

(b) Driven by continuous adjoint, 9149 cells

Figure 4.3: Michigan "M": final adapted meshes.

Qualitatively, the comparison in Figure 4.3 shows that meshes adapted by the discrete adjoint and continuous adjoint resemble each other. Both adaptations concentrate on the part the computational domain swept out by the advecting Gaussian pulse. No substantial differences are evident in the domain areas targeted for refinement between the continuous and discrete adjoints.

Figure 4.4 shows a quantitative study of the error evolution with mesh refinement. Using uniform mesh refinement as a reference curve, both discrete-adjoint-based adaptation and continuous-adjoint-based adaptation converge similarly.

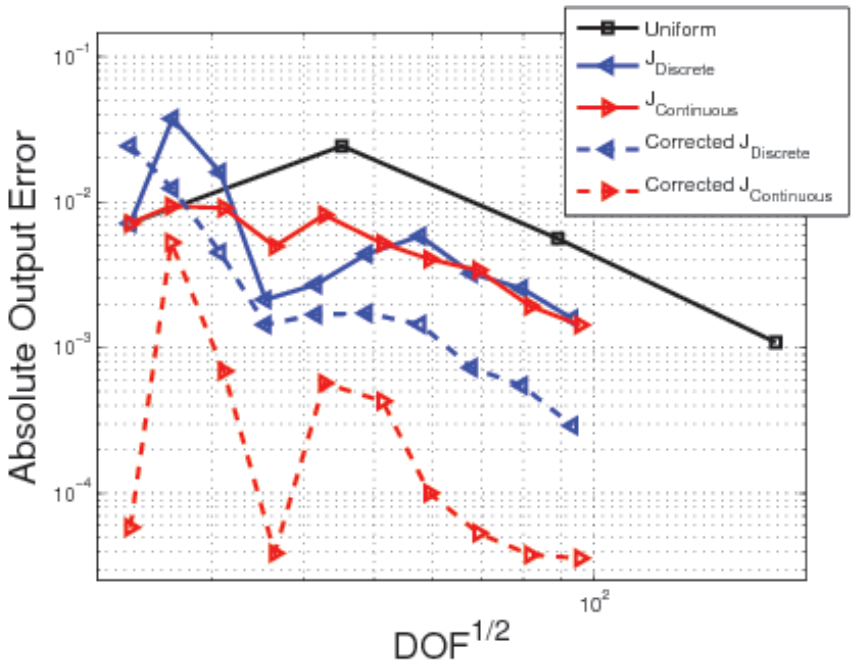


Figure 4.4: Michigan "M": output error convergence with degrees of freedom, for both the discrete and continuous adjoint methods.

Output-based methods generate error estimates for the targeted outputs. Error estimates can correct outputs, and thus improve the accuracy of the outputs. In Figure 4.4, the error level in the continuous adjoint corrected output is around one magnitude lower than in the discrete adjoint corrected output.

Consistent or asymptotically-consistent discrete adjoints produce error estimates that converge at the same rate as the corresponding continuous adjoint. In Figure 4.5, we compare the discrete adjoint error estimates and the continuous adjoint error estimates, under uniform refinement. For smooth problems, i.e.õnes with no singularities in the solution, uniform mesh refinement and adaptive mesh refinement yield the same convergence rate, asymptotically. Through Figure 4.5, we observe that the continuous adjoint and the discrete adjoint error estimates do asymptotically converge at the same rate.

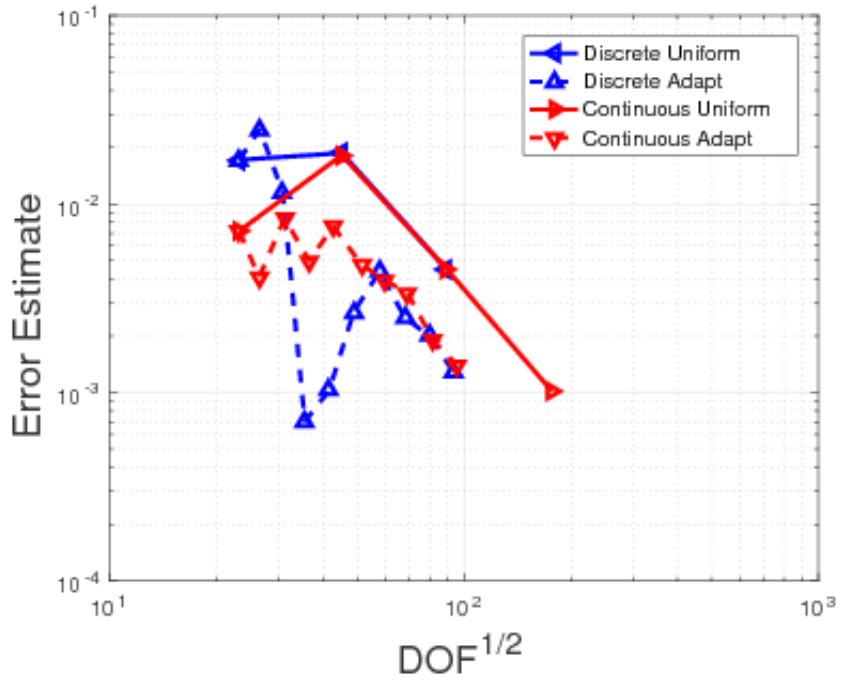


Figure 4.5: Michigan "M": convergence of the output error estimate for uniform and adapted mesh sequences. The slopes of all four curves in the asymptotic range are approximately 3.0.

As our studies show, there is not a significant performance difference between discrete adjoint-based h-adaptation and continuous adjoint-based h-adaptation. Continuous adjoints do have the disadvantage of requiring changes to the derivation for any new combination of boundary condition and output definition. However, for lean discretizations such as the Active Flux method, they can yield performance benefits due to a simpler implementation (similar to the primal) and lower storage requirements (no matrices).

4.2.3 Additional Discrete Adjoint Based h-Adaptation Simulations

In this section, we present more results that demonstrate the benefits of output-based adaptation relative to uniform refinement. These are all implemented for the discrete-adjoint formulation of the Active Flux method. In all cases, the output of

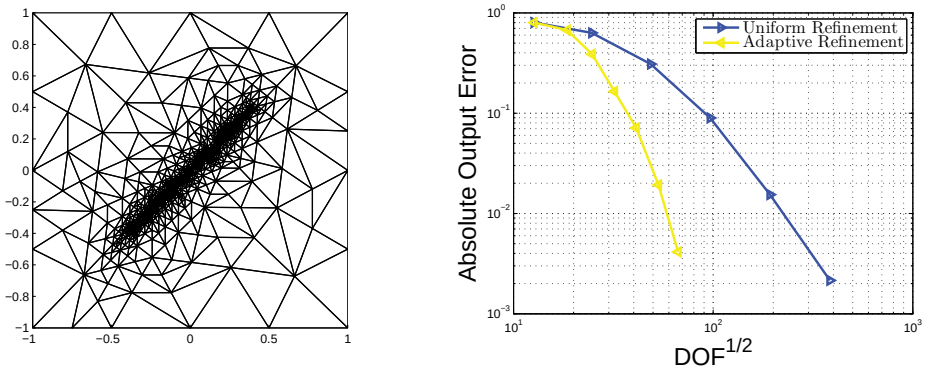
interest is a point quantity measured at the final time.

Three different computational domain geometries are considered: a square domain mesh, a crescent domain, and a circular domain. These are all very similar problems, and the primary capability tested with these simulations is the ability of the local h-adaptation mechanics to faithfully follow the prescribed refinement indicators, even on non-trivial domains. We briefly describe the problem setup for each case, present the initial mesh, the resulting adapted mesh, the primal solution, and the error convergence performance comparison between the adaptive approach and uniform mesh refinement.

Square Domain

Using the initial mesh and primal problem that has already been described previously, see Figure 2.11, we implement both adaptive mesh refinement and uniform mesh refinement to solve the scalar advection problem. The output error of each adapted mesh was recorded, and comparison results are shown in Figure 4.6(b). The adaptive refinement method uses fewer degrees of freedom but generates more accurate outputs. In addition, the benefits of adaptive refinement grow as tighter accuracy is required. The resulting final adapted mesh is shown in Figure 4.6(a). The diagonal part of the mesh, through which the scalar profile advects, is much finer than the rest of the domain, which, as expected, appears untouched by the adaptation.

The present result makes intuitive sense. Due to the advective nature of the unsteady flow field, it is not sufficient to only adapt the area where our output is defined. So, the mesh should not only be refined around the output point. Second, our point output ultimately advects from the upstream of the computational domain. Along the advection path of the flow, errors can be introduced into the simulation,



(a) Output-adapted mesh

(b) Error convergence

Figure 4.6: Square domain: adapted mesh and error convergence.

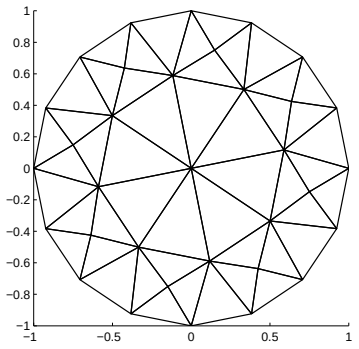
and these will eventually pollute the point output. Thus, besides adapting the area where our output is defined, the area through which the advected profile travelled should also be refined. Third, for a localized point output, not all of the area covered by the advected profile matters equally, since a point output only depends on the information from a small part of the computational domain. Figure 4.6(a) shows the area around the diagonal of the computational domain has more weight in affecting the accuracy of the output.

With the purpose of illustrating our research approach, we adopted a simple square geometry in the above discussion. However, for two-dimensional cases, the computational domain can be more complex. We therefore next test the adaptive approach on meshes with more complex boundaries.

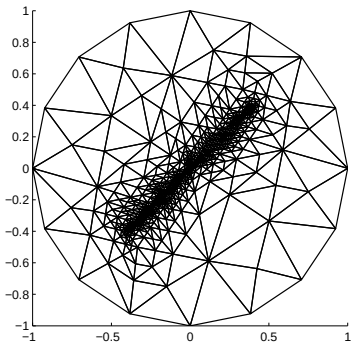
Circular Mesh

The purpose of doing this circular mesh case is solely to show that more complex boundaries can be handled by the solver. Inflow/outflow boundary conditions are dynamically enforced on the circular boundary depending on the flow field advection direction. The initial mesh is shown in Figure 4.7(a), and other than the domain

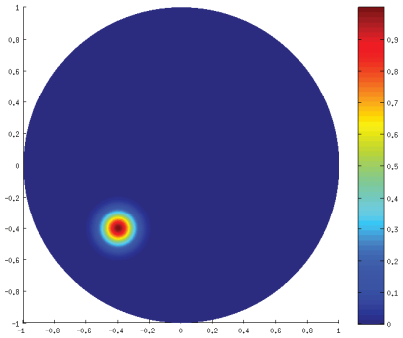
difference, the problem setup is exactly the same as the square geometry case. The primal initial condition is shown in Figure 4.7(c). We apply uniform mesh refinement and adaptive mesh refinement methods to solve this problem, and Figure 4.7(d) shows the results: the adaptive mesh refinement method outperforms the uniform mesh refinement method in degrees of freedom.



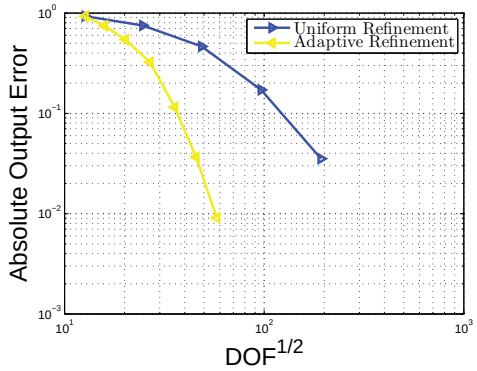
(a) Initial mesh



(b) Resulting adapted mesh



(c) Primal initial condition



(d) Error convergence comparison between adaptative mesh refinement and uniform mesh refinment

Figure 4.7: Application of theoretical work on circular mesh

Crescent Mesh

The crescent mesh is another test case of a more complex geometry. Inflow/outflow boundary conditions are again dynamically enforced on the border of this mesh. The initial mesh is shown in Figure 4.8(a). The flow field advection velocity is $[1,\ 0.1]$. A Gaussian pulse originally centered at $[-0.4,\ 0]$, Figure 4.2(c), advects with the

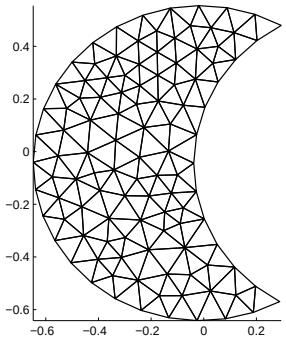
flow until it arrives at $[-0.1,\ 0.03]$. The output of interest is the point output at $[-0.1,\ 0.03]$. We apply uniform mesh refinement and adaptive mesh refinement on this problem Figure 4.8(d) shows the results: again, the adaptive mesh refinement method converges faster than the uniform mesh refinement method.

From these results, we observe that varying the geometry doesn't significantly affect the performance of the developed mesh adaptation mechanics. This is expected because the advection region of interest is not affected by the boundaries. In addition, we see that the adapted mesh conveys information about what happened in the unsteady simulation. The darkened areas of the meshes in Figure 4.6(b), Figure 4.2(b), Figure 4.7(b) and Figure 4.8(b) indicate the region through which the profile advected before being measured at the point output. The advective nature of the physics is clearly evident in the resulting adapted mesh.

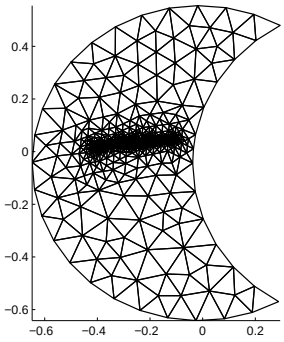
4.3 Comparison of Active Flux and Discontinuous Galerkin h-Adaptation

Adaptive methods have been studied extensively for the discontinuous Galerkin discretization [38, 40, 41]. In addition, recent works have compared the performance of DG scheme and the Active Flux scheme. In this section, we address the question of how DG and Active Flux compare in an h-adaptive setting.

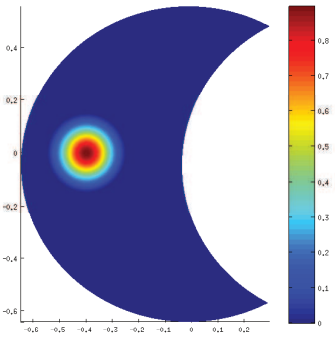
For the same test case described in Section 4.2.3, we perform h-adaptation using the discontinuous Galerkin method. The DG code for this comparison is Xflow, and the adaptation is output-based. The details of the adaptive method are outlined in previous work [57]. Briefly, we use a implicit time marching, from the diagonally-implicit Runge-Kutta family. The output error is estimated through the solution of a spatially-discrete but temporally-continuous unsteady adjoint, marched backwards in time from the terminal condition dictated by the output. The error estimate at



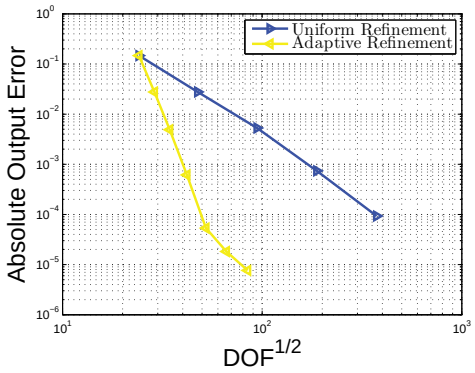
(a) Initial mesh



(b) Resulting adapted mesh



(c) Primal problem illustration



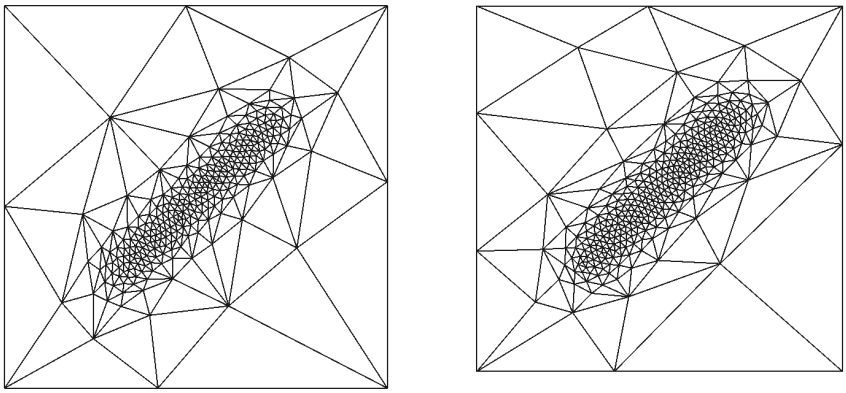
(d) Error convergence comparison between adaptative mesh refinement and uniform mesh refinment

Figure 4.8: Application of theoretical work on crescent mesh

each space-time element is then marginalized to spatial elements through a summation of absolute values over time steps. In addition, error estimates associated with refinement samples, to be used for mesh optimization [57, 58], are also marginalized in time. These samples drive unstructured spatial mesh adaptation and global remeshing using the BAMG mesh generation code.

A difference between the Active Flux and the discontinuous Galerkin simulations is the definition of the output. Whereas the AF output consists of a point value, the DG output is a weighted integral of the final-time state, to make the terminal condition for the adjoint well-posed. However, the outputs are similar as the weight function chosen for the DG output is a two-dimensional Gaussian with a sharp peak at the point output location. The standard deviation for this Gaussian is $\sigma = .05$.

Figure 4.9 shows two adapted meshes generated using $p = 1$ (DG1) and $p = 2$ (DG2) spatial approximation, respectively. As shown in Figure 4.9, the meshes are



(a) DG1, DIRK3, 5^{th} spatial adaptation iteration

(b) DG2, DIRK4, 5^{th} spatial adaptation iteration

Figure 4.9: Adapted meshes for DG1 and DG2, respectively.

not significantly different. This is because the adapted meshes are optimized for the same total space-time degree of freedom count. If the time stepping schemes of one

of the methods were of a lower order, different sized spatial meshes would arise due to a different weighting between spatial and temporal degrees of freedom. However, these results use DIRK3 and DIRK4 time stepping, and these are consistent with the spatial orders of convergence.

We now perform a quantitative study of the h-adaptation performance difference between DG and Active Flux. The h-adaptation mechanics in the Active Flux solver consist of a local operations. On the other hand, h-adaptation in Xflow uses *BAMG*, the Bidimensional Anisotropic Mesh Generator, library and MOESS, mesh optimization through error sampling and synthesis [59].

To compare DG and AF outputs, we scale data from the DG solver to be representative of the point output, by dividing the DG output by the integral of the weight function. Figure 4.10 shows the resulting output convergence.

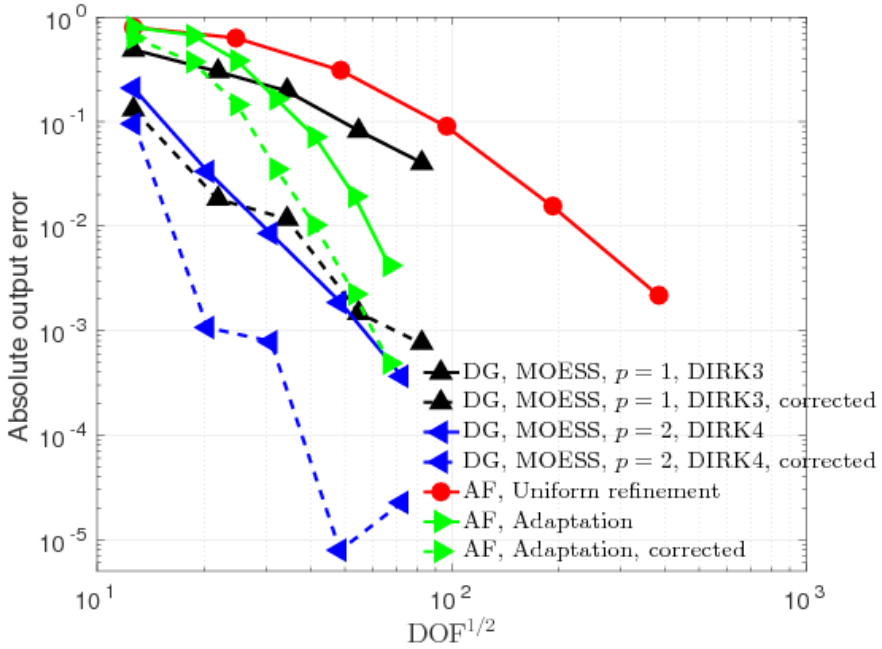


Figure 4.10: Convergence of h-adapted DG and AF for a scalar advection simulation.

As illustrated in Figure 4.10, the adaptation paths of DG h-adaptation and Ac-

tive Flux adaptation are clearly different. Both schemes show benefits from error estimation, with reduced error levels in the corrected output error curves, and they all perform better in terms of degrees of freedom relative to uniform refinement. In particular, DG1 h-adaptation behaves comparably to Active Flux h-adaptation. On the other hand, DG2 h-adaptation performs better than both DG1 and Active Flux based h-adaptation, which can be explained as increased accuracy in the error estimate and primal solve due to increased solution approximation order.

4.4 Summary

The research contribution from this chapter include,

- Described an error localization and adaptation strategy for the Active Flux method. Several scalar advection test cases demonstrated the efficiency and accuracy improvements of the adaptive method compared to uniform refinement.
- Compared discontinuous Galerkin (DG) h-adaptation and the Active Flux (AF) method h-adaptation. AF h-Adaptation showed comparable performance to DG1, whereas higher-order DG fared better in terms of degrees of freedom for the smooth problem tested.

CHAPTER V

Adaptation Acceleration

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 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 in detail in [38].

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 chapter, 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 a fine-space adjoint solution for more than just one adaptation iteration. The second shortcut relies on using 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.

5.1 The Adjoint-Weighted Residual

Output-based error estimates rely on the concept of an adjoint-weighted residual [32, 33, 38, 60]. 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 $J_H(\mathbf{U}_H) = J_h\left(\mathbf{U}_h^H\right)$, 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 [32, 38] reads

$$\underbrace{J_h\left(\mathbf{U}_h^H\right) - J_h\left(\mathbf{U}_h\right)}_{\delta J} = \boldsymbol{\Psi}_h^T \delta\mathbf{R}_h = -\boldsymbol{\Psi}_h^T \mathbf{R}_h\left(\mathbf{U}_h^H\right), \tag{5.1}$$

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

$$\left(\frac{\partial \mathbf{R}_h}{\partial \mathbf{U}_h}\right)^T \boldsymbol{\Psi}_h + \frac{\partial J_h}{\partial \mathbf{U}_h} = \mathbf{0}. \tag{5.2}$$

$\boldsymbol{\Psi}_h$ 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. 5.1 are not trivial, as we can see by breaking down each of the terms:

$$\delta J \approx \underbrace{-\boldsymbol{\Psi}_h^T}_{\boxed{\text{fine space adjoint}}} \underbrace{\mathbf{R}_h}_{\boxed{\text{fine space residual operator}}} \Big(\underbrace{\mathbf{U}_h^H}_{\boxed{\text{injected state}}} \Big). \tag{5.3}$$

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. 5.3 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 several techniques 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 two sections we thus also introduce an approach to avoid fully-repeating this calculation at each adaptation iteration.

5.2 Active Flux Method Adaptation Acceleration

5.2.1 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 the solution regularity and certain choices in the error estimation procedure [32]. 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.

We present one particular shortcut for bypassing a rigorous formulation of adjoit based error estimation: 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

$$\delta J \approx -\mathbf{\Psi}_H^T \mathbf{R}_H(\mathbf{U}_H^{\tilde{H}}), \tag{5.4}$$

where $\mathbf{U}_H^{\tilde{H}}$ is the "coarser" solution injected into the current space. Eqn. 5.4 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. 5.4 could still provide useful adaptive information (albeit with a possible lag in adaptive iterations). Figure 5.1 illustrates schematically the use of the coarser space.

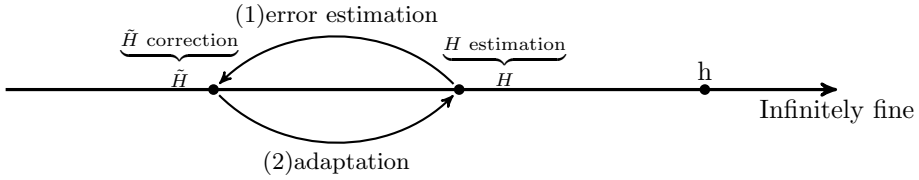


Figure 5.1: Illustration of discretization spaces used for coarser-space error estimation and adaptation.

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

In order to implement our idea, we need to come up with strategies to create a 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 5.2. To create discretization errors due to the drop of state approximation error, we have

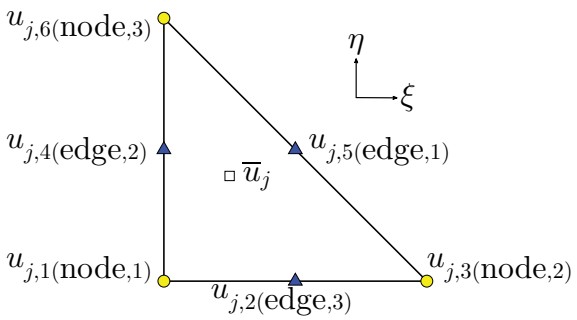


Figure 5.2: Unknown placement in the Active Flux method.

many options. Below we outline five strategies.

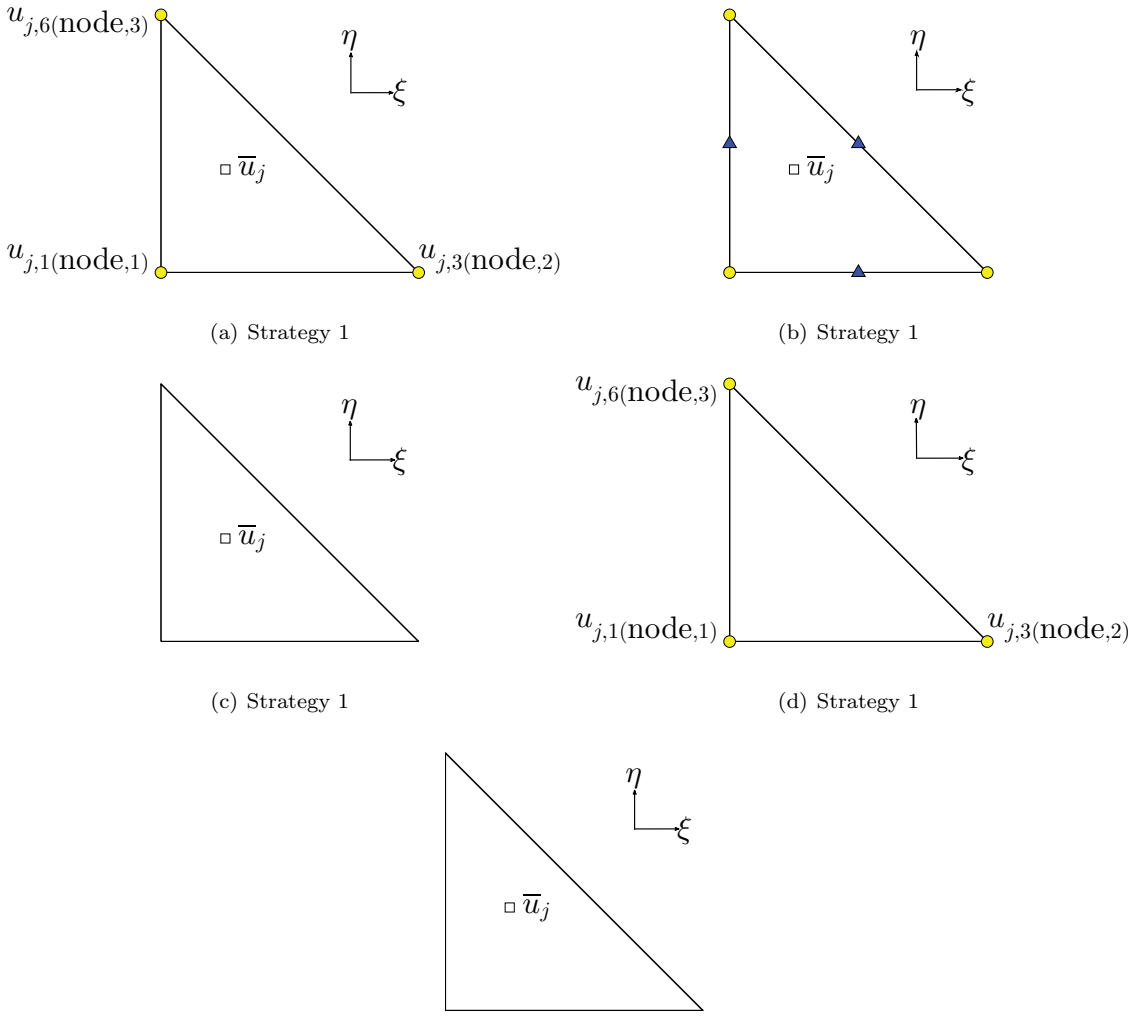
- **Strategy 1**

  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 5.3(a).

  We then treat the remaining three node states as basis coefficients in a linear approximation of the state, as in a $p = 1$ finite-element method. New edge states are calculated by interpolating 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 keep the cell average state, as shown in Figure 5.3(b). Looping over elements in a predefined arbitrary order, whichever nodes and edges are hit first become the new node and edge states. As a result, the approximation is ad hoc and non-unique. We lose accuracy in our approximation. Specifically, we expect first-order accuracy.



(a) Strategy 1

(b) Strategy 1

(c) Strategy 1

(d) Strategy 1

(e) Strategy 1

Figure 5.3: Illustration of five strategies proposed for coarsening the approximation space in the Active Flux method.

- **Strategy 3**

  In this strategy, we only keep the cell average state, as shown in Figure 5.3(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 5.3(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 neighboring elements in the Active Flux method, whenever we have multiple 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 shown in Figure 5.3(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.

  The Active Flux method places an unknown in the center of the cell that is

equal to the cell average. Accordingly, the single state least-square projection value turns out to be the same as the cell average value. Consequently, this strategy serves as a canonical test for strategy 4 as well as an efficient approach for adaptation.

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.

5.2.2 Coarse-Space Error Estimate Instructed h-Adaptation

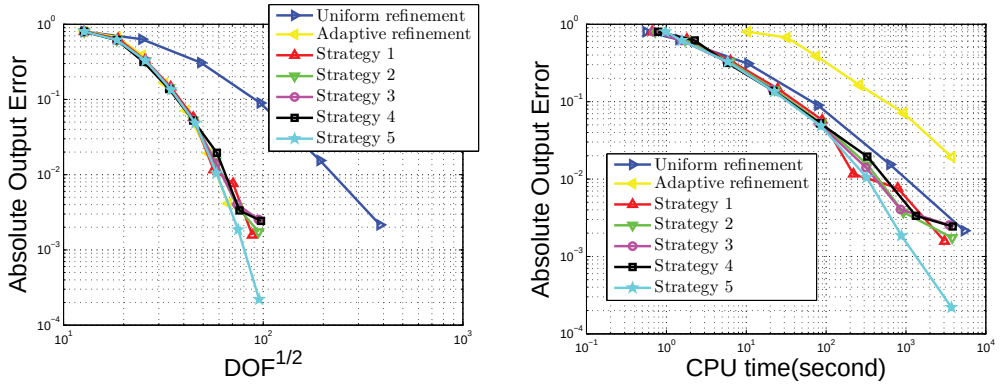
In this section, we present results showing how the use of a coarser space fares in driving mesh adaptation in the Active Flux method, i.e., coarse-space error estimate instructed h-adaptation.

The current mesh adaptation mechanics is static, meaning that the mesh does not change in time. According to the discussion in Section 5.2.1, 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. At present, we test the coarse-space selection strategies outlined in Section 5.2.1 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 2.11. Figure 2.11(a) shows the initial unstructured mesh, and Figure 2.11(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 the 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.



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

(b) Error convergence comparison in terms of computational time

Figure 5.4: Scalar advection with Active Flux: error convergence comparison.

Figure 5.4(b) compares the absolute error convergence performance of conventional mesh adaptation, all five of our "adaptation acceleration" strategies, and uniform refinement. In our implementation, conventional output-based adaptation is only able to outperform uniform mesh refinement when using spatial degrees of freedom to measure cost. With this measurement, we find all five curves of "adaptation acceleration" strategies in Figure 5.4(a) have very similar behavior 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 5.4(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 5.4(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, it may be possible to make use of the information in the error history to predict the remaining error on the current mesh. Specifically, future work may consider a priori error estimates and extrapolation techniques, both of which are expected to be valid in the asymptotic regime when these error estimates would be necessary.

5.3 Discontinuous Galerkin Adaptation Acceleration

5.3.1 Adaptive Sub-Iterations

Figure 5.5 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 estimate then drives adap-

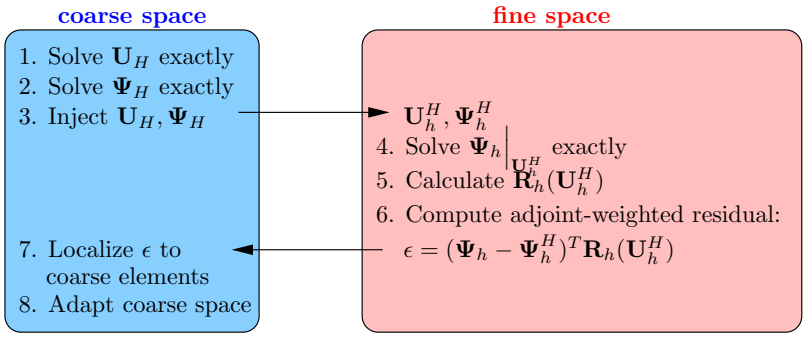


Figure 5.5: Schematic of a "standard" error estimation and adaptation iteration in which the fine space adjoint is solved exactly at every iteration.

tation; 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 [20, 61]. However, the error estimates often suffer when using such approximations.

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 5.6 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 cost. Note that in these

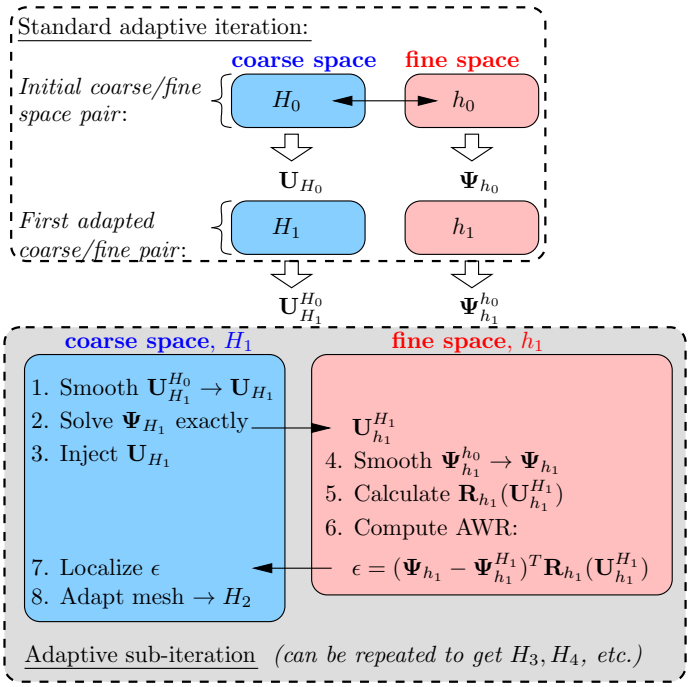


Figure 5.6: Schematic of the proposed error estimation and adaptation iteration in which approximate sub-iterations piggy-back on a standard adaptive iteration. In particular, the fine-space adjoint solve is reused in the sub-iterations, where it is only smoothed via an inexpensive iterative solver, thereby saving computational time compared to the standard approach in which the fine space adjoint is re-solved on every adaptive iteration.

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.

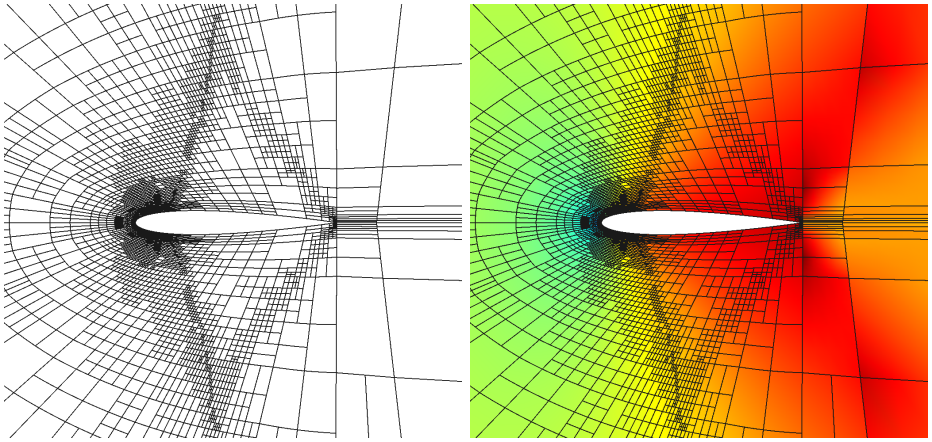
5.3.2 Adaptive Sub-Iteration Results

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

Transonic airfoil with a fishtail shock

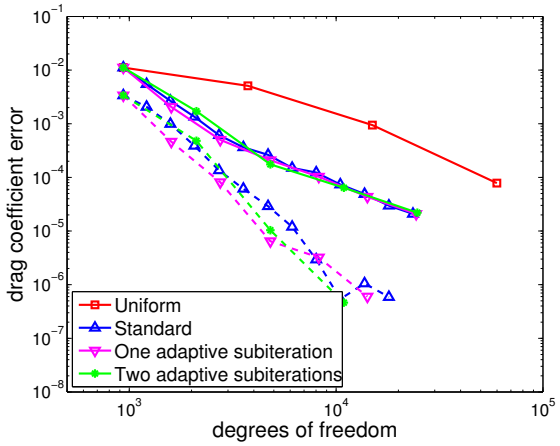
We first consider a NACA 0012 airfoil in inviscid (Euler) $M = 0.95$ flow at $\alpha = 0$. The flow is transonic and we use element-wise artificial viscosity, discretized using the second-form of Bassi and Rebay [43], to stabilize the solution. We consider drag prediction using an approximation order 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 5.7 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 5.7, 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

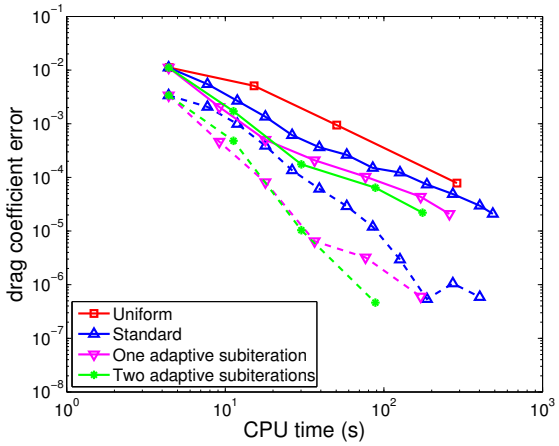


(a) Final drag-adapted mesh

(b) Mach contours (0 to 1.5)



(c) Drag convergence with DOF



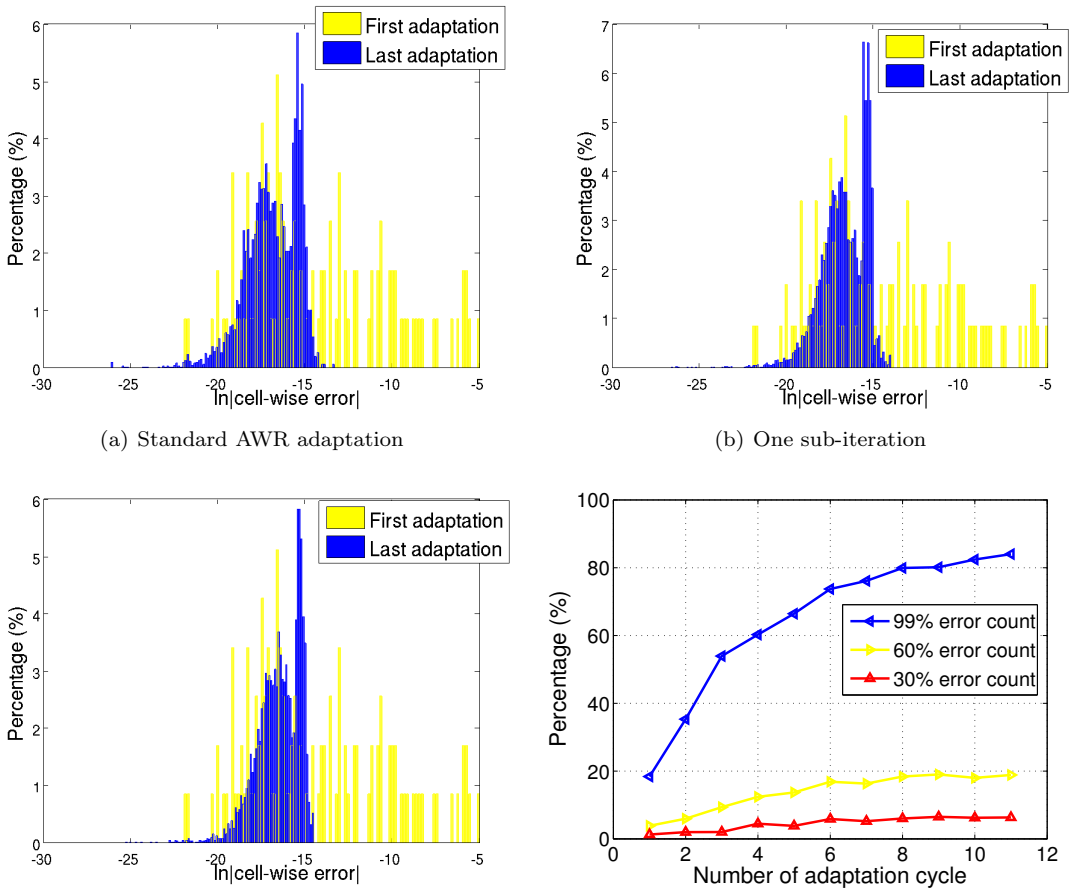
(d) Drag convergence with CPU time

Figure 5.7: 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. CPU wall time is measured by running our code, Xflow, with one processor, on a fully subscribed Haswell architecture compute node configured with 24 cores, two twelve-core 2.5 GHz Intel Xeon E5-2680v3 processors.

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 the adaptive indicator that dictates which elements are chosen for refinement. Furthermore, the outputs corrected by the error estimates (dashed line) 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 5.7(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 latter adaptations, the benefit of sub-iterations is at times as much as an order of magnitude error reduction for a given computational time.

Figure 5.8(a) shows 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 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 5.8(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 5.9 further illustrates this point: both the mean and standard deviation of the error indicator drop with each adaptive



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

iteration.

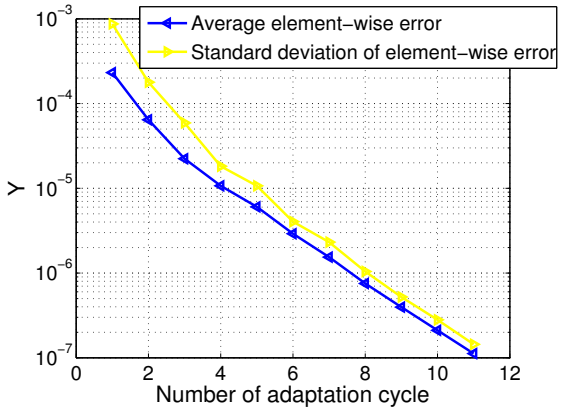
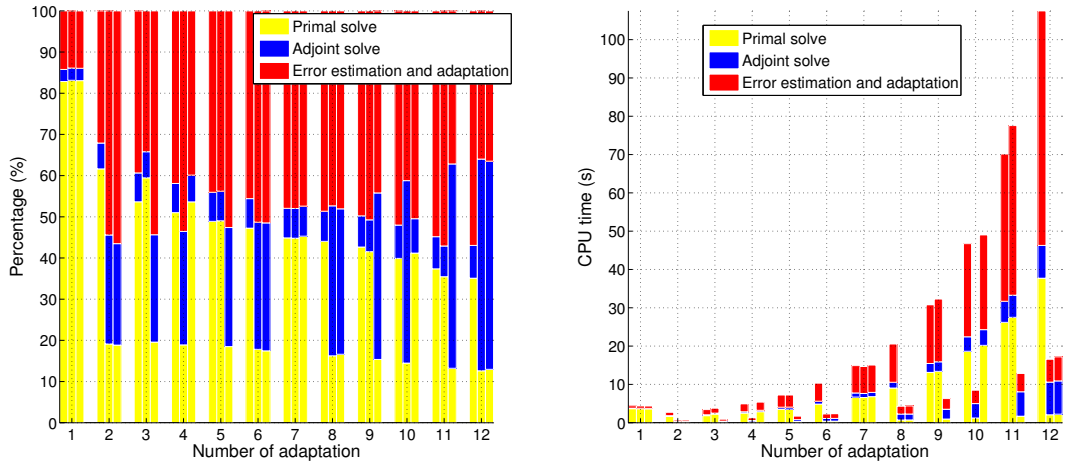


Figure 5.9: NACA 0012, $M = 0.95$, $\alpha = 0°$: convergence of the mean and standard deviation of the error indicator with adaptive mesh refinement for the standard adjoint-weighted residual method.

Figure 5.10 shows the CPU time breakdown for the different adaptation strategies. Figure 5.10(a) shows the CPU time percentage breakdown and Figure 5.10(b) shows the actual CPU time breakdown [1].

Figure 5.10(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 5.10(a) confirms this trend,

[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) Percentage CPU time breakdown

(b) Raw CPU time breakdown

Figure 5.10: 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 any fine-space solves.

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 5.7 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.

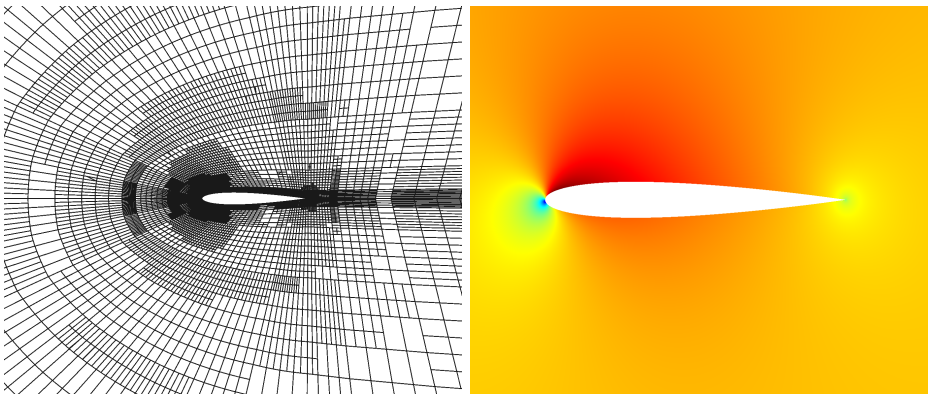
The transonic fishtail case demonstrates the strength of sub-iterations To further test the capability of the proposed technique, we next present test cases for a subsonic airfoil and a three dimensional wing.

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

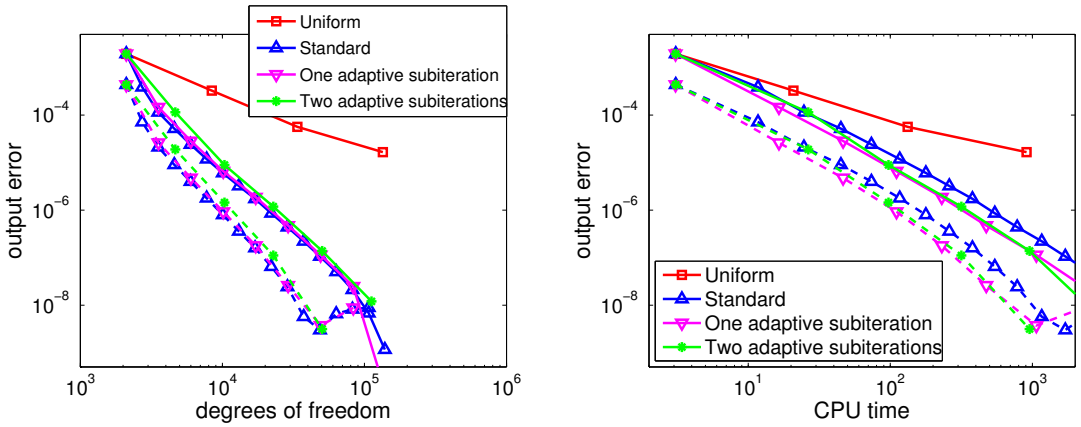
adaptation is also the same, $f = 0.1$, and the approximation order is $p = 2$.

Figure 5.11 shows a comparison of the various adaptive strategies for this case. Figure 5.11(c) shows that the methods with sub-iterations exhibit a similar output



(a) Final drag-adapted mesh

(b) Mach contours (0 to 0.72)



(c) Drag convergence with DOF

(d) Drag convergence with CPU time

Figure 5.11: 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. CPU wall time is measured by running our code, Xflow, with one processor, on a fully subscribed Haswell architecture compute node configured with 24 cores, two twelve-core 2.5 GHz Intel Xeon E5-2680v3 processors.

error convergence behavior with degrees of freedom compared to standard adaptation. However, as shown in Figure 5.11(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.

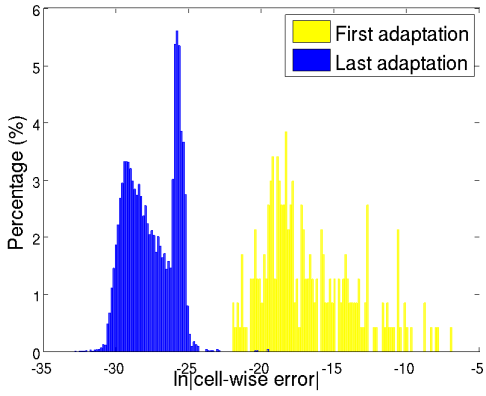
Figure 5.12 shows histograms of the error indicator distribution for the different adaptation strategies. Again, we see a similar trend for all three methods: the error distribution tightens and shifts to the left from the first to the last adaptive iteration. There are some differences in the histogram for lowest errors, but these are least important to adaptation: at the larger error values, the histograms appear very similar. Figure 5.12(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.

Figure 5.13 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.

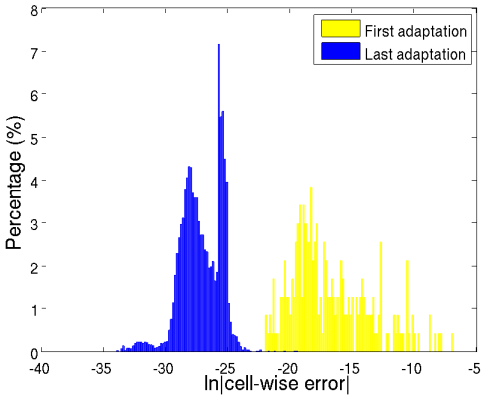
Figure 5.10 shows the CPU time breakdown for the different adaptation strategies. Figure 5.10(a) shows the CPU time percentage breakdown and Figure 5.10(b) shows the actual CPU time breakdown [2].

Figure 5.14 shows the CPU-time breakdown comparison among standard and sub-iterative adaptation. The results are similar to the fishtail case in the previous

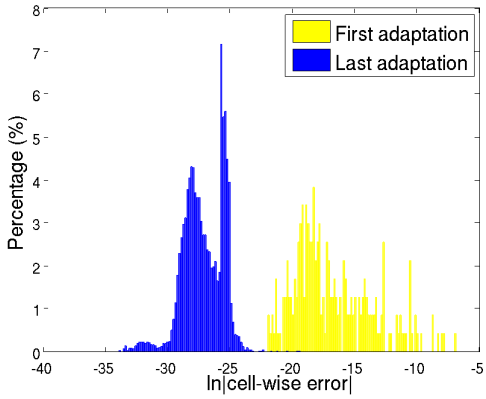
[2]Timings were performed on the University of Michigan Flux cluster, on nodes that each had two six-core 2.67 GHz Intel Xeon X5650 processors and 48GB RAM.



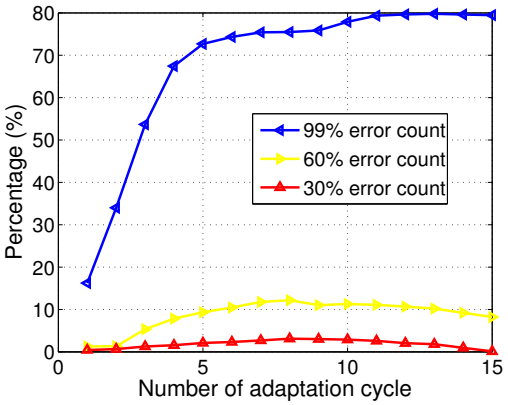
(a) Standard AWR adaptation



(b) One sub-iteration



(c) Two sub-iterations



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

Figure 5.12: NACA 0012, $M = 0.5$, $\alpha = 2°$: comparison of error indicator distributions.

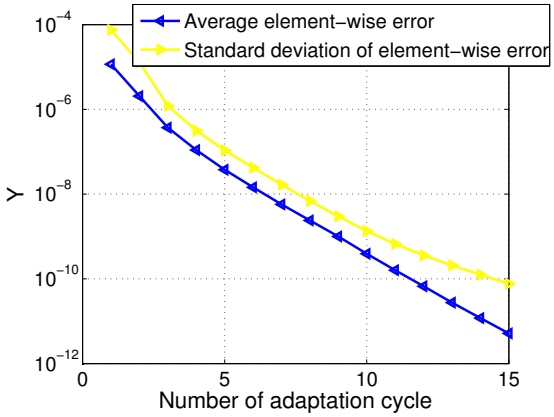
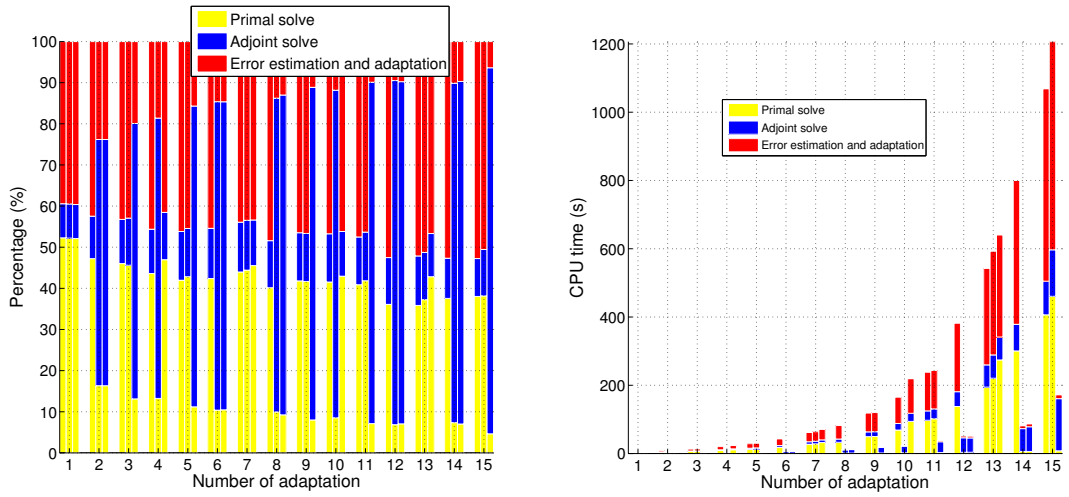


Figure 5.13: NACA 0012, $M = 0.5$, $\alpha = 2°$: convergence of the mean and standard deviation of the error indicator with adaptive mesh refinement for the standard adjoint-weighted residual method.

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 5.14(b)) while the percent of the time taken by the adjoint solve increases (Figure 5.14(a)).



(a) Percentage CPU time breakdown

(b) Raw CPU time breakdown

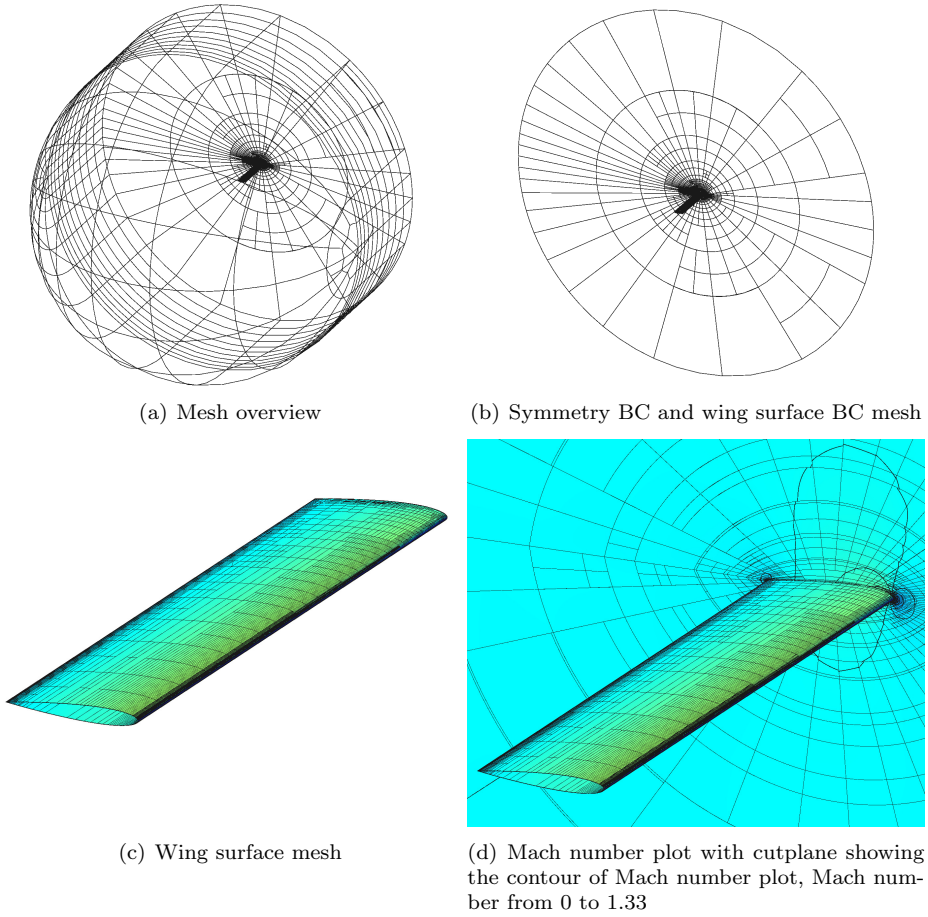
Figure 5.14: NACA 0012, $M = 0.5$, $\alpha = 2°$: CPU time breakdown results. At each of the 15 adaptive iterations, we show three bar plots, which are, from left to right: standard adaptation, adaptation with one sub-iteration, and adaptation with two sub-iterations. Each of these bars is divided vertically 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 any fine-space solves.

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 5.15 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 5.16 shows the output error con-



(a) Mesh overview

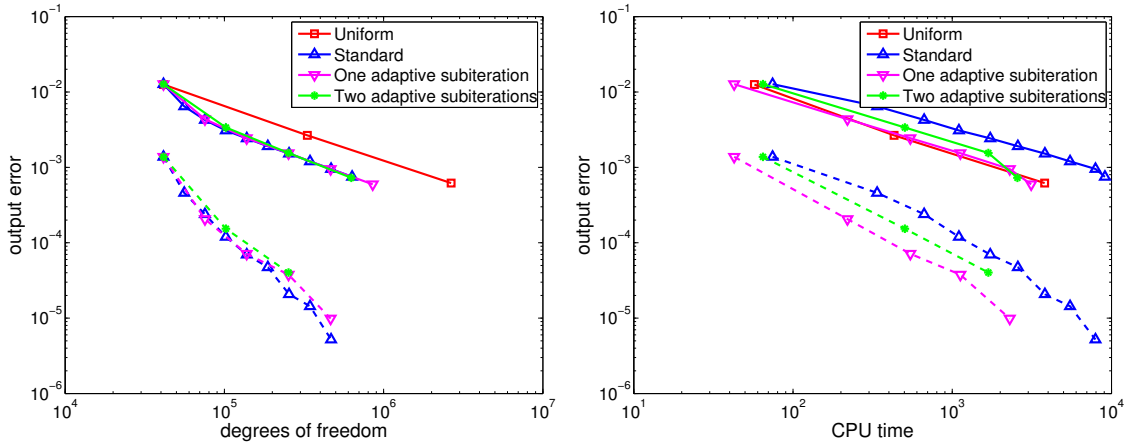
(b) Symmetry BC and wing surface BC mesh

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

vergence 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 5.17 shows the error histograms for the various methods. These again appear similar among the methods, each showing a decreasing-error trend from the



(a) Convergence with degrees of freedom

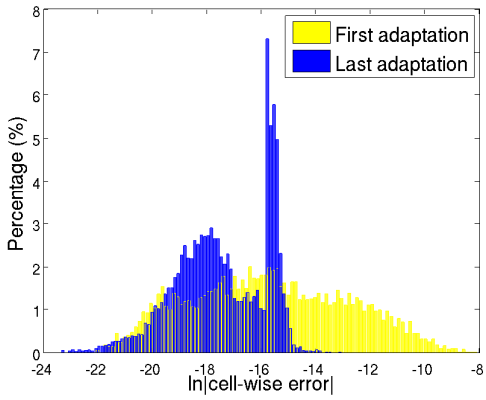
(b) Convergence with CPU time

Figure 5.16: 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. CPU wall time is measured by running our code, Xflow, with one Haswell architecture compute node, configured with 24 cores, two twelve-core 2.5 GHz Intel Xeon E5-2680v3 processors.

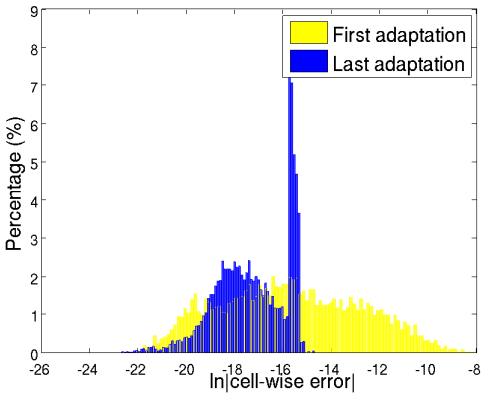
first to the last adaptation iteration. Figure 5.17(d) shows that the number of elements responsible for 99% of the error rises from 40% to just over 80% in the course of adaptation. The number of elements responsible for 30% and 60% of the error also increases but then stagnates, likely due to large error contributions from elements in singular areas of the flow, as observed in the previous section.

The CPU time breakdown is shown in Figure 5.18. We again observe a sharp drop in computational time during the sub-iterations in Figure 5.18(b), and an enlargement of the blue section of the columns, the relative adjoint solve time, in Figure 5.18(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.

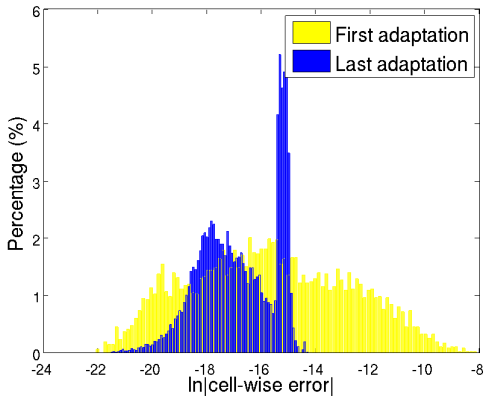
In summary, at least for the cases tested, the use of sub-iterations has a minimal



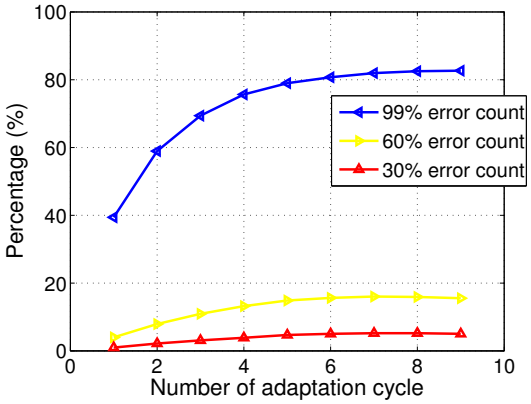
(a) Standard AWR adaptation



(b) One sub-iteration

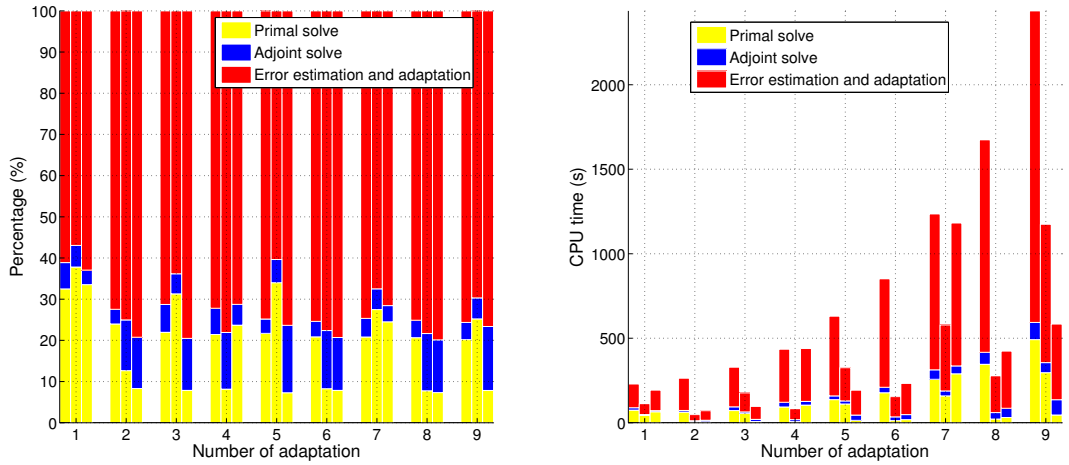


(c) Two sub-iterations



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

Figure 5.17: NACA 0012 wing, $M = 0.4$, $\alpha = 3°$: comparison of error indicator distributions.



(a) Percentage CPU time breakdown

(b) Raw CPU time breakdown

Figure 5.18: 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 any fine-space solves.

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.

5.4 Summary

In this chapter, we have presented two general techniques for accelerating output-based error estimation and mesh adaptation. The research contribution from this chapter include,

- Constructed the acceleration algorithm 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 for the discontinuous Galerkin finite-element for steady-state Euler simulations.

- Established the acceleration algorithm of coarse spaces for creating the adaptive indicator. This retrospective error estimation strategy does not 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.

CHAPTER VI

r-Adaptation

This chapter investigates an alternative approach to adapting a computational mesh that is different from standard h or p adaptation. The purpose of mesh adaptation is to reduce errors that are inherent to numerical simulations. The error in consistent and stable numerical methods depends on the mesh size, h, which can refer to a measure of the size of elements or the spacing between nodes, and on the approximation order, p, which can refer to the order of polynomials in a finite-element method or to the order of a finite-difference scheme. To reduce the error, we must decrease h and/or increase p, and this gives rise to what we call standard h [60, 62, 63], p [47, 54, 64, 65], or hp [66–70] refinement strategies.

Numerous techniques exist for adapting the size of a mesh. These include hanging-node refinement [71–73], local mesh operations [74–76], and global re-meshing [59,77]. All of these techniques have the capability to increase resolution in some areas and to decrease the resolution in other areas, so that the total number of degrees of freedom can grow, shrink, or remain the same. A related adaptation technique, that is more restricted, but relatively underutilized, is r adaptation [78–83] , in which the nodes of the mesh undergo motion in order to redistribute the resolution to areas that need it the most. The total number of degrees of freedom remains constant, and hence r

adaptation is limited in the maximum possible error reduction that it can achieve. However, when coupled with h and/or p refinement, the combined strategy yields a flexible and efficient means for reducing the output error.

In steady-state calculations, r adaptation is expected to produce final meshes that are similar to other h-refinement mechanics, when the number of degrees of freedom is held constant. For example, if global re-meshing is available, there may be no incentive to use r refinement, except possibly to reduce the cost of mesh re-generation. However, in unsteady calculations, r adaptation presents a distinct advantage in its ability to dynamically and smoothly track important features without abrupt refinement changes (e.g. sudden element splits) and without error-prone inter-grid solution transfers. As long as the mesh motion is smooth, solution techniques on deformable domains/meshes can be applied without the need for solution interpolation or projection.

r-Adaptation possesses unique properties that set it apart from h and p adaptations. These properties include intrinsic unsteadiness, global scale mesh alteration and degree-of-freedom conservation [81–83].

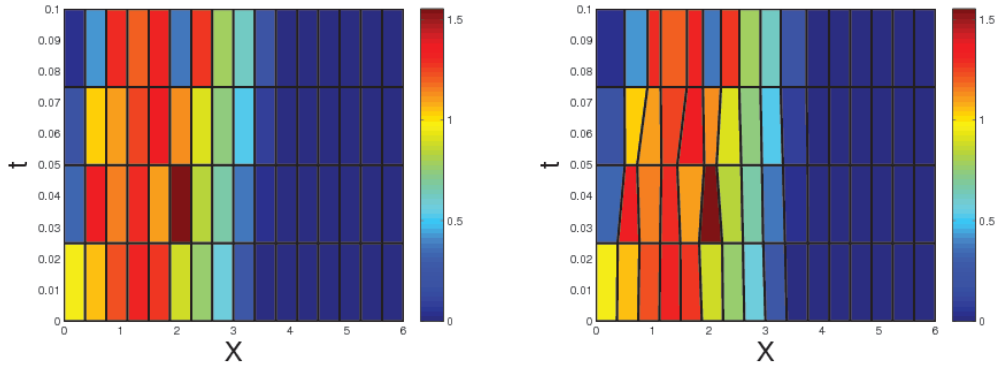
For unsteady dynamic adaptation, since solution of the adjoint problem requires backwards time marching, the motion history and the mesh topology between the current time, t_{dynamic}, and the end time, t_{end}, are unknown. As a refinement option, dynamic p-adaptation is a simple procedure for unsteady problems because it does not affect the mesh topology. However, precisely because of this lack of topology change, p-adaptation does not perform well when the solution is not smooth. Local h-adaptation does change the mesh topology, and depending on the mechanics employed, locally h-adapted meshes may remain nested and retain the original mesh topology. Dynamic adaptation via local h-adaptation can be challenging due

to the required mesh structures and state mappings, though it does offer a direct approach for increasing the degrees of freedom. Global h-adaptation, in which the entire mesh is regenerated, involves the high cost of global mesh generation at every adaptive iteration, and hence this strategy is significantly more expensive than local h-adaptation or r refinement.

The unsteady adjoint solve and residual evaluation are particularly challenging when performing r-adaptation. There are two ways to approach the solution of the unsteady adjoint equation for r-adaptation. One approach for solving the unsteady adjoint equation is, at every time step, to solve for the adjoint until the end time by assuming that there is no motion between time points t_{dynamic} and t_{end}, yielding a relatively accurate terminal condition at time point t_{dynamic}. However, this introduces $n \times (n+1)/2$ loops for the adjoint solve and the residual re-evaluation, where n is the number of time steps, and for practical purposes this is computationally unfeasible.

In our adaptation strategy, we first obtain the space-time error indicator on a prescribed motion, e.g. the r-adaptation history from running the unsteady simulation for the i^{th} time. Then, 1) for the Active Flux method, we map our existing error indicators onto the new mesh topology, while 2) for discontinuous Galerkin, we use the error indicator information from the i^{th} time step directly. Our motivation is that when performing adaptations incrementally, the changes between adjacent adaptation cycles are relatively small. We can either use the approximated error indicator or map the error indicators from the i^{th} cycle onto the $(i+1)^{\text{th}}$ cycle using projection. Figure 6.1(a) shows the localized error indicators when there is no motion, i.e. the 0^{th} cycle. Figure 6.1(b) shows the 1^{st} cycle's r-adapted error indicators distribution, mapped from the 0^{th} cycle.

In one dimension, the projected error is proportional to the length of cells. In



(a) Space-time error map before r-refinement

(b) Space-time error map after r-refinement

Figure 6.1: One dimensional space-time illustration of the challenges of r-refinement with general adaptive motion in one spatial dimension.

two dimension, sampling nodes have to be created to collect statistically reliable error projections. Through the present approach, we save time solving the unsteady adjoint, and the mapped error information proves to be sufficient for r-adaptation because only the relative values among error indicators matter for r-adaptation purposes.

6.1 Active Flux Method r-Adaptation

For r-adaptation applied to the Active Flux method, the continuous adjoint is used, due to its relative inexpense compared to the discrete adjoint. The continuous adjoint avoids the computational burden of linearzing and transposing systems for a dynamic adaptation algorithm, which would have to be done for the discrete adjoint. Continuous adjoint-based error estimation with mesh motion has been presented in detail in Section 3.1.

6.1.1 Adaptive Indicator

r-Adaptation indicators reside on a warped space-time control volume due to mesh motion. To obtain an accurate error indicator, we integrate the contribution of the

continuous adjoint error estimate,

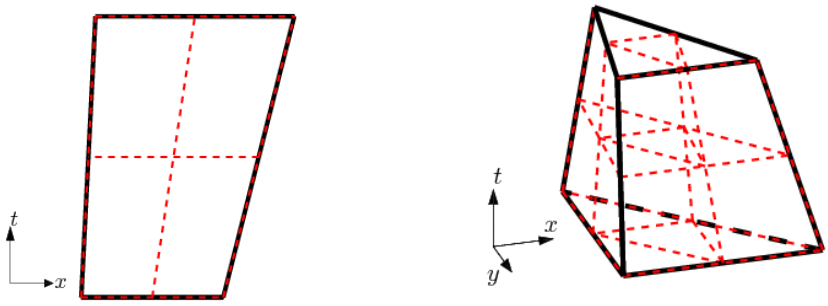
$$\delta J \approx \int_T \int_\Omega \psi_h \, r(u_H) \, d\Omega \, dt. \tag{6.1}$$

We perform this integration by parts. For example, for element e and time step k, this contribution is, for the advection equation,

$$\varepsilon_{e,k} = -\int_{\Omega_e} \delta\psi_h u_H \, dx \Big|_{t^k}^{t^{k+1}} - \int_{\partial\Omega_e} \int_{t^k}^{t^{k+1}} \delta\psi_h u_H \vec{V} \cdot \vec{n} \, dt \, ds. \tag{6.2}$$

The elements contributing to this error estimate are shown in Figure 6.2(a) for 1D. Similarly, Figure 6.2(b) shows the contributing elements in two dimensions. Taking the absolute value of this contribution yields an adaptive indicator,

$$\epsilon_{e,k} = |\varepsilon_{e,k}| = \left| \int_{\Omega_e} \delta\psi_h u_H \, dx \Big|_{t^k}^{t^{k+1}} + \int_{\partial\Omega_e} \int_{t^k}^{t^{k+1}} \delta\psi_h u_H \vec{V} \cdot \vec{n} \, dt \, ds \right|. \tag{6.3}$$



(a) Subelements in 1D

(b) Subelements in 2D

Figure 6.2: In 1D, the error indicator contributions come from 4 quadrilateral sub-cells of each space-time cell. In 2D, the error contributions come from 8 prismatic sub-cells of each space-time cell.

6.1.2 r-Refinement for Unsteady Problems

Generally, we expect more advantages from mesh refinement in higher dimensions because the error in the computational domain may be more unevenly distributed compared to lower dimensions, due to a higher number of degrees of freedom. For the

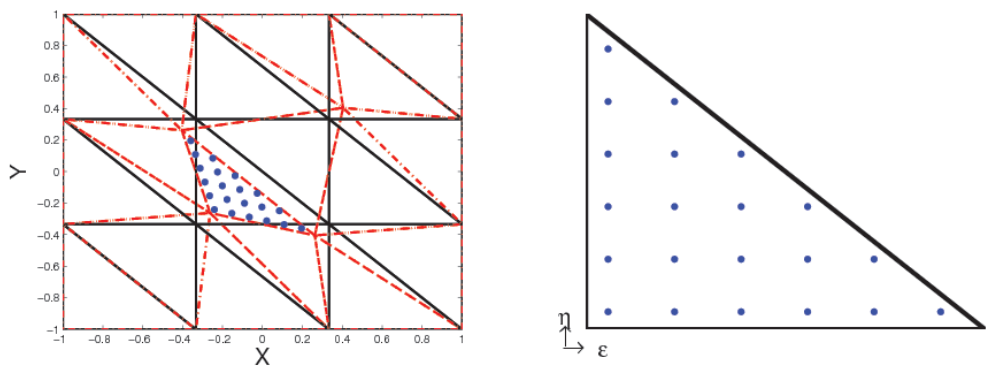
present study, we apply many different strategies of r-refinement on one dimensional problems to gain insights applicable to two-dimensional r-refinement. For the final adaptation results, we only demonstrate the two dimensional strategy.

For an unsteady problem, r-refinement constitutes efficient adaptation mechanics, as no degrees of freedom are added. We strive for a strategy of error equi-distribution over the space-time elements. Since the space-time mesh changes with each iteration, we need a way to map error indicators from one mesh to another.

Figure 6.3(a) illustrates such a mapping. The error indicators computed at the previous iteration reside on a mesh, the black mesh, that is different from the r-refined (red) mesh. To obtain error indicator information on the refined mesh, we use evenly distributed sampling nodes inside each triangle to collect error information for the new cells. Figure 6.3(b) shows the distribution of sampling nodes in a reference triangle inside reference space. This distribution should be sufficiently fine to provide accurate error sampling and the points should be located only inside the elements. The average of the indicators at all of the sampled nodes is taken as the approximate error indicator on each cell. To locate the sampling nodes, we implement a fast line-based search technique, in which a fictional line is drawn between two sampling points to determine which elements need to be traversed to reach the next sampling point, instead of looping over the whole computational domain in a brute-force search.

6.1.3 r-Refinement Mechanics

Given an adaptive indicator on each space-time element, a spring analogy [84] is used to drive the mesh motion. In this analogy, edges of the mesh are treated as springs so that the mesh is a web of springs. The error information is associated to each spring in the web via averaging of the element-based error indicator, and a force (via Hooke's law) is created by relating the error indicator to the spring equilibrium



(a) Mesh mapping illustration

(b) 21 sampling nodes in a reference triangle

Figure 6.3: Error mapping mechanics illustration: the black mesh is the baseline mesh, the red mesh indicates the r-refined mesh, which is dynamically changing throughout the simulation time, and the blue dots are the sampling points for the error indicator transfer.

length, through a prescribed scaling. This force is

$$\text{Force} = -k\,(L' - L)\,, \tag{6.4}$$

where L' is the error-indicator-determined equilibrium length and L is the current edge length. A balance of forces at each node causes displacements, and the balance of forces in the mesh is then analogous to error equi-distribution, which is a goal of mesh adaptation.

The parameter that influences how well the spring analogy works is the ratio of the spring stiffness to the nodal mass, k/m. After fixing $m = 1$, the only parameter that matters is k. In our current implementation, k is a tunable parameter in the spring analogy and a determining factor of how fast the mesh responds to error indicator information. Its value could be related in non-dimensional form to the time step size through analysis of a one-dimensional harmonic oscillator – however, in the present work with O(1) units, we simply tune it to produce reasonable results. Figure 6.4 presents a flow chart of how the r-refinement mechanics works for our problem. This includes how the adaptive indicator computation and spring analogy are coupled

together.

6.1.4 r-Adaptation Result

The presented primal problem is a clipped Gaussian pulse with center located at coordinates, $\vec{X}_0 = [-0.4, -0.4]$, and convecting at the bulk flow advection velocity, $\vec{a} = [2, 2]$, until the center of the Gaussian vortex reaches, $\vec{X}_{\text{end}} = [0.4, 0.4]$, with inflow / outflow boundary condition. The Gaussian function takes the form

$$u = \begin{cases} \exp\left(-\dfrac{(x-x_0)^2 + (y-y_0)^2}{\sigma^2}\right) - C & \left\|\vec{X}_i - \vec{X}_0\right\| \leqslant R \\ 0 & \left\|\vec{X}_i - \vec{X}_0\right\| > R \end{cases} \tag{6.5}$$

where, we pick a clipping radius of $R = 0.5$, a spread of $\sigma = 0.1$, and a clipping offset of $C = \exp(-R^2/\sigma^2)$. Our output is a domain weighted integral $J = \int_\Omega W(x,y) \times u(x,y)\, d\Omega$, where the weight function is $W(x,y) = [\sin((x+1)\pi) \times \sin(y+1)\pi)]^2$.

	$\lvert J_{\text{exact}} - J_H\rvert_{(\text{static mesh solve})}$	$\lvert J_{\text{exact}} - J_H\rvert_{(r \text{ refinement})}$
$N_{\text{cell}} = 50$	2.883×10^{-1}	2.686×10^{-2}
$N_{\text{cell}} = 200$	2.329×10^{-2}	7.253×10^{-3}
$N_{\text{cell}} = 450$	1.292×10^{-2}	5.738×10^{-3}

Table 6.1: Error comparison of output error on a static mesh solve and output error on a r-refinement solve, CFL = 0.7, simulation time = 0.1 period, the third cycle, $k = \dfrac{2}{\text{time step size}}$

The sequence of error indicators and the r-refinement history in Figure 6.5 show clear convection dominance in this problem. The mesh nodes agglomerate diagonally, toward the same direction along which the scalar is moving (up and to the right). This means that the adaptive indicators captured some of the physics of the scalar advection problem. The error prone area is along the diagonal of the computational domain. If the mesh were plotted at every time step of r-refinement, the mesh nodes would appear to move with the advecting scalar. This shows that the r-refinement strategy is dynamically adjusting the degrees of freedom towards the error prone area(s).

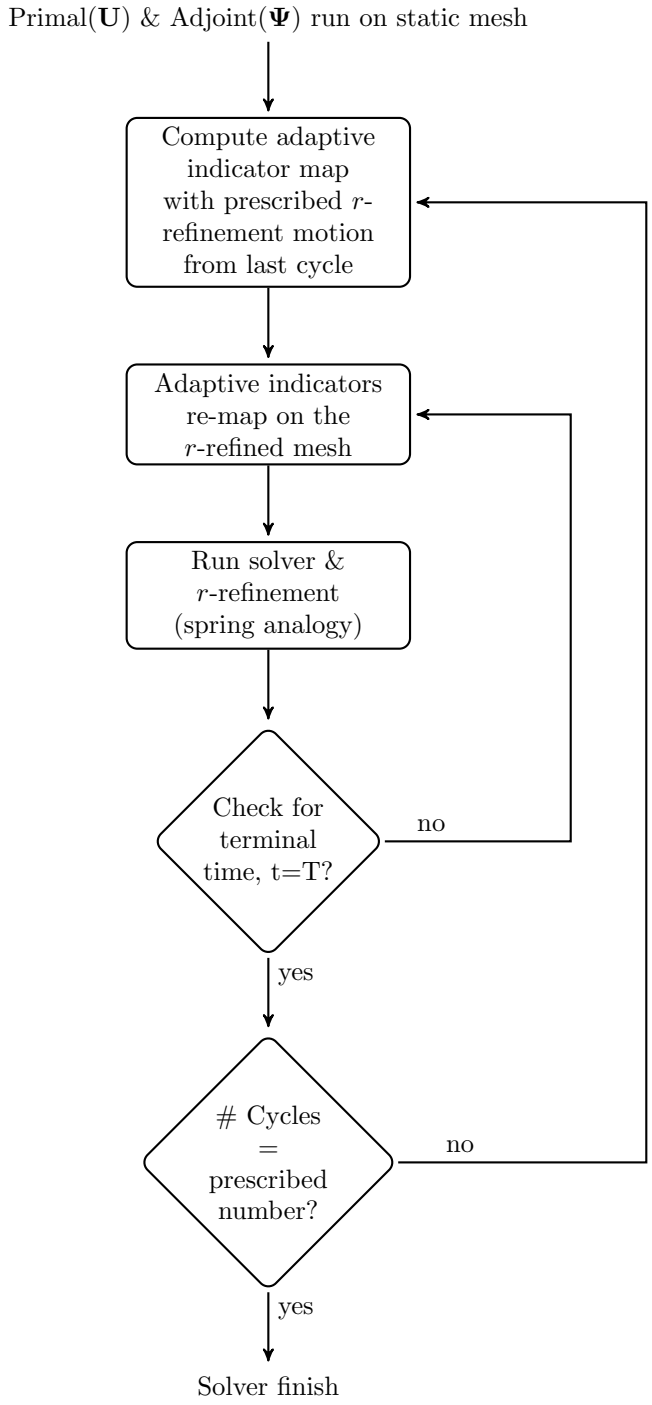
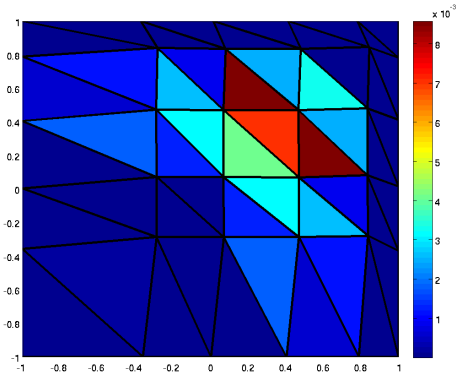
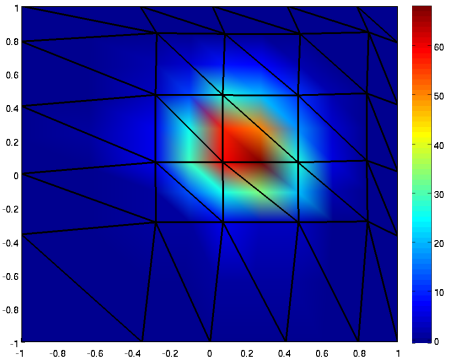


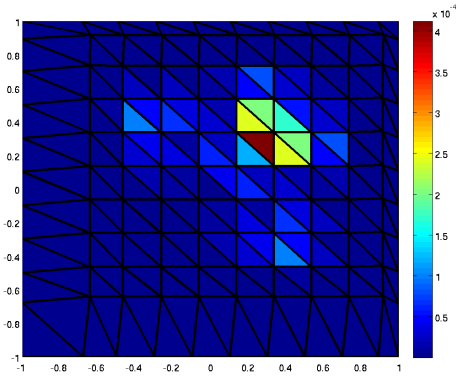
Figure 6.4: r-Refinement mechanics for the Active Flux method.



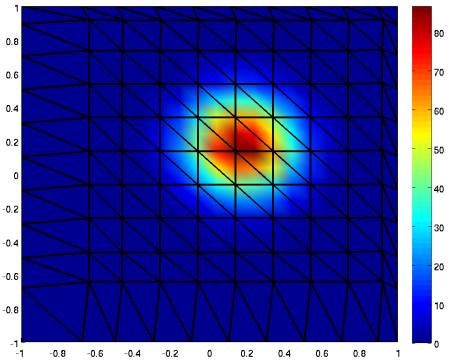
(a) Adaptive indicators, $N_{\text{cell}} = 50$



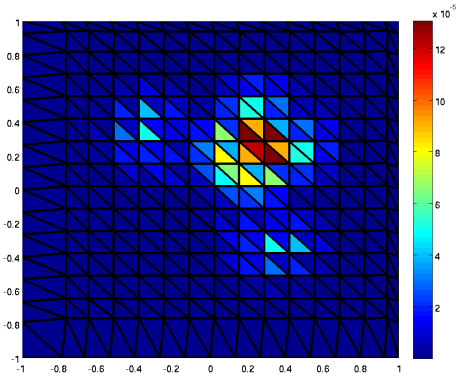
(b) r-Refinement state, $N_{\text{cell}} = 50$



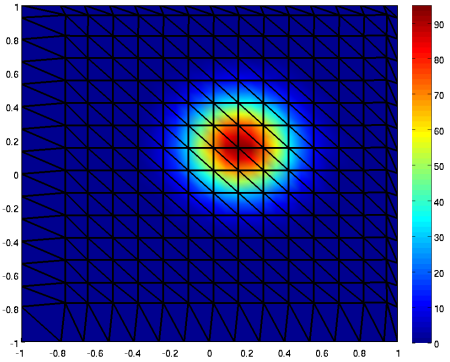
(c) Adaptive indicators, $N_{\text{cell}} = 200$



(d) r-Refinement state, $N_{\text{cell}} = 200$



(e) Adaptive indicators, $N_{\text{cell}} = 450$



(f) r-Refinement state, $N_{\text{cell}} = 450$

Figure 6.5: Comparison of r-refinement adaptive indicators and r-refinement primal snapshots, both captured at $t = 0.8 \times$ (total simulation time) for the third r-refinement cycle. In this case, CFL $= 0.7$, and the simulation time $= 0.1$ period.

Table 6.1 shows the comparison of error levels on static meshes and r-refined meshes. With exactly the same number of degrees of freedom, r-refinement decreases the error level by more than 50% within three cycles of r-refinement.

6.2 Discontinuous Galerkin r-Adaptation

The arbitrary Lagrangian-Eulerian (ALE) mesh motion algorithm used for discontinuous Galerkin (DG) r-adaptation is not strictly conservative on a finite-size mesh. The mesh motion algorithm itself introduces additional error into the solution process compared to a solve on a static mesh. For r-adaptation to be useful, the error introduced by mesh motion should be smaller than the scheme discretization error. The choice of the mesh motion algorithm for r-adaptation is dictated by two considerations: (1) introducing as little error as possible, and (2) handling general motions, because r-adaptation needs to be flexible enough to work with arbitrary error distributions.

Mesh motion based on analytic functions can be made smooth and hence of low error due to mesh motion, yet it is limited in the types of motion that can be realized to control an arbitrary error distribution. On the other hand, mesh motion based on interpolation of nodal velocities is much more flexible in addressing general error distributions, though it is more difficult to implement and it introduces more errors arising from non-smooth features of the mapping. In this work we present implementations of both methods and compare their results.

6.2.1 Analytic Function Based Mesh Motion

Mapping

In standard applications of ALE, the mapping from reference to global space is chosen to move or deform the geometry in a desired fashion. In the present work,

we consider mesh motions that increase or decrease resolution in certain areas of the mesh, at certain targeted times. Such a mapping, categorized as contraction or dilation, follows the following form:

$$\vec{x} = \left(\vec{X} - \vec{X}_0(t)\right) f(\vec{X}, t) + \vec{X}_0(t), \tag{6.6}$$

where $\vec{X}$ represents the reference-space coordinates, $\vec{X}_0(t)$ represents the center of the contraction/dilation, and $f(\vec{X}, t)$ is a function that dictates the magnitude of the contraction or dilation. We henceforth call f the *contraction function*, though specifically, $f < 1$ indicates a contraction, while $f > 1$ indicates a dilation. Limiting cases include $f = 1$, in which $\vec{x} = \vec{X}$ (no motion), and $f = 0$, in which all of the mesh points coalesce to $\vec{x} = \vec{X}_0$.

We note that the mapping in Eqn. 6.6 is only a contraction/dilation locally near $\vec{X}_0$. In the surrounding region, the effect of the mapping is opposite, as a contraction near one point causes stretching of the mesh (larger elements) further away from the point, with the distance dependent on the form of $f(\vec{X}, t)$. We choose the following parametrized form for the contraction function,

$$f(\vec{X}, t) \;=\; 1 - A(t) \exp\left[-(\vec{X} - \vec{X}_0(t))^T \mathbf{W} (\vec{X} - \vec{X}_0(t))\right], \tag{6.7}$$

where $\mathbf{W}$ is a dim $\times$ dim symmetric, positive-definite matrix that encodes the stretching magnitudes and directions. Specifically, the eigenvectors of $\mathbf{W}$ give the direction of stretching, whereas the eigenvalues relate to the stretching magnitude. To make the contraction active for only a certain time, we choose a localized form of the contraction amplitude, $A(t)$,

$$A(t) \;=\; A_0 \exp[-w(t - t_0)^2]. \tag{6.8}$$

This is a Gaussian, illustrated in Figure 6.6, with w related to the duration of the contraction (i.e. the width of the Gaussian), and t_0 the time of peak deformation.

The amplitude A_0 is set based on user-defined maximum contraction ratio, g_0, which

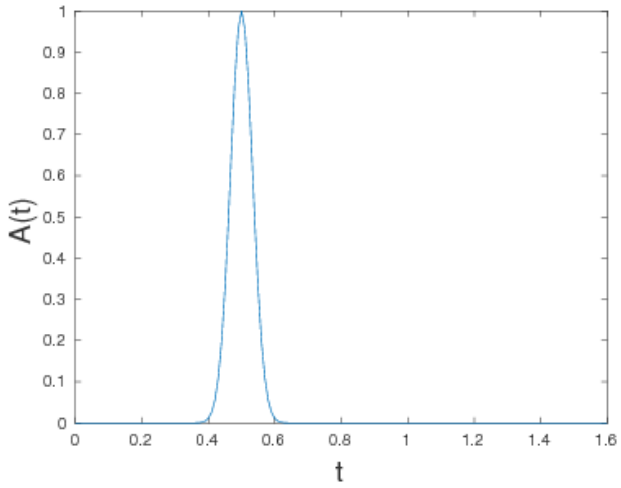


Figure 6.6: Sample temporal amplitude function for a contraction/dilation map.

is defined as the volume fraction of an element in physical space relative to reference space.

For all the elements, the contraction ratio is,

$$g = \det(G) = \frac{\text{Volume of an Element in Physical Space}}{\text{Volume of the Reference Element}} \tag{6.9}$$

The value of g serves as an indicator of area change for the mapping G. With our mapping for two dimensional problem,

$$G = \begin{bmatrix} f + (X_1 - X_1(t_0))f_x & (X_1 - X_1(t_0))f_y \\ (X_2 - X_2(t_0))f_x & f + (X_2 - X_2(t_0))f_y \end{bmatrix} \tag{6.10}$$

$$
\begin{aligned}
g &= \det(G) \\
&= f^2 + f\left[\left(X_1 - X_1(t_{\text{motion}})\right) f_x + \left(X_2 - X_2(t_{\text{motion}})\right) f_y\right] \\
&= 1 - 2A(t,x,y) - A_x(t,x,y)\left(X_1 - X_1(t_{\text{motion}})\right) \\
&- A_y(t,x,y)\left(X_2 - X_2(t_{\text{motion}})\right) \\
&+ A^2(t,x,y) + A(x,y,t)A_x(t,x,y)\left(X_1 - X_1(t_{\text{motion}})\right) \\
&+ A(x,y,t)A_y(t,x,y)\left(X_2 - X_2(t_{\text{motion}})\right). \qquad (6.11)
\end{aligned}
$$

Mathematically, the peak of the contraction, g_0, is reached when $t = t_{\text{motion}}$ and $\vec{X} = [x_0,\ y_0]$, at this time point, $A(t,x,y) = A_0$, $A_x = 0$ and $A_y = 0$. Hence, the most ridicule value of g,

$$
g(t_0, x_0, y_0) = g_0 = (1 - A_0)^2. \qquad (6.12)
$$

As a result,

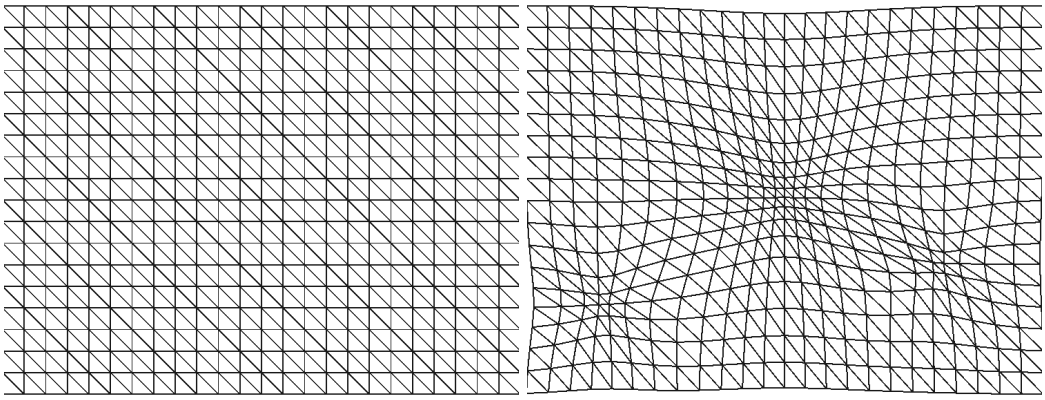
$$
A_0 = 1 - \sqrt{g_0}. \qquad (6.13)
$$

Eqn. 6.11 reveals the corresponding mapping choice of A_0 according to the maximum allowable contraction ratio should be $1 - \sqrt{g_0}$.

Superposition

To handle multiple locations and times of large error, we superimpose several contraction mappings, centered at locations of largest error. When dealing with domains of finite extent, we have to take care not to inadvertently alter the geometry under consideration when applying the r-adaptation maps. As the presented spatial

function in Eqn. 6.7 has global effect, we must clip its region of influence. Otherwise we will observe deformation of the computational boundary, as shown in Figure 6.7.



(a) Original static mesh

(b) Mesh boundary distortion caused by three contraction sources

Figure 6.7: Effect of superimposed contractions without clipping on the computational domain boundary.

To avoid this problem, we measure the wall distance, defined as the distance to the closest wall, from each proposed contraction center, $\vec{X}_0$, and use this wall distance to limit the stretching magnitudes. Specifically, we clip the eigenvalues of $\mathbf{W}$ in Eqn. 6.7 so that the location of the closest wall corresponds to a minimum of three standard deviations of the Gaussian contraction function.

6.2.2 Node-Interpolated Mesh Motion Based Mesh Motion for r-Adaptation

In node-interpolated mesh motion (NIMM), the position and velocity of each node are degrees of freedom that can be used to design a tailored mesh motion. In contrast to analytic function based mesh motion (AFBMM), NIMM does not rely on analytic functions and can hence express more complex motions that are often necessary in r-adaptation.

The NIMM approach to mesh motion is still based on the ALE framework, which requires the mapped coordinates, nodal velocities, and mapping Jacobians and determinant. These are obtained in two steps: 1) interpolation and 2) Jacobian $\underline{G}$ smoothing.

Interpolation refers to mapping mesh motion information from the mesh nodes to element interiors. This introduces an error that depends on the order of interpolation: for example, a second order error will be introduced when using linear interpolation.

The ALE formulation requires six geometric properties: physical coordinates $\vec{x}$, velocities $\vec{v}$, mapping Jacobian $\underline{G}$, Jacobian determinant g, and Jacobian derivatives $\partial G/\partial \vec{x}$ and $\partial G/\partial t$. Two of these, in our case $\vec{x}$ and $\vec{v}$, can be prescribed. With $\vec{x}$ and $\vec{v}$ information, $\underline{G} = \frac{\partial \vec{x}}{\partial \vec{\xi}} \times \left(\frac{\partial \vec{X}}{\partial \vec{\xi}}\right)^{-1}$ and $g = \det \underline{G}$. Lastly, $\partial \underline{G}/\partial \vec{X}$ and $\partial \underline{G}/\partial t$ are zero within an element because the prescribed velocity $\vec{v}$ is constant, which results in zero second-order derivatives with respect to $\vec{x}$.

$\vec{v}$ and $\vec{x}$ are prescribed at mesh nodes (vertices). For any other points where no information was provided, we calculate the information required for ALE through interpolation as explained in Figure 6.8, with the example of a $2D$ triangular reference element. This interpolation is used to obtain $\vec{x}$, $\vec{v}$, and $\underline{G}$ inside the element. The Jacobian determinant, g, is computed from the interpolated Jacobian: $g = \det \underline{G}$.

Jacobian smoothing

As a result of the linear interpolation of coordinates from vertices to the interior of the element, the reference-to-global mapping Jacobian matrix $\underline{G}$ is constant within each element. For general mesh motion, the discontinuous representation of $\underline{G}$ across elements causes errors that pollute the results and destroy the high-order accuracy of the method. We reduce this error by smoothing $\underline{G}$ to create a piecewise-linear, continuous representation.

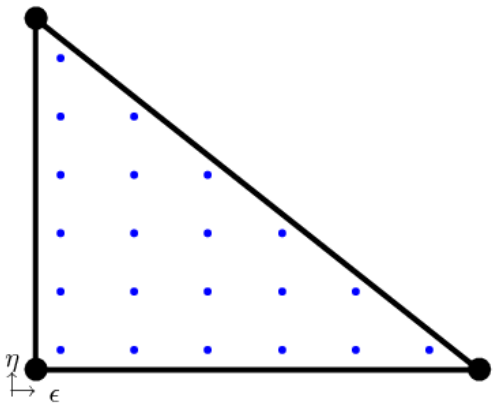


Figure 6.8: Interpolation in a $2D$ reference space, (ξ, η). Black dots represent linear Lagrange basis function nodes, which are used to interpolate ALE information from the nodes to any other points (blue dots) inside an element or on its edges.

The Jacobian is smoothed by first averaging the matrix from elements to nodes: $\underline{G}$ at each node is set equal to the average of $\underline{G}$ over the adjacent elements. The Jacobian is then interpolated linearly back to the element interiors to provide the desired piecewise-linear representation.

As an example of the effect of smoothing, we consider a scalar advection simulation. The primal initial condition is a constant box function that is nonzero in a region defined by four vertices, [0.5 0.5], [1.5 0.5], [1.5 1.5] and [0.5 1.5]. This scalar initial condition advects through the computational domain at a velocity $\vec{a} = [2.0 \;\; 0.1]$, for time $t = 1.0$ with a prescribed sinusoidal mesh motion,

$$\begin{aligned} x_0 &= X_0 + A_x \sin(2\pi x_0/X_0) \sin(2\pi x_1/X_1) \sin(2\pi t/t_0) \\ x_1 &= X_1 + A_y \sin(2\pi x_0/X_0) \sin(2\pi x_1/X_1) \sin(4\pi t/t_0) \end{aligned} \tag{6.14}$$

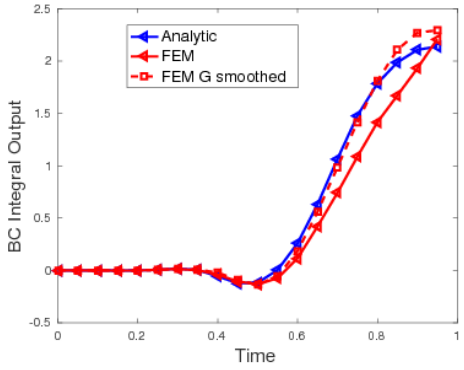
where, X_i are reference-space coordinates and x_i are physical-space coordinates, $A_x = 0.3$, $A_y = 0.2$ and $t_0 = 1.0$, and the computational domain is a 3 by 2 rectangle. The output is a boundary integral at the right boundary of the computational domain measured at every time step.

The prescribed motion in Equation 6.14 is an analytic expression, and hence we

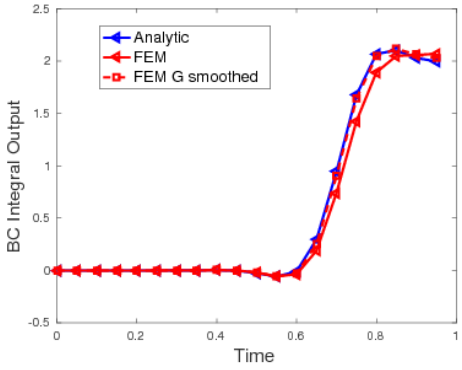
can run AFBMM: we treat the result as a reference solution for the NIMM runs. We perform an L_2 error convergence study on NIMM with and without Jacobian smoothing. For the refinement study, the L_2 error is defined as

$$E_{L_2} = \frac{1}{N_t}\sqrt{\sum_{i=0}^{i=N_t}\left(U_i^{\text{analytic}} - U_i^{\text{NIMM}}\right)^2}, \tag{6.15}$$

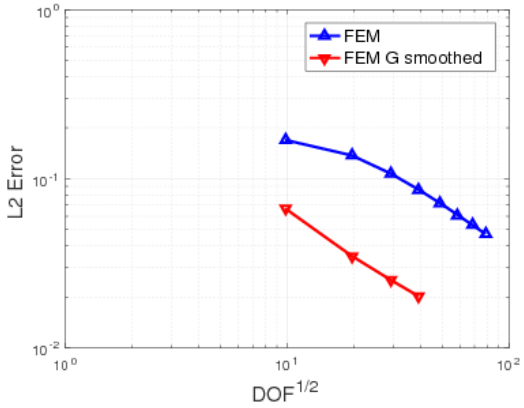
where N_t is the total number of time steps. Figure 6.9 shows the resulting benefits of Jacobian smoothing on the L_2 error through a uniform mesh refinement study.



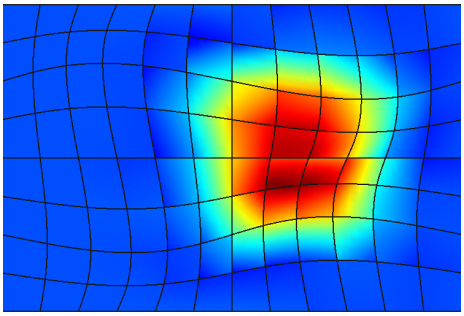
(a) Output evolution with time, first refinement cycle, 96 elements



(b) Output evolution with time, fourth refinement cycle, 1536 elements



(c) Refinement study



(d) Primal solution and mesh at $t = 0.45$

Figure 6.9: Error reduction through a smoother nodal $\underline{G}$ values.

6.2.3 Error Estimation and r-Adaptation

Error Estimation

An adjoint solution allows us to estimate the numerical error in the corresponding output of interest, $\bar{J}$, through the adjoint-weighted residual [32,38]. To estimate the error, we use this adjoint to weight the unsteady residuals obtained by injecting the primal solution into a finer space, both spatially and temporally.

For the present study, we use a unit order increment in both space and time to obtain this *fine space*, denoted by subscript h. The resulting final form of the error estimate is

$$\delta \bar{J} \approx -\int_0^T \mathbf{\Psi}_h^T \bar{\mathbf{R}}_h(\mathbf{U}_h^H)\, dt, \tag{6.16}$$

where $\mathbf{U}_h^H$ is the injection of the primal from space H to space h. The integral in Eqn. 6.16 is a summation of integrals over time intervals, each performed with appropriate quadrature.

The output error estimate in Eqn. 6.16 is separated into spatial and temporal components by selectively refining the fine space only in space or only in time [61]. This separation then drives the decision of whether to refine in space or in time.

Error Localization and r-Adaptation

For r-adaptation instructed by analytic function based mesh motion, we employ a simplified localization and adaptation procedure in which a separate contraction mapping is created for every space-time element in the top $f^{\text{adapt}} = 10\%$ fraction of elements with the largest error. The spatial centroid of each selected element serves as the center of the contraction, $\vec{X}_0$. The magnitude of the contraction is set to be as large as possible, subject to the no-wall motion constraint discussed in subsection 6.2.1. In addition, we heuristically set w in Eqn. 6.8 to produce a

temporal Gaussian function with standard deviation equal to two time steps.

For node-interpolated r-adaptation, the adaptation is driven by a spring analogy previously used for the Active Flux method [80]. The localized error within each element is re-distributed onto element edges via averaging, and we relate this error to the equilibrium length of a spring on the edge. The whole mesh is treated as a web of springs, and a force balance is used to equidistribute the error. At each time step, we move nodes by balancing the forces inside the spring web, where the force on an edge is given by

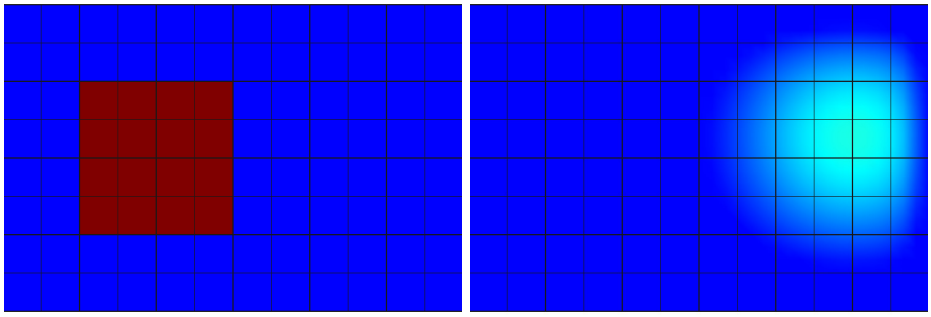
$$F = -K \times (L_{\text{edge length}} - L_{\text{edge-wise error}}) \tag{6.17}$$

The force balance process is enforced via several iterations during each time step of the unsteady solve. After mesh movement, the error indicator information obtained from the previous adaptive iteration will not be accurate, though after many adaptation iterations, the variation in error indicators becomes smaller and the motion converges. Presently, a termination condition consists of a fixed number of adaptation iterations.

6.2.4 Results

We consider a scalar advection-diffusion problem for the unknown concentration u, with advection velocity $\vec{V} = [1, 0.1]$, viscosity = 1.0 and homogeneous Dirichlet boundary conditions. The domain is a rectangle, and the initial condition is a zero distribution of the state throughout the domain, with the exception of a square region on the left portion, where $u = 1$. Figure 6.10 shows this initial condition, as well as the solution at the final time.

The spatial domain is discretized with quadrilaterals in reference space, and the order of approximation is $p_{\text{spatial}} = 1$. The total simulation time is $t_{\text{total}} = 1.6$. The



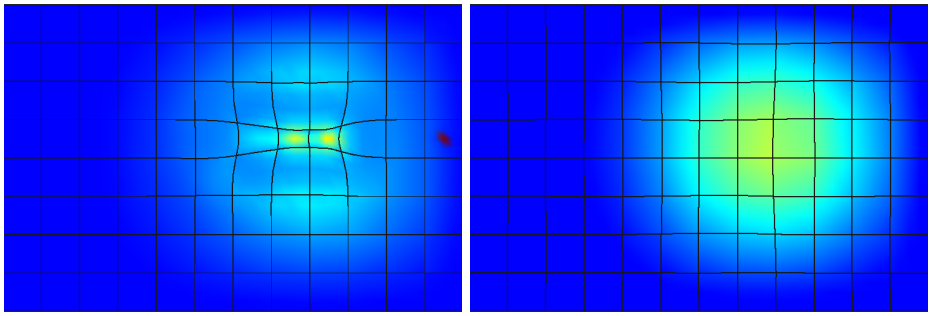
(a) Initial condition (b) Final time

Figure 6.10: Initial condition and final-time primal solution of the scalar advection-diffusion problem.

problem is solved using DG-in-time with 20 time steps of order $p_{\text{temporal}} = 1$ initially. r-adaptation may generate smaller elements and result in additional time steps, as determined by the space-time adaptation algorithm. The output of interest is a boundary integral of the state on the right-hand side boundary at the final time.

We apply the output-based error estimation procedure to this problem to identify the localized elemental error. We then use the space-time indicators to drive the proposed r-adaptation strategy. Figure 6.11 and Figure 6.12 show the resulting mesh at $t = 0.88$ and $t = 1.6$ respectively in the simulation. The contractions of the mesh near the output measurement for both cases are clearly evident.

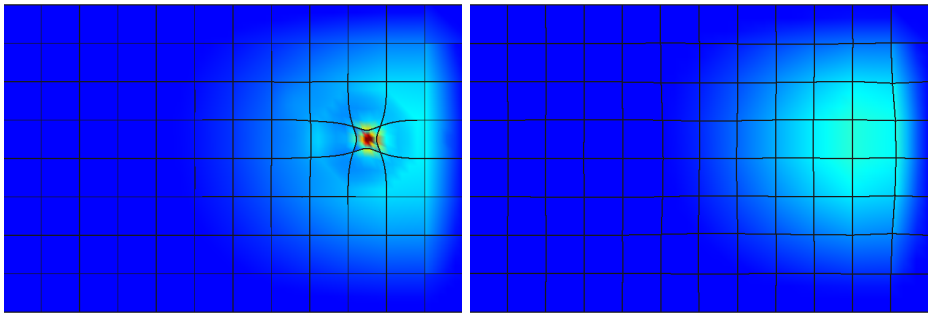
Figure 6.11 shows that AFBMM tends to locally alternate mesh resolution while NIMM tends to re-distribute mesh resolution globally. Similarly, Figure 6.12 displays the same observation, where AFBMM picks the single element with the largest error to place a contraction source, while NIMM shrinks the whole column of elements at the right boundary of the computational domain to achieve a more accurate right-hand-side boundary integral output.



(a) AFBMM, snapshot at $t = 0.88$. Several contraction sources are placed to redistribute the mesh resolution.

(b) NIMM, snapshot at $t = 0.88$. The mesh is mildly contracted near the center of the scalar profile and a contracting wave that propagates through the domain is observed in the animation.

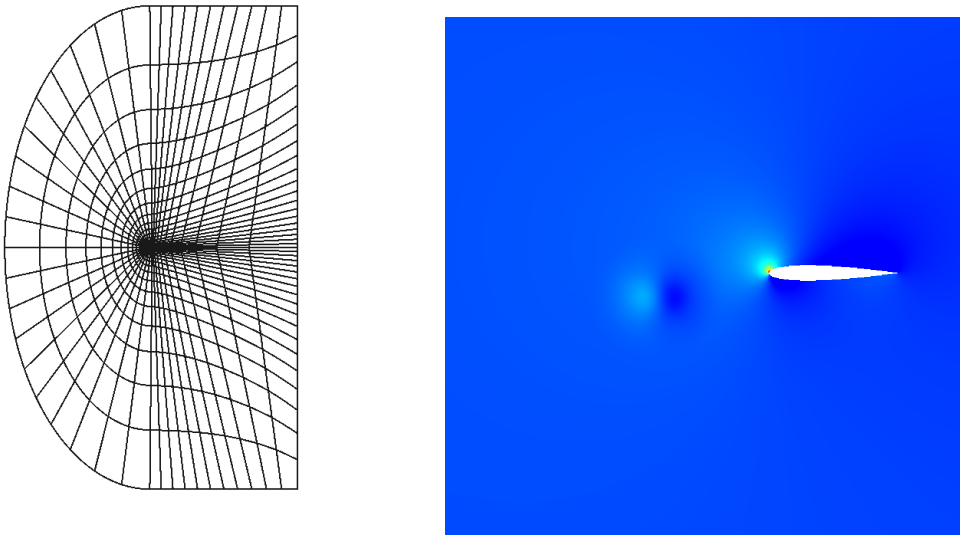
Figure 6.11: Solution and mesh snapshots generated by AFMM and NIMM at $t = 0.88$. AFBMM tends to pick out elements with the highest error and places contraction sources on these elements. On the other hand, NIMM motion deforms the mesh in a more uniform/global fashion.



(a) AFBMM, snapshot at $t = 1.6$. One contraction function is placed around the element with the highest error.

(b) NIMM, snapshot at $t = 1.6$. Elements near the right boundary are warped by NIMM.

Figure 6.12: Snapshot of AFMM and NIMM at $t = 1.6$.



(a) Computational domain (b) Mach number plot

Figure 6.13: Airfoil vortex encounter case using a farfield inflow boundary condition and a static pressure outflow boundary condition. $p_{\text{spatial}} = 1$, $p_{\text{temporal}} = 1$, DG in time, output of interest is the lift integral over time $= \int_{t=0}^{t=t_{\text{end}}} F_{\text{lift}} dt$, Euler equations.

For a quantitative comparison, we measure the error estimate without mesh motion as -1.03×10^{-2}. After three iterations of r-adaptation, i.e. using the no-motion error indicators to drive the r refinement procedure, the error estimate drops to -1.31×10^{-3} for AFBMM, which is almost an order of magnitude error reduction. The error estimate drops to -6.04×10^{-3} for NIMM, which is around a 50% error reduction.

6.3 Additional Simulations

Euler cases were tested for both AFBMM and NIMM as well. However, due to parallel code implementation (introducing new data structure to the main code), the turn-around time for the cases tested so far are extremely long and the simulations that were run show small displacements induced by r-adaptation. Nevertheless, these displacements still lead to error reductions.

For the case in Figure 6.13, the output error estimated on a static mesh solve is

-1.60×10^{-3} and the output error estimated on the r-adapted mesh is -2.54×10^{-5}. Due to the minor observable motion issue, future work in this area is still needed.

6.4 Summary

Work in previous chapters has already investigated h-refinement with various adaptation strategies for the Active Flux scheme [16,17] using static mesh refinement. In the present work, we demonstrate the Active Flux method with mesh motion and dynamic mesh adaptation using r-refinement for unsteady simulations, driven by a continuous adjoint. In addition, we created a output-based r-adaptation mechanics for discontinuous Galerkin method.

The research contribution from this chapter include,

- Discussed two key aspects of the r-refinement strategy: the error indicator mapping to create an adaptive indicator on an arbitrarily-deformed mesh, and the spring analogy for driving mesh motion. The final result shows that with these models, the error in the desired output could be quickly reduced by at least 50% within the first few r-refinement cycles.

- Introduced a method for mesh adaptation using r-refinement for the discontinuous Galerkin discretization. The method uses an arbitrary Lagrangian-Eulerian framework for moving the mesh, with two different mesh movement algorithms, AFBMM and NIMM. Both methods show positive results in dropping the total error in the simulation, though the NIMM method is expected to be more robust for arbitrary error indicator distributions.

CHAPTER VII

Conclusion

7.1 Contributions

This dissertation makes contributions to solution verification methods for high-order computational fluid dynamics (CFD) discretizations, with focus on the discontinuous Galerkin (DG) and the Active Flux methods. These verification methods consist of two primary components: estimation of numerical error and adaptation of the discretization to reduce this error. An advantage of such approaches compared to uniform (uninformed) refinement is a savings in both degrees of freedom and computational time required to solve a given problem to an acceptable level of accuracy. This advantage was demonstrated in the present work for both discretizations considered. Another advantage of the adaptive approach is quantified error estimates that provide termination criteria in an iterative solution setting.

The following list enumerates the research contribution of this dissertation.

1. **Adjoint implementation for the Active Flux method:** A comparison study of discrete adjoints versus continuous adjoints shows that, although the derivation process of the discrete adjoint is straightforward, its implementation could be costly and requires adjoint consistency studies. This is particularly the case for finite-volume methods, like Active Flux, which do not have a finite-

element variational formulation. Adjoint consistency studies are necessary to ensure that the discrete adjoint approximates the continuous adjoint in the limit of infinite resolution. On the other hand, though continuous adjoint formulation has adjoint consistency built into the derivation, and with this sound theoretical support doesn't require adjoint consistency studies. However, it requires re-derivation upon changes of boundary condition and physics.

2. **Error estimation for the Active Flux method:** output error estimates are implemented using the adjoint-weighted residual method. These error estimates then assist in automated CFD solution management. However, their fidelity is asymptotic, so that oscillatory error estimates may occur on coarse grid levels. An automated CFD solution management should take both the absolute value of error estimates as well as error estimates convergence trend into consideration. For example, on a coarse mesh, the absolute value of error estimates might be falsely small, whereas upon refinement, the error estimates may increase and then converge at the appropriate rate.

3. **h-adaptation mechanics for the Active Flux method:** upon creation of h-adaptation mechanics, we examine two aspects of h-adaptation, (1) a comparison of isotropic h-adaptation and anisotropic h-adaptation and (2) global re-meshing versus local element splitting. Regarding (1), anisotropic mesh refinement shows advantages when the physics of interest contain anisotropic features. In space-time formulations, the anisotropy may mix spatial and temporal dimensions, so that large time steps can be combined with stretched spatial grids. Regarding (2), we note that global re-meshing is a more expensive procedure compared to local element splitting. Global re-meshing is desirable when

global nodal re-positioning is performed at the same time.

4. **Solution adaptation acceleration frameworks for the Active Flux method and the DG method:** solution adaptation acceleration research is inspired by the fact that approximate adaptive indicators have merit in instructing adaptation. An empirical study on computational time expenditure discloses that most of the simulation time is spent on the fine-space adjoint solve for a solution verified simulation. Thus, it makes sense for a time-efficient acceleration method to focus on reducing the fine space adjoint solve computational time. An effective solution adaptation acceleration method should also provide an accurate error estimate.

5. **Output-based r-adaptation for the Active Flux and DG methods:** r-Adaptation takes many shapes and forms, e.g., global nodal position optimization and nodal movement through analogies to physical systems, such as springs. Despite the choices of r-adaptation, as a degree-of-freedom conservative adaptation choice, r-adaptation shows many merits for use in CFD codes and can be seamlessly integrated with any other adaptation choices without adding significant computational time cost.

7.2 Future Research

This work introduced new solution verification methods for high-order CFD discretizations. The following research topics could be worth investigating in the future:

1. An investigation on output-based r-adaptation with different mesh motion strategies: any analogy-based model contains tunable parameters. Though parameters could be reduced, tunable parameters usually cannot be eliminated. To

realize a fully automated $r - adaptation$, especially for unsteady flow problems, an investigation of r-adaptation with different motion strategies should be performed.

2. An integration of different adaptation choices: i.e. h, p and r. Both h and p adaptive methods have already been well-studied, as well as the combination p. r refinement has been studied to a lesser extent. However, there is a lack of research regarding hpr adaptation work. If implemented correctly and driven by accurate error estimates, hpr adaptation should perform better than any adaptation choices alone or any other adaptation choice combinations.

3. A standardization of solution adaptation acceleration: the findings in this dissertation suggest that solution adaptation acceleration is advantageous for multiple high-order methods. Combinations of such acceleration techniques and rigorous error estimates should be investigated as a method for reducing cost while maintaining accuracy.

APPENDICES

APPENDIX A

Acoustics Cases Error Convergence in Xflow

The discontinuous Galerkin method can exhibit convergence rates in certain outputs that are higher than the expected solution errors predicted by simple interpolation theory. This behavior is termed "super-convergence", although mathematically the high rate should not come as a surprise: it is provable and expected under relatively broad assumptions. Thus, a more apt term may be "appropriate" or "expected" convergence. Nevertheless, there are certain conditions that must be satisfied in order for these rates to be attained, and especially for unsteady problems, these conditions may not be intuitive. In this appendix we present a study of the discontinuous Galerkin method applied to the unsteady linear acoustics equations, and we show that super-convergence can only be attained with a least-squares projection of the intial conditions.

A.1 Linear Acoustics

By only taking the acoustic terms from the Euler equations, i.e., linearizing the acoustic subsystem by assuming constant coefficients ρ_0 and c_0, and relating the pressure and density through $c^2 = p/\rho$, the linearized Euler equations, aka, the

acoustics equations, are:

$$\frac{\partial p^*}{\partial t} + c_0 \left(\frac{\partial u^*}{\partial x} + \frac{\partial v^*}{\partial y} + \frac{\partial w^*}{\partial z} \right) = 0 \tag{A.1a}$$

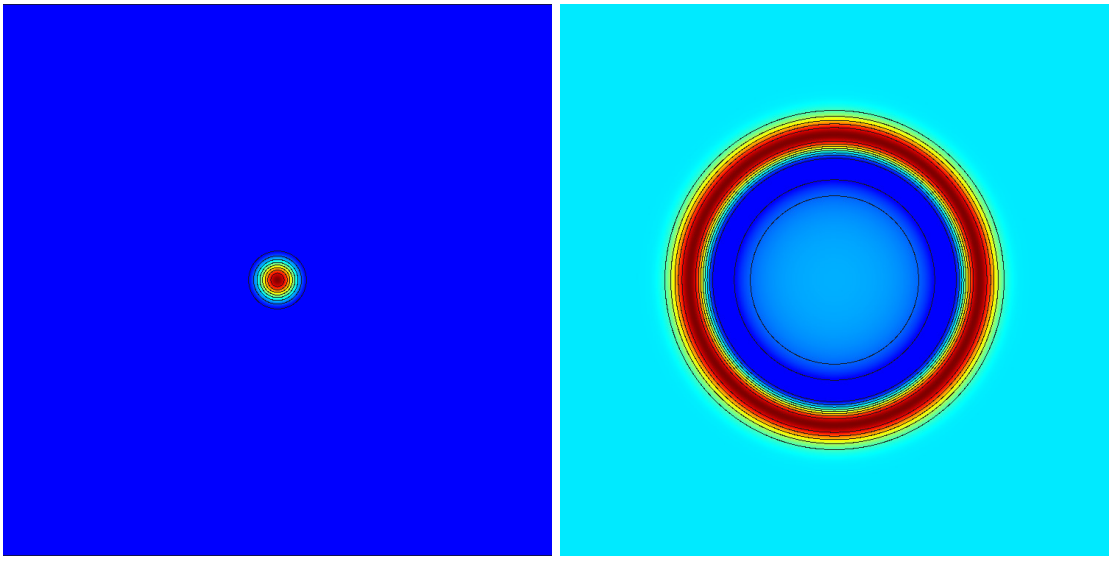
$$\frac{\partial u^*}{\partial t} + c_0 \frac{\partial p^*}{\partial x} = 0 \tag{A.1b}$$

$$\frac{\partial v^*}{\partial t} + c_0 \frac{\partial p^*}{\partial y} = 0 \tag{A.1c}$$

$$\frac{\partial w^*}{\partial t} + c_0 \frac{\partial p^*}{\partial z} = 0 \tag{A.1d}$$

In Equation A.1, the non-dimensional pressure is $p^* = p/\rho_0 c_0^2$, and the non-dimensional, velocity is $\mathbf{u}^* = \mathbf{u}/c_0$.

Figure A.1 illustrates the primal acoustics problem, evolution of a centered Gaussian pressure disturbance in a box.



(a) Initial pressure distribution, p^* (b) Final pressure distribution, p^* at time $t = 1$

Figure A.1: Linearized Euler (acoustics), Initial condition: $p^* = \exp(-50r^2), u' = 0, v' = 0$.

The output of interest in this simulation is a domain-weighted pressure integral at the final time,

$$J = \int_\Omega w(\vec{x}) p^*(\vec{x}) d\Omega, \qquad w(\vec{x}) = r^4 e^{-4r}. \tag{A.2}$$

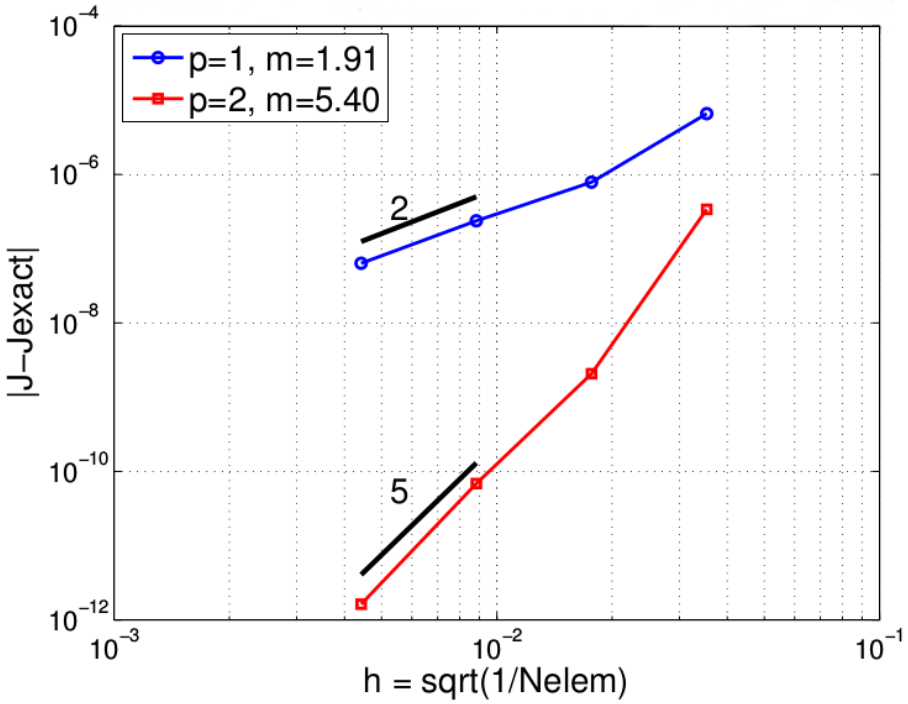


Figure A.2: Acoustics cases convergence study on right-triangular element meshes, with $N = 20, 40, 80, 160$ elements and two spatial orders, p. The truth output is obtained from $p = 3$ on 160 elements.

With this weighted pressure domain integral, An output error versus degrees of freedom convergence study is performed as shown in Figure A.2.

For the convergence in Figure A.2, we expect to observe a super-convergence rate of $2p + 1$, since this is a smooth problem with a smooth output that is a direct function of the state (not a least-squares error). However, after observing the plot, the convergence rate is 2 for $p = 1$ (suboptimal) and 5 for $p = 2$ (optimal). To better understand the observed convergence rate, a scalar advection case convergence is carried out.

A.2 A Step Back: One Dimensional Scalar Advection:

Running a similar problem with 1D scalar advection also shows suboptimality for $p = 1$ (a rate of 2). Even the $p = 2$ rate isn't exactly $2p + 1 = 5$: the observed

convergence rate is between 4 and 5 depending on the problem. More tests show that the $p = 2$ rate depends on how long the simulation ran: shorter run times produced a rate close to 4. This information suggested a problem with the initial condition.

A.2.1 One Dimensional Scalar Initial Condition

By only looking at the output computed from the initial condition, we note that the exact/analytic initial condition must be projected to the current mesh/order prior to the start of the simulation. We observe the mesh-convergence rate from a (weighted integral) of a scalar initial condition in Figure A.3

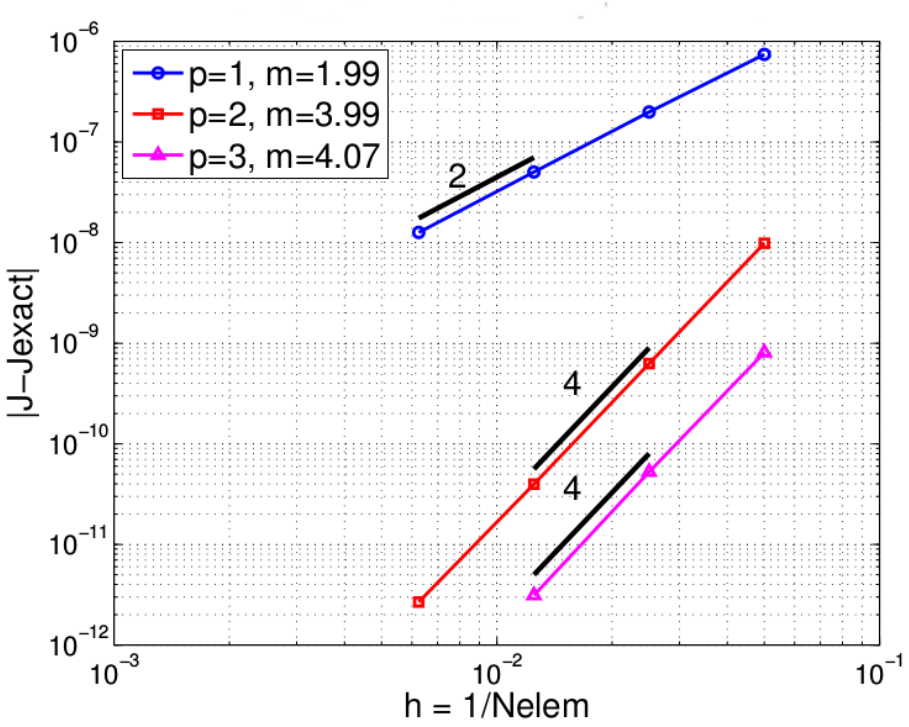


Figure A.3: One-dimensional scalar advection problem, $T = 0$, $J(u_0)$, Xflow 2015.

Figure A.3 shows an apparent rate of $p + 1$ for odd orders and a rate of $2p$ for even orders. This is a problem if we are trying to observe super-convergence later. Indeed, in our original acoustics problem, the $p = 2$ result got lucky because the super-convergent discretization error dominated this initial condition projection

error.

To demonstrate expected error from an initial condition projection, a one dimension problem is set up. The output evaluated from an exact solution, $u_{\text{exact}}(x)$, projected to order p on N elements, $u_h(x)$, is

$$\begin{aligned} J_h &= \int_0^L w(x)u_h(x)dx \\ &= \int_0^L w(x)\left[u_{\text{exact}}(x) + \delta u_h(x)\right]\,dx \\ &= J_{\text{exact}} + \int_0^L w(x)\delta u_h(x)\,dx \end{aligned} \tag{A.3}$$

where $\delta u_h(x) = u_h(x) - u_{\text{exact}}(x)$. Standard interpolation theory tells us that for an order p representation on elements of size Δx, we have $\delta u_h(x) = O(\Delta x^{p+1})$. Based on this estimate, if we don't expect any cancellation in the weighted integral, we expect the error in the output (for just the initial condition) to be

$$\delta J_h = \int_0^L w(x)\delta u_h(x)\,dx = O(\Delta x^{p+1}). \tag{A.4}$$

Undeniably, this is the rate that we observe, at least for $p = 1$ in Figure A.3 above. $p = 2$ has a higher rate because of cancellation in the projection. If we just interpolated the initial condition at the nodes, we would generally see the $p + 1$ rate for all orders.

If we were to realize a least-squares projection of the initial condition, on each element, the integral of $\delta u_h(x)$ multiplied by any polynomial of order p should be zero. So our IC output error in this case should converge as,

$$\begin{aligned} \delta J_h &= \int_0^L w(x)\delta u_h(x)\,dx \\ &= \int_0^L (w(x) - w_h(x))\delta u_h(x)\,dx \\ &= \int_0^L \delta w_h(x)\delta u_h(x)\,dx \\ &= O(\Delta x^{2p+2}) \end{aligned} \tag{A.5}$$

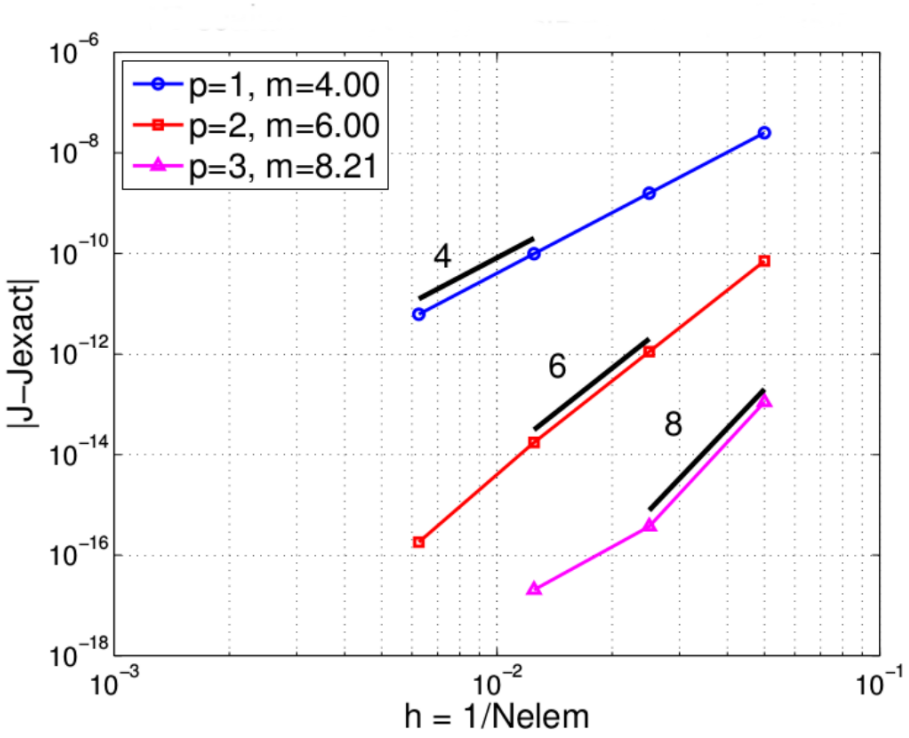


Figure A.4: One dimensional scalar, $T = 0$, $J(u_0)$ with least-squares projected initial condition.

where $w_h(x)$ is the least-squares projection of $w(x)$ onto order p polynomials on each element and $\delta w_h(x) = O(\Delta x^{p+1})$ is the resulting difference. It turns out that the initial condition routine used for these tests is not exactly doing least-squares projection. Instead, when preparing the code (Xflow) was performing nodal sampling at the Lagrange nodes without any projection. This sampling procedure for initialization is common in DG codes, but this procedure leads to a reduction in the accuracy compared to using a least-squares projection of the initial condition. Using least-squares initial condition projection gives the error convergence result in Figure A.4. After making sure the least square projection is done correctly, the super-convergence convergence rate for the initial condition is observed. Running the scalar advection cases, then, gives the correct rate of $2p + 1$.

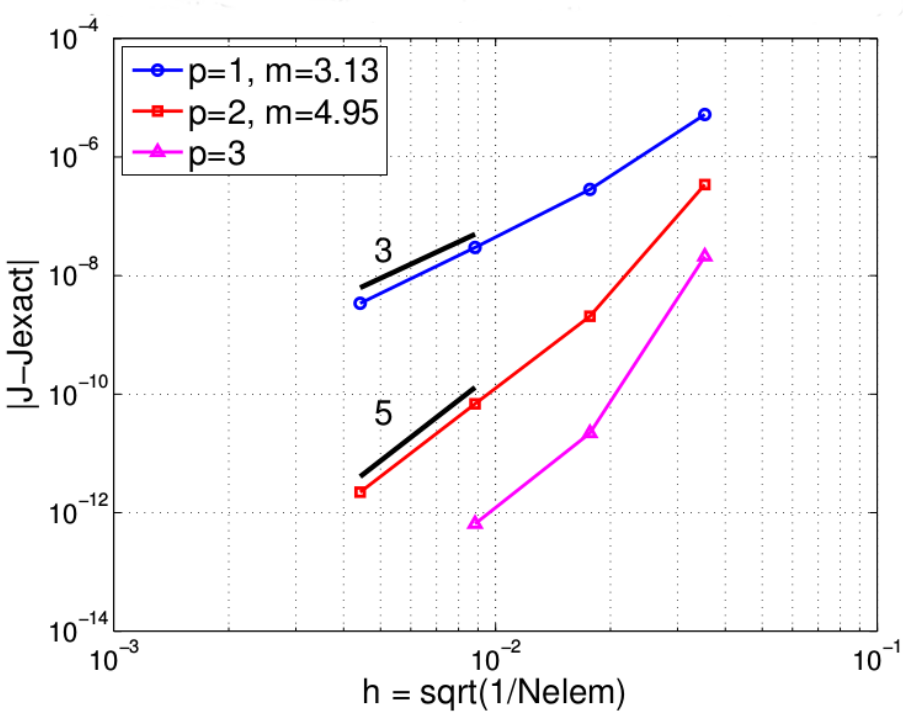


Figure A.5: Acoustics, initial condition $f = e^{-50r^2}$, T=1, $J\left(u(T)\right)$, with least square projected initial condition.

A.3 Back to Acoustics

Following the initial condition least squares discovery for the scalar advection case mentioned above, we turn back to the original acoustics problem. Running it again gives the correct $2p+1$ rates, as shown in Figure A.5.

Finally, we note that using an initial condition that is exactly representable in the space of, say, $p = 1$ (e.g. a "tent" function) also gives the optimal $2p+1$ rates, even with the slope discontinuities in the initial data, as long as these are on element interfaces. For the purpose of obtaining more accurate DG simulation result, setting up initial condition with least-square projection instead of Lagrangian nodal sampling is strongly recommended.

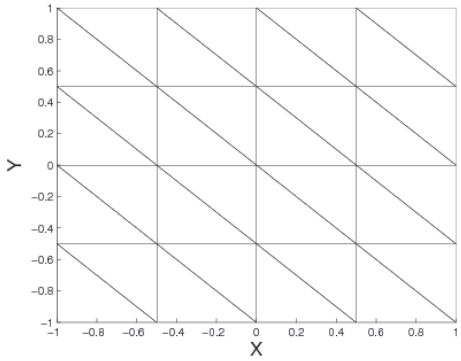
APPENDIX B

Active Flux Method Anisotropic Mesh Refinement Studies

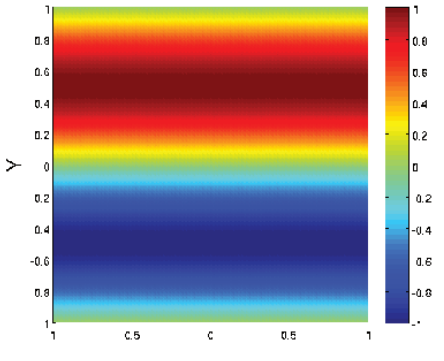
h-Adaptation adds degrees of freedom to the computational mesh. Added degrees of freedom could be evenly distributed in the adapted region, i.e., uniform adaptation, or they could be distributed in a way that favors one particular spatial direction in the adapted region, i.e., anisotropic adaptation.

Anisotropic adaptation is more computationally expensive than uniform adaptation because anisotropic adaptation requires extra computation in the decision of which spatial direction to adapt. However, anisotropic adaptation can reduce the computational cost of resolving the solution to a given accuracy, by only refining in the directions that need more resolution. To demonstrate the power of anisotropic mesh adaptation over uniform refinement for the Active Flux method, we consider a scalar advection test case, shown in Figure B.1.

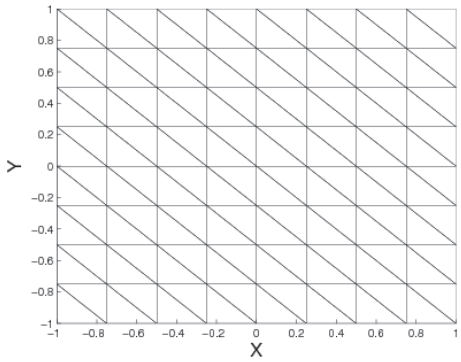
In Figure B.1, a steady scalar advection case is chosen as the test problem. The simulation is deemed converged after the residual drops seven orders of magnitude following free-stream initialization. We carry out two sequences of mesh refinement, uniform and anisotropic (elements that are long horizontally, short vertically), from the same initial mesh, Figure B.1(a). The refinement process is designed in way that upon each refinement cycle, the two refinement sequence contain same cell counts. We use the number of cells in the mesh to represent the number of degrees of free-



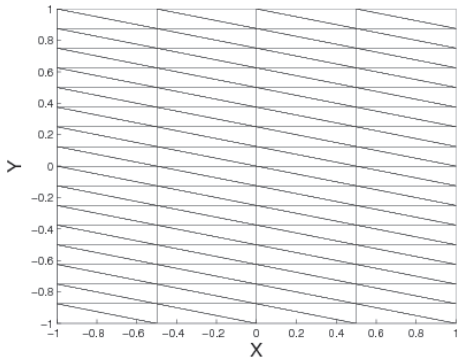
(a) Original Mesh



(b) Primal problem, $U = \sin(y\pi)$



(c) Uniformly refined mesh, second refinement cycle, DOF = 128

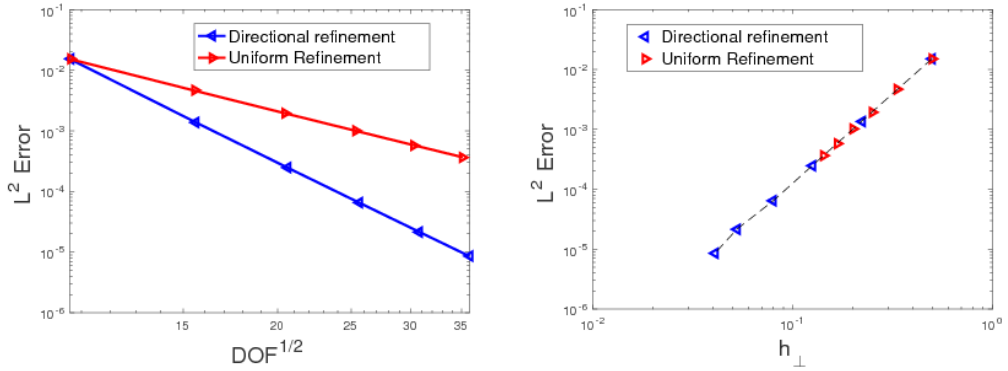


(d) Flow-aligned anisotropic mesh refinement, second refinement cycle, DOF = 128

Figure B.1: The initial uniform mesh is shown in (a). The primal solution in (b) represents horizontal advection of sinusoidal boundary data, $U = \sin(y\pi)$. (c) and (d) show meshes after two refinement cycles for uniform mesh refinement and anisotropic mesh refinement, respectively.

dom in the computational domain and denote it as, DOF. The designed sequences of DOF are as follows: $\{16 \times 2,\ 36 \times 2,\ 64 \times 2,\ 100 \times 2,\ 144 \times 2,\ 196 \times 2\}$. Figure B.1 shows two meshes, one uniform and one anisotropic, both with 128 cells.

Figure B.2 shows the L_2 solution errors for both mesh refinement sequences. In



(a) $(\text{DOF})^{1/2}$ versus L^2 error

(b) The mesh characteristic length perpendicular to the flow direction, $h_\perp$ versus L^2 error

Figure B.2: Convergence studies for the uniform mesh refinement and the anisotropic mesh refinement with different cost measurements.

Figure B.2(a), the anisotropic mesh refinement error converges at a rate of 6, while the uniform mesh refinement error converges at a rate of 3. However, when we plot the L_2 error against the mesh characteristic length perpendicular to the flow direction, Figure B.2(b), both refinements generate a slope of 3, which indicates that they both converge at third order, the convergence rate of the Active Flux scheme. This indicates that accuracy of the Active Flux scheme is dictated by the resolution of the mesh in the direction perpendicular to the flow. The direction parallel to the flow need not be as resolved, at all in this simple advection case. Consequently, if we were to perform anisotropic h-adaptation for the Active Flux scheme, we should observe a steeper error reduction with anisotropic h-adaptation than with uniform h-adaptation. Future work will consider automating the process of detecting such anisotropy for general flow fields, and generating the corresponding meshes.

BIBLIOGRAPHY

[1] Wang, Z., Fidkowski, K., Abgrall, R., Bassi, F., Caraeni, D., Cary, A., Deconinck, H., Hartmann, R., Hillewaert, K., Huynh, H., Kroll, N., May, G., Persson, P.-O., van Leer, B., and Visbal, M., "High-order CFD methods: current status and perspective," *International Journal for Numerical Methods in Fluids*, Vol. 00, 2012, pp. 1–42.

[2] Lohner, R., "An adaptive finite element scheme for transient problems in CFD," *Computer Methods in Applied Mechanics and Engineering*, Vol. 61, 1987, pp. 323–338.

[3] Dwight, R., "Heuristic *a posteriori* estimation of error due to dissipation in finite volume schemes and application to mesh adaptation," *Journal of Computatonal Physics*, Vol. 227, 2008, pp. 2845–2863.

[4] Park, M., "Adjoint-based, three-dimensional error prediction and grid adaptation," *AIAA Journal*, Vol. 42, 2004, pp. 2845–2863.

[5] Yano, M., Modisette, J. M., and Darmofal, D. L., "The importance of mesh adaptation for higher-order discretizations of aerodynamics flows," 20th AIAA Computational Fluid Dynamics Conference AIAA 2011-3852, 2011.

[6] Venkatakrishnan, V., Allmaras, S. R., Kamenetskii, D. S., and Johnson, F. T., "Higher order schemes for the compressible Navier-Stokes equations," AIAA Paper 2003-3987, 2003.

[7] Cockburn, B. and Shu, C.-W., "Runge-Kutta discontinuous Galerkin methods for convection-dominated problems," *Journal of Scientific Computing*, Vol. 16, No. 3, 2001, pp. 173–261.

[8] Nguyen, N., Peraire, J., and Cockburn, B., "An implicit high-order hybridizable discontinuous Galerkin method for linear convection-diffusion equations," *Journal of Computational Physics*, Vol. 228, 2009, pp. 3232–3254.

[9] Demkowicz, L. and Gopalakrishnan, J., "A class of discontinuous Petrov-Galerkin methods. Part I: The transport equation," *Computer Methods in Applied Mechanics and Engineering*, Vol. 199, No. 23-24, 2010, pp. 1558–1572.

[10] Hughes, T. J., Scovazzi, G., and Tezduyar, T. E., "Stabilized methods for compressible flows," *Journal of Scientific Computing*, Vol. 43, 2010, pp. 343–368.

[11] Tezduyar, T. E. and Senga, M., "Stabilization and shock-capturing parameters in supg formulation of compressible flow," *Computer Methods in Applied Mechanics and Engineering*, Vol. 195, 2006, pp. 1621–1632.

[12] Altair Engineering, "Acusolve," `https://altairhyperworks.com/product/AcuSolve`, [Online].

[13] Eymann, T. A., *Active Flux Schemes*, Ph.D. thesis, The University of Michigan, Ann Arbor, 2013.

[14] Eymann, T. A. and Roe, P. L., "Active flux schemes," 49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition 2011–382, 2011.

[15] Eymann, T. A. and Roe, P. L., "Active flux schemes for systems," 20th AIAA Computational Fluid Dynamics Conference 2011–3840, 2011.

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

[17] Ding, K., Fidkowski, K. J., and Roe, P. L., "Adjoint-based error estimation and mesh adaptation for the active flux method," 21st AIAA Computational Fluid Dynamics Conference AIAA 2013-2942, 2013.

[18] Ding, K., Fidkowski, K. J., and Roe, P. L., "Continuous adjoint based error estimation and r-refinement for the active-flux method," 54th AIAA Aerospace Sciences Meeting AIAA 2016-0832, 2016.

[19] Persson, P.-O., Bonet, J., and Peraire, J., "Discontinuous Galerkin solution of the Navier-Stokes equations on deformable domains," *Computer Methods in Applied Mechanical Engineering*, Vol. 198, 2009, pp. 1585–1595.

[20] Kast, S. M. and Fidkowski, K. J., "Output-based mesh adaptation for high order Navier-Stokes simulations on deformable domains," *Journal of Computational Physics*, Vol. 252, No. 1, 2013, pp. 468–494.

[21] Fidkowski, K. J. and Luo, Y., "Output-based space-time mesh adaptation for the compressible Navier-Stokes equations," *Journal of Computational Physics*, Vol. 230, 2011, pp. 5753–5773.

[22] Ding, K. and Fidkowski, K. J., "Output error control using r-adaptation," 23rd AIAA Computational Fluid Dynamics Conference AIAA 2013-4111, 2017.

[23] Eymann, T. A. and Roe, P. L., "Multidimensional active flux schemes," 21st AIAA Computational Fluid Dynamics Conference 2011–3840, 2013-2940.

[24] Roe, P., "Approximate Riemann solvers, parameter vectors, and difference schemes," *Journal of Computational Physics*, Vol. 43, 1981, pp. 357–372.

[25] van Leer, B., "Towards the ultimate conservative difference scheme iv. a new approach to numerical convection," *Journal of Computational Physics*, Vol. 23, 1977, pp. 276–299.

[26] Zalesak, S. T., "Fully multidimensional flux-corrected transport algorithms for fluids," *Journal of Computational Physics*, Vol. 31, 1979, pp. 335–362.

[27] Antony Jameson, L. M., "Aerodynamic shape optimization techniques based on control theory," *Computational Mathematics Driven by Industrial Problems*, edited by A. J. Rainer E. Burkard and G. Strang, Springer, 2007, pp. 151–221.

[28] de Baar, J. H., Scholcz, T. P., Verhoosel, C. V., Dwight, R. P., van Zuijlen, A. H., and Bijl, H., "Efficient uncertainty quantification with gradient-enhanced kriging: applications in FSI," European Congress on computational methods in Applied Sciences and Engineering ECCOMAS 2012, 2012.

[29] Giannakoglou, K. C. and Papadimitriou, D. I., "Adjoint methods for shape optimization," *Optimization and Computational Fluid Dynamics*, edited by D. Thevenin and G. Janiga, Springer, 2007, pp. 79–106.

[30] Asch, M., Bocquet, M., and Nodet, M., *Data assimilation: methods, algorithms, and applications*, Society for Industrial and Applied Mathematics, 2016.

[31] Zhoujie Lyu, G. K. K. and Matins, J. R. R. A., "Rans-based aerodynamic shape optimiztion investigaitons of the common research model wing," 52nd Aerospace Sciences Meeting AIAA 2014-0567, 2014.

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

[33] Hartmann, R. and Houston, P., "Adaptive discontinuous Galerkin finite element methods for the compressible Euler equations," *Journal of Computational Physics*, Vol. 183, No. 2, 2002, pp. 508–532.

[34] Giles, M. B. and Süli, E., "Adjoint methods for PDEs: a posteriori error analysis and post-processing by duality," *Acta Numerica*, Cambridge University Press,, 2003, pp. 145–236.

[35] Ding, K., Fidkowski, K. J., and Roe, P. L., "Adjoint and defect error bounding and correction for functional estimates," 16th AIAA Computational Fluid Dynamics Conference AIAA 2003-3846, 2003.

[36] Drzewiecki, T. J., *Adjoint Based Uncertainty Quantification and Sensitivity Analysis for Nuclear Thermal Fluids Codes*, Ph.D. thesis, The University of Michigan, Ann Arbor, 2013.

[37] Lala Yi Li, Y. A. and Jameson, A., "Continuous adjoint approach for adaptive mesh refinement," 20th AIAA Computational Fluid Dynamics Conference AIAA 2011-3982, 2011.

[38] Fidkowski, K. J. and Darmofal, D. L., "Review of output-based error estimation and mesh adaptation in computational fluid dynamics," *American Institute of Aeronautics and Astronautics Journal*, Vol. 49, No. 4, 2011, pp. 673–694.

[39] Duraisamy, K., Alonso, J., Palacios, F., and Chandrashekar, P., "Error estimation for high speed flows using continuous and discrete adjoints," AIAA Paper 2010-128, 2010.

[40] Fidkowski, K., "High-order output-based adaptive methods for steady and unsteady aerodynamics," *37^{th} Advanced CFD Lectures series; Von Karman Institute for Fluid Dynamics (December 9–12 2013)*, edited by H. Deconinck and R. Abgrall, von Karman Institute for Fluid Dynamics, 2013.

[41] Fidkowski, K., "Output-based error estimation and mesh adaptation for steady and unsteady flow problems," *38^{th} Advanced CFD Lectures Series; Von Karman Institute for Fluid Dynamics (September 14–16 2015)*, edited by H. Deconinck and T. Horvath, von Karman Institute for Fluid Dynamics, 2015.

[42] Arnold, D. N., Brezzi, F., Cockburn, B., and Marini, L. D., "Unified analysis of discontinuous Galerkin methods for elliptic problems," *SIAM Journal on Numerical Analysis*, Vol. 39, No. 5, 2002, pp. 1749–1779.

[43] Bassi, F. and Rebay, S., "GMRES discontinuous Galerkin solution of the compressible Navier-Stokes equations," *Discontinuous Galerkin Methods: Theory, Computation and Applications*, edited by K. Cockburn and Shu, Springer, Berlin, 2000, pp. 197–208.

[44] Saad, Y. and Schultz, M. H., "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific Computing*, Vol. 7, No. 3, 1986, pp. 856–869.

[45] Saad, Y., "A flexible inner-outer preconditioned GMRES algorithm," *SIAM Journal on Scientific Computing*, Vol. 14, No. 2, 1993, pp. 461–469.

[46] Ceze, M. A. and Fidkowski, K. J., "A robust adaptive solution strategy for high-order implicit CFD solvers," AIAA Paper 2011-3696, 2011.

[47] Lu, J., *An a Posteriori Error Control Framework for Adaptive Precision Optimization Using Discontinuous Galerkin Finite Element Method*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2005.

[48] Giles, M. B. and Pierce, N. A., "Adjoint equations in CFD: duality, boundary conditions and solution behavior," AIAA Paper 97-1850, 1997.

[49] Giles, M. and Pierce, N., "Analytic adjoint solutions for the quasi-one-dimensional Euler equations," *Journal of Fluid Mechanics*, Vol. 426, 2001, pp. 327–345.

[50] Hartmann, R., "Adjoint consistency analysis of discontinuous Galerkin discretizations," *SIAM Journal on Numerical Analysis*, Vol. 45, No. 6, 2007, pp. 2671–2696.

[51] Oliver, T. A. and Darmofal, D. L., "Impact of turbulence model irregularity on high–order discretizations," AIAA Paper 2009-953, 2009.

[52] Despres, B., "Lax theorem and finite volume schemes," *Mathematics of Computation*, Vol. 73, No. 247, 2003, pp. 1203–1234.

[53] Fan, D., *On the Acoustic Component of Active Flux Schemes for Nonlinear Hyperbolic Conservation Laws*, Ph.D. thesis, The University of Michigan, Ann Arbor, 2017.

[54] Kast, S. M., Ceze, M. A., and Fidkowski, K. J., "Output-adaptive solution strategies for unsteady aerodynamics on deformable domains," Seventh International Conference on Computational Fluid Dynamics ICCFD7-3802, 2012.

[55] Frederic Alauzet, Adrien Loseille, A. D. and Frey, P., "Multi-dimensional continuous metric for mesh adaptation," 15th international meshing roundtable, 2006.

[56] Darmofal, D. L., Allmaras, S. R., Yano, M., and Kudo, J., "An adaptive, higher-order discontinuous Galerkin finite element method for aerodynamics," AIAA paper AIAA 2013-2871, 2013.

[57] Fidkowski, K. J., "Output-based space-time mesh optimization for unsteady flows using continuous-in-time adjoints," *Journal of Computational Physics*, Vol. 341, No. 15, July 2017, pp. 258–277.

[58] Fidkowski, K. J., "A local sampling approach to anisotropic metric-based mesh optimization," AIAA Paper 2016–0835, 2016.

[59] Yano, M., *An Optimization Framework for Adaptive Higher-Order Discretizations of Partial Differential Equations on Anisotropic Simplex Meshes*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2012.

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

[61] Fidkowski, K. J., "Output error estimation strategies for discontinuous Galerkin discretizations of unsteady convection-dominated flows," *International Journal for Numerical Methods in Engineering*, Vol. 88, No. 12, 2011, pp. 1297–1322.

[62] Fidkowski, K. J., Oliver, T. A., Lu, J., and Darmofal, D. L., "p-Multigrid solution of high–order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations," *Journal of Computational Physics*, Vol. 207, 2005, pp. 92–113.

[63] Hartmann, R., "Adaptive discontinuous Galerkin methods with shock-capturing for the compressible Navier-Stokes equations," *International Journal for Numerical Methods in Fluids*, Vol. 51, No. 9–10, 2006, pp. 1131–1156.

[64] Szabo, B. A., "Estimation and control of error based on p convergence," *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, edited by I. Babuška, O. C. Zienkiewicz, J. Gago, and E. R. de Oliveira, John wiley & Sons Ltd., 1986, pp. 61–78.

[65] "Adjoint-based error estimation and adaptive mesh refinement for the RANS and $k-\omega$ turbulence model equations," *Journal of Computational Physics*, Vol. 230, No. 11, 2011, pp. 4268 – 4284.

[66] Houston, P. and Süli, E., "*hp*-adaptive discontinuous Galerkin finite element methods for first-order hyperbolic problems," *SIAM Journal on Scientific Computing*, Vol. 23, No. 4, 2001, pp. 1226–1252.

[67] Wang, L. and Mavriplis, D., "Adjoint-based $h-p$ adaptive discontinuous Galerkin methods for the 2D compressible Euler, equations," *Journal of Computational Physics*, Vol. 228, 2009, pp. 7643–7661.

[68] Burgess, N. K. and Mavriplis, D. J., "An *hp*-adaptive discontinuous Galerkin, solver for aerodynamic flows on mixed-element meshes," AIAA Paper 2011-490, 2011.

[69] Ceze, M. A. and Fidkowski, K. J., "An anisotropic hp-adaptation framework for functional prediction," *American Institute of Aeronautics and Astronautics Journal*, Vol. 51, 2013, pp. 492–509.

[70] Woopen, M., Balan, A., May, G., and Schütz, J., "A comparison of hybridized and standard DG methods for target-based hp-adaptive simulation of compressible flow," *Computers & Fluids*, Vol. 98, 2014, pp. 3–16.

[71] Bey, K. S. and Oden, J. T., "*hp*-version discontinuous Galerkin methods for hyperbolic conservation laws," *Computer Methods in Applied Mechanics and Engineering*, Vol. 133, 1996, pp. 259–286.

[72] Rannacher, R., "Adaptive Galerkin finite element methods for partial differential equations," *Journal of Computational and Applied Mathematics*, Vol. 128, 2001, pp. 205–233.

[73] Ceze, M. A. and Fidkowski, K. J., "Output-driven anisotropic mesh adaptation for viscous flows using discrete choice optimization," AIAA Paper 2010-0170, 2010.

[74] Buscaglia, G. C. and Dari, E. A., "Anisotropic mesh optimization and its application in adaptivity," *International Journal for Numerical Methods in Engineering*, Vol. 40, No. 22, November 1997, pp. 4119–4136.

[75] Wood, W. A. and Kleb, W. L., "On multi-dimensional unstructured mesh adaptation," AIAA Paper 99-3254, 1999.

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

[77] Fidkowski, K. J. and Darmofal, D. L., "A triangular cut-cell adaptive method for high-order discretizations of the compressible Navier-Stokes equations," *Journal of Computational Physics*, Vol. 225, 2007, pp. 1653–1672.

[78] Capon, P. J. and Jimack, P. K., "On the adaptive finite element solution of partial differential equations using h-r refinement," Tech. Rep. 96.03, University of Leeds, School of Computing, 1996.

[79] Bank, R. E. and Smith, R. K., "Mesh smoothing using a posteriori error estimates," *SIAM Journal on Numerical Analysis*, Vol. 34, No. 3, 1997, pp. 979–997.

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

[81] McRae, D. S., "r-refinement grid adaptation algorithms and issues," *Computer Methods in Applied Mechanics and Engineering*, Vol. 1, No. 189, 2000, pp. 1161–1182.

[82] Zegeling, P. A., "r-refinement for evolutionary PDEs with finite elements or finite differences," *Applied Numerical Mathematics*, Vol. 26, 1998, pp. 97–104.

[83] G. Beckett, J. A. Mackenzie, A. R. and Solan, D. M., "On the numerical solution of one-dimensional PDEs using adaptive methods based on equidistribution," *Journal of Computational Physics*, Vol. 167, 2001, pp. 372–392.

[84] Schmidt, J. and Stoevesandt, B., "Dynamic mesh optimization based on the spring analogy," ITM WEb of Conferences 2 03001, 2014.